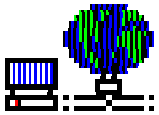
Marinetti

Version 2.0

Programmers' Guide

"For the Apple IIGS®, the world just got a whole lot closer!"

Designed and written by Richard Bennett



This revision: 20th May 2001

Contents

Acknowledgements

Sections of this document may be based on or lifted from discussions with programmers and developers who assisted in testing Marinetti during its initial and on-going development cycles, and as such, some of their copyrighted material may have accidentally been included in this document. Any use of individually copyrighted text was unintentional and purely in the spirit of making Marinetti a reality. Concerned copyright owners should contact the author to immediately resolve any conflicts.

Special thanks to Mike Westerfield for providing the headers and declarations for C, Pascal and BASIC, in this document.

Special thanks to Geoff Weiss for his continuing confidence and his initial guiding light. Marinetti exists only because of him.

Development phase testers

Tony Diaz
Dave Hecker
David Miller
Andrew Roughan
Erick Wagner
Ewen Wannop
Geoff Weiss

Specification feedback

Sönke Behrens
Joachim Lange
Devin Reade
Antoine Vignau
Mike Westerfield

Alpha testers

Jeff Blakeney
Tony Diaz
Dave Hecker
David Miller
Andrew Roughan
Erick Wagner
Ewen Wannop
Geoff Weiss
Mike Westerfield

FAQs

Ryan Suenaga
Erick Wagner (1.0)

Beta testers

Jeff Blakeney
Tony Diaz
Dave Hecker
Max Jones
Joe Kohn
David Miller
Andrew Roughan
Erick Wagner
Ewen Wannop
Tony Ward
Geoff Weiss
Mike Westerfield
Chris Vavruska

Those kind enough to email kind words about Marinetti 1.0

Rodney Abel
Cindy Adams
Jim Bauer
Jawaid Bazyar
Greg Buchne
Bruce Clark
Jon Christopher Sy Co
Art Coughlin
Dan DeDona
Jim Dwyer
Dean Esmay
Henrik Gudat
Jeremy Hack
Harold Hislop
Dave Johnson
Max Jones
James Keim
Tim Kellers
Joe Kohn
Daniel Krass
Gene Linkoski
Robert Liptak
Mark Marr-Lyon
Michael Malady
Nathan Mates
Ricardo Matinata
Mike McGovern
Kirk Mitchell
Todd Myers
Greg Nelson
Kevin Noonan
Aaron Pulver
J. Marshall Reber
Toby Reiter
Lucas Scharenbroich
Anne Schnaubelt
Paul Schultz
Kelvin Sherlock
Mitchell Spector
Ryan Suenaga
Joyce Sullivan
Greg Thompson
Gary Utter
Chris Vavruska

Introduction

Marinetti is a TCP/IP protocol suite for the Apple IIGS. It allows applications on an Apple IIGS with System 6.0.1 to connect to and interact with, an internet.

The Marinetti software is free of charge, and is available from various locations, including the Marinetti Home Page:

```
http://www.apple2.org/marinetti/
```

This document describes how to use Marinetti in your own programs, and the various tool calls which Marinetti accepts.

Using this document in conjunction with the Apple IIGS Toolbox Reference Manuals and widely available protocol RFCs, you should be able to add TCP/IP support to your Apple IIGS application.

This documentation refers to and assumes a prior knowledge of the Apple IIGS toolbox. Apple IIGS toolbox reference manuals are available from:

The Byte Works
8000 Wagon Mound Driver N.W.
Albuquerque, NM 87120
U.S.A.

505-989-4092 (fax)
505-898-8183 (voice)

MikeW50@aol.com
http://www.hypermall.com/byteworks/

This document refers only to Marinetti 2.0, and not its predecessor, 1.0.

Marinetti is in no way connected to or with the vaporware product commonly referred to as "GS/TCP", Derek Taubert's Apple IIGS port of public domain TCP/IP source code which requires GNO/ME to run and as of the date of this document, has not been released.

Summary of new features

Marinetti version 2.0 is almost a complete rewrite of the original. So much so that all applications, without exception, will need to be modified to use it.

The main changes between 1.0b1, the only public release, and 2.0, which may affect developers, include:

- The single Control Panel has had sections of code split off into a tool set stub, an init, and individually loaded link layer modules.
- Preferences and link layer configuration data are now stored in a common `TCPIP` folder inside the `System` folder.
- Marinetti now uses a toolbox interface. *An interim* `Tool054`*, which provided tool access to the version 1.0b1 requests was released to the public in December 1997, however this has now been superceded and must be overwritten with the newer version 2.0* `Tool054` *file.*
- Link layers are now separate load modules, with a documented interface for developers. Marinetti ships with a number of modules supporting various link layer types.
- Marinetti now includes Domain Name resolution, allowing applications to use domain names instead of IP addresses.
- Many calls have had their names changed to more accurately reflect what they do.
- Many calls have had their calling parameters changed.
- Better support for servers. *While both versions allow you to write server applications, in 2.0 it is more like BSD UNIX (unfortunately, but developers requested it).*
- Marinetti no longer needs to be officially registered with the author.
- Many bugs have been fixed, making Marinetti much more stable. *See the* `CHANGES` *file, which ships with Marinetti, for more details.*

Because Marinetti uses a toolbox interface, you will need to issue the tool locator call `_LoadOneTool(#54,#$200)` before using it. The tool locator calls `_StartUpTools` and `_ShutDownTools` do not support the Marinetti tool set.

Programming with Marinetti

Marinetti was written for the Apple IIGS with Apple IIGS programmers in mind. With this document, along with the supplied header and declaration files, your current development environment, and some minimal TCP/IP knowledge, you should be able to add TCP/IP facilities to your applications.

While previous knowledge of how TCP/IP and UNIX sockets work would be helpful, it is not mandatory for getting Marinetti support into your applications. This chapter should give newcomers enough information to get started, and those experienced with TCP/IP on other platforms a firm idea of how Marinetti differs from traditional UNIX socket implementations.

If you are already familiar with how TCP/IP works, you might like to skip ahead to the section “Calling Marinetti.”

What is TCP/IP

TCP/IP is actually two different pieces of software, yet they usually go together because applications on an internet use them in conjunction with one another.

The term internet, note the lower case “i”, refers to a networking protocol which allows computers to talk to each other in a fairly relaxed environment.

The term Internet, note the upper case “I”, refers to the most popular network in the world currently using the internet protocol.

An internet is usually explained using a network layer model. Although more detailed models are fairly common, this is the basic four layer model which they are derived from.

Application	Telnet, FTP, Finger, Gopher, email etc.
Transport	TCP, UDP
Network	IP — ICMP, IGMP
Link	Device driver and interface card (SLIP, PPP etc.)

On the Apple II, there is another layer at the bottom for driving the serial ports or interface card.

This layering is usually referred to as a stack, thus the terminology, TCP/IP stack.

The link layer

For two computers on an internet to communicate, they need to be connected so they can send data back and forth. This is the link layer, named so because it looks after the two computers being linked together. It could be a simple direct cable connection, or a modem to modem connection. For personal computers

using modems, the most common protocols are SLIP, the Serial Line Internet Protocol, and PPP, the Point to Point Protocol.

SLIP is the most basic form of communication, and simply sends the data it is told to send, much like a telecom program does. A newer version of SLIP, called C/SLIP, for Compressed Serial Line Internet Protocol, compresses some of the data as it is transferred. The main problem with SLIP, is that there is no handshaking for the computers to send administrative information back and forth, such as connection tuning, compression options, and IP address management, leaving the user to provide a number of key pieces of information for the whole thing to work properly.

PPP, on the other hand, provides the same serial connection as SLIP, yet it includes compression, and basic handshaking. The handshaking lets the host tell the client what its IP address is, and which compression options to use.

The network layer

The next layer up, the network layer, is the backbone of the connection.

IP, the Internet Protocol, takes packets of information, called datagrams, and sends them between the various computers on the network.

Each computer in the network is allocated an address, called an IP address. Each datagram sent by IP contains the destination computer's address. If there are a number of computers connected together, IP looks at the address in each datagram to decide which computer it is intended for.

Addresses are 32 bit numbers, but are usually expressed in the more readable dotted decimal notation, such as 255.102.10.103. Each of the four numbers contains 8 bits of the complete 32 bit address. When a datagram arrives, IP looks at the destination address in the datagram and decides whether it belongs to the computer it is running on. If not, it simply sends it on to the next computer in the network. This way datagrams hop from computer to computer until they arrive at the correct destination.

IP also has a facility where it can chop up large datagrams into fragments, which are really mini-datagrams. The fragments may take different paths to the destination, depending on the network loading, or other factors. When the fragments arrive at their destination, IP puts them all back together again as the original datagram, and the receiver never knows they were fragmented.

IP on its own is fairly unreliable, as it never knows whether a datagram it sent has actually reached its destination.

The application layer

Applications, such as email packages or Web browsers , simply open a direct path from themselves to the destination server. Imagine it as running a hose from the garden tap to the garden. Turn it on and the data comes flooding out in a continuous stream. In the case of a Web browser, we're most likely talking about an HTML document, or a GIF/JPEG.

Once made, the connection will remain until the application decides to close it, or as in the garden hose example, it turns off the tap.

For Web browsers, a connection is opened and closed for each file, so in an HTML document that contains say four GIFs, the browser would first read the HTML document, then read each of the GIFs. In fact you can see this in action in Netscape Navigatir as it draws the images as they are received. In fact, in Netscape Navigator there is a preference option for the maximum number of connections it may open simultaneously. The default is 4, as this seems to be the optimum amount on low speed serial connections.

This is also where multi-tasking or threading comes into play, as each of these is considered a separate task.

These connections are made and maintained by TCP, the Transmission Control Protocol.

The transport layer

TCP looks after management of the connection. You could think of it as the garden hose itself. It winds its way throughout the garden, delivering the continuous stream of water to its destination, making sure every drop arrives, and in the correct order.

TCP accepts a stream of data from the client or server and splits it up into segments. It then tells IP to send these segments to the destination.

IP encapsulates the segments within IP datagrams and sends them across the network. IP at the other end receives the datagrams and passes them to TCP as segments. TCP then starts rebuilding the original data stream.

As each IP datagram arrives, which may need to be rebuilt from fragments, TCP sends back another datagram which says “Yep! Got that one!” The sending TCP waits for these acknowledgement datagrams before continuing. The application however, simply sees it as a continuous stream of data.

And that’s the basics of TCP.

What makes TCP complex, is the timing. Segments, and therefore datagrams, must arrive and be acknowledged in a certain amount of time. If not, TCP resends the segment. If the acknowledgement gets lost, then the destination TCP may start receiving segments that it already has, causing unecessary overheads.

TCP also uses windowing, which means it sends a number of datagrams at once, waits for responses and then sends another bunch. The ZMODEM file transfer protocol derives its efficiency from the same windowing technique.

Because IP datagrams and fragments may take different paths to the destination, segments may also arrive at their destination out of order, and the receiving TCP must wait and reorder them for the application. Indeed the fragments of a datagram may also arrive out of order, adding another level of complexity.

Using datastreams

The one problem with the garden hose analogy, is that unlike a garden hose, TCP connections may transport data in either direction at the same time. In effect, it is two garden hoses lying next to each other.

Any application using this continuous stream approach, is said to be using TCP/IP, because TCP is managing, or transporting, the data stream, and IP is handling all the underlying network management.

What the applications do with their stream of data is completely up to them. This may mean a number of connections, say one for telling the destination what to send, and another for the actual data being returned, or could be a simple question and answer type protocol utilising a single connection.

FTP, POP3, NNTP, SMTP, are all based on this connection stuff being run by TCP.

UNIX sockets

Most implementations of TCP/IP are considered to be a part of the operating system, whether it be UNIX, Mac OS, Amiga DOS, GS/OS, or whatever. For an application to use a TCP/IP connection, it needs a way to identify it, much like a file reference number identifies individual files open by an operating system. In the UNIX world, each connection is identified by a data structure called a socket.

The term socket may be used in a number of ways, so it is important that it is understood. At its most theoretical, a socket is the TCP connection, or data stream between the two computers. For example, in the Netscape Navigator example above, it would use five sockets, with probably four of them open at once, to read in the HTML document and its four GIFs..

Another use of the term socket, is at the application runtime level. Each TCP connection knows its source and destination by using an IP address to identify the computer, and a port number to identify a specific connection on that computer. A number of connections may share a port, so another unique number is used to identify each individual connection on the port. In UNIX TCP/IP implementations, a unique id for each connection on the computer is usually generated by concatenating the port number with the unique number. This unique id is called a socket, and is used by the application to uniquely identify each connection.

Port numbers are used on servers to help identify applications. For example, if a computer is running a Web server, then it is usually accessed through port 80. It may use other ports as well, or instead, but the standard port is 80. For the Web server to identify each connection to port 80, of which there may be many, it uses the socket assigned as described above.

There are other uses of the term socket, but these are the two main ones. Because there are so many uses for the term, it is not used within Marinetti. Marinetti instead uses an `ipid` to identify individual TCP connections. It is important to note that while most implementations do use the term socket, it is not actually included in the official TCP or IP specifications. It is purely an artifact of the UNIX world.

Applications make calls to the TCP/IP stack via socket calls, which are usually kept in a socket library, along with other operating system calls. With Marinetti, these calls are implemented as a tool set.

Other transport protocols

There are over 200 different protocols which use IP for datagram delivery, including TCP, UDP and ICMP. Some of these are proprietary, and some are publicly documented in RFCs.

Each IP datagram contains an indicator byte describing the protocol the datagram conforms to. This way, each protocol may have its own receive queue, and IP knows which queue the datagram should be dropped into.

UDP, is a basic datagram delivery protocol, where the application takes care of timeouts and reordering of data. ICMP is the administrative protocol which IP uses for returning timeout and network errors.

Calling Marinetti

Marinetti is a system tool, and as such will need to be loaded before use with the toolbox tool locator call `_LoadOneTool(#54,#$200)`. Once loaded, you will need to call `_TCPIPStartUp` to initialise the tool set. The tool locator calls `_StartUpTools` and `_ShutDownTools` do not as yet support the Marinetti tool set.

Connecting to the network

The first step, is to make a network connection. To see if the network is up, the application calls `_TCPIPGetConnectStatus`, which returns a word indicating if the network is up.

If Marinetti is not yet connected to the network, the application may either make the connection itself by making a `_TCPIPConnect` call, or issue a warning dialog indicating that the network is currently down. The `_TCPIPConnect` call assumes that the user has set up Marinetti correctly using the TCP/IP CDev.

Obtaining an `ipid`

Once the network is up, the application may start making socket calls.

Each time you create a connection with a specific IP address and port number, no matter which protocol you wish to use, a 16 bit integer, called an `ipid`, is allocated by Marinetti to reference it. The `ipid` may then be used by your application to make requests to the connection, much like a GS/OS reference is used to reference files. To assign an `ipid`, you use the `_TCPIPLogin` call.

`_TCPIPLogin` accepts a number of parameters, most notably the destination IP address and port number, as well as a number of network performance variables. It returns the new `ipid` to you.

`_TCPIPLogin` also chooses a unique source port number, which can be examined using the `_TCPIPGetSourcePort` call, and changed if necessary by calling `_TCPIPSetSourcePort`. If you wish to change the source port, you must do so immediately after logging in, or network connections may fail.

Opening a TCP connection

To open a TCP connection, call _TCPIPOpenTCP. This call accepts a single input, the ipid.

_TCPIPOpenTCP simply tells Marinetti to start initiation of a connection. Keep in mind that it may take Marinetti some time to make the connection for you, depending on how busy the network is, and the speed of both the link layer and the Apple IIGS it is running on.

The application then uses the _TCPIPStatusTCP call to check if the connection has been made. When srState becomes tcpsESTABLISHED, you're done. If the state goes to tcpsCLOSED, then the connection failed.

Sending and receiving data

Once the TCP connection has been made, data may flow in either direction simultaneously, with Marinetti doing all the work for you.

To send data, the application calls _TCPIPWriteTCP. This call simply copies the data into an internal buffer, and initiates the send.

Again, the application must call _TCPIPStatusTCP to see when the data was transferred. When all the buffers have been emptied, srSndQueued will be nil.

To receive data, the application calls _TCPIPReadTCP. This call attempts to fill the user supplied buffer with data already received from the connection. The amount of data actually received is returned in rrBuffCount.

If the receive buffer cannot be completely filled, then no data is returned, unless the push flag was set by the sender.

_TCPIPReadTCP and _TCPIPReadLineTCP are the only TCP calls which immediately return with a result. All other TCP calls simply initiate an action and return.

Closing a TCP connection

If the application wishes to close a TCP connection, it calls _TCPIPCloseTCP. The close is queued, and won't be initiated until all the data in the send buffer has been sent. Once the TCP connection has been closed, _TCPIPStatusTCP will indicate an srState of tcpsCLOSED.

If the other end of the connection issues a close first, then _TCPIPStatusTCP will indicate a number of varying close states. At this stage, the application may either make additional receive calls to empty out the receive buffer, or it may issue a close of its own to force the connection to close. Once closed, again the states will vary while the close is negotiated, and eventually the state will become tcpsTIMEWAIT.

The tcpsTIMEWAIT state will remain for quite a while, and is designed to let any lost segments expire before letting this ipid open another connection. Once the time wait period has elapsed, the state automatically becomes tcpsCLOSED.

In summary, both ends must issue direct close calls before the connection will close. If one end closes, the other end is still free to receive data before it too closes the connection. Once in the `tcpsCLOSED` state, all local data and control blocks have been purged.

Releasing an `ipid`

Once the application is finished with a particular destination IP address and port, it must call `_TCPIPLogout`, to release the assigned `ipid`. You may only logout the `ipid` if the TCP connection is in the `tcpsTIMEWAIT` or `tcpsCLOSED` state.

If the state is `tcpsTIMEWAIT`, the logout is queued for later, and actioned once Marinetti notices the state becomes `tcpsCLOSED`. In this case, the `ipid` is no longer available until the socket is closed.

Disconnecting from the network

If the application made the original network connection, it may wish to disconnect from the network as well. To do this, simply call `_TCPIPDisconnect`. In order to disconnect, every `ipid` must be logged out.

How Marinetti obtains control

Marinetti depends on a number of administrative tasks running concurrently, such as handling administrative duties and control of the underlying communications. To do this, it uses a RunQ entry.

However, if the RunQ is not active, because either it has been disabled, or the Event Manager has been shut down, Marinetti will choke with a backlog of tasks and data. Data will still be received, however it will not be actioned upon. To fix this, there is a call named `_TCPIPPoll`, which the application should issue as often as possible. `_TCPIPPoll` checks the various pending Marinetti tasks and performs a set number of iterations of each, so the more often `_TCPIPPoll` is called, the faster the system throughput.

The standard way of calling `_TCPIPPoll`, is to simply add one `_TCPIPPoll` call inside the application's main event loop, to be called when you receive a null event from the Event Manager (or Task Master).

However, it is much easier to simply let the RunQ task do everything for you. In fact, you can even issue `_TCPIPPoll` calls while the RunQ task is active, if you really wish to speed up throughput.

Finally, because different Apple IIGS systems have different speeds and loads, there are a number of tuning parameters available using the `_TCPIPGetTuningTable` and `_TCPIPSetTuningTable` calls.

Information on internet protocols (RFCs)

Protocol specifications are usually presented to the Internet public via RFCs, or Request For Comment documents. These documents are numbered and may be found on the InterNIC mail server.

To retrieve an RFC, send an email message to:

```
mailserv@ds.internic.net
```

Before replacing the `xxx` with the number of the RFC you wish to retrieve, the content of your message should read:

```
file /ftp/rfc/rfcxxx.txt
```

Alternatively, you could use one of the RFC HyperText Archives for search and retrieval, with all the RFCs interlinked using HTML. I tend to use one of the mirror sites, such as the following:

```
http://sunsite.auc.dk/RFC/
```

Here is a list of current RFCs for a number of internet protocols. These are by no means all.

RFC977	NNTP	Network News Transfer Protocol
RFC1939	POP3	Post Office Protocol - Version 3
RFC959	FTP	File Transfer Protocol
RFC821	SMTP	Simple Mail Transfer Protocol
RFC854		Telnet Protocol

Here are some of the Telnet negotiated option RFCs:

RFC856	Binary (8 bit)
RFC857	Echo
RFC858	Suppress go ahead
RFC859	Status
RFC860	Timing Mark
RFC1073	Window size
RFC1079	Terminal speed
RFC1091	Terminal type
RFC1184	Linemode
RFC1372	Remote flow control
RFC1408	Environment variables

Housekeeping tool calls

The following tool calls are mandatory tool locator calls.

TCPIPBootInit $0136

Initialises Marinetti.

▲ **Warning** This call must not be made by an application. ▲

Parameters

The stack is not affected by this call.

Errors None

BASIC
```
SUB TCPIPBootInit
```

C
```
extern pascal void TCPIPBootInit (void);
```

Pascal
```
procedure TCPIPBootInit;
```

TCPIPStartUp $0236

Starts Marinetti for use by an application. This call must be made by the application before making any other calls to Marinetti.

Parameters

The stack is not affected by this call.

Errors None

BASIC
```
SUB TCPIPStartUp
```

C
```
extern pascal void TCPIPStartUp (void);
```

Pascal
```
procedure TCPIPStartUp;
```

TCPIPShutDown $0336

Shuts down Marinetti, once an application has finished with it.

Parameters

The stack is not affected by this call.

Errors None

BASIC
```
SUB TCPIPShutDown
```

C
```
extern pascal void TCPIPShutDown (void);
```

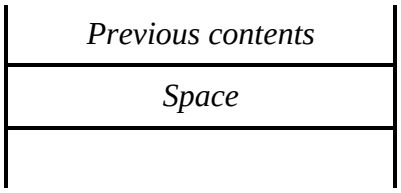
Pascal
```
procedure TCPIPShutDown;
```

TCPIPVersion $0436

Returns the Marinetti version number. For Marinetti 2.0, the version returned is $0200.

Parameters

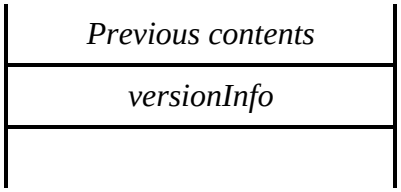
Stack before call



Word — Space for result

<— SP

Stack after call



Word — Version information

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPVersion as %
```

C
```
extern pascal Word TCPIPVersion (void);
```

Pascal
```
function  TCPIPVersion: integer;
```

TCPIPReset $0536

Resets Marinetti.

▲ **Warning** This call must not be made by an application. ▲

Parameters

The stack is not affected by this call.

Errors None

BASIC
```
SUB TCPIPReset
```

C
```
extern pascal void TCPIPReset (void);
```

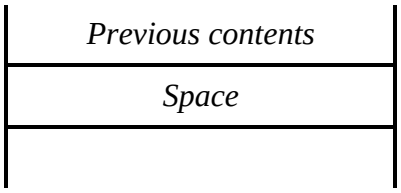
Pascal
```
procedure TCPIPReset;
```

TCPIPStatus $0636

Returns a boolean flag indicating whether or not Marinetti is active.

Parameters

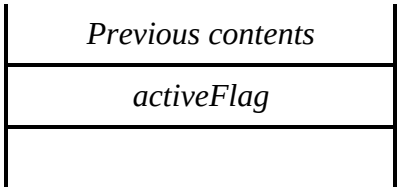
Stack before call



Word — Space for result

<— SP

Stack after call



Word — Boolean; TRUE if Marinetti is active

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPStatus as %
```

C
```
extern pascal Boolean TCPIPStatus (void);
```

Pascal
```
function  TCPIPStatus: boolean;
```

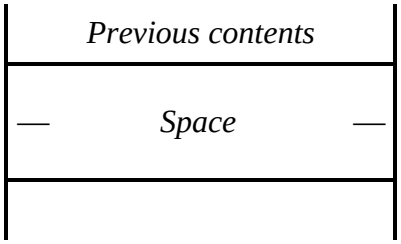
`activeFlag` The value returned is TRUE (non-zero) if Marinetti is active, and FALSE ($0000) if it is not.

TCPIPLongVersion $0836

Returns the Marinetti rVersion number.

Parameters

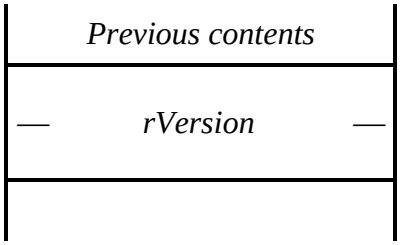
Stack before call



Long — Space for result

<— **SP**

Stack after call



Long — rVersion

<— **SP**

Errors None.

BASIC
```
FUNCTION TCPIPLongVersion as &
```

C
```
extern pascal Long TCPIPLongVersion (void);
```

Pascal
```
function  TCPIPLongVersion: longint;
```

Administrative tool calls

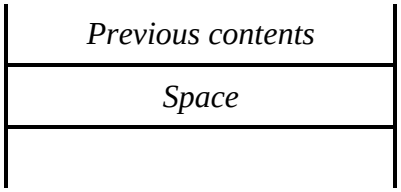
The following calls deal with specific Marinetti administrative tasks.

TCPIPGetConnectStatus $0936

Asks Marinetti if it is currently connected to the network.

Parameters

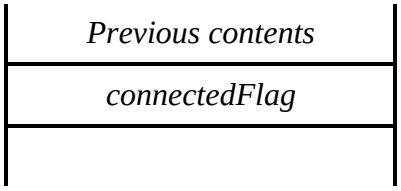
Stack before call



Word — Space for result

<— SP

Stack after call



Word — Boolean; TRUE if currently connected

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPGetConnectStatus as %
```

C
```
extern pascal Boolean TCPIPGetConnectStatus (void);
```

Pascal
```
function  TCPIPGetConnectStatus: boolean;
```

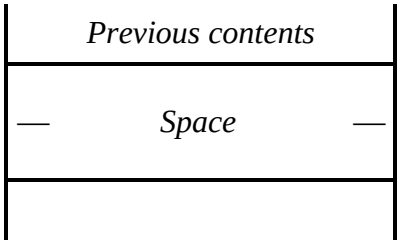
`connectedFlag` The value returned is TRUE (non-zero) if Marinetti is currently connected to the network, and FALSE ($0000) if it is not.

TCPIPGetErrorTable $0A36

Returns a pointer to a list of longwords, Marinetti's error table.

Parameters

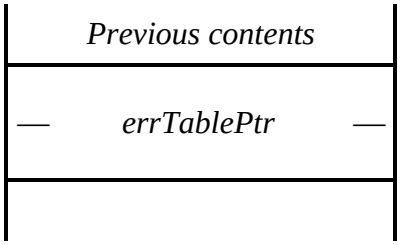
Stack before call



Long — Space for result

<— **SP**

Stack after call



Long — Pointer to error table

<— **SP**

Errors None.

BASIC

```
FUNCTION TCPIPGetErrorTable as errTablePtr

type errTable
    tcpDGMSTBLEN as long
    tcpDGMSTOTAL as long
    tcpDGMSFRAGSIN as long
    tcpDGMSFRAGSLOST as long
    tcpDGMSBUILT as long

    tcpDGMSOK as long

    tcpDGMSBADCHK as long
    tcpDGMSBADHEADLEN as long
    tcpDGMSBADPROTO as long
    tcpDGMSBADIP as long

    tcpDGMSICMP as long
    tcpDGMSICMPUSER as long
    tcpDGMSICMPKERNEL as long

    tcpDGMSICMPBAD as long
    tcpDGMSICMPBADTYPE as long
    tcpDGMSICMPBADCODE as long
```

```
                tcpDGMSICMPECHORQ as long
                tcpDGMSICMPECHORQOUT as long
                tcpDGMSICMPECHORP as long
                tcpDGMSICMPECHORPBADID as long

                tcpDGMSUDP as long
                tcpDGMSUDPBAD as long
                tcpDGMSUDPNOPORT as long

                tcpDGMSTCP as long
                tcpDGMSTCPBAD as long
                tcpDGMSTCPNOPORT as long
                tcpDGMSTCPQUEUED as long
                tcpDGMSTCPOLD as long

                tcpDGMSOFRAGMENTS as long
                tcpDGMSFRAGMENTED as long
            end type
            type errTablePtr as pointer to errTable

C           extern pascal errTablePtr TCPIPGetErrorTable (void);

            typedef struct {
                long tcpDGMSTBLEN;
                long tcpDGMSTOTAL;
                long tcpDGMSFRAGSIN;
                long tcpDGMSFRAGSLOST;
                long tcpDGMSBUILT;

                long tcpDGMSOK;

                long tcpDGMSBADCHK;
                long tcpDGMSBADHEADLEN;
                long tcpDGMSBADPROTO;
                long tcpDGMSBADIP;

                long tcpDGMSICMP;
                long tcpDGMSICMPUSER;
                long tcpDGMSICMPKERNEL;

                long tcpDGMSICMPBAD;
                long tcpDGMSICMPBADTYPE;
                long tcpDGMSICMPBADCODE;
                long tcpDGMSICMPECHORQ;
                long tcpDGMSICMPECHORQOUT;
                long tcpDGMSICMPECHORP;
                long tcpDGMSICMPECHORPBADID;

                long tcpDGMSUDP;
                long tcpDGMSUDPBAD;
                long tcpDGMSUDPNOPORT;
```

```
    long tcpDGMSTCP;
    long tcpDGMSTCPBAD;
    long tcpDGMSTCPNOPORT;
    long tcpDGMSTCPQUEUED;
    long tcpDGMSTCPOLD;

    long tcpDGMSOFRAGMENTS;
    long tcpDGMSFRAGMENTED;
    } errTable, *errTablePtr;
```

Pascal

```
function  TCPIPGetErrorTable: errTablePtr;

errTable = record
   tcpDGMSTBLEN: longint;
   tcpDGMSTOTAL: longint;
   tcpDGMSFRAGSIN: longint;
   tcpDGMSFRAGSLOST: longint;
   tcpDGMSBUILT: longint;

   tcpDGMSOK: longint;

   tcpDGMSBADCHK: longint;
   tcpDGMSBADHEADLEN: longint;
   tcpDGMSBADPROTO: longint;
   tcpDGMSBADIP: longint;

   tcpDGMSICMP: longint;
   tcpDGMSICMPUSER: longint;
   tcpDGMSICMPKERNEL: longint;

   tcpDGMSICMPBAD: longint;
   tcpDGMSICMPBADTYPE: longint;
   tcpDGMSICMPBADCODE: longint;
   tcpDGMSICMPECHORQ: longint;
   tcpDGMSICMPECHORQOUT: longint;
   tcpDGMSICMPECHORP: longint;
   tcpDGMSICMPECHORPBADID: longint;

   tcpDGMSUDP: longint;
   tcpDGMSUDPBAD: longint;
   tcpDGMSUDPNOPORT: longint;

   tcpDGMSTCP: longint;
   tcpDGMSTCPBAD: longint;
   tcpDGMSTCPNOPORT: longint;
   tcpDGMSTCPQUEUED: longint;
   tcpDGMSTCPOLD: longint;

   tcpDGMSOFRAGMENTS: longint;
   tcpDGMSFRAGMENTED: longint;
   end;
errTablePtr = ^errTable;
```

`errTablePtr` The value returned is a pointer to the error table. The error table is read only, and is provided for reference only.

The currently defined error table offsets are:

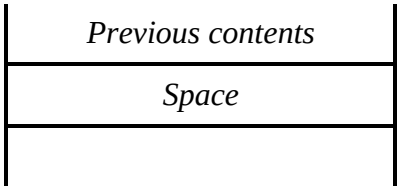
`tcpDGMSTBLEN`	+0000	The total length of the error table, in bytes, including `tcpDGMSTBLEN`
`tcpDGMSTOTAL`	+0004	Total datagrams received (good and bad)
`tcpDGMSFRAGSIN`	+0008	Got a fragment (datagram is queued to frag list)
`tcpDGMSFRAGSLOST`	+0012	Fragment purged after timeout in queue
`tcpDGMSBUILT`	+0016	Built a datagram from fragments (is then queued)
`tcpDGMSOK`	+0020	Datagrams queued from link or `tcpDGMSBUILT`
`tcpDGMSBADCHK`	+0024	Bad IP checksum (datagram is purged)
`tcpDGMSBADHEADLEN`	+0028	Bad IP header lengths (datagram is purged)
`tcpDGMSBADPROTO`	+0032	Unsupported protocols (added to misc queue)
`tcpDGMSBADIP`	+0036	Not my or loopback IP (datagram is purged)
`tcpDGMSICMP`	+0040	ICMP total datagrams in (good and bad)
`tcpDGMSICMPUSER`	+0044	ICMP user datagrams
`tcpDGMSICMPKERNEL`	+0048	ICMP kernel datagrams
`tcpDGMSICMPBAD`	+0052	ICMP bad checksum or datagram too short
`tcpDGMSICMPBADTYPE`	+0056	ICMP bad ic_type
`tcpDGMSICMPBADCODE`	+0060	ICMP bad ic_code
`tcpDGMSICMPECHORQ`	+0064	ICMP ECHORQs in
`tcpDGMSICMPECHORQOUT`	+0068	ICMP ECHORQ replies sent out
`tcpDGMSICMPECHORP`	+0072	ICMP ECHORPs in
`tcpDGMSICMPECHORPBADID`	+0076	ICMP ECHORPs unclaimed
`tcpDGMSUDP`	+0080	UDPs OK (added to UDP queue)
`tcpDGMSUDPBAD`	+0084	Bad UDP header (datagram is purged)
`tcpDGMSUDPNOPORT`	+0088	No such logged in port (datagram is purged)
`tcpDGMSTCP`	+0092	TCPs OK (returned to TCP main logic)
`tcpDGMSTCPBAD`	+0096	Bad TCP header or checksum (datagram is purged)
`tcpDGMSTCPNOPORT`	+0100	No such logged in port (datagram is purged)
`tcpDGMSTCPQUEUED`	+0104	Arrived before required (datagram is queued)
`tcpDGMSTCPOLD`	+0108	Already received this segment (datagram is purged)
`tcpDGMSOFRAGMENTS`	+0112	Fragments transmitted
`tcpDGMSFRAGMENTED`	+0116	Datagrams fragmented for transmission

TCPIPGetReconnectStatus $0B36

Asks Marinetti if there is enough information for it to dynamically reconnect to the network.

Parameters

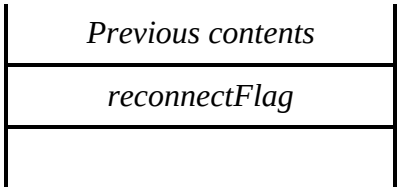
Stack before call



Word — Space for result

<— SP

Stack after call



Word — Boolean; TRUE if reconnect possible

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPGetReconnectStatus as %
```

C
```
extern pascal Boolean TCPIPGetReconnectStatus (void);
```

Pascal
```
function  TCPIPGetReconnectStatus: boolean;
```

`reconnectFlag` The value returned is TRUE (non-zero) if Marinetti has enough information for it to reconnect to the network, and FALSE ($0000) if it has not.

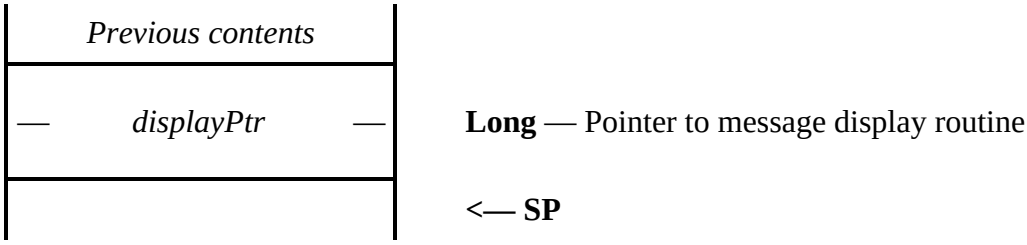
TCPIPReconnect $0C36

If the Apple IIGS crashes, or for whatever reason needs to be reboot, then Marinetti provides a reconnect facility, so it can dynamically reconnect without having to re-dial or renegotiate the connection, depending upon the connect method.

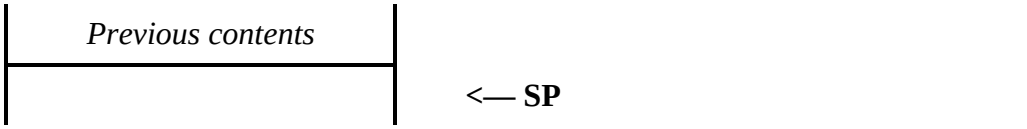
Reconnection assumes that there is enough internally saved information for Marinetti to reconnect (*see the* `TCPIPReconnectData` $0B36 *call)*, such as a modem or similar connection device still being connected to the network, as well as Marinetti link management variables, which may have been saved to disk before the crash.

Parameters

Stack before call



Stack after call



Errors	`terrLINKERROR`	There was an error with the link. In the case of the built in serial ports, they may be in use by another product or AppleTalk
	`terrCONNECTED`	Marinetti is already connected
	`terrNORECONDATA`	There is no reconnect data available to perform the reconnect
	`terrLINKBUSY`	Modem or interface is busy
	`terrNOLINKINTERFACE`	No dial tone or similar
	`terrNOLINKRESPONSE`	No modem answer or similar

BASIC

```
FUNCTION TCPIPReconnect as displayPtr
```

C

```
extern pascal displayPtr TCPIPReconnect (void)
```

Pascal

```
function  TCPIPReconnect: displayPtr;
```

`displayPtr` This routine is called by Marinetti with various pstrings for display during the reconnection process. The routine must be available to be called for the duration of the `TCPIPReconnect` call. If you do not wish to display reconnection messages, pass `displayPtr` as `nil`.

The routine is called in full native, with 16 bit accumulator and index registers. The accumulator, index registers, data bank and direct page registers are undefined on entry. The data bank and direct page registers must be restored on exit.The pointer to the pstring is on the stack, and must be removed before returning.

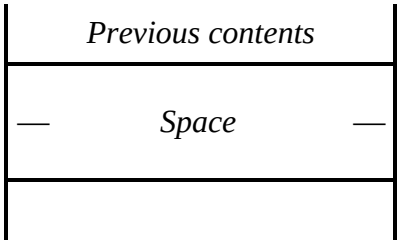
◆NOTE: *Currently, for connections using the serial port, only a 19200 baud connection may be reconnected to. This request was added mainly as a developer facility, and corners were cut to provide it.*

TCPIPGetMyIPAddress $0F36

Returns Marinetti's IP address.

Parameters

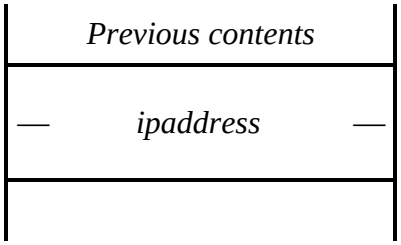
Stack before call



Long — Space for result

<— **SP**

Stack after call



Long — The IP address

<— **SP**

Errors `terrNOCONNECTION` Not currently connected

BASIC
```
FUNCTION TCPIPGetMyIPAddress as &
```

C
```
extern pascal Long TCPIPGetMyIPAddress (void);
```

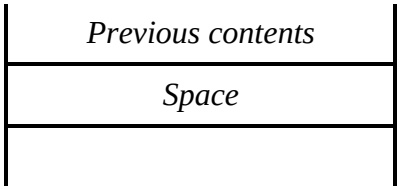
Pascal
```
function  TCPIPGetMyIPAddress: longint;
```

TCPIPGetConnectMethod $1036

Returns the current method which Marinetti is using, or will use to connect to the network.

Parameters

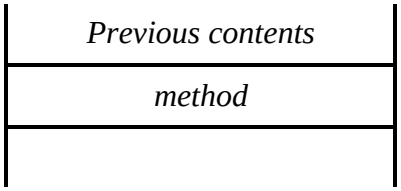
Stack before call



Word — Space for result

<— SP

Stack after call



Word — Current connect method

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPGetConnectMethod as %
```

C
```
extern pascal Word TCPIPGetConnectionMethod (void);
```

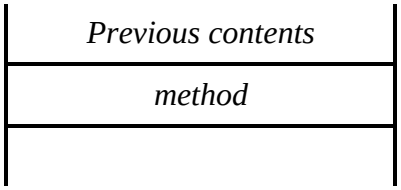
Pascal
```
function  TCPIPGetConnectionMethod: integer;
```

TCPIPSetConnectMethod $1136

Tells Marinetti the default connect method to use.

Parameters

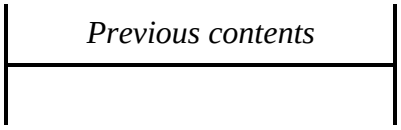
Stack before call



Word — The new connect method

<— SP

Stack after call



<— SP

Errors	`terrCONNECTED`	Marinetti is already connected

BASIC

```
SUB TCPIPSetConnectMethod (%)
```

C

```
extern pascal void TCPIPSetConnectionMethod (Word);
```

Pascal

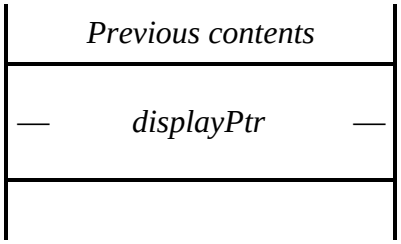
```
procedure TCPIPSetConnectionMethod (method: integer);
```

TCPIPConnect $1236

Tells Marinetti to connect to the network, using the current connect method.

Parameters

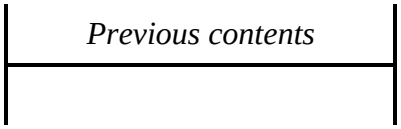
Stack before call



Long — Pointer to message display routine

<— SP

Stack after call



<— SP

Errors	`terrSCRIPTFAILED`	The connect script failed
	`terrLINKERROR`	There was an error with the link. In the case of the built in serial ports, they may be in use by another product or AppleTalk
	`terrCONNECTED`	Marinetti is already connected
	`terrNOLINKLAYER`	Unable to load link layer module for the selected connect method
	`terrBADLINKLAYER`	Not a link layer module
	`terrUSERABORTED`	The user aborted the connect
	`terrLINKBUSY`	Modem or interface is busy
	`terrNOLINKINTERFACE`	No dial tone or similar
	`terrNOLINKRESPONSE`	No modem answer or similar

BASIC

```
FUNCTION TCPIPConnect as displayPtr
```

C

```
extern pascal displayPtr TCPIPConnect (void);
```

Pascal

```
function  TCPIPConnect: displayPtr;
```

`displayPtr`

This routine is called by Marinetti with various pstrings for display during the connection process. The routine must be available to be called for the duration of the `TCPIPConnect` call. If you do not wish to display connection messages, pass `displayPtr` as `nil`.

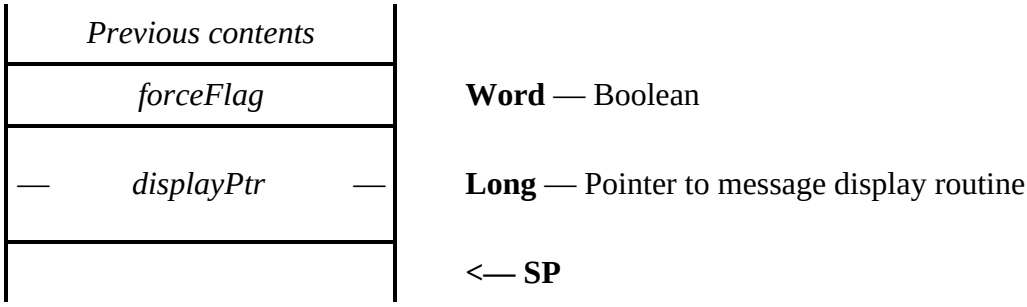
The routine is called in full native, with 16 bit accumulator and index registers. The accumulator, index registers, data bank and direct page registers are undefined on entry. The data bank and direct page registers must be restored on exit.The pointer to the pstring is on the stack, and must be removed before returning.

TCPIPDisconnect $1336

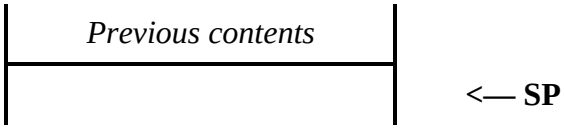
Tells Marinetti to disconnect from the network.

Parameters

Stack before call



Stack after call



Errors	`terrSCRIPTFAILED`	The connect script failed
	`terrLINKERROR`	There was an error with the link. In the case of the built in serial ports, they may be in use by another product or AppleTalk
	`terrNOCONNECTION`	Not currently connected
	`terrLOGINSPENDING`	There are still `ipid`s logged in
	`terrUSERABORTED`	The user aborted the disconnect
	`terrLINKBUSY`	Modem or interface is busy
	`terrNOLINKINTERFACE`	No dial tone or similar
	`terrNOLINKRESPONSE`	No modem answer or similar

BASIC

```
SUB TCPIPDisconnect (%, displayPtr)
```

C

```
extern pascal void TCPIPDisconnect (Boolean,
                        displayPtr);
```

Pascal

```
procedure TCPIPDisconnect (forceFlag: boolean; dPtr:
                        displayPtr);
```

`forceFlag` Ordinarily, the `TCPIPDisconnect` call will not disconnect unless every `ipid` has been logged out. This is so as not to interrupt network tasks waiting to be serviced. Remember, there may be more than one application running at a time, as well as NDAs and CDAs, which may be using the network as well. However, if the user knows that the pending `ipid`s are either hung or can be forced, set this flag to `true` and `TCPIPDisconnect` will forceably disconnect from the network.

Normal procedure would be to issue `TCPIPDisconnect` with `forceFlag` set to `false`. If a `terrLOGINSPENDING` error is returned, double check which `ipid`s are still logged in, or ask the user if they wish to continue, then if all is OK, issue `TCPIPDisconnect` again with `forceFlag` set to `true`.

`displayPtr` This routine is called by Marinetti with various pstrings for display during the disconnection process. The routine must be available to be called for the duration of the `TCPIPDisconnect` call. If you do not wish to display disconnection messages, pass `displayPtr` as `nil`.

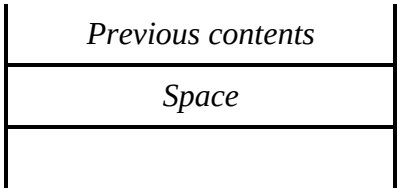
The routine is called in full native, with 16 bit accumulator and index registers. The accumulator, index registers, data bank and direct page registers are undefined on entry. The data bank and direct page registers must be restored on exit.The pointer to the pstring is on the stack, and must be removed before returning.

TCPIPGetMTU $1436

Returns the current MTU (Maximum Transmission Unit), or the maximum IP datagram size. This value is set by the link layer module once it knows the host MRU (Maximum Receive Unit) size.

Parameters

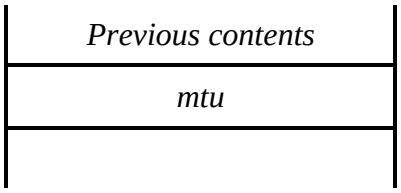
Stack before call



Word — Space for result

<— SP

Stack after call



Word — Maximum Transmission Unit size

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPGetMTU as %
```

C
```
extern pascal Word TCPIPGetMTU (void);
```

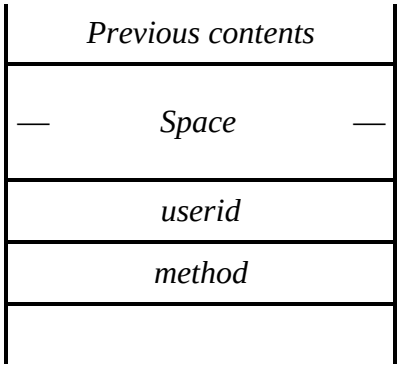
Pascal
```
function  TCPIPGetMTU: integer;
```

TCPIPGetConnectData $1636

Returns the connect data for the specified connect method.

Parameters

Stack before call



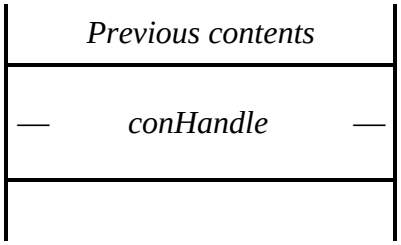
Long — Space for result

Word — userID to use with `NewHandle`

Word — Connect method to return

<— SP

Stack after call



Long — Handle to the connect data

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPGetConnectData (%, %) as conHandle
```

C
```
extern pascal conHandle TCPIPGetConnectData (Word,
                        Word);
```

Pascal
```
function  TCPIPGetConnectData (userid: integer; method:
                        integer): conHandle;
```

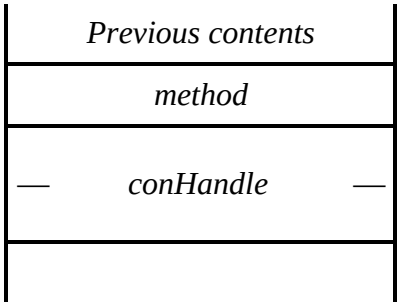
`conHandle` The returned handle is now owned by the `userid` which was passed from `dataIn`. Marinetti no longer owns or keeps track of this handle.

TCPIPSetConnectData $1736

Passes Marinetti the connect data for the specified connect method.

Parameters

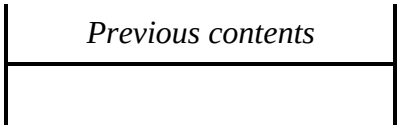
Stack before call



Word — Connect method of connect data to set

Long — Handle to the connect data

Stack after call



Errors None.

BASIC
```
SUB TCPIPSetConnectData (%, conHandle)
```

C
```
extern pascal void TCPIPSetConnectData (Word,
                        conHandle);
```

Pascal
```
procedure TCPIPSetConnectData (method: integer; cHand:
                        conHandle);
```

`conHandle` Handle containing the connect data

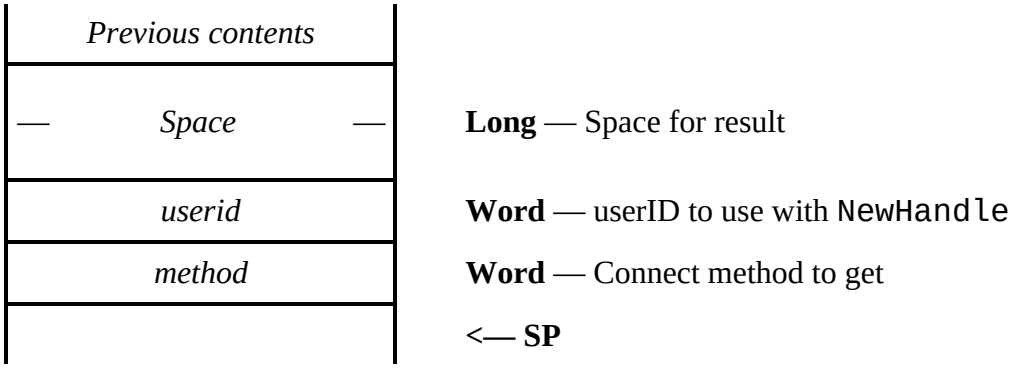
Once passed, the handle is owned by Marinetti and you must not perform any more actions on it.

TCPIPGetDisconnectData $1836

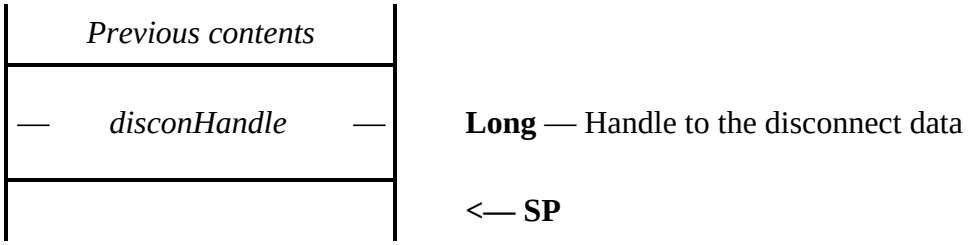
Returns the disconnect data for the specified connect method.

Parameters

Stack before call



Stack after call



Errors None.

BASIC
```
FUNCTION TCPIPGetDisconnectData (%, %) as disconHandle
```

C
```
extern pascal disconHandle TCPIPGetDisconnectData
                        (Word, Word);
```

Pascal
```
function  TCPIPGetDisconnectData (userid: integer;
                        method: integer): disconHandle;
```

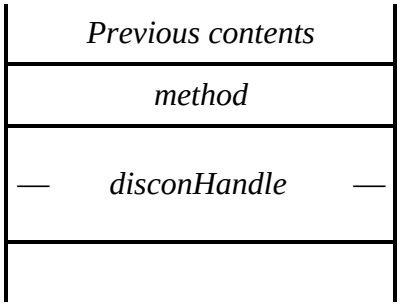
`disconHandle` The returned handle is now owned by the `userid` which was passed from `dataIn`. Marinetti no longer owns or keeps track of this handle.

TCPIPSetDisconnectData $1936

Passes Marinetti the disconnect data for the specified disconnect method.

Parameters

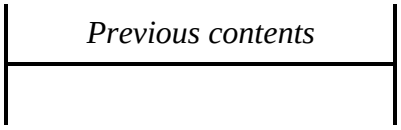
Stack before call



Word — Connect method of disconnect data to set

Long — Handle to the disconnect data

Stack after call



Errors None.

BASIC
```
SUB TCPIPSetDisconnectData (%, disconHandle)
```

C
```
extern pascal void TCPIPSetDisconnectData (Word,
                        disconHandle);
```

Pascal
```
procedure TCPIPSetDisconnectData (userid: integer;
                        dHand: disconHandle);
```

`disconHandle` Handle containing the disconnect data.

Once passed, the handle is owned by Marinetti and you must not perform any more actions on it.

TCPIPLoadPreferences $1A36

Loads the default preferences from disk.

Parameters

The stack is not affected by this call.

Errors None

BASIC
```
SUB TCPIPLoadPreferences
```

C
```
extern pascal void TCPIPLoadPreferences (void);
```

Pascal
```
procedure TCPIPLoadPreferences;
```

TCPIPSavePreferences $1B36

Saves the default preferences to disk. If you wish to make a changes to preferences permanent, you must make this call.

Parameters

The stack is not affected by this call.

Errors None

BASIC
```
SUB TCPIPSavePreferences
```

C
```
extern pascal void TCPIPSavePreferences (void);
```

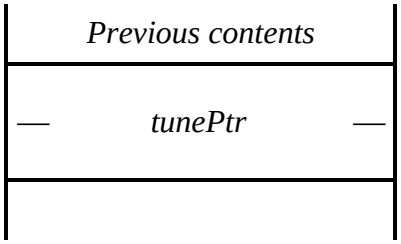
Pascal
```
procedure TCPIPSavePreferences;
```

TCPIPGetTuningTable $1E36

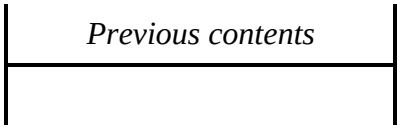
Returns the current tuning table.

Parameters

Stack before call



Stack after call



Errors None.

BASIC

```
FUNCTION TCPIPGetTuningTAble as tunePtr

type tuneRecord
    tcpTUNECOUNT as integer
    tcpTUNEIPUSERPOLLCT as integer
    tcpTUNEIPRUNQFREQ as integer
    tcpTUNEIPRUNQCT as integer
    tcpTUNETCPUSERPOLL as integer
end type
type tunePtr as pointer to tuneRecord
```

C

```
extern pascal tunePtr TCPIPGetTuningTable (void);

typedef struct {
    Word tcpTUNECOUNT;
    Word tcpTUNEIPUSERPOLLCT;
    Word tcpTUNEIPRUNQFREQ;
    Word tcpTUNEIPRUNQCT;
    Word tcpTUNETCPUSERPOLL;
    } tuneStruct, *tunePtr;
```

Pascal

```
function  TCPIPGetTuningTable: tunePtr;

tuneRecord = record
   tcpTUNECOUNT: integer;
   tcpTUNEIPUSERPOLLCT: integer;
   tcpTUNEIPRUNQFREQ: integer;
```

```
   tcpTUNEIPRUNQCT: integer;
   tcpTUNETCPUSERPOLL: integer;
   end;
tunePtr = ^tuneRecord;
```

`tunePtr` Points to a 10 byte buffer where the tuning table is to be returned.

The currently defined tuning table offsets are:

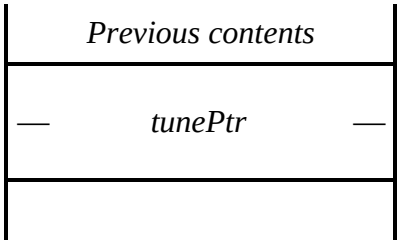
`tcpTUNECOUNT`	+0000	The total length of the tuning table, in bytes, including `tcpTUNECOUNT`. Currently 10.
`tcpTUNEIPUSERPOLLCT`	+0002	The number of datagrams Marinetti will build per `TCPIPPoll` request. The valid range is 1 through 10 inclusive. The default is 2.
`tcpTUNEIPRUNQFREQ`	+0004	The RunQ frequency value (60ths of a second). The default is 30 (half a second).
`tcpTUNEIPRUNQCT`	+0006	The number of datagrams Marinetti will build per RunQ dispatch. The valid range is 1 through 10 inclusive. The default is 2.
`tcpTUNETCPUSERPOLL`	+0008	The TCP steps to perform per user, per `TCPIPPoll` request and RunQ dispatch. The valid range is 1 through 10 inclusive. The default is 2.

TCPIPSetTuningTable $1F36

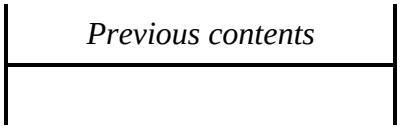
Replaces the current tuning table.

Parameters

Stack before call



Stack after call



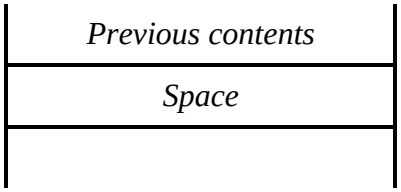
Errors	`terrBADTUNETABLELEN` Tune table length in Marinetti 2.0 must be 10
BASIC	`SUB TCPIPSetTuningTAble (tunePtr)`
C	`extern pascal void TCPIPSetTuningTAble (tunePtr);`
Pascal	`procedure TCPIPSetTuningTable (tPtr: tunePtr);`
`tunePtr`	Points to a new tuning table, which Marinetti will copy into its internal tuning table.

TCPIPGetConnectMsgFlag $4236

Returns the connect message flag, which tells the link layer module whether or not to display connect messages.

Parameters

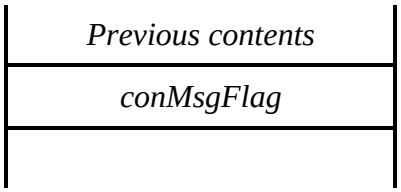
Stack before call



Word — Space for result

<— SP

Stack after call



Word — Boolean

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPGetConnectMsgFlag as %
```

C
```
extern pascal Boolean TCPIPGetConnectMsgFlag (void);
```

Pascal
```
function  TCPIPGetConnectMsgFlag: boolean;
```

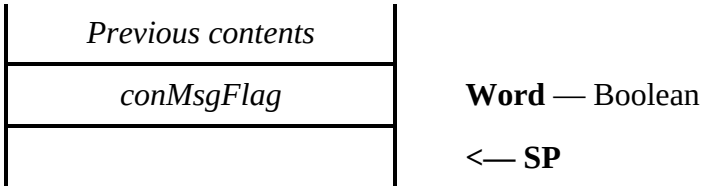
`conMsgFlag` The value returned is TRUE (non-zero) if link layer modules are to display connect messages, and FALSE ($0000) if they are not.

TCPIPSetConnectMsgFlag $4336

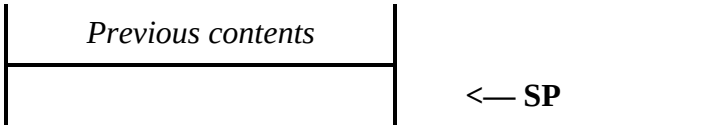
Tells Marinetti to tell link layer modules whether or not to display connect messages.

Parameters

Stack before call



Stack after call



Errors None.

BASIC
```
SUB TCPIPSetConnectMsgFlag (%)
```

C
```
extern pascal void TCPIPSetConnectMsgFlag (Boolean);
```

Pascal
```
procedure TCPIPSetConnectMsgFlag (flag: boolean);
```

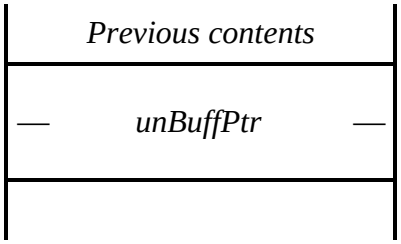
`conMsgFlag` The value is TRUE (non-zero) if link layer modules are to display connect messages, and FALSE ($0000) if they are not.

TCPIPGetUsername $4436

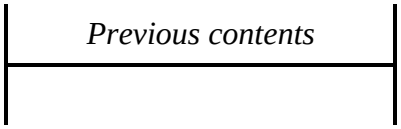
Returns the current username.

Parameters

Stack before call



Stack after call



Errors None.

BASIC
```
FUNCTION TCPIPGetUsername as usernamePtr
```

C
```
extern pascal usernamePtr TCPIPGetUsername (void)
```

Pascal
```
function  TCPIPGetUsername: usernamePtr;
```

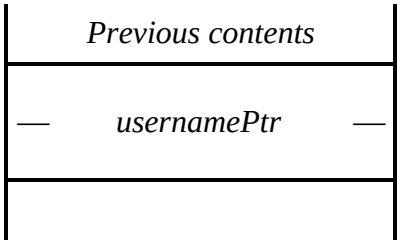
`unBuffPtr` Pointer to a 51 byte response buffer, for the returned username pstring.

TCPIPSetUsername $4536

Sets the current username.

Parameters

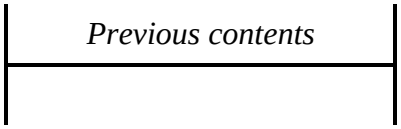
Stack before call



Long — Pointer to username pstring

<— SP

Stack after call



<— SP

Errors None.

BASIC
```
SUB TCPIPSetUsername (usernamePtr)
```

C
```
extern pascal void TCPIPSetUsername (usernamePtr);
```

Pascal
```
procedure TCPIPSetUsername (name: usernamePtr);
```

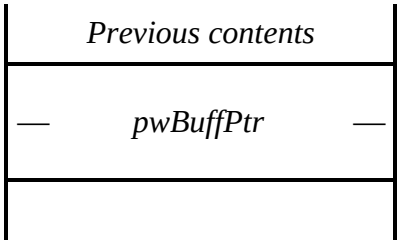
`usernamePtr` Usernames may contain a maximum of 50 characters.

TCPIPGetPassword $4636

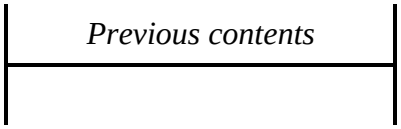
Returns the user's current password.

Parameters

Stack before call



Stack after call



Errors None.

BASIC

```
FUNCTION TCPIPGetPassword as passwordPtr

type password
    length as byte
    name(49) as char
end type
type passwordPtr as pointer to password
```

C

```
extern pascal passwordPtr TCPIPGetPassword (void)

typedef struct {
    Byte length;
    char name[50];
    } password, *passwordPtr;
```

Pascal

```
function  TCPIPGetPassword: passwordPtr;

password = string[50];
passwordPtr = ^password;
```

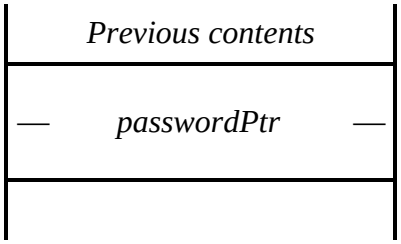
`pwBuffPtr` Pointer to a 51 byte response buffer, for the returned password pstring.

TCPIPSetPassword $4736

Sets the user's password.

Parameters

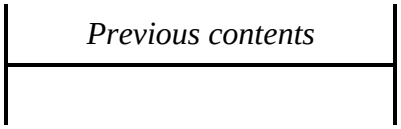
Stack before call



Long — Pointer to password pstring

<— SP

Stack after call



<— SP

Errors None.

BASIC
```
SUB TCPIPSetPassword (passwordPtr)
```

C
```
extern pascal void TCPIPSetPassword (passwordPtr);
```

Pascal
```
procedure TCPIPSetPassword (name: passwordPtr);
```

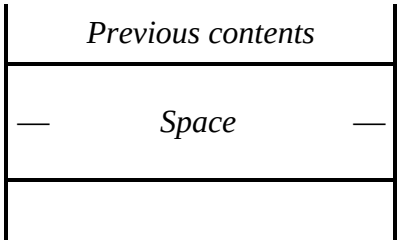
`passwordPtr` Passwords may contain a maximum of 50 characters.

TCPIPGetLinkVariables $4A36

Returns a pointer to the variables maintained by the current link layer module.

Parameters

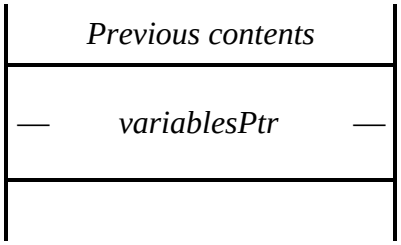
Stack before call



Long — Space for result

<— **SP**

Stack after call



Long — Pointer to variables

<— **SP**

Errors `terrNOCONNECTION` Not currently connected

BASIC

```
FUNCTION TCPIPGetLinkVariables as variablesPtr

type variablesRecord
    inwLength as integer
    inwIP as long
    inwMethod as integer
    inwMTU as integer
    inwLVPtr as long
end type
type variablesPtr as pointer to variablesRecord
```

C

```
extern pascal variablesPtr TCPIPGetLinkVariables
                    (void);

typedef struct {
    Word inwLength;
    Long inwIP;
    Word inwMethod;
    Word inwMTU;
    Long inwLVPtr;
    } variablesStruct, *variablesPtr;
```

Pascal

```
function  TCPIPGetLinkVariables: variablesPtr;
```

```
variablesRecord = record
   inwLength: integer;
   inwIP: longint;
   inwMethod: integer;
   inwMTU: integer;
   inwLVPtr: longint;
   end;
variablesPtr = ^variablesRecord;
```

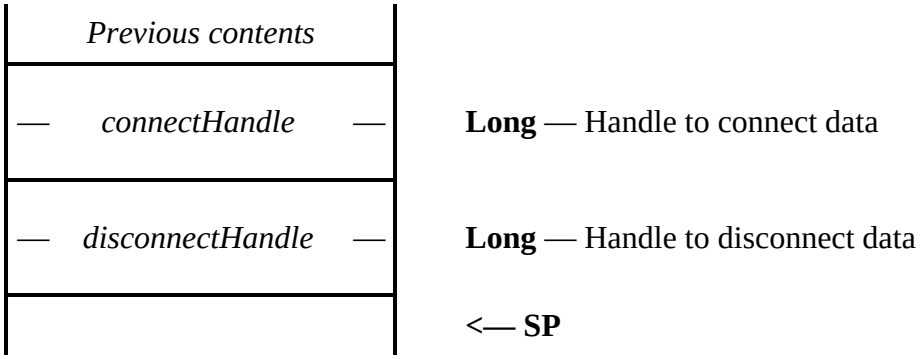
`variablePtr` Points to the variables maintained by the current link layer module. The layout of the variables is described in the `LinkGetVariables` link layer module call.

TCPIPEditLinkConfig $4B36

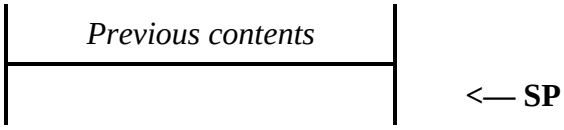
Presents a window allowing the user to edit configuration parameters required by the link layer module. This call is currently only made by the Control Panel, but may be made by other applications which may wish to control Marinetti's setup.

Parameters

Stack before call



Stack after call



Errors	`terrNOLINKLAYER`	Unable to load link layer module for the selected connect method
	`terrBADLINKLAYER`	Not a link layer module
	`terrBADENVIRONMENT`	Either the desktop is not currently displayed, or the correct tools are not started.

BASIC
```
SUB TCPIPEditLinkConfig (Handle, Handle)
```

C
```
extern pascal void TCPIPEditLinkConfig (Handle,
                        Handle);
```

Pascal
```
procedure TCPIPEditLinkConfig (connectHand: handle;
                        disconnectHAnd: handle);
```

This call passes two handles, containing the connect and disconnect data respectively. When the call returns, the same handles will contain the alterned data.

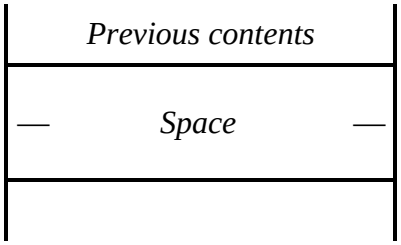
This call must be made while the desktop is displayed, as the code which presents the data to the user depends on certain tool sets to be already started. The complete list may found in the description of the `LinkConfigure` link layer module call later in this document.

TCPIPGetModuleNames $4C36

Returns a pointer to an array of `linkInfoBlk` records, indicating which link layer modules are available for use.

Parameters

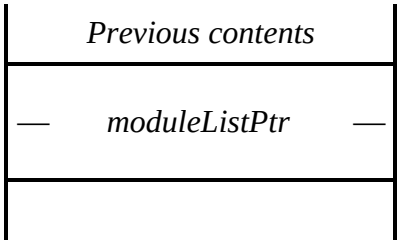
Stack before call



Long — Space for result

<— SP

Stack after call



Long — Pointer to module name list

<— SP

Errors None.

BASIC

```
FUNCTION TCPIPGetModuleNames as moduleListPtr

type module
    liMethodID as integer
    liName(20) as byte
    liVersion as long
    liFlags as integer
    liFilename(16) as byte
    liMenuItem(14) as byte
end type
type moduleListPtr as pointer to module
```

C

```
extern pascal void TCPIPGetModuleNames (ModuleListPtr);

typedef struct {
    Word liMethodID;
    char liName[21];
    Long liVersion;
    Word liFlags;
    char liFilename[16];
    Byte liMenuItem[14];
    } module, (*moduleListPtr)[];
```

Pascal

```
function  TCPIPGetModuleNames: moduleListPtr;

module = record
   liMethodID: integer;
   liName: string[20];
   liVersion: longint;
   liFlags: integer;
   liFilename: string[15];
   liMenuItem: array[0..13] of byte;
   end;
moduleList = array[0..99] of module;
moduleListPtr = ^moduleList;
```

`moduleListPtr` Points to an array of 64 byte extended `linkInfoBlk` records. The list is terminated by a `nil` word. Each record is defined as follows:

`+00`	`liMethodID`	`word`	The connect method. See the `conXXX` equates at the end of this document
`+02`	`liName`	`21 bytes`	Pstring name of the module
`+23`	`liVersion`	`longword`	rVersion (type $8029 resource layout) of the module
`+27`	`liFlags`	`word`	Contains the following flags:

bit15	This link layer uses the built in Apple IIGS serial ports
bits14-0	Reserved – set to zeros

`+29`	`liFilename`	`16 bytes`	Pstring filename of the module
`+45`	`liMenuItem`	`14 bytes`	`rMenuItem` template ready for use, which defines this connect method as a menu item

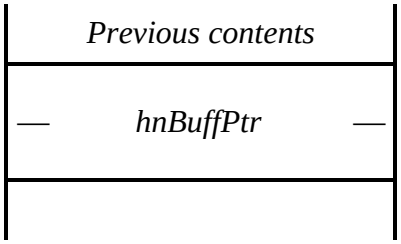
◆NOTE: *The link layer module call* `LinkModuleInfo` *also refers to a* `linkInfoBlk`*, but with less entries in it. This is because Marinetti fills in the rest of the information itself before returning the records in the* `TCPIPGetModuleNames` *call.*

TCPIPGetHostName $5136

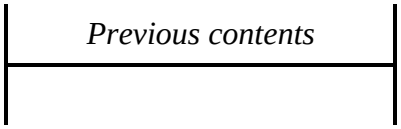
Returns the current host name.

Parameters

Stack before call



Stack after call



Errors None.

BASIC

```
FUNCTION TCPIPGetHostName as hostNamePtr

type hostName
    length as byte
    name(29) as byte
end type
type hostNamePtr as pointer to hostName
```

C

```
extern pascal hostNamePtr TCPIPGetHostName (void);

typedef struct {
    Byte length;
    char name[50];
    } hostName, *hostNamePtr;
```

Pascal

```
function  TCPIPGetHostName: hostNamePtr;

hostName = string[30];
hostNamePtr = ^hostName;
```

`hnBuffPtr` Pointer to a 31 byte response buffer, for the returned host name pstring.

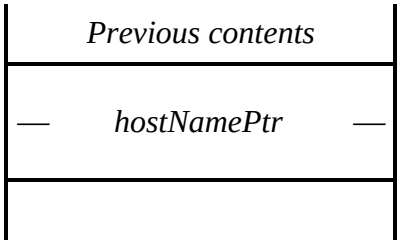
◆NOTE: *The host name is not actually used for anything, but is provided for ease of porting of BSD applications. The default is is set to "appleiigs" if no previous host name has been set.*

TCPIPSetHostName $5236

Sets the current host name.

Parameters

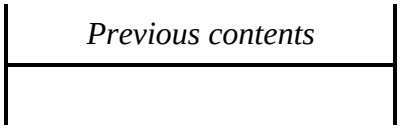
Stack before call



Long — Pointer to host name pstring

<— SP

Stack after call



<— SP

Errors None.

BASIC
```
SUB TCPIPSetHostName (hostNamePtr)
```

C
```
extern pascal void TCPIPSetHostName (hostNamePtr);
```

Pascal
```
procedure TCPIPSetHostName (hPtr: hostNamePtr);
```

`hostNamePtr` Host names may contain a maximum of 30 characters.

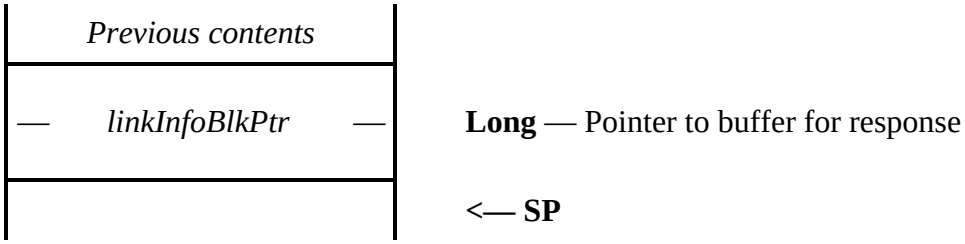
◆NOTE: *The host name is not actually used for anything, but is provided for ease of porting of BSD applications. The default is is set to "appleiigs" if no previous host name has been set.*

TCPIPGetLinkLayer $5436

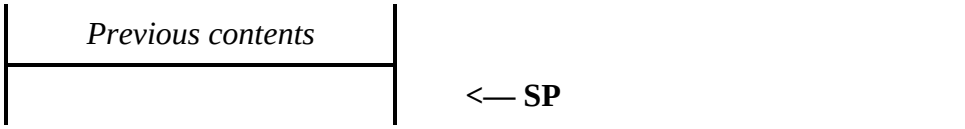
Returns information about the module.

Parameters

Stack before call



Stack after call



Errors None.

BASIC

```
FUNCTION TCPIPGetLinkLayer as linkInfoBlkPtr

type linkInfoBlk
    liMethodID as integer
    liName(20) as byte
    liVersion as long
    liFlags as integer
end type
type linkInfoBlkPtr as pointer to linkInfoBlk
```

C

```
extern pascal linkInfoBlkPtr TCPIPGetLinkLayer (void);

typedef struct {
    Word liMethodID;
    char liName[21];
    Long liVersion;
    Word liFlags;
    } linkInfoBlk, *linkInfoBlkPtr;
```

Pascal

```
function  TCPIPGetLinkLayer: linkInfoBlkPtr;

linkInfoBlk = record
   liMethodID: integer;
   liName: string[20];
   liVersion: longint;
   liFlags: integer;
   end;
```

```
linkInfoBlkPtr = ^linkInfoBlk;
```

`linkInfoBlkPtr` Points to a fixed length 27 byte response buffer as follows:

`+00`	`liMethodID`	`word`	The connect method. New modules will need to apply to the author for a unique ID to use. See `conXXX` equates for details of already defined values
`+02`	`liName`	`21 bytes`	Pstring name of the module
`+23`	`liVersion`	`longword`	rVersion (type $8029 resource layout) of the module
`+27`	`liFlags`	`word`	Contains the following flags:

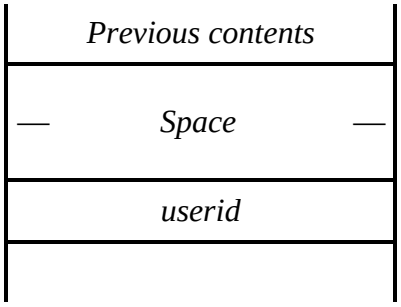
bit15	This link layer uses the built in Apple IIGS serial ports
bits14-1	Reserved – set to zeros
bit0	Indicates whether the module contains an rIcon resource

TCPIPGetAuthMessage $5736

Returns the authentication message returned from the link layer module during connection.

Parameters

Stack before call

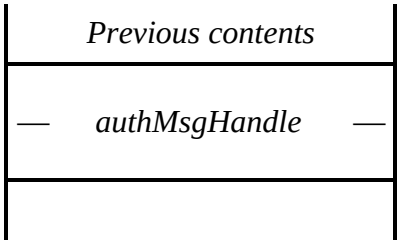


Long — Space for result

Word — userID for Marinetti to use with `NewHandle`

<— SP

Stack after call



Long — Handle containing authentication message

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPGetAuthMessage (%) as handle
```

C
```
extern pascal handle TCPIPGetAuthMessage (Word);
```

Pascal
```
function  TCPIPGetAuthMessage (userID: integer):
                        handle;
```

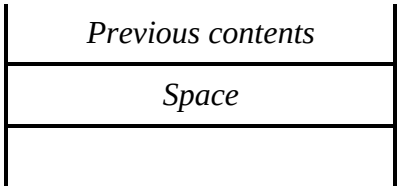
`authMsgHandle` If the current link layer supports authentication messages, then this handle will contain the ASCII text of the message sent by the host, else it will be empty. Either way, the handle belongs to the userid passed on the stack. The message may be for a successful or unsuccessful connection, and varies depending on the host system being used. It will always be less than 256 characters in length.

TCPIPGetAliveFlag $5A36

Returns the alive flag, which tells Marinetti whether to automatically keep the link alive.

Parameters

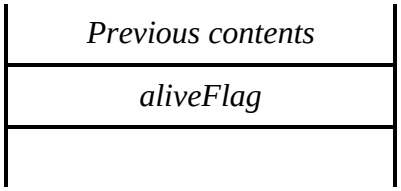
Stack before call



Word — Space for result

<— SP

Stack after call



Word — Boolean

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPGetAliveFlag as %
```

C
```
extern pascal Boolean TCPIPGetAliveFlag (void);
```

Pascal
```
function  TCPIPGetAliveFlag: boolean;
```

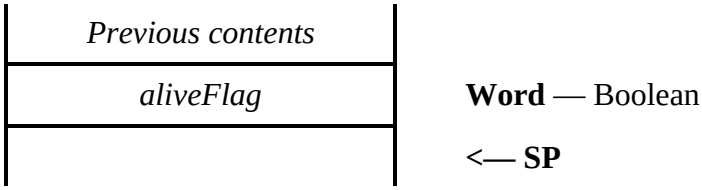
`aliveFlag` The value returned is TRUE (non-zero) if Marinetti is to automatically keep the link alive, and FALSE ($0000) if it is not.

TCPIPSetAliveFlag $5B36

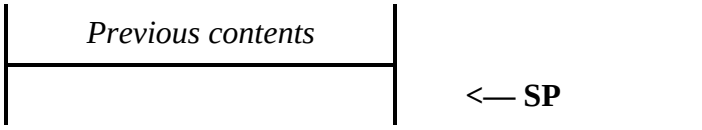
Tells Marinetti whether to automatically keep the link alive.

Parameters

Stack before call



Stack after call



Errors None.

BASIC
```
SUB TCPIPSetAliveFlag (%)
```

C
```
extern pascal void TCPIPSetAliveFlag (Boolean);
```

Pascal
```
procedure TCPIPSetAliveFlag (alive: boolean);
```

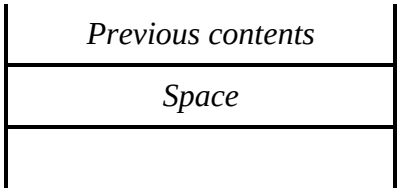
`aliveFlag` The value is TRUE (non-zero) if Marinetti is to automatically keep the link alive, and FALSE ($0000) if it is not.

TCPIPGetAliveMinutes $5C36

Returns how often Marinetti should present traffic to the network in an attempt to stop it disconnecting due to inactivity. The `aliveFlag` must be set to `true` to support this feature.

Parameters

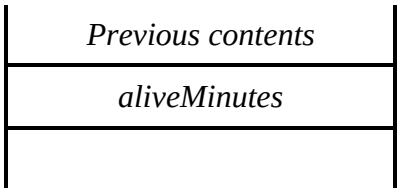
Stack before call



Word — Space for result

<— SP

Stack after call



Word — Boolean

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPGetAliveMinutes as %
```

C
```
extern pascal Word TCPIPGetAliveMinutes (void);
```

Pascal
```
function  TCPIPGetAliveMinutes: integer;
```

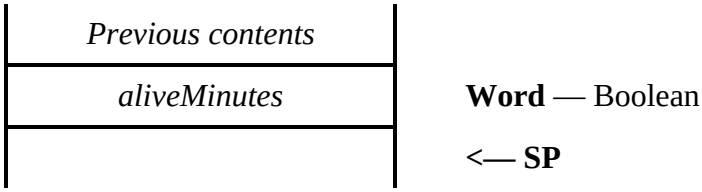
`aliveMinutes` The number of minutes between network checks. A value of zero also forces `aliveFlag` to `false`.

TCPIPSetAliveMinutes $5D36

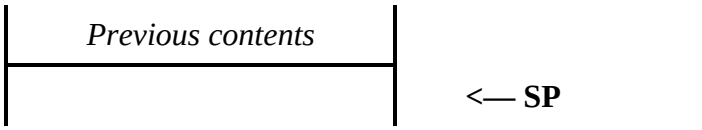
Tells Marinetti how often to present traffic to the network in an attempt to stop it disconnecting due to inactivity. The `aliveFlag` must be set to `true` to support this feature.

Parameters

Stack before call



Stack after call



Errors `terrBADALIVEMINUTES` Minutes value is invalid

BASIC `SUB TCPIPSetAliveMinutes (%)`

C `extern pascal void TCPIPSetAliveMinutes (Word);`

Pascal `procedure TCPIPSetAliveMinutes (aliveMinutes: integer);`

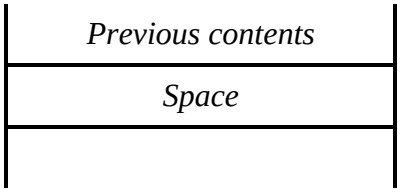
`aliveMinutes` A number from 1 to 999, indicating the number of minutes between network checks.

TCPIPGetBootConnectFlag $5F36

Returns the boot connect flag, which tells Marinetti whether to automatically connect to the network each time GS/OS is booted.

Parameters

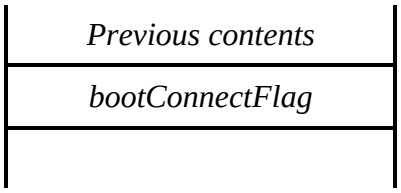
Stack before call



Word — Space for result

<— SP

Stack after call



Word — Boolean

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPGetBootConnectFlag as %
```

C
```
extern pascal Boolean; TCPIPGetBootConnectFlag (void);
```

Pascal
```
function  TCPIPGetBootConnectFlag: boolean;
```

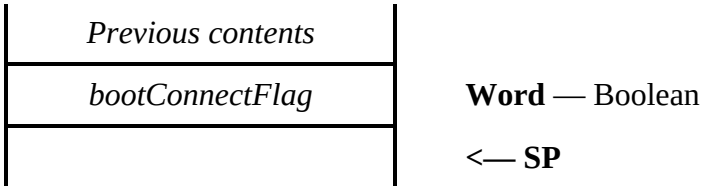
`bootConnectFlag` The value returned is TRUE (non-zero) if Marinetti is to automatically connect to the network, and FALSE ($0000) if it is not.

TCPIPSetBootConnectFlag $6036

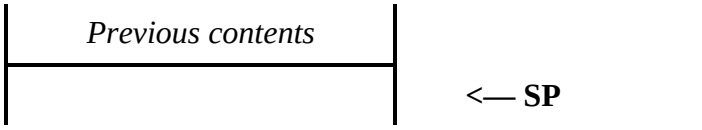
Tells Marinetti whether to automatically connect to the network each time GS/OS is booted.

Parameters

Stack before call



Stack after call



Errors None.

BASIC
```
SUB TCPIPSetBootConnectFlag (%)
```

C
```
extern pascal void TCPIPSetBootConnectFlag (Boolean);
```

Pascal
```
procedure TCPIPSetBootConnectFlag (bootConnect:
                                   boolean);
```

`bootConnectFlag` The value is TRUE (non-zero) if Marinetti is to automatically connect to the network, and FALSE ($0000) if it is not.

Domain Name Resolution

Domain names are what most people traditionally think of when giving an address for a machine on an internet. The problem is that the internet protocol requires you to address the machines with a numeric IP address rather than a domain name. Your machine is responsible for looking up the numeric IP address of the machine it wants to talk to before it can do so.

Domain names are purely administrative data, contained within a database on a server somewhere on the network, which applications must refer to when converting to and from the actual numeric IP addresses required by the network. Obviously Marinetti must start with a numeric IP address somewhere, and this is provided by calling `TCPIPSetDNS` with the numeric IP addresses of Domain Name Servers on the network you wish to use. Fields for this information are also provided in the CDev, and are saved with the preferences.

Once Marinetti knows which Domain Name Servers to use, the application may start converting domain names to numeric IP addresses by calling `TCPIPDNRNameToIP`.

Because the information for conversion is kept elsewhere on the network, looking up a domain name is not instantaneous, and the application may do other things while it is waiting for an answer. Therefore, making a `TCPIPDNRNameToIP` call initiates a request, and won't necessarily immediately return an answer. You can do whatever you want while you wait for a reply, just make sure you're either calling `SystemTask` (or `TaskMaster`) or `TCPIPPoll` every so often, to allow the resolver to do its job.

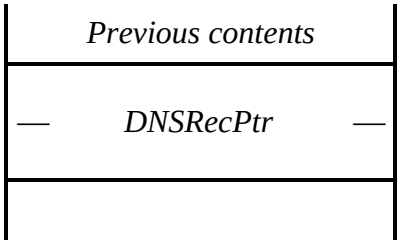
Once the call has been made, check the return buffer every so often. While the call is pending, the initial word, or DNR status code, will be set to `DNR_Pending`. Once the call has completed, this will change to something else, and if successful, your answer will have been returned.

TCPIPGetDNS $1C36

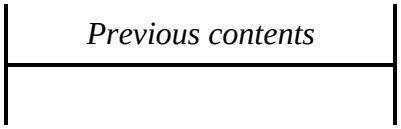
Returns the IP addresses of the main and auxiliary Domain Name Servers.

Parameters

Stack before call



Stack after call



Errors None.

BASIC

```
FUNCTION TCPIPGetDNS as DNSRecPtr

type DNSRec
    DNSMain as long
    DNSAux as long
end type
type DNSRecPtr as pointer to DNSRec
```

C

```
extern pascal DNSRecPtr TCPIPGetDNS (void);

typedef struct {
    Long DNSMain;
    Long DNSAux;
    } DNSRec, *DNSRecPtr;
```

Pascal

```
function  TCPIPGetDNS: DNSRecPtr;

DNSRec = record
   DNSMain: longint;
   DNSAux: longint;
   end;
DNSRecPtr = ^DNSRec;
```

`DNSRecPtr` Points to the response record. The layout is as follows:

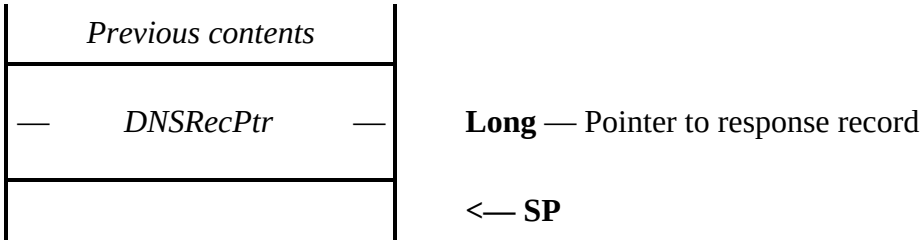
```
+00  DNSMain        longword   Main DNS IP address
+04  DNSAux         longword   Auxiliary DNS IP address
```

TCPIPSetDNS $1D36

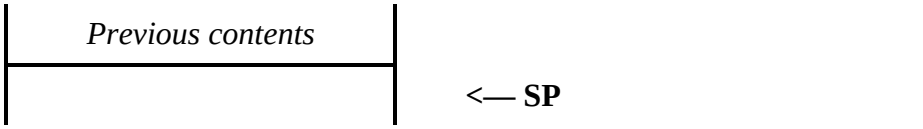
Sets the IP addresses of the main and secondary Domain Name Servers.

Parameters

Stack before call



Stack after call



Errors None.

BASIC
```
SUB TCPIPSetDNS (DNSRecPtr)
```

C
```
extern pascal void TCPIPSetDNS (DNSRecPtr);
```

Pascal
```
procedure TCPIPSetDNS (DNS: DNSRecPtr);
```

`DNSRecPtr` Points to the desired DNS record. The layout is as follows:

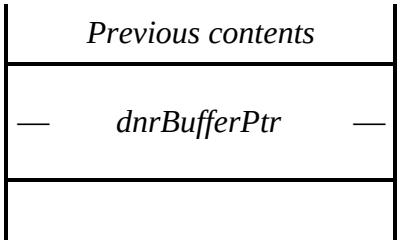
```
+00  DNSMain        longword   Main DNS IP address
+04  DNSAux         longword   Auxilliary DNS IP address
```

TCPIPCancelDNR $2036

Cancels a pending request to the Domain Name Servers.

Parameters

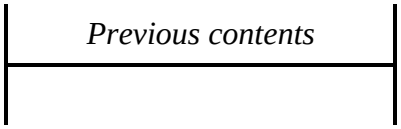
Stack before call



Long — Pointer to return buffer

<— SP

Stack after call



<— SP

Errors	`terrNODNRPENDING`	No such entry in DNR list
	`terrDNRBUSY`	DNR is currently busy - try again later

BASIC
```
SUB TCPIPCancelDNR (DNRBufferPtr)
```

C
```
extern pascal void TCPIPCancelDNR (dnrBufferPtr);
```

Pascal
```
procedure TCPIPCancelDNR (dnr: dnrBufferPtr);
```

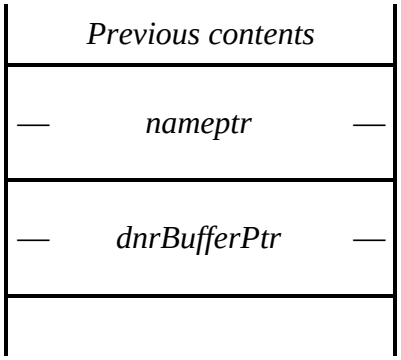
`dnrBufferPtr` The pointer to the return buffer indicates which request to cancel.

TCPIPDNRNameToIP $2136

Initiates a request to the Domain Name Servers, via Marinetti's Domain Name Resolver, to look up an ASCII domain name and return it as a numeric IP address.

Parameters

Stack before call

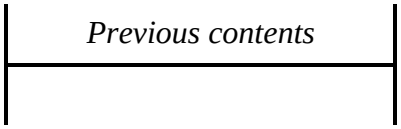


Long — Pointer to pstring domain name to look up

Long — Pointer to return buffer

<— SP

Stack after call



<— SP

Errors	`terrNODNSERVERS`	No servers registered with Marinetti
	`terrDNRBUSY`	DNR is currently busy - try again later

BASIC

```
SUB TCPIPDNRNameToIP (pStringPtr, DNRBufferPtr)

type dnrBuffer
    DNRstatus as integer
    DNRIPaddress as long
end type
type dnrBufferPtr as pointer to dnrBuffer
```

C

```
extern pascal void TCPIPDNRNameToIP (char *,
                    dnrBufferPtr);

typedef struct {
    Word DNRstatus;
    Long DNRIPaddress;
    } dnrBuffer, *dnrBufferPtr;
```

Pascal

```
procedure TCPIPDNRNameToIP (name: pstring; dnr:
                    dnrBufferPtr);

dnrBuffer = record
   DNRstatus: integer;
   DNRIPaddress: longint;
```

```
   end;
dnrBufferPtr = ^dnrBuffer;
```

`dnrBufferPtr` Points to the following DNR return buffer:

`+00 DNRstatus`	`word`	Current status of DNR for this request
`+02 DNRIPaddress`	`longword`	Returned IP address

The `DNRstatus` codes are as follows:

`DNR_Pending`	$0000	Request is still being processed
`DNR_OK`	$0001	Your request completed successfully, and `dnrBuffer` contains the requested data
`DNR_Failed`	$0002	The request failed. Either the connection timed out, or some other network error
`DNR_NoDNSEntry`	$0003	Requested domain has no DNS entry
`DNR_Cancelled`	$0004	Cancelled by user

IP network tool calls

These calls provide access to network layer functions

TCPIPPoll $2236

Tells Marinetti to execute a set number of steps in all its pending tasks. See `TCPIPGetTuneTable` for more details.

Parameters

The stack is not affected by this call.

Errors None

BASIC
```
SUB TCPIPPoll
```

C
```
extern pascal void TCPIPPoll (void);
```

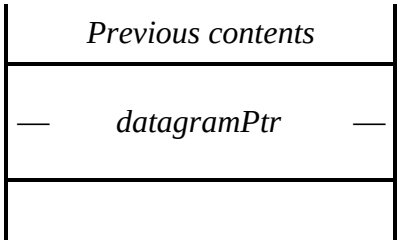
Pascal
```
procedure TCPIPPoll;
```

TCPIPSendIPDatagram $4036

Sends a raw IP datagram across the network.

Parameters

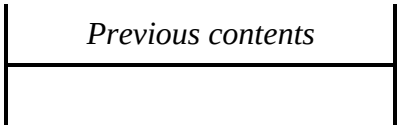
Stack before call



Long — Pointer to the datagram to send

<— SP

Stack after call



<— SP

Errors `terrNOCONNECTION` Not currently connected to the network

BASIC
```
SUB TCPIPSendIPDatagram (datagramPtr)
```

C
```
extern pascal void TCPIPSendIPDatagram (datagramPtr);
```

Pascal
```
procedure TCPIPSendIPDatagram (dPtr: datagramPtr);
```

This call assumes that the IP header has been formatted correctly with the appropriate length indicators, and uses this to determine the checksum and final datagram length. While the destination address must be embedded in the header, Marinetti will copy in its current IP address for you.

Network and Transport layer tool calls

These calls provide access to protocol functions of the network and transport layers, excluding TCP, which is described in its own section.

Most requests involve using an `ipid`, which is assigned when you login to Marinetti.

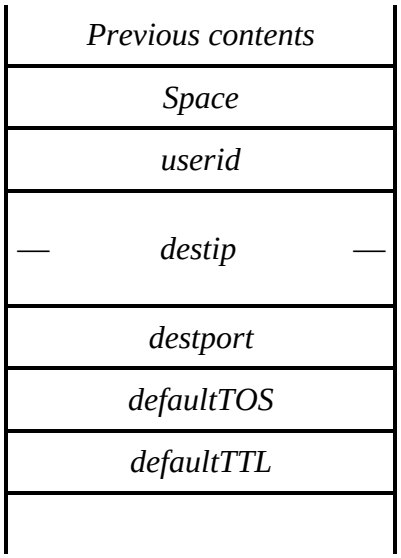
You may only make one connection of each type, such as TCP or UDP, per `ipid`.

TCPIPLogin $2336

This is the initial login for a task, telling Marinetti the network destination, and IP management parameters. In return, Marinetti assigns a source port number, and returns you an `ipid` to use with subsequent calls.

Parameters

Stack before call



Word — Space for result

Word — userID for Marinetti to use with `NewHandle`

Long — Destination IP address

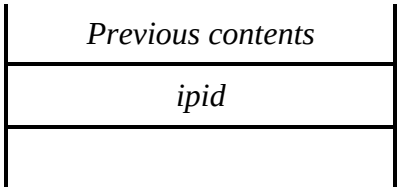
Word — Destination port number

Word — Default Type Of Service

Word — Default Time To Live

<— SP

Stack after call



Word — ipid to use for subsequent calls

<— SP

Errors	`terrIPIDTABLEFULL`	There are too many connections already
	`terrNOCONNECTION`	Not connected to the network

BASIC `FUNCTION TCPIPLogin (%, &, %, %, %) as %`

C

```
extern pascal Word TCPIPLogin (Word, Long, Word, Word,
                    Word)
```

Pascal

```
function  TCPIPLogin (userID: integer; destip: longint;
                    destport: integer; defaultTOS:
                    integer; defaultTTL: integer):
                    integer;
```

`userid` This must be a valid Memory Manager userID, which Marinetti may use on your behalf when returning data to you. Handles returned which contain data, such as those from `TCPIPReadTCP`, will belong to you, and be allocated with this userID.

`destIP` The destination IP address for all connections using this `ipid`. Some standard special case IP addresses are valid, such as 127.0.0.1, which is for loopback. Using the loopback address, or Marinetti's current IP address, two applications on the same Apple IIGS may talk to each other via a TCP connection.

`destPort` The destination port for all connections using this `ipid`. Using a destination port of $0000, tells Marinetti to use this login as a service dispatcher for incoming connections.

`defaultTOS` This is the initial `TOS` ("Type Of Service") value to use for all IP services for this `ipid`. If unsure, use a value of $0000, which assigns equal priority to each TOS bit. The following are the valid bit flags, of which only one may be set at a time.

`%0001 0000`	Minimise delay
`%0000 1000`	Maximise throughput
`%0000 0100`	Maximise reliability
`%0000 0010`	Minimise monetary cost

`defaultTTL` This is the initial TTL ("Time To Live") value to use for all IP services for this `ipid`. If unsure, use a value of $0040, which means each IP datagram will hop at least 64 hosts before expiring. Values larger than $00FF will pinned to 255.

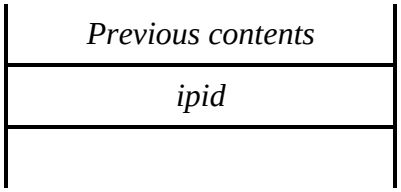
`ipid` This is the value assigned to this destination/port pair. It must be supplied with any calls which access this connection.

TCPIPLogout $2436

Tells Marinetti to logout this `ipid`, thus freeing all its control blocks, and making it available for subsequent `TCPIPLogin` calls.

Parameters

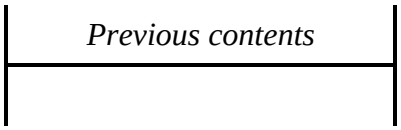
Stack before call



Word — ipid to logout

<— SP

Stack after call



<— SP

Errors	`terrSOCKETOPEN`	This `ipid` still has a pending connection, and Marinetti will not log it out until the connection has closed/ended
	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC

```
SUB TCPIPLogout (%)
```

C

```
extern pascal void TCPIPLogout (Word);
```

Pascal

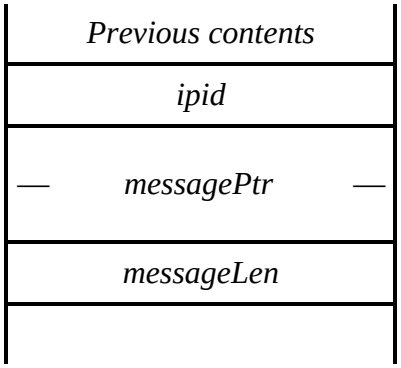
```
procedure TCPIPLogout (ipid: integer);
```

TCPIPSendICMP $2536

Sends an ICMP message datagram across the network.

Parameters

Stack before call



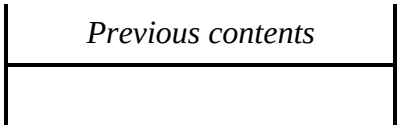
Word — Connection to use

Long — Pointer to the ICMP message

Word — Length of the ICMP message

<— SP

Stack after call



<— SP

Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC
```
SUB TCPIPSendICMP (%, messagePtr, %)
```

C
```
extern pascal void TCPIPSendICMP (Word, messagePtr,
                                  Word);
```

Pascal
```
procedure TCPIPSendICMP (ipid: integer; mPtr:
                                  messagePtr; messageLen: integer);
```

`messagePtr` Points to just the ICMP message. Marinetti takes care of generating the correct checksum, encapsulating it in an appropriate IP datagram, and sending it across the network.

If sending echo request and echo reply messages, you must store your `ipid` as the message identifier, or instead use the `TCPIPSendICMPEcho` call, which was designed specifically for this purpose.

Ordinarily a ICMP datagram should have an IP header TOS value of 255 if performing network administration functions, so that the destination has the best possible chance of receiving the message. However, datagrams sent by `TCPIPSendICMP` use the current TOS value for the requested `ipid`.

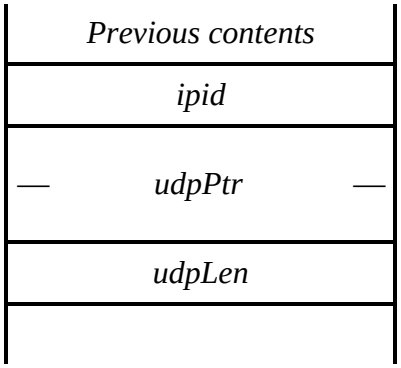
`TCPIPSendICMP` automatically initialises and calculates the embedded ICMP checksum for you.

TCPIPSendUDP $2636

Sends a UDP datagram across the network.

Parameters

Stack before call



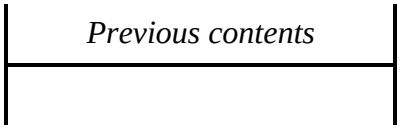
Word — Connection to use

Long — Pointer to the UDP data to send

Word — Length of the UDP data to send

<— SP

Stack after call



<— SP

Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC

```
SUB TCPIPSendUDP (%, udpPtr, %)
```

C

```
extern pascal void TCPIPSendUDP (Word, udpPtr, Word);
```

Pascal

```
procedure TCPIPSendUDP (ipid: integer; uPtr: udpPtr;
                        udpLen: integer);
```

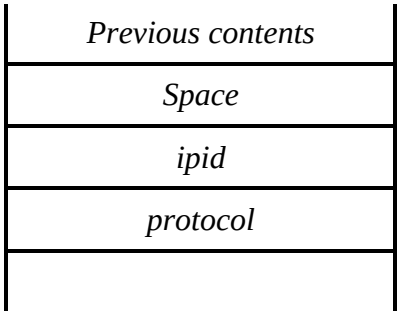
`datagramPtr` Points to just the UDP data. The UDP header is built for you inside Marinetti, encapsulated in an appropriate IP datagram, and sent across the network.

TCPIPGetDatagramCount $2736

Returns the number of pending input datagrams for a specific `ipid`.

Parameters

Stack before call



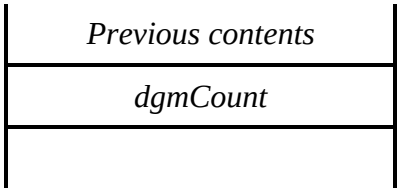
Word — Space for result

Word — Connection to use

Word — Protocol of the queue to count

<— SP

Stack after call



Word — Count of datagrams in the queue

<— SP

Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC

```
FUNCTION TCPIPGetDatagramCount (%, %) as %
```

C

```
extern pascal Word TCPIPGetDatagramCount (Word, Word);
```

Pascal

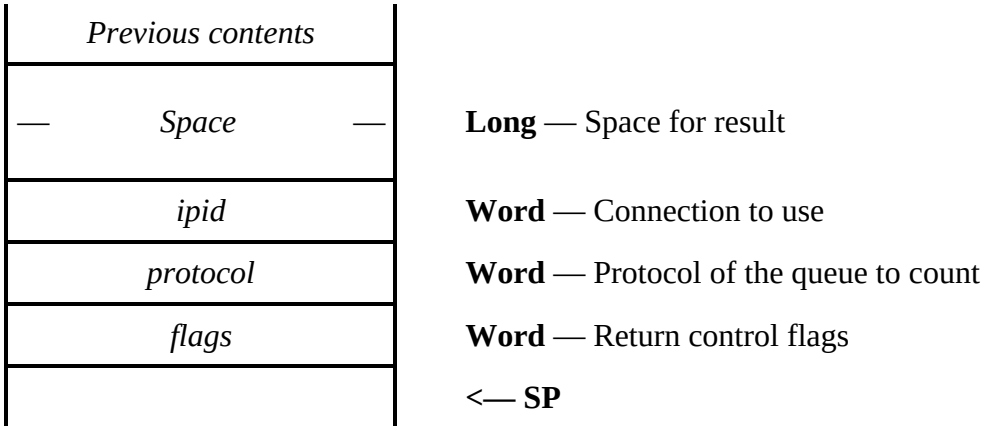
```
function  TCPIPGetDatagramCount (ipid: integer;
                                 protocol: integer): integer;
```

TCPIPGetNextDatagram $2836

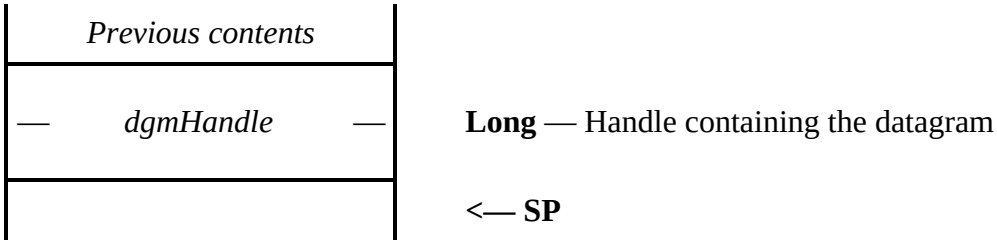
Returns the next datagram for a specific protocol.

Parameters

Stack before call



Stack after call



Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC
```
FUNCTION TCPIPGetNextDatagram (%, %, %) as dgmHandle
```

C
```
extern pascal dgmHandle TCPIPGetNextDatagram (Word,
                    Word, Word);
```

Pascal
```
function  TCPIPGetNextDatagram (ipid, protocol, flags:
                    integer): dgmHandle;
```

`protocol` The values `protocolICMP`, `protocolUDP` and `protocolTCP` return entries from the ICMP, UDP and TCP queues respectively. Any other value will return the next IP queue entry.

`flags` Controls how the datagram is returned:

Bit 15 controls the IP header; 0 = keep, 1 = remove
Bit 14 controls the embedded header; 0 = keep, 1 = remove

Removing the embedded header also forces bit 15 to remove the IP header.

`dgmHandle` Contains the returned data, or is `nil` if there is no currently available datagram for that protocol.

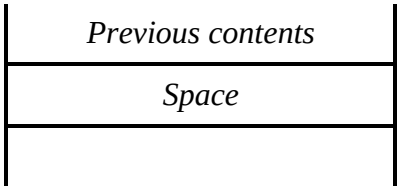
◆NOTE: *For TCP, this request is primarily a test routine, left over from previous versions of Marinetti. Instead, TCP should be read using the appropriate TCP requests.*

TCPIPGetLoginCount $2936

Returns the current number of Marinetti log ins.

Parameters

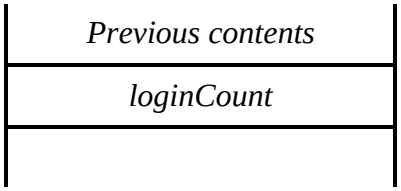
Stack before call



Word — Space for result

<— SP

Stack after call



Word — Number of current log ins

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPGetLoginCount as %
```

C
```
extern pascal Word TCPIPGetLoginCount (void);
```

Pascal
```
function  TCPIPGetLoginCount: integer;
```

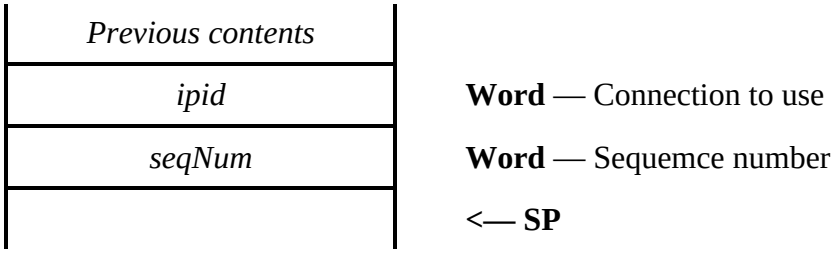
Marinetti will not disconnect from the network while there are pending log ins. All `TCPIPLogin` calls must be balanced with a `TCPIPLogout` call.

TCPIPSendICMPEcho $2A36

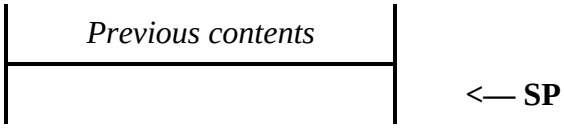
Sends an ICMP echo request across the network.

Parameters

Stack before call



Stack after call



Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC `SUB TCPIPSendICMPEcho (%, %)`

C `extern pascal void TCPIPSendICMPEcho (Word, Word)`

Pascal `procedure TCPIPSendICMPEcho (ipid, seqNum: integer);`

`seqNum` The sequence number to include in the Echo request. This should ordinarily start at 1, and be incremented for each subsequent send.

This request builds an appropriate ICMP message, encapsulates it with an IP datagram, and sends it across the network. The `ipid` is used as the embedded identifier.

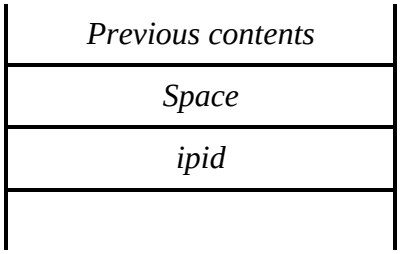
Ordinarily a ICMP datagram should have an IP header TOS value of 255 if performing network administration functions, so that the destination has the best possible chance of receiving the message. However, datagrams sent by `TCPIPSendICMPEcho` use the current TOS value for the requested `ipid`.

TCPIPReceiveICMPEcho $2B36

Scans the ICMP protocol queue for the first echo reply message, deletes it, and returns its sequence number.

Parameters

Stack before call

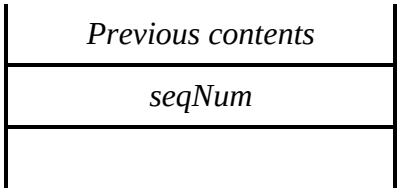


Word — Space for result

Word — Connection to use

<— SP

Stack after call



Word — Sequence number of first echo reply

<— SP

Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in
	`terrNOICMPQUEUED`	No ICMP datagrams in the queue

BASIC
```
FUNCTION TCPIPReceiveICMPEcho (%) as %
```

C
```
extern pascal Word TCPIPReceiveICMPEcho (Word);
```

Pascal
```
function  TCPIPReceiveICMPEcho (ipid: integer):
                        integer;
```

`seqNum` The sequence number of the first echo reply message found in the ICMP queue.

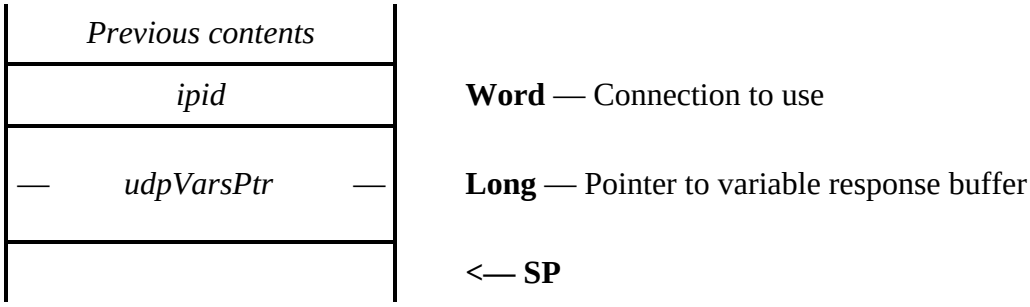
◆NOTE: *This request may return no echo replies, even though `TCPIPGetDatagramCount` says there are messages in the ICMP queue. This is because `TCPIPGetDatagramCount` counts all messages, not just the echo replies.*

TCPIPStatusUDP $5336

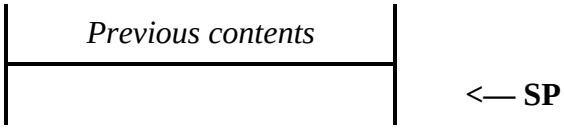
Returns a number of variables relating to UDP.

Parameters

Stack before call



Stack after call



Errors

	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC

```
SUB TCPIPStatusUDP (%, udpVarsPtr)

type udpVars
    uvQueueSize as integer
    uvError as integer
    uvErrorTick as long
    uvCount as long
    uvTotalCount as long
    uvDispatchFlag as integer
end type
type udpVarsPtr as pointer to udpVars
```

C

```
extern pascal void TCPIPStatusUDP (Word, udpVarsPtr);

typedef struct {
    Word uvQueueSize;
    Word uvError;
    Long uvErrorTick;
    Long uvCount;
    Long uvTotalCount;
    Word uvDispatchFlag;
    } udpVars, *udpVarsPtr;
```

Pascal

```
procedure TCPIPStatusUDP (ipid: integer; uPtr:
                    udpVarsPtr);

udpVars = record
   uvQueueSize: integer;
   uvError: integer;
   uvErrorTick: longint;
   uvCount: longint;
   uvTotalCount: longint;
   uvDispatchFlag: integer;
   end;
udpVarsPtr = ^udpVars;
```

On return from the call, the response buffer looks like this:

+00	uvQueueSize	word	Number of entries in receive queue
+02	uvError	word	Last ICMP type 3 error code
+04	uvErrorTick	longword	Tick of when error occurred
+08	uvCount	longword	Total received for this ipid
+12	uvTotalCount	longword	Total received for all ipids
+16	uvDispatchFlag	word	UDP dispatch flag

`uvError` If an ICMP Port Unreachable Error was received, then bit15 will be set, and the remaining bits will contain an error code as follows:

$8000	Network unreachable
$8001	Host unreachable
$8002	Protocol unreachable
$8003	Port unreachable
$8004	Fragmentation needed but DF bit set
$8005	Source route failed
$8006	Destination network unknown
$8007	Desination host unknown
$8009	Destination network administratively prohibited
$800A	Desination host administratively prohibited
$800B	Network unreachable for TOS
$800C	Host unreachable for TOS

If an ICMP Time Expired Error was received, then bit15 will be set, and the remaining bits will contain an error code as follows:

$8010	TTL expired, never reached destination

`uvTotalCount` This is the same value as that returned in the `tcpDGMSUDP` field of the error table.

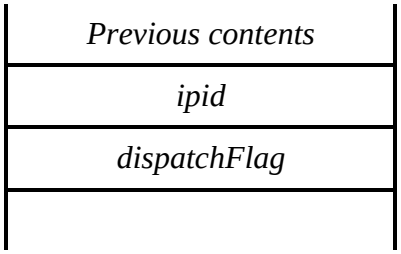
`uvDispatchFlag` This is the `dispatchFlag` boolean, which was set by the `TCPIPSetUDPDispatch` call, or `false` if not yet set. It indicates whether this `ipid` can accept incoming UDP packets from different ports. `destPort` must also be set to $0000.

TCPIPSetUDPDispatch $6136

Tells Marinetti whether this `ipid` is to be used for incoming UDP dispatch. To dispatch correctly, `destPort` must be set to $0000.

Parameters

Stack before call

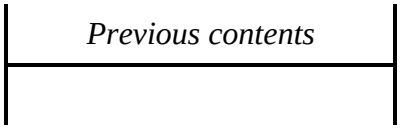


Word — Connection to use

Word — Indicates whether this `ipid` will dispatch

<— SP

Stack after call



<— SP

Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC
```
SUB TCPIPSetUDPDispatch (%, %)
```

C
```
extern pascal void TCPIPSetUDPDispatch (Word, Boolean);
```

Pascal
```
procedure TCPIPSetUDPDispatch (ipid: integer;
                               dispatchFlag: boolean);
```

`dispatchFlag` The value is TRUE (non-zero) if Marinetti is dispatch incoming UDP datagrams to this `ipid`, and FALSE ($0000) if it is not.

TCP tool calls

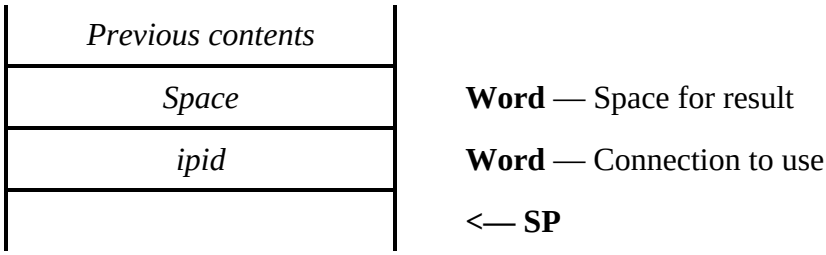
These calls are the TCP specific socket functions, and are similar to the BSD socket interfaces available on UNIX systems. It should be noted that `ipid`s using TCP may also use other protocols at the same time.

TCPIPOpenTCP $2C36

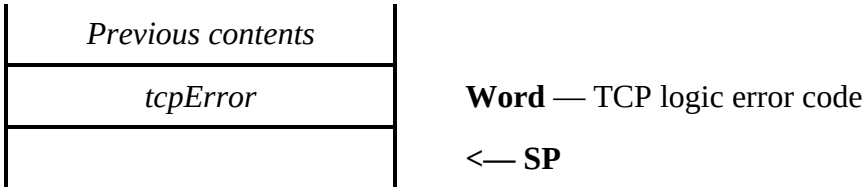
Initiates a TCP open request.

Parameters

Stack before call



Stack after call



Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC `FUNCTION TCPIPOpenTCP (%) as %`

C `extern pascal Word TCPIPOpenTCP (Word);`

Pascal `function  TCPIPOpenTCP (ipid: integer): integer;`

`tcpErr` This will be one of the `tcperr*` equates.

This request initiates an open connection request, it does not complete the opening of the connection.

All successful `TCPIPOpenTCP` calls must be balanced at some stage with a successful `TCPIPCloseTCP` call.

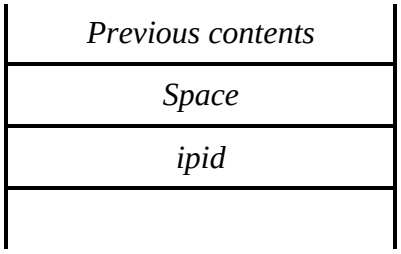
◆NOTE: *The current `TOS` and `TTL` values for the `ipid` are saved, and new preferred internal values for TCP are substituted.*

TCPIPListenTCP $4E36

Initiates a TCP listen request, to listen for incoming connection initiations.

Parameters

Stack before call

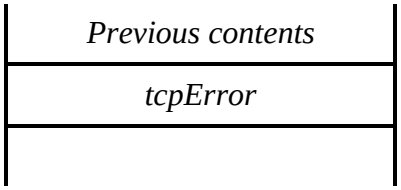


Word — Space for result

Word — Connection to use

<— SP

Stack after call



Word — TCP logic error code

<— SP

Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC `FUNCTION TCPIPListenTCP (%) as %`

C `extern pascal Word TCPIPListenTCP (Word);`

Pascal `function  TCPIPListenTCP (ipid: integer): integer;`

`tcpErr` This will be one of the `tcperr*` equates.

All successful `TCPIPListenTCP` calls must be balanced at some stage with a successful `TCPIPCloseTCP` call.

If the `ipid` has logged in with a non-zero `destPort`, then this listen request will only respond to incoming requests from that port number. Use a `destPort` of $0000 to catch all incoming requests.

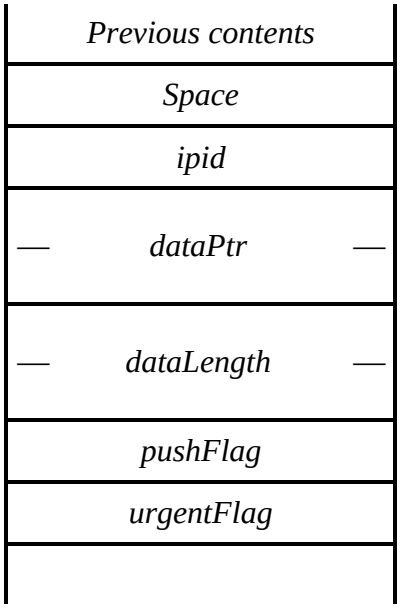
◆NOTE: *The current `TOS` and `TTL` values for the `ipid` are saved, and new preferred internal values for TCP are substituted.*

TCPIPWriteTCP $2D36

Writes data to the send queues, ready to be sent across a TCP connection.

Parameters

Stack before call



Word — Space for result

Word — Connection to use

Long — Pointer to data to queue

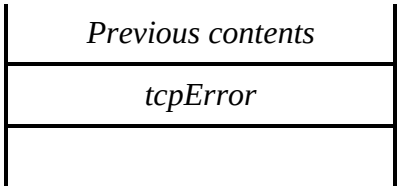
Long — Length of data to queue

Word — Push this data?

Word — Mark this data as urgent?

<— SP

Stack after call



Word — TCP logic error code

<— SP

Errors

`terrNOCONNECTION`	Not currently connected to the network
`terrBADIPID`	This `ipid` has not yet been logged in

BASIC

```
FUNCTION TCPIPWriteTCP (%, dataPtr, &, %, %) as %
```

C

```
extern pascal Word TCPIPWriteTCP (Word, dataPtr, Long,
                                  Boolean, Boolean);
```

Pascal

```
function  TCPIPWriteTCP (ipid: integer; dPtr: dataPtr;
                         dataLength: longint; pushFlag,
                         urgentFlag: boolean): integer;
```

`pushFlag` A boolean indicating whether to queue this with a local and destination push (`TRUE` – non-zero) or to queue as per normal (`FALSE` - $0000).

`urgentFlag` A boolean indicating whether to interrupt normal transmission and queue this as urgent data (`TRUE` – non-zero) or to queue as per normal (`FALSE` - $0000).

`tcpErr` This will be one of the `tcperr*` equates.

This request returns immediately, after queuing the data. Marinetti will actually send the data when and as it is able to.

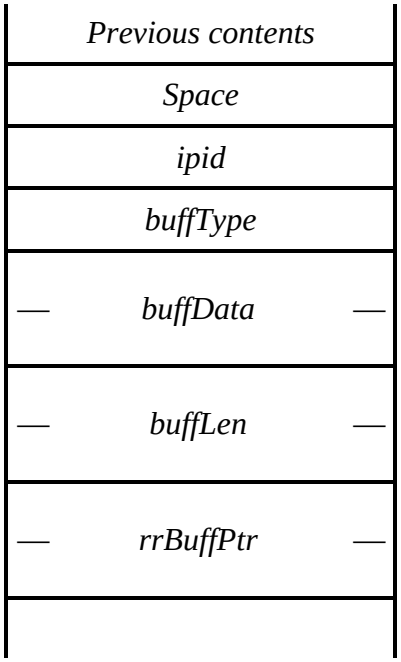
◆NOTE: *For those new to TCP programming, the urgentFlag parameter does not indicate that the data is simply urgent. It is a standard TCP function which initiates a number of events which may or may not include purging of data already in transit. It is advised not to use this parameter unless you fully understand the consequences. The pushFlag is also a standard TCP function, and while it will not purge data, you should be familiar wiuh the concept of a TCP "push" before using it.*

TCPIPReadTCP $2E36

Reads data from the TCP receive buffer, into a user buffer.

Parameters

Stack before call



Word — Space for result

Word — Connection to use

Word — Type of buffer in buffData

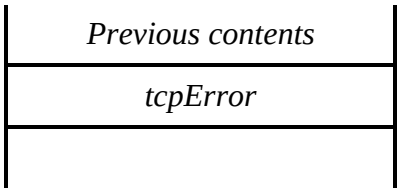
Long — Buffer descriptor

Long — Length of buffer

Long — Pointer to read response buffer

<— SP

Stack after call



Word — TCP logic error code

<— SP

Errors

`terrNOCONNECTION`	Not currently connected to the network
`terrBADIPID`	This `ipid` has not yet been logged in

BASIC

```
FUNCTION TCPIPReadTCP (%, %, univ, &, rrBuffPtr) as %

type rrBuff
    rrBuffCount as long
    rrBuffHandle as handle
    rrMoreFlag as boolean
    rrPushFlag as boolean
    rrUrgentFlag as boolean
end type
type rrBuffPtr as pointer to rrBuff
```

C

```
extern pascal Word TCPIPReadTCP (Word, Word, Ref, Long,
                    rrBuffPtr);

typedef struct {
    Long rrBuffCount;
    Handle rrBuffHandle;
    Word rrMoreFlag;
    Word rrPushFlag;
    Word rrUrgentFlag;
    } rrBUff, *rrBuffPtr;
```

Pascal

```
function  TCPIPReadTCP (ipid, buffType: integer; data:
                    univ longint; buffLen: longint;
                    bPtr: rrBuffPtr): integer;

rrBuff = record
   rrBuffCount: longint;
   rrBuffHandle: handle;
   rrMoreFlag: boolean;
   rrPushFlag: boolean;
   rrUrgentFlag: boolean;
   end;
rrBuffPtr = ^rrBuff;
```

`buffType` Describes the buffer type in `buffData`, and must be one of the following:

`$0000`	`buffData` is a pointer to a buffer for the read data.
`$0001`	`buffData` is a handle to contain the read data, and is resized by Marinetti to fit.
`$0002`	`buffData` is ignored. A new handle is created, and returned containing the read data.

`buffLength` This is the maximum length of the read data. Marinetti will only read up to `buffLen` number of bytes into the buffer.

On return from the call, the requested 14 byte read response buffer is completed as follows:

`+00`	`rrBuffCount`	`longword`	Length of the returned data
`+04`	`rrBuffHandle`	`handle`	Handle to the data
`+08`	`rrMoreFlag`	`word`	Is there more data received?
`+10`	`rrPushFlag`	`word`	Was this buffer pushed?
`+12`	`rrUrgentFlag`	`word`	Is this urgent data?

`rrBuffHandle` Contains the handle to the data, only if `buffType` was $0002, and `rrbuffCount` > 0.

`rrMoreFlag` A boolean indicating whether there is any data left in the queue still to read (`TRUE` – non-zero) or this read has emptied the queue (`FALSE` - $0000).

`rrPushFlag` A boolean indicating whether this data was pushed (`TRUE` – non-zero) or not (`FALSE` - $0000).

`rrUrgentFlag` A boolean indicating whether this is urgent data (`TRUE` – non-zero) or not (`FALSE` - $0000).

When you issue a `TCPIPReadTCP` call, there are a number of logic steps which dictate how much data is actually read. The Marinetti logic goes roughly like this:

1. Check how much data we have actually received from the connection. This is our maximum return count, or `maxrec`.

2. See how much data the user is asking for, via the `buffLength` parameter.

3. Whichever is smallest out of `buffLength` and `maxrec`, becomes the amount to read, which becomes `rrBuffCount`.

4. Was there an `rrPushFlag` set inside the data stream from the head of the queue up until `rrBuffCount`? If so, `rrBuffCount` becomes the offset into the data stream of the end of the push, so only the pushed data is returned.

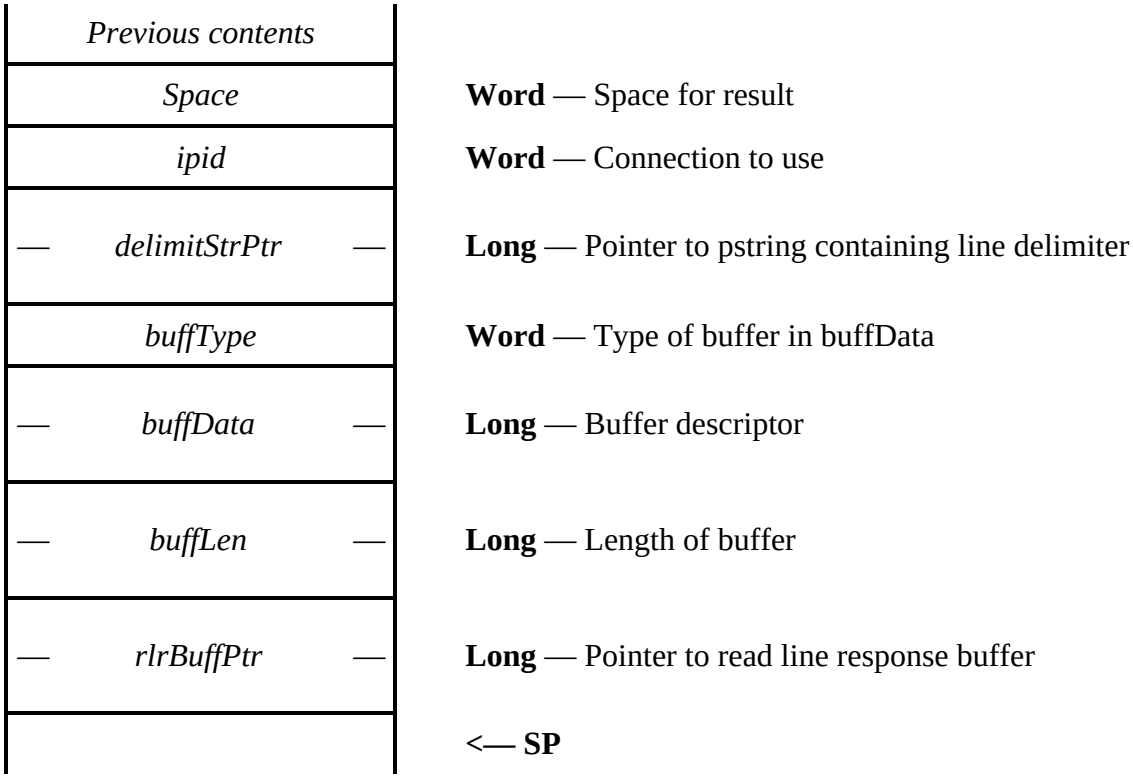
5. Return `rrBuffCount` bytes to the user.

TCPIPReadLineTCP $5E36

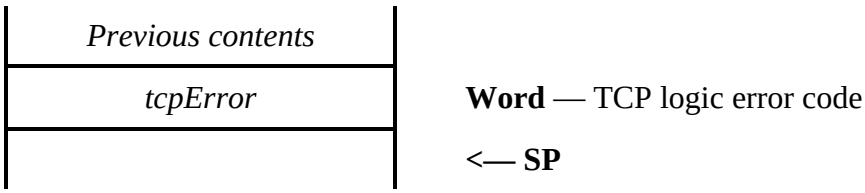
Reads a line of data from the TCP receive buffer, into a user buffer.

Parameters

Stack before call



Stack after call



Errors

`terrNOCONNECTION`	Not currently connected to the network
`terrBADIPID`	This `ipid` has not yet been logged in
`terrBUFFERTOOSMALL`	Buffer is too small

BASIC

```
FUNCTION TCPIPReadLineTCP (%, pstringPtr, %, univ, &,
                           rlrBuffPtr) as %

type rlrBuff
    rlrBuffCount as long
    rlrBuffHandle as handle
    rlrIsDataFlag as boolean
    rlrMoreFlag as boolean
```

```
    rlrBuffSize as long
end type
type rlrBuffPtr as pointer to rlrBuff
```

C

```
extern pascal Word TCPIPReadLineTCP (Word, char *,
                    Word, Ref, Long, rlrBuffPtr)

typedef struct {
    Long rlrBuffCount;
    Handle rlrBuffHandle;
    Word rlrIsDataFlag;
    Word rlrMoreFlag;
    Long rlrBuffSize;
    } rlrBuff, *rlrBuffPtr;
```

Pascal

```
function  TCPIPReadLineTCP (ipid, delimitStrPtr:
                    pStringPtr; buffType: integer;
                    data: univ longint; buffLen:
                    longint; bPtr: rlrBuffPtr):
                    integer;

rlrBuff = record
   rlrBuffCount: longint;
   rlrBuffHandle: handle;
   rlrIsDataFlag: boolean;
   rlrMoreFlag: boolean;
   rlrBuffSize: longint;
   end;
rlrBuffPtr = ^rlrBuff;
```

`delimitStrPtr` Points to a pstring to use as a line delimiter. Pushes and urgents are ignored, and the user buffer is only filled if the delimiter string has been received. If `delimitStrPtr` is nil, then this call is routed to `TCPIPReadTCP` instead.

If bit31 is set, then the delimiter is not stripped from the line before it is returned.

All other parameters are the same as the `TCPIPReadTCP` call.

On return from the call, the requested 16 byte read response buffer is completed as follows:

+00	rlrBuffCount	longword	Length of the returned data
+04	rlrBuffHandle	handle	Handle to the data
+08	rlrIsDataFlag	word	Was a line actually read?
+10	rlrMoreFlag	word	Is there more data received?
+12	rlrBuffSize	longword	Required buffer size

If a `terrBUFFERTOOSMALL` error is returned, then the line was too large for the supplied buffer. The required size, whether the buffer was filled or not, is always returned in `rrBuffSize`.

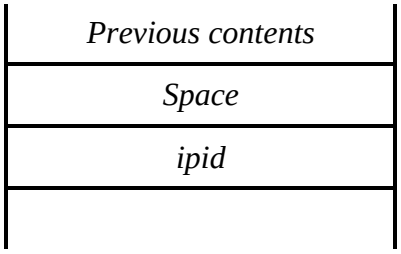
It is possible for `rlrBuffCount` to be `nil` and `rlrIsDataFlag` to be `true`, indicating that a null line was read.

TCPIPCloseTCP $2F36

Issues a close of a connection.

Parameters

Stack before call

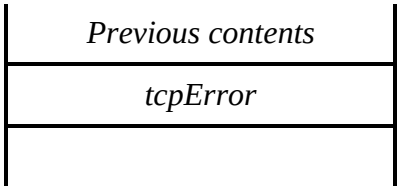


Word — Space for result

Word — Connection to use

<— SP

Stack after call



Word — TCP logic error code

<— SP

Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC `FUNCTION TCPIPCloseTCP (%) as %`

C `extern pascal Word TCPIPCloseTCP (Word);`

Pascal `function  TCPIPCloseTCP (ipid: integer): integer;`

`tcpErr` This will be one of the `tcperr*` equates.

Closing a connection involves handshaking across the network. As such, this call simply sets a flag indicating that Marinetti is to close the connection as soon as possible. Use the `TCPIPStatusTCP` call to check when the connection is finally `CLOSED`.

Closing state transition will normally go from `CLOSING`, which is set by `TCPIPCloseTCP`, through `FINWAIT1` and `FINWAIT2` while the close is negotiated by each end, on to `TIMEWAIT` and then finally `CLOSED`.

`TIMEWAIT` indicates that the connection is effectively closed as far as each end of the connection is concerned, and Marinetti is simply reserving the port so as to expire lost network datagrams. Once Marinetti is happy with making the port number available again, it will do so. The timeout period will vary depending on the network time, but will be a minimum of two minutes.

Again, the `TCPIPStatusTCP` call will tell you when the state has finally gone to `CLOSED`, but for all intents, unless you wish to use the same port number, `TIMEWAIT` indicates a successful close. You may

issue a TCPIPLogout call in either TIMEWAIT or CLOSED state, and Marinetti will take care of the rest of the close for you.

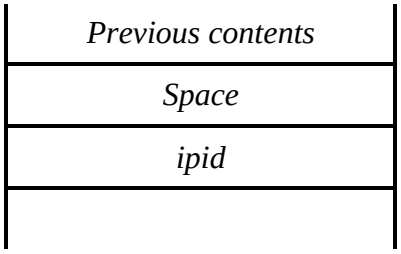
When the connection finally closes (ie. CLOSED), Marinetti restores the original TOS and TTL values which were saved when the connection was opened, although if you have already logged out, this obviously won't be an issue.

TCPIPAbortTCP $3036

Forces a connection to abnormally close.

Parameters

Stack before call

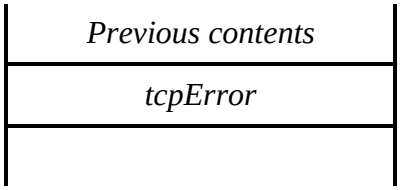


Word — Space for result

Word — Connection to use

<— SP

Stack after call



Word — TCP logic error code

<— SP

Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC
```
FUNCTION TCPIPAbortTCP (%) as %
```

C
```
extern pascal Word TCPIPAbortTCP (Word);
```

Pascal
```
function  TCPIPAbortTCP (ipid: integer): integer;
```

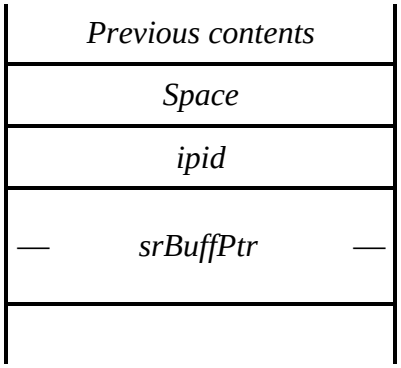
`tcpErr` This will be one of the `tcperr*` equates.

TCPIPStatusTCP $3136

Returns the status of a connection.

Parameters

Stack before call



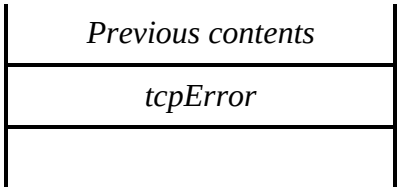
Word — Space for result

Word — Connection to use

Long — Pointer to status response buffer

<— SP

Stack after call



Word — TCP logic error code

<— SP

Errors		
	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC

```
FUNCTION TCPIPStatusTCP (%, srBuffPtr) as %

type srBuff
    srState as integer
    srNetworkError as integer
    srSndQueued as long
    srRcvQueued as long
    srDestIP as long
    srDestPort as integer
    srConnectType as integer
    srAcceptCount as integer
end type
type srBuffPtr as pointer to srBuff
```

C

```
extern pascal Word TCPIPStatusTCP (Word, srBuffPtr);

typedef struct {
    Word srState;
    Word srNetworkError;
    Long srSndQueued;
    Long srRcvQueued;
```

```
    Long srDestIP;
    Word srDestPort;
    Word srConnectType;
    Word srAcceptCount;
    } srBuff, *srBuffPtr;
```

Pascal

```
function  TCPIPStatusTCP (ipid: integer; sPtr:
                    srBuffPtr): integer;

srBuff = record
   srState: integer;
   srNetworkError: integer;
   srSndQueued: longint;
   srRcvQueued: longint;
   srDestIP: longint;
   srDestPort: integer;
   srConnectType: integer;
   srAcceptCount: integer;
   end;
srBuffPtr = ^srBuff;
```

`tcpErr` This will be one of the `tcperr*` equates.

On return from the call, the requested status response buffer is completed as follows:

+00	srState	word	TCP state
+02	srNetworkError	word	ICMP error code
+04	srSndQueued	longword	Bytes left in send queue
+08	srRcvQueued	longword	Bytes left in receive queue
+12	srDestIP	longword	Destination IP address
+16	srDestPort	word	Destination port
+18	srConnectType	word	Connection type
+20	srAcceptCount	word	If in listen mode, number of pending incoming requests

`srState` Indicates the current state of the TCP state machine. This will be one of the `tcps*` equates.

`srNetworkError` If the connection fails, normally indicated by the state going to `tcpsTIMEWAIT` or `tcpsCLOSED`, without application involvement, then the error code from an ICMP Port Unreachable Error message will indicate what caused the problem. Because $0000 is a valid error code, bit15 is used to indicate whether the error is relevant to this connection.

$8000	Network unreachable
$8001	Host unreachable
$8002	Protocol unreachable
$8003	Port unreachable
$8004	Fragmentation needed but DF bit set
$8005	Source route failed

$8006	Destination network unknown
$8007	Desination host unknown
$8009	Destination network administratively prohibited
$800A	Destination host administratively prohibited
$800B	Network unreachable for TOS
$800C	Host unreachable for TOS

If an ICMP Time Expired Error is received, then the segment is simply re–sent, in the hope that a shorter path may be found. This differs from UDP, where an error is reported back to the caller.

`srConnectType` Indicates the type of connection that is open, and will be one of the following:

$0000	Active (client) connection (See TCPIPOpenTCP)
$0001	Passive (listen/server) connection (See TCPIPListenTCP)

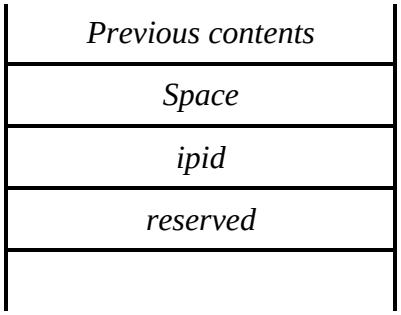
`srAcceptCount` Indicates the number of incoming requests queued if this is a passive connection.

TCPIPAcceptTCP $4F36

If a TCP connection in listen mode (see `TCPIPListenTCP`) has accepted an incoming connection request, then this call will create a new `ipid` to control the connection. The original `ipid` is left open, ready to accept more incoming requests.

Parameters

Stack before call



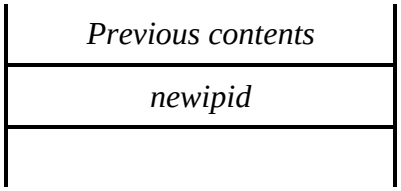
Word — Space for result

Word — Connection to use

Word — Reserved for future use. Use $0000

<— SP

Stack after call



Word — `ipid` of new connection

<— SP

Errors		
	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in
	`terrNOINCOMING`	There is no pending incoming request
	`terrNOTSERVER`	This `ipid` is not set up as a server

BASIC

```
FUNCTION TCPIPAcceptTCP (%, %) as %
```

C

```
extern pascal Word TCPIPAcceptTCP (Word, Word);
```

Pascal

```
function  TCPIPAcceptTCP (ipid, reserved: integer):
                          integer;
```

If successful, `TCPIPAcceptTCP` implicitly issues `TCPIPOpenTCP` on the new `ipid`, and thus must be closed by the application when no longer required, by calling `TCPIPCloseTCP`. Likewise the actual listen request, initiated by `TCPIPListenTCP`, must also be closed by the application when no longer required, by calling `TCPIPCloseTCP`.

Transport administration tool calls

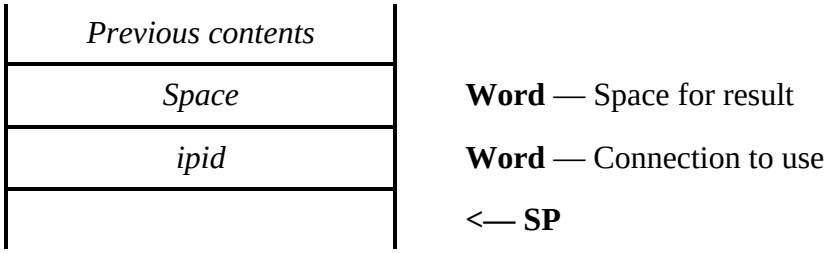
These calls deal with transport adminstration, such as parameters in control blocks, and transport layer performance functions.

TCPIPGetSourcePort $3236

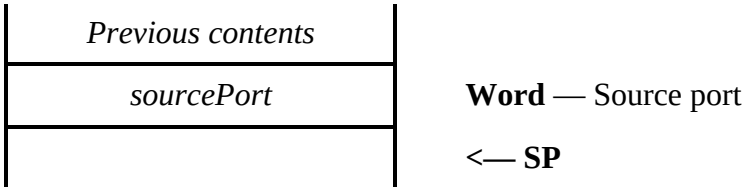
Returns the current source port for the specified `ipid`.

Parameters

Stack before call



Stack after call



Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC

```
FUNCTION TCPIPGetSourcePort (%) as %
```

C

```
extern pascal Word TCPIPGetSourcePort (Word);
```

Pascal

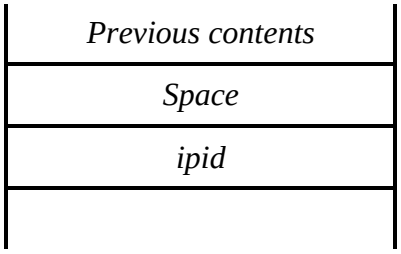
```
function  TCPIPGetSourcePort (ipid: integer): integer;
```

TCPIPGetTOS $3336

Returns the current Type Of Service value for a specified `ipid`.

Parameters

Stack before call

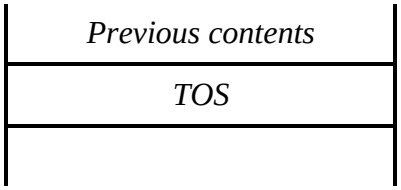


Word — Space for result

Word — Connection to use

<— SP

Stack after call



Word — Type Of Service (TOS)

<— SP

Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC
```
FUNCTION TCPIPGetTOS (%) as %
```

C
```
extern pascal Word TCPIPGetTOS (Word);
```

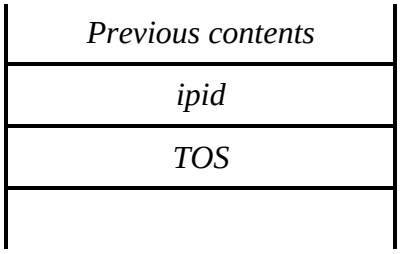
Pascal
```
function  TCPIPGetTOS (ipid: integer): integer;
```

TCPIPSetTOS $3436

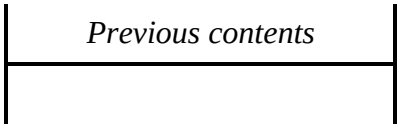
Sets a new Type Of Service value for a specified `ipid`.

Parameters

Stack before call



Stack after call



Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC

```
SUB TCPIPSetTOS (%, %)
```

C

```
extern pascal void TCPIPSetTOS (Word, Word);
```

Pascal

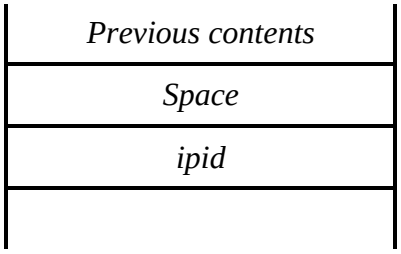
```
procedure TCPIPSetTOS (ipid, TOS: integer);
```

TCPIPGetTTL $3536

Returns the current Time To Live value for a specified `ipid`.

Parameters

Stack before call

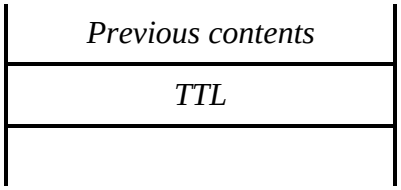


Word — Space for result

Word — Connection to use

<— SP

Stack after call



Word — Time To Live (TTL)

<— SP

Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC
```
FUNCTION TCPIPGetTTL (%) as %
```

C
```
extern pascal Word TCPIPGetTTL (Word);
```

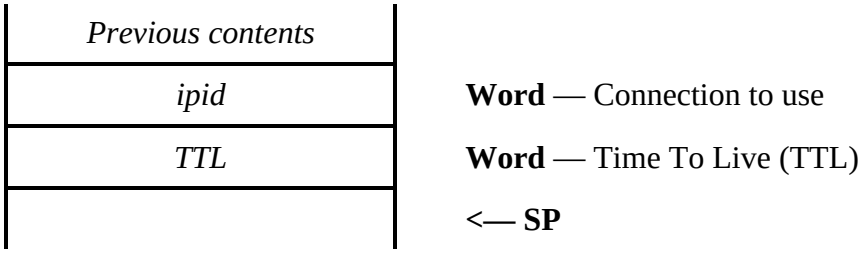
Pascal
```
function  TCPIPGetTTL (ipid: integer): integer;
```

TCPIPSetTTL $3636

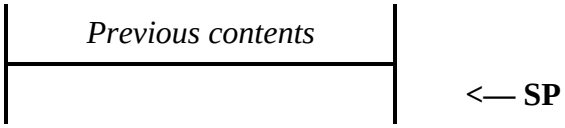
Sets a new Time To Live value for a specified `ipid`.

Parameters

Stack before call



Stack after call



Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC

```
SUB TCPIPSetTTL (%, %)
```

C

```
extern pascal void TCPIPSetTTL (Word, Word);
```

Pascal

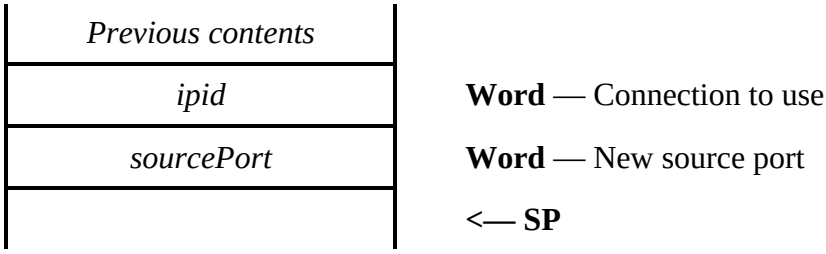
```
procedure TCPIPSetTTL (ipid, TTL: integer);
```

TCPIPSetSourcePort $3736

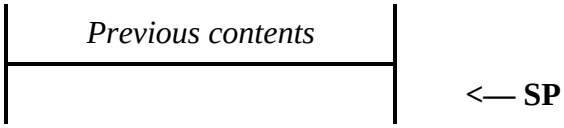
Sets a new source port for a specified `ipid`.

Parameters

Stack before call



Stack after call



Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC

```
SUB TCPIPSetSourcePort (%, %)
```

C

```
extern pascal void TCPIPSetSourcePort (Word, Word);
```

Pascal

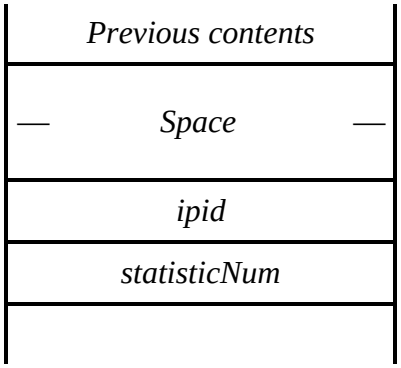
```
procedure TCPIPSetSourcePort (ipid, sourcePort:
                              integer);
```

TCPIPGetUserStatistic $4936

Returns a specific statistic for the specified `ipid`.

Parameters

Stack before call



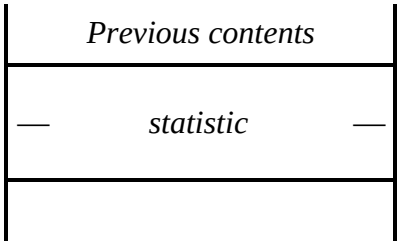
Long — Space for result

Word — Connection to use

Word — Which statistic to return

<— SP

Stack after call



Long — Returned statistic

<— SP

Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in
	`terrBADPARAMETER`	Invalid parameter for this call

BASIC

```
FUNCTION TCPIPGetUserStatistic (%, %) as &
```

C

```
extern pascal Long TCPIPGetUserStatistic (Word, Word);
```

Pascal

```
function  TCPIPGetUserStatistic (ipid, statisticNum:
                        integer): longint;
```

`statisticNum` Indicates which statistic to return.

`$0001`	Number of data bytes (octets) received by the current or most recent TCP connection by this `ipid`
`$0002`	Number of data bytes (octets) sent by the current or most recent TCP connection by this `ipid`

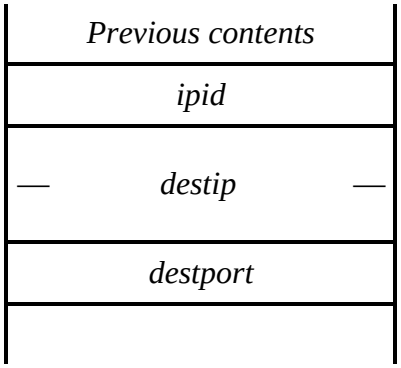
`statistic` The returned value depends upon the statistic requested.

TCPIPSetNewDestination $5036

Sets a new destination IP address and port, which will take affect next time a connection is initiated.

Parameters

Stack before call



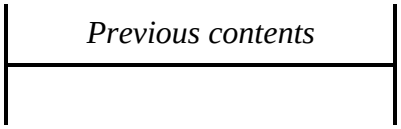
Word — Connection to use

Long — Destination IP address

Word — Destination port number

<— SP

Stack after call



<— SP

Errors	`terrNOCONNECTION`	Not currently connected to the network
	`terrBADIPID`	This `ipid` has not yet been logged in

BASIC

```
SUB TCPIPSetNewDestination (%, &, %)
```

C

```
extern pascal void TCPIPSetNewDestination (Word, Long,
                    Word);
```

Pascal

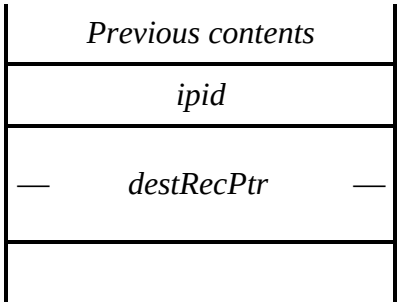
```
procedure TCPIPSetNewDestination (ipid: integer;
                    destip: longint; destPort:
                    integer);
```

TCPIPGetDestination $6236

Returns the destination IP address and port which is being used for this `ipid`.

Parameters

Stack before call

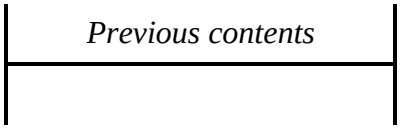


Word — Connection to use

Long — Pointer to response record

<— SP

Stack after call



<— SP

Errors

`terrNOCONNECTION`	Not currently connected to the network
`terrBADIPID`	This `ipid` has not yet been logged in

BASIC

```
SUB TCPIPGetDestination (%, destRecPtr)

type destRec
    drUserID as integer
    drDestIP as long
    drDestPort as integer
end type
type destRecPtr as pointer to destRec
```

C

```
extern pascal void TCPIPGetDestination (Word,
                    destRecPtr);

typedef struct {
    Word drUserID;
    Long drDestIP;
    Word drDestPort;
    } destRec, *destRecPtr;
```

Pascal

```
procedure TCPIPGetDestination (ipid: integer; dPtr:
                    destRecPtr);

destRec = record
   drUserID: integer;
   drDestIP: longint;
```

```
   drDestPort: integer;
   end;
destRecPtr = ^destRec;
```

`destRecPtr` Points to the following response record:

+00	drUserID	word	UserID used by this ipid
+02	drDestIP	longword	Destination IP address
+06	drDestPort	word	Destination port number

Library type calls

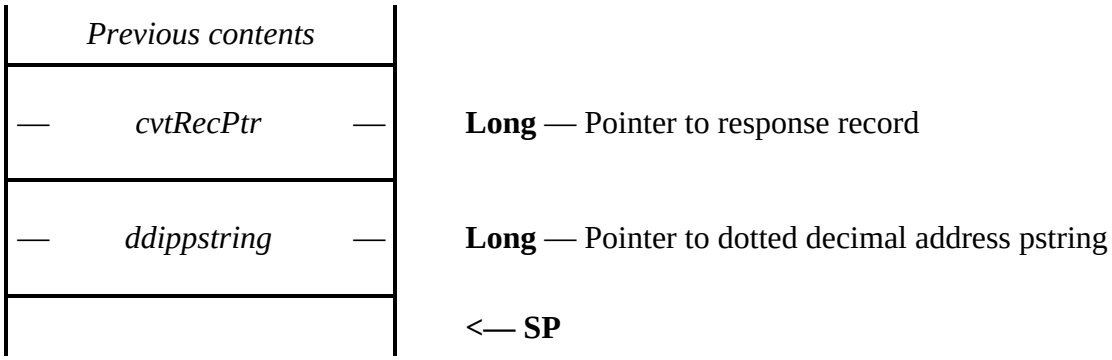
These calls are typical of generic library functions, and do not directly deal with networking. They are primarily internal routines which may also be useful for developers.

TCPIPConvertIPToHex $0D36

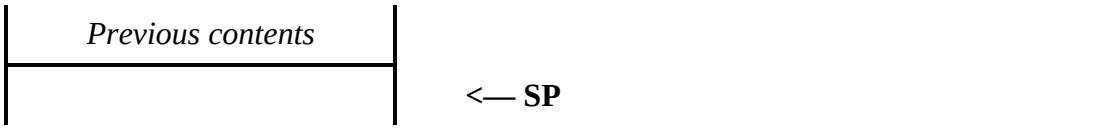
Convert an ASCII text string representing a dotted decimal IP address, optionally followed by a comma delimited port number, into their equivalent number values.

Parameters

Stack before call



Stack after call



Errors None.

BASIC

```
SUB TCPIPConvertIPToHex (cvtRecPtr, pstringPtr)

type cvtRec
    cvtIPAddress as long
    cvtPort as integer
end type
type cvtRecPtr as pointer to cvtRec
```

C

```
extern pascal void TCPIPConvertIPToHex (cvtRecPtr,
                    char *);

typedef struct {
    Long cvtIPAddress;
    Word cvtPort;
    } cvtRec, *cvtRecPtr;
```

Pascal

```
procedure TCPIPConvertIPToHex (cvt: cvtRecPtr; sPtr:
                    pStringPtr);

cvtRec = record
   cvtIPAddress: longint;
   cvtPort: integer;
   end;
cvtRecPtr = ^cvtRec;
```

`cvtRecPtr` Points to the response record. The layout is as follows:

`+00`	`cvtIPAddress`	`longword`	Returned IP address
`+04`	`cvtPort`	`word`	Port number or `nil` if none

`ddippstring` A pointer to a pstring containing an ASCII string for the dotted decimal address to convert.

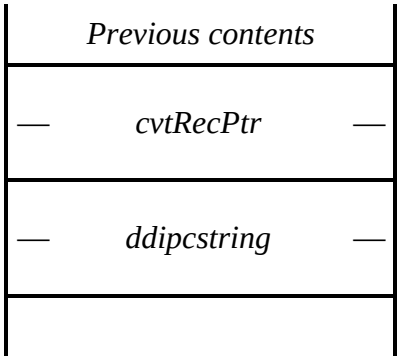
If the dotted decimal IP address is followed by a “,” (comma) a “:” (colon) or a “;” (semi–colon) and then a number in the range 1 to 65535, then it will be returned as the port number. eg. “192.80.63.5:23” returns the Telnet port, which is 23.

TCPIPConvertIPCToHex $3F36

Convert an ASCII text string representing a dotted decimal IP address, optionally followed by a comma delimited port number, into their equivalent number values.

Parameters

Stack before call

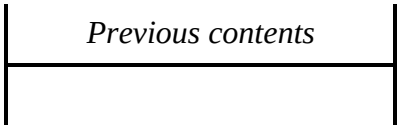


Long — Pointer to response record

Long — Pointer to dotted decimal address cstring

<— **SP**

Stack after call



<— **SP**

Errors None.

BASIC
```
SUB TCPIPConvertIPCToHex (cvtRecPtr, pstringPtr)
```

C
```
extern pascal void TCPIPConvertIPCToHex (cvtRecPtr,
                    char *);
```

Pascal
```
procedure TCPIPConvertIPCToHex (cvt: cvtRecPtr; sPtr:
                    pStringPtr);
```

`cvtRecPtr` Points to the response record. The layout is as follows:

`+00`	`cvtIPAddress`	`longword`	Returned IP address
`+04`	`cvtPort`	`word`	Port number or `nil` if none

`ddipcstring` A pointer to a cstring containing an ASCII string for the dotted decimal address to convert.

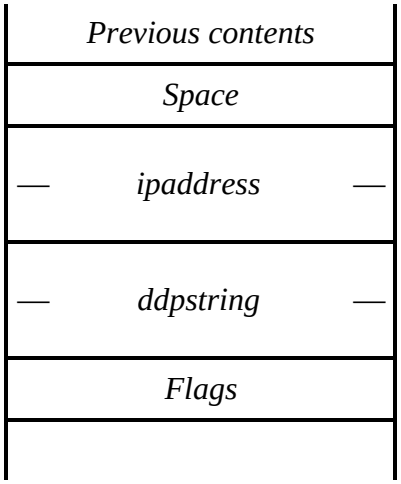
If the dotted decimal IP address is followed by a “,” (comma) a “:” (colon) or a “;” (semi–colon) and then a number in the range 1 to 65535, then it will be returned as the port number. eg. “192.80.63.5:23” returns the Telnet port, which is 23.

TCPIPConvertIPToASCII $0E36

Converts a longword IP address into a pstring ASCII text string of the dotted decimal equivalent.

Parameters

Stack before call



Word — Space for result

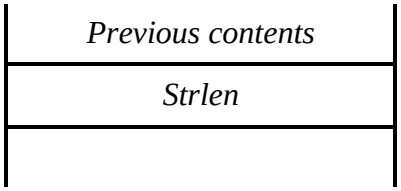
Long — The IP address to convert

Long — Pointer to the 16 byte return buffer

Word — Formatting flags

<— SP

Stack after call



Word — The length of the returned string

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPConvertIPToASCII (&, pstringPtr, %) as %
```

C
```
extern pascal Word TCPIPConvertIPToASCII (Long, char *,
                    Word);
```

Pascal
```
function  TCPIPConvertIPToASCII (ipaddress: longint;
                    ddpstring: pString15Ptr; flags:
                    integer): integer;
```

`flags` Various formatting flags:

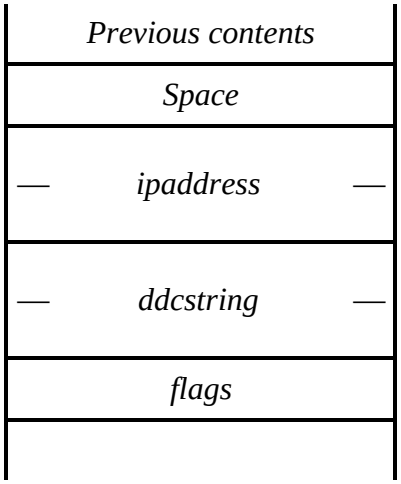
Bit 15 is for ASCII type; 0 = low ASCII, 1 = high ASCII

TCPIPConvertIPToCASCII $5836

Converts a longword IP address into a zero terminated ASCII text string of the dotted decimal equivelent.

Parameters

Stack before call



Word — Space for result

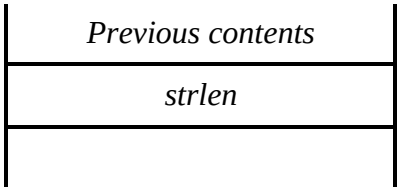
Long — The IP address to convert

Long — Pointer to the 16 byte return buffer

Word — Formatting flags

<— SP

Stack after call



Word — The length of the returned string

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPConvertIPToCASCII (&, pstringPtr, %) as %
```

C
```
extern pascal Word TCPIPConvertIPToCASCII (Long, char
                    *, Word);
```

Pascal
```
function  TCPIPConvertIPToCASCII (ipaddress: longint;
                    ddpstring: pString15Ptr; flags:
                    integer): integer;
```

`flags` Various formatting flags:

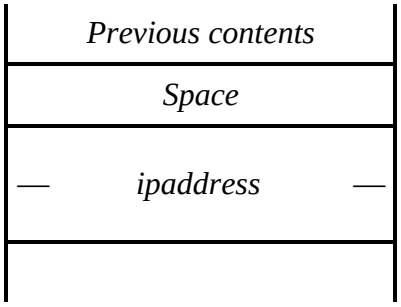
Bit 15 is for ASCII type; 0 = low ASCII, 1 = high ASCII

TCPIPConvertIPToClass $4136

Returns the class of a given IP address.

Parameters

Stack before call

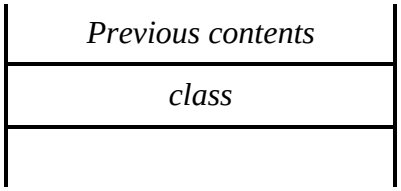


Word — Space for result

Long — The IP address

<— SP

Stack after call



Word — The class of the IP address

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPConvertIPToClass (&) as %
```

C
```
extern pascal Word TCPIPConvertIPToClass (Long);
```

Pascal
```
function  TCPIPConvertIPToClass (ipaddress: longint):
                    integer;
```

`ipclass` The class of the returned IP address will be one of the following:

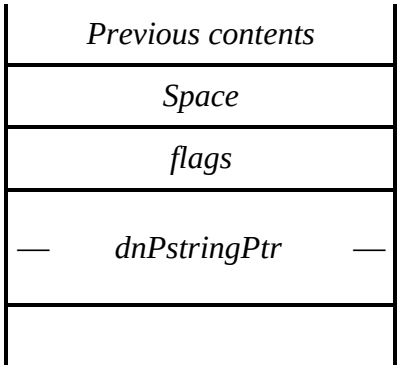
`$0000`	Class A
`$0001`	Class B
`$0002`	Class C
`$0003`	Class D
`$0004`	Class E

TCPIPMangleDomainName $5936

Takes an ASCII string as input, and modifies it for use as a syntactically correct domain name, which may be used as input to the DNR module.

Parameters

Stack before call



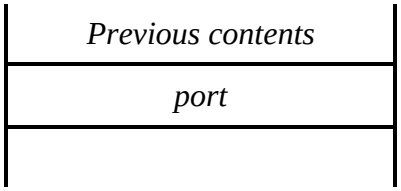
Word — Space for result

Word — Indicates which functions to perform

Long — Pointer to the domain name pstring

<— SP

Stack after call



Word — Returned port number

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPMangleDomainName (%, pstringPtr) as %
```

C
```
extern pascal Word TCPIPMangleDomainName (Word, char
                      *);
```

Pascal
```
function  TCPIPMangleDomainName (flags: integer;
                      dnPstringPtr: pStringPtr): integer;
```

`flags` Indicates which functions to perform. (Recommended = $F800):

bit15	Remove port suffix
bit14	Translate “%” tokens
bit13	Conversion to lower case
bit12	Delete illegal characters
bit11	Strip off high bit
bit10-bit0	Reserved – set to zeros

If all bits are set to zero, then the string will not be modified.

`dnPstringPtr` A pointer to a pstring containing an ASCII string domain name.

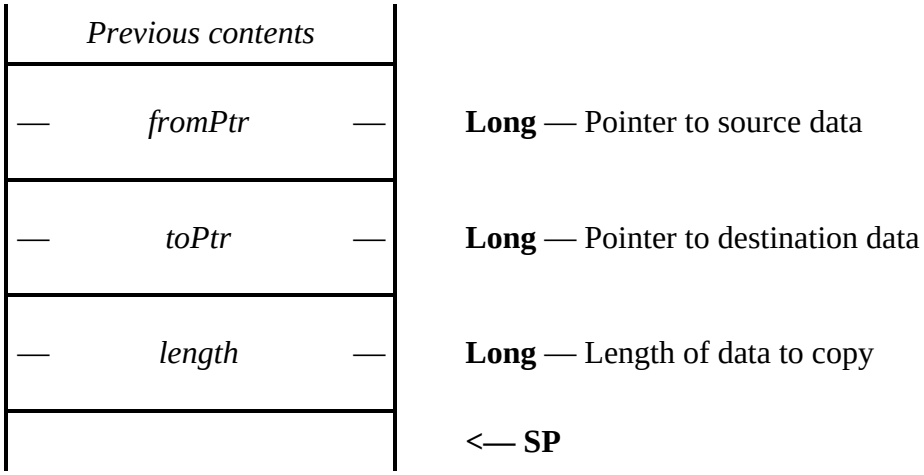
`port` If the domain name is is followed by a “,” (comma) a “:” (colon) or a “;” (semi–colon) and then a number in the range 1 to 65535, then it will be returned as the port number, else it will be `nil`. eg. “delphi.com:23” returns the Telnet port, which is 23.

TCPIPPtrToPtr $5536

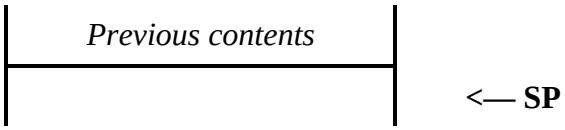
Copies memory in a forward direction from one location to another, optimised for speed.

Parameters

Stack before call



Stack after call



Errors None.

BASIC
```
SUB TCPIPPtrToPtr (ptr, ptr, &)
```

C
```
extern pascal void TCPIPPtrToPtr (ptr, ptr, Long);
```

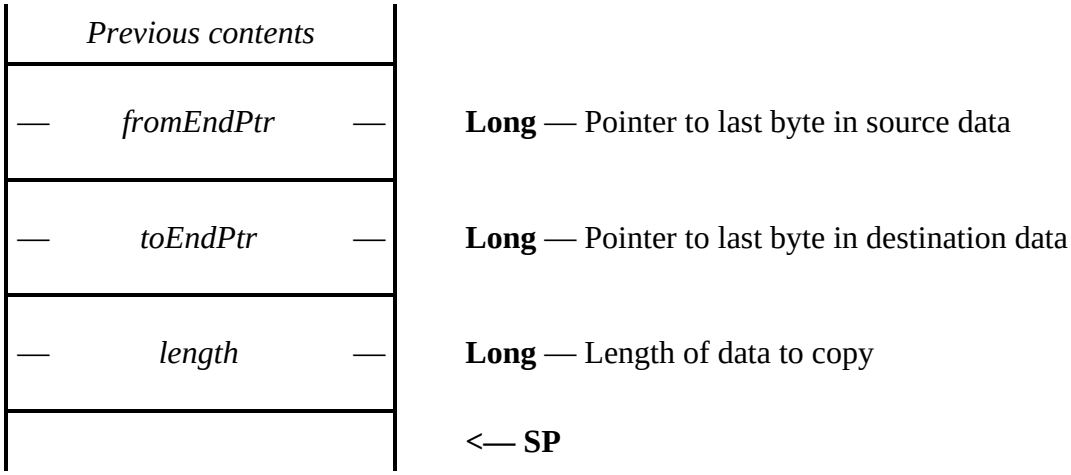
Pascal
```
procedure TCPIPPtrToPtr (fromPtr, toPtr: ptr; length:
                         longint);
```

TCPIPPtrToPtrNeg $5636

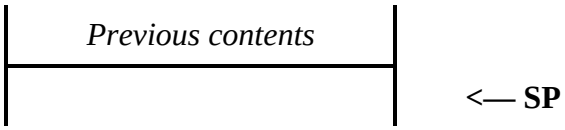
Copies memory in a backward direction from one location to another.

Parameters

Stack before call



Stack after call



Errors None.

BASIC
```
SUB TCPIPPtrToPtrNeg (ptr, ptr, &)
```

C
```
extern pascal void TCPIPPtrToPtrNeg (ptr, ptr, Long);
```

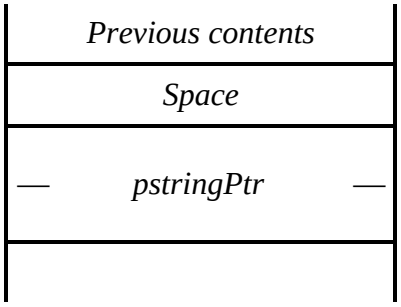
Pascal
```
procedure TCPIPPtrToPtrNeg (fromEndPtr, toEndPtr: ptr;
                            length: longint);
```

TCPIPValidateIPString $4836

Returns a flag indicating whether the passed pstring is a valid ASCII representation of an IP address.

Parameters

Stack before call

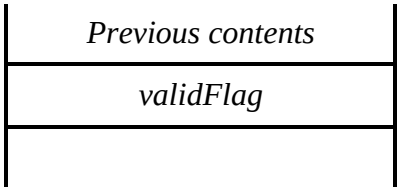


Word — Space for result

Long — Pointer to IP address pstring

<— SP

Stack after call



Word — Boolean

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPValidateIPString (pstringPtr) as %
```

C
```
extern pascal Word TCPIPValidateIPString (char *);
```

Pascal
```
function  TCPIPValidateIPString (sPtr: pStringPtr):
                        boolean;
```

`validFlag` The value returned is TRUE (non-zero) if this is a valid pstring, or FALSE ($0000) if it is not.

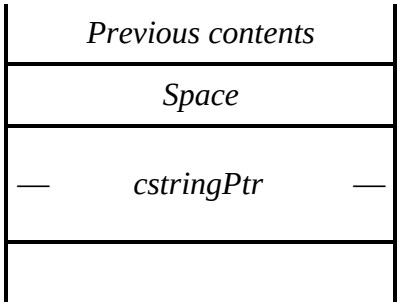
This call does not interrogate the resultant IP address to see if it exists or is valid from a network administration standpoint. It simply checks to make sure it is a valid dotted decimal address. ie. four numeric arguments, each between 0 and 255 inclusive, delimited by decimal point symbols.

TCPIPValidateIPCString $1536

Returns a flag indicating whether the passed cstring is a valid ASCII representation of an IP address.

Parameters

Stack before call

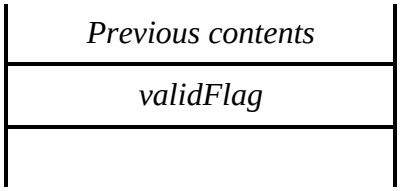


Word — Space for result

Long — Pointer to IP address cstring

<— SP

Stack after call



Word — Boolean

<— SP

Errors None.

BASIC
```
FUNCTION TCPIPValidateIPCString (cstringPtr) as %
```

C
```
extern pascal Word TCPIPValidateIPCString (char *);
```

Pascal
```
function  TCPIPValidateIPCString (sPtr: cStringPtr):
                        boolean;
```

`validFlag` The value returned is TRUE (non-zero) if this is a valid cstring, or FALSE ($0000) if it is not.

This call does not interrogate the resultant IP address to see if it exists or is valid from a network administration standpoint. It simply checks to make sure it is a valid dotted decimal address. ie. four numeric arguments, each between 0 and 255 inclusive, delimited by decimal point symbols.

Link layer modules

Link layer protocols, such as PPP, are handled through the Marinetti link layer module interface, and are individually loaded as required from the `*:System:TCPIP` folder.

Marinetti link layer modules are OMF files of file type $00BC and auxilliary type $00004083.

Once loaded, the module load point is used as a dispatch procedure, much like the toolbox, and is called with the accumulator and index registers long. Upon entry, the registers will be as follows:

`A`	Module's direct page if one was loaded as OMF, else $0000
`X`	Call number
`Y`	Marinetti UserID (While loaded, modules are consider a part of Marinetti, and as such, all memory allocations must use Marinetti's UserID, and not the module's)
`DBK`	Unknown
`DP`	Marinetti's direct page
`S`	RTL address then parameters

If the module was built with its own direct page, then `A` will contain its address in bank 0. A value of $0000 indicates there is no direct page allocated and you should either allocate your own now, or use part of Marinetti's. The direct page register will contain Marinetti's direct page, on which you have exclusive access to offsets $E0-$FF. These locations will be preserved for you across calls, and so may be used for permanent variables while loaded and started.

On exit, `A` will contain a Marinetti error code in the `terr_*` range ANDed with `terrmask`, with the carry flag indicating any errors. The `DBK` and `DP` registers must be restored, and the stack fixed to remove the input parameters passed by the call.

How your module works is up to you, as long as it conforms to the calling interface. All of the included modules which use the serial port, allocate an internal interrupt buffer at `LinkStartup` time, and build datagrams from there each time `LinkGetDatagram` is called. However, there is nothing to stop a module building within an interrupt loop and queuing internally, or letting an external processor, such as a network card, do the work, so long as IP datagrams are returned to Marinetti via the `LinkGetDatagram` call.

Configuration data for link layer modules are stored within Marinetti, and applications may access them using the `TCPIPGetConnectData`, `TCPIPSetConnectData`, `TCPIPGetDisconnectData` and `TCPIPSetDisconnectData` tool calls. Additionally, they may be edited using the `TCPIPEditLinkConfig` call.

Link layer modules must be careful when changing the layout of their connect and disconnect data, as users may have an older versions currently installed. Modules should either include a version word at the beginning of the data, or be able to recognise earlier layouts of the data.

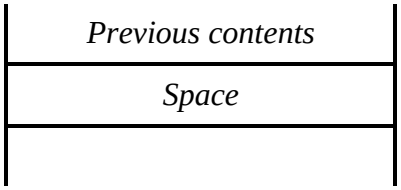
While Marinetti looks after saving the configuration data for each module, the data itself is private to the module concerned. For reference, the configuration data for SLIP and PPP (scripted) are currently defined as ASCII scripts, and all the rest which ship with Marinetti are proprietary.

LinkInterfaceV $0000

Returns the maximum link layer module interface which this link layer module supports.

Parameters

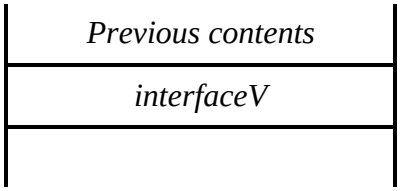
Stack before call



Word — Space for result

<— SP

Stack after call



Word — Interface version = $0001

<— SP

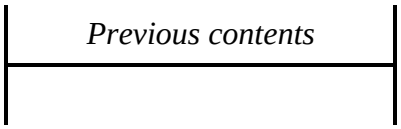
`interfaceV` The interface described in this document is $0001

LinkStartup $0002

Starts the link layer module once it is loaded. The module should perform any initialisation tasks short of actually starting a connection.

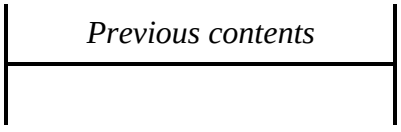
Parameters

Stack before call



<— SP

Stack after call



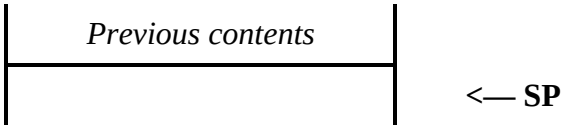
<— SP

LinkShutDown $0004

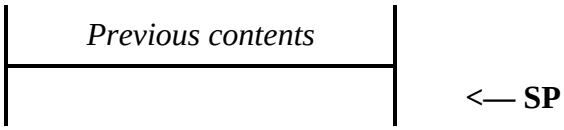
Marinetti will purge the module from memory, once this call has completed. The module has no choice in the matter.

Parameters

Stack before call



Stack after call

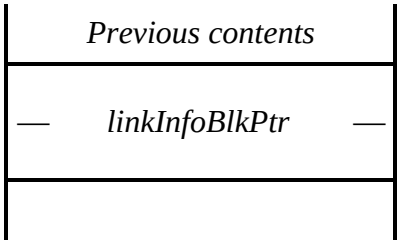


LinkModuleInfo $0006

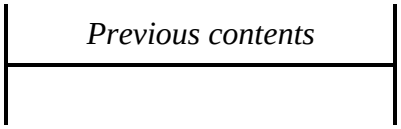
Returns information about the module.

Parameters

Stack before call



Stack after call



`linkInfoBlkPtr` Points to a fixed length 27 byte response buffer as follows:

`+00`	`liMethodID`	`word`	The connect method. New modules will need to apply to the author for a unique ID to use. *See* `conXXX` *equates for details of already defined values*
`+02`	`liName`	`21 bytes`	Pstring name of the module
`+23`	`liVersion`	`longword`	rVersion (type $8029 resource layout) of the module
`+27`	`liFlags`	`word`	Contains the following flags:

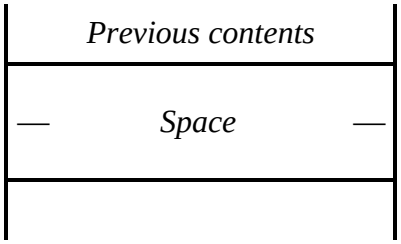
bit15	This link layer uses the built in Apple IIGS serial ports
bits14-1	Reserved – set to zeros
bit0	Indicates whether the module contains an rIcon resource

LinkGetDatagram $0008

Returns a raw data datagram from the network.

Parameters

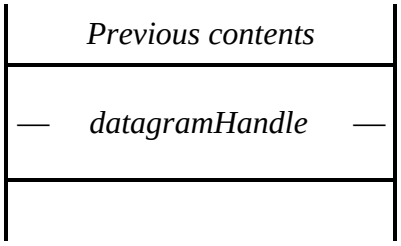
Stack before call



Long — Space for result

<— SP

Stack after call



Long — Handle containing the datagram

<— SP

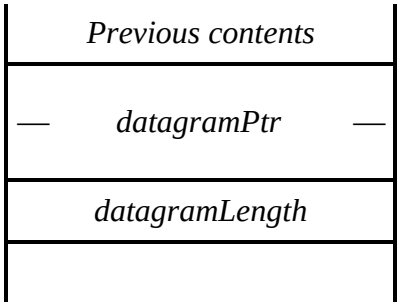
`datagramHandle` The handle must be allocated with Marinetti's UserID, which was passed to the module during the call, and must contain a valid IP datagram, stripped of any underlying network headers. If there is no datagram waiting, then `datagramHandle` will be returned as `nil`.

LinkSendDatagram $000A

Sends an IP datagram to the network via the module's datagram encapsulation.

Parameters

Stack before call

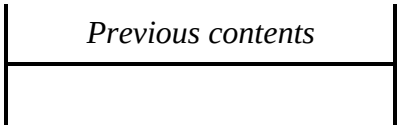


Long — Pointer to the datagram data

Word — Length of the datagram to send

<— SP

Stack after call



<— SP

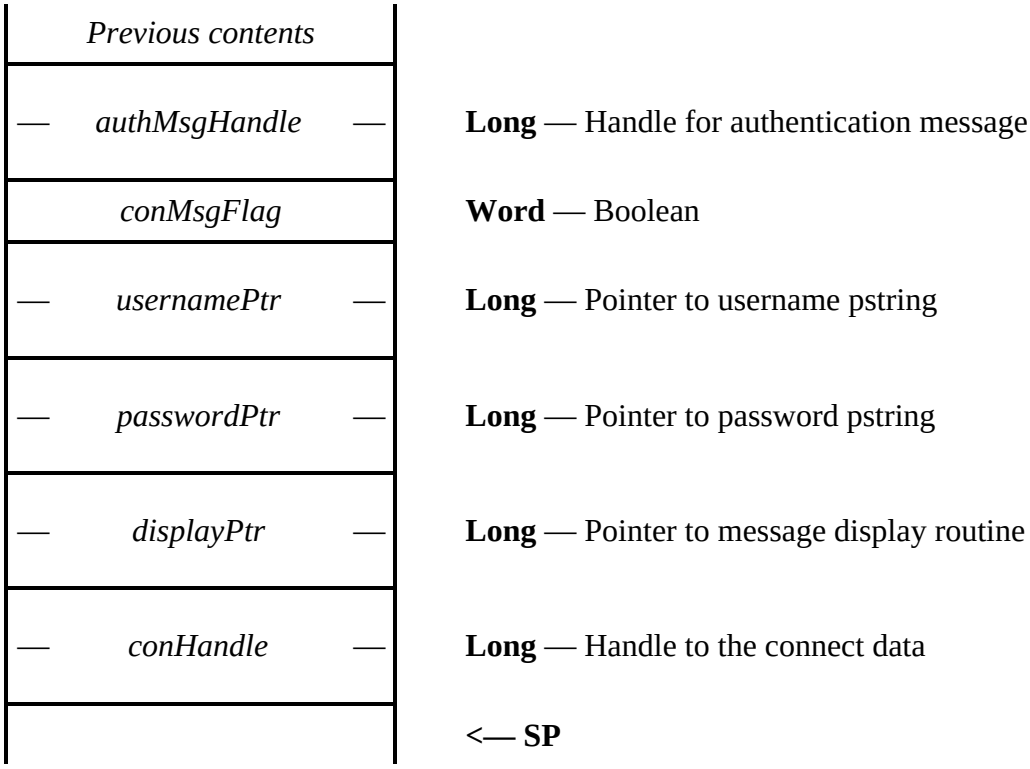
The module should wrap the datagram in the appropriate datagram encapsulation, and send it to the network.

LinkConnect $000C

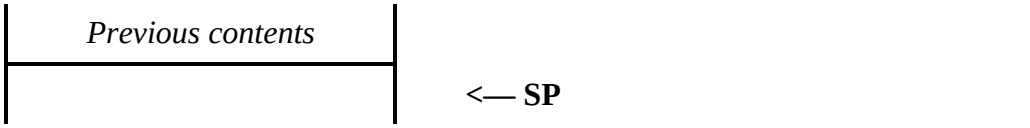
Attempts to connect Marinetti to the network.

Parameters

Stack before call



Stack after call



`authMsgHandle` If the link layer supports an authentication method, then any optional authentication messages should be copied into this handle, which is supplied as initially empty. If the link layer does not support authentication messages, then the handle should simply be ignored. The data may then be retrieved by an application by using the `TCPIPGetAuthMessage` call.

This is one of Marinetti's permanent data handles and must remain valid. You may resize it as required, by using `_SetHandleSize`.

`conMsgFlag` The value is TRUE (non-zero) if link layer modules are to display connect messages, and FALSE ($0000) if they are not.

`displayPtr` Points to the calling applications message display routine. It is possible for `conMsgFlag` to be `true`, yet `displayPtr` is `nil`. The link layer module is expected to handle this situation correctly, and not issue any display call backs. The module is also completely in charge of calling the displayPtr routine, including any register preservation required.*See* `TCPIPConnect` *for more details*.

`conHandle` The handle content must not be altered or purged, as it belongs to Marinetti.

Valid error codes are those returned by the `TCPIPConnect` tool call, ANDed with `terrmask`.

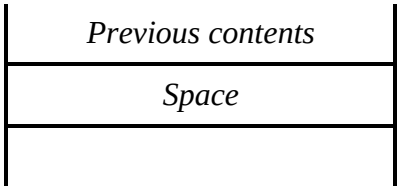
Once the link is active, the link layer must fill in the link layer variables (see `LinkGetVariables` call) correctly before returning.

LinkReconStatus $000E

Returns a flag indicating whether the module is in a state to reconnect.

Parameters

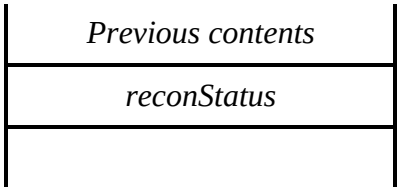
Stack before call



Word — Space for result

<— SP

Stack after call



Word — Boolean

<— SP

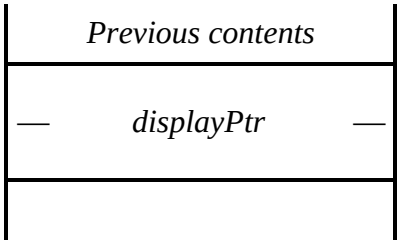
`reconStatus` The value returned is TRUE (non-zero) if this link layer module is able to reconnect to the network, and FALSE ($0000) if it is not.

LinkReconnect $0010

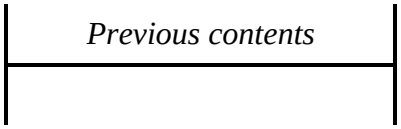
Attempts to reconnect to the network, assuming the physical connection is still active, but the logical connection is not. An example would be a serial connection such as SLIP, where the modem is still connected to an ISP after a reboot, and the user wants to continue from where they left off.

Parameters

Stack before call



Stack after call



Modules do not have to support this call. It is provided primarily for developers to stay connected while testing. If not supported, simply return an error in the `terr_*` range, and ANDed with `terrmask`, indicating an appropriate problem with the link. The only variable (see the `LinkGetVariables` call) the link layer module should touch during this call, is `lvConnected`.

If a link layer module wishes to save its own data so it may better support the reconnect facility, it should write its data to a file in `*:System:TCPIP:`, preferrably one named after the link layer, for example `PPP.state`.

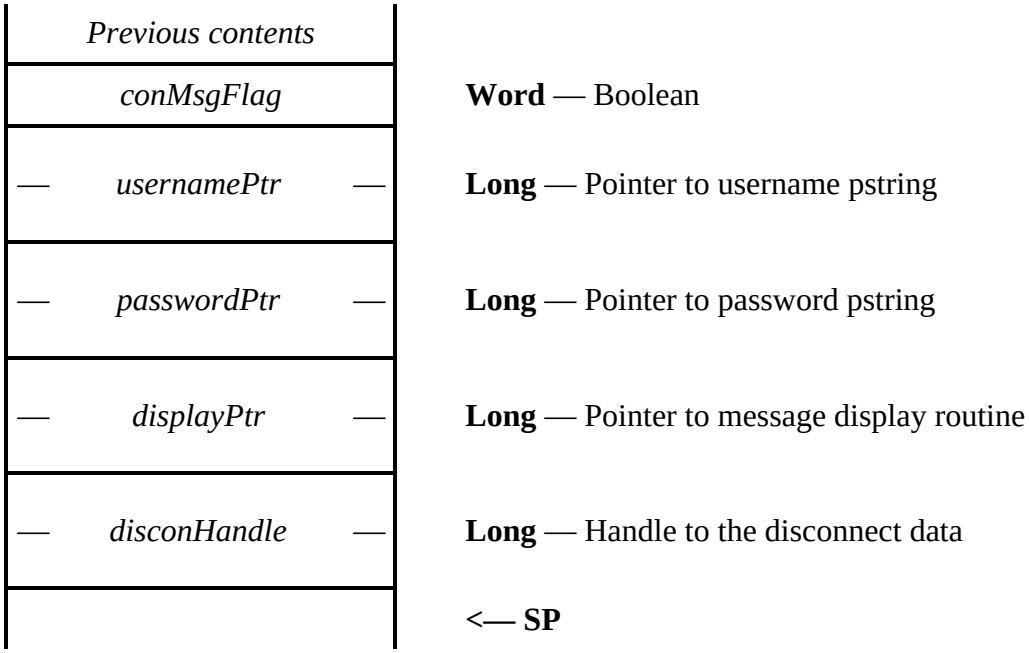
Before `LinkReconnect` is called, Marinetti will store the reconnection IP address in `lvIPaddress` in case the link layer module requires it. There is no facility for using a different IP address on a reconnection.

LinkDisconnect $0012

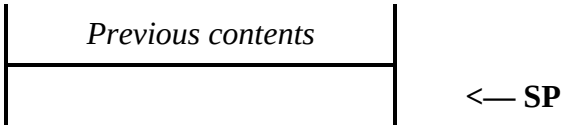
Attempts to disconnect Marinetti from the network.

Parameters

Stack before call



Stack after call



`conMsgFlag` The value is TRUE (non-zero) if link layer modules are to display connect messages, and FALSE ($0000) if they are not.

`displayPtr` Points to the calling application's message display routine. *See* `LinkConnect` *and* `TCPIPDisconnect` *for more details.*

`disconHandle` The handle content must not be altered or purged, as it belongs to Marinetti.

Valid error codes are those return by the `TCPIPDisconnect` tool call, and ANDed with `terrmask`.

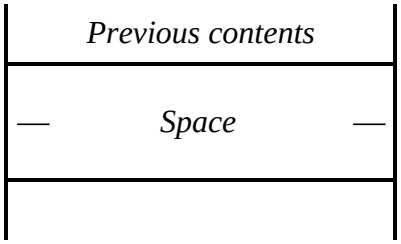
Before returning, the link layer module should set the `lvConnected` flag (see the `LinkGetVariables` call) appropriately. All other variables may be left as is, even though the link may have been dropped, and in fact may be used by Marinetti for post connection processing.

LinkGetVariables $0014

Returns a pointer to the link layer module's variables.

Parameters

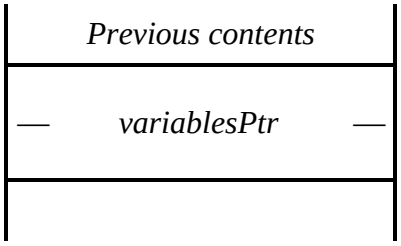
Stack before call



Long — Space for result

<— **SP**

Stack after call



Long — Pointer to variables

<— **SP**

`variablesPtr` Points to the following data block:

`+00`	`lvVersion`	`word`	Version of this record. The only record currently defined, is this one, which must be $0001.
`+02`	`lvConnected`	`word`	`true` ($8000) if currently connected, `false` ($0000) if not. Marinetti checks this often to see if the link is still up
`+04`	`lvIPaddress`	`longword`	Current IP address being used by the link layer module
`+08`	`lvRefCon`	`longword`	For internal use by the link layer module. This usually contains a pointer to another record containing variables specific to this link layer module
`+12`	`lvErrors`	`longword`	Total number of datagram errors
`+14`	`lvMTU`	`word`	Maximum Transmission Unit size for this host

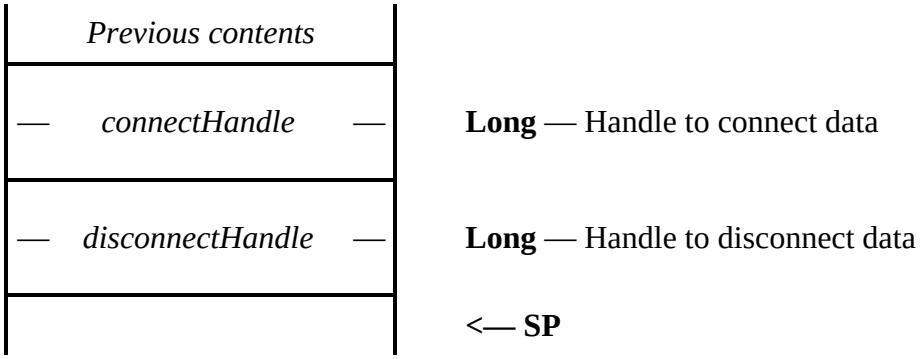
The above record must remain fixed in memory while the module is loaded.

LinkConfigure $0016

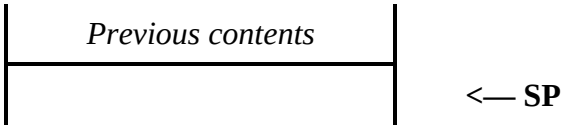
Presents a window allowing the user to edit configuration parameters required by the link layer module. This call is currently only made by the Control Panel, but may be made by other applications which may control Marinetti's setup.

Parameters

Stack before call



Stack after call



This call passes two handles, containing the connect and disconnect data respectively. The handles may be resized and edited as required. If either of the handles are empty, then there is currently no configuration data, and the handle should be resized and initialised before presenting any dialogs to the user.

When called, the desktop will be displayed, and the following tool sets will guarantee to have been started. Other tool sets may have also been started, but the module should check before using them and start them if necessary, and shut them down again on exit.

Tool Set Name	Tool Set No.
Tool Locator	#01 $01
Memory Manager	#02 $02
Miscellaneous Toolset	#03 $03
Quickdraw II	#04 $04
Event Manager	#05 $05
Integer Math Toolset	#11 $0B
Text Toolset	#12 $0C
Window Manager	#14 $0E
Menu Manager	#15 $0F
Control Manager	#16 $10
System Loader	#17 $11
Quickdraw II Auxilliary	#18 $12
LineEdit Toolset	#20 $14

Dialog Manager	#21	$15
Scrap Manager	#22	$16
TCP/IP	#54	$36

◆NOTE: *The module's resource fork is not available during calls to the module. Attempts by a module to open its resource fork may cause the module and Marinetti to crash.*

Outward bound notifications

It may be useful for some applications to be notified asynchronously when certain events occur within Marinetti, such as the network going up and down. Marinetti provides this facility via System 6 IPC requests.

An application that wishes to receive requests sent by Marinetti, should first call the Tool Locator tool set call `_AcceptRequests` with a `nameString` such as `TCP/IP~CompanyName~ProductName~`. `CompanyName` is the name of your company and `ProductName` is the name of your product. Marinetti sends its requests to every application with a `nameString` beginning with `TCP/IP`.

The requests which Marinetti sends out should not be accepted by your routine. They are informational only.

TCPIPSaysHello $8101

Marinetti sends this request once it has completed its startup procedure.

`dataIn` is reserved

`dataOut` is reserved

TCPIPSaysNetworkUp $8102

Marinetti sends this request immediately a network connection is made.

`dataIn` is a pointer to the following data buffer:

+00	`inwLength`	integer	Length of buffer, including this, is $000E
+02	`inwIP`	longword	Your IP address
+06	`inwMethod`	integer	The connect method currently being used
+08	`inwMTU`	integer	The MTU currently being used
+10	`inwLVPtr`	longword	Pointer to link layer variables currently being used

`dataOut` is reserved

TCPIPSaysNetworkDown $8103

Marinetti sends this request immediately after it has disconnected from the network.

`dataIn` is a pointer to the following data buffer:

+00	`inwCount`	integer	Length of buffer, including this, is $000E
+02	`inwIP`	longword	The IP address which was being used
+06	`inwMethod`	integer	The connect method which was being used
+08	`inwMTU`	integer	The MTU which was being used
+10	`inwLVPtr`	longword	Pointer to link layer variables which were being used

`dataOut` is reserved

◆NOTE: *The link layer module which was being used will be purged from memory after this request, so you should save off any parameters you will need from the link layer variables before returning.*

Debugging and testing

Previous versions of Marinetti were called using the toolbox IPC interface, and supported a number of built-in debugging requests. A test application called TESTER was also included with the Marinetti Developers' Kit to make it easier to test requests. Unfortunately, this was in Merlin source code, making it difficult for ORCA/M programmers to make their own changes. There were also no test utilities for high level compiled languages.

From Marinetti version 2.0 onwards, all calls are made using the toolbox interface. This makes debugging and testing a lot easier, by using Dave Lyons' Nifty List. The `NList.Data` file contains the call syntax and error codes for tool calls, and is easily modified to allow you to issue Marinetti tool calls from the Nifty List command line.

As the Nifty List solution is neater, easier to use, and language independent, the original TESTER application is no longer included or supported. Also, due to the abundance of utilities which provide tool breaks, such as Apple's GSBug, a number of break point debugging requests have also been removed.

To add Marinetti support to Nifty List, load the `NList.Data` file into a text editor, and make the following changes.

Find the line starting "2E26 MCSetVolume(…" and add the following after it:

```
0036 === Marinetti ===
0136 TCPIPBootInit()
0236 TCPIPStartUp()
0336 TCPIPShutDown()
0436 TCPIPVersion():Vers
0536 TCPIPReset()
0636 TCPIPStatus():ActFlg
0836 TCPIPLongVersion():rVersion/4
0936 TCPIPGetConnectStatus():connectedFlag
0A36 TCPIPGetErrorTable():@errTablePtr
0B36 TCPIPGetReconnectStatus():reconnectFlag
0C36 TCPIPReconnect(@displayPtr)
0D36 TCPIPConvertIPToHex(@cvtRecPtr,@ddippstring)
0E36 TCPIPConvertIPToASCII(ipaddress/4,@ddpstring,flags):strlen
0F36 TCPIPGetMyIPAddress():ipaddress/4
1036 TCPIPGetConnectMethod():method
1136 TCPIPSetConnectMethod(method)
1236 TCPIPConnect(@displayPtr)
1336 TCPIPDisconnect(forceflag,@displayPtr)
1436 TCPIPGetMTU():mtu
1536 TCPIPValidateIPCString(@cstringPtr):validFlag
1636 TCPIPGetConnectData(userid,method):H
1736 TCPIPSetConnectData(method,H)
1836 TCPIPGetDisconnectData(userid,method):H
1936 TCPIPSetDisconnectData(method,H)
1A36 TCPIPLoadPreferences()
1B36 TCPIPSavePreferences()
1C36 TCPIPGetTuningTable(@tunePtr)
1D36 TCPIPSetDNS(@DNSRecPtr)
1E36 TCPIPGetDNS(@DNSRecPtr)
1F36 TCPIPSetTuningTable(@tunePtr)
2036 TCPIPCancelDNR(@dnrBufferPtr)
2136 TCPIPDNRNameToIP(@nameptr,@dnrBufferPtr)
2236 TCPIPPoll()
```

```
2336 TCPIPLogin(userid,destip/4,destport,defaultTOS,defaultTTL):ipid
2436 TCPIPLogout(ipid)
2536 TCPIPSendICMP(ipid,@messagePtr,messageLen)
2636 TCPIPSendUDP(ipid,@udpPtr,udpLen)
2736 TCPIPGetDatagramCount(ipid,protocol):dgmCount
2836 TCPIPGetNextDatagram(ipid,protocol,flags):H
2936 TCPIPGetLoginCount():loginCount
2A36 TCPIPSendICMPEcho(ipid,seqNum)
2B36 TCPIPReceiveICMPEcho(ipid):seqNum
2C36 TCPIPOpenTCP(ipid):tcpError
2D36 TCPIPWriteTCP(ipid,@dataPtr,dataLength/4,pushFlag,urgentFlag):tcpError
2E36 TCPIPReadTCP(ipid,buffType,buffData/4,buffLen/4,@rrBuffPtr):tcpError
2F36 TCPIPCloseTCP(ipid):tcpError
3036 TCPIPAbortTCP(ipid):tcpError
3136 TCPIPStatusTCP(ipid,@srBuffPtr):tcpError
3236 TCPIPGetSourcePort(ipid):sourcePort
3336 TCPIPGetTOS(ipid):TOS
3436 TCPIPSetTOS(ipid,TOS)
3536 TCPIPGetTTL(ipid):TTL
3636 TCPIPSetTTL(ipid,TTL)
3736 TCPIPSetSourcePort(ipid,sourcePort)
3F36 TCPIPConvertIPCToHex(@cvtRecPtr,@ddipcstring)
4036 TCPIPSendIPDatagram(@datagramPtr)
4136 TCPIPConvertIPToClass(ipaddress/4):class
4236 TCPIPGetConnectMsgFlag():conMsgFlag
4336 TCPIPSetConnectMsgFlag(conMsgFlag)
4436 TCPIPGetUsername(@unBuffPtr)
4536 TCPIPSetUsername(@usernamePtr)
4636 TCPIPGetPassword(@pwBuffPtr)
4736 TCPIPSetPassword(@passwordPtr)
4836 TCPIPValidateIPString(@pstringPtr):validFlag
4936 TCPIPGetUserStatistic(ipid,statisticNum):statistic/4
4A36 TCPIPGetLinkVariables():@variablesPtr
4B36 TCPIPEditLinkConfig(connectHandle/4,disconnectHandle/4)
4C36 TCPIPGetModuleNames():@moduleListPtr
4E36 TCPIPListenTCP(ipid):tcpError
4F36 TCPIPAcceptTCP(ipid,reserved):newipid
5036 TCPIPSetNewDestination(ipid,destip/4,destport)
5136 TCPIPGetHostName(@hnBuffPtr)
5236 TCPIPSetHostName(@hostNamePtr)
5336 TCPIPStatusUDP(ipid,@udpVarsPtr)
5436 TCPIPGetLinkLayer(@linkInfoBlkPtr)
5536 TCPIPPtrToPtr(@from,@to,len/4)
5636 TCPIPPtrToPtrNeg(@fromend,@toend,len/4)
5736 TCPIPGetAuthMessage(userid):authMsgHandle/4
5836 TCPIPConvertIPToCASCII(ipaddress/4,@ddcstring,flags):strlen
5936 TCPIPMangleDomainName(flags,@dnPstringPtr):port
5A36 TCPIPGetAliveFlag():aliveFlag
5B36 TCPIPSetAliveFlag(aliveFlag)
5C36 TCPIPGetAliveMinutes():aliveMinutes
5D36 TCPIPSetAliveMinutes(aliveMinutes)
5E36
TCPIPReadLineTCP(ipid,@delimitStrPtr,buffType,buffData/4,buffLen/4,@rrBuffPtr):tcpErr
or
5F36 TCPIPGetBootConnectFlag():bootConnectFlag
6036 TCPIPSetBootConnectFlag(bootConnectFlag)
6136 TCPIPSetUDPDispatch(ipid,dispatchFlag)
6236 TCPIPGetDestination(ipid,@destRecPtr)
```

Find the line which contains “2613 mcCallNotSupported”, and add the following after it.

```
3601 terrBADIPID
3602 terrNOCONNECTION
3603 terrNORECONDATA
3604 terrLINKERROR
3605 terrSCRIPTFAILED
3606 terrCONNECTED
3607 terrSOCKETOPEN
3608 terrINITNOTFOUND
3609 terrVERSIONMISMATCH
360A terrBADTUNETABLELEN
360B terrIPIDTABLEFULL
360C terrNOICMPQUEUED
360D terrLOGINSPENDING
360E terrTCPIPNOTACTIVE
360F terrNODNSERVERS
3610 terrDNRBUSY
3611 terrNOLINKLAYER
3612 terrBADLINKLAYER
3613 terrENJOYCOKE
3614 terrNORECONSUPPORT
3615 terrUSERABORTED
3616 terrBADUSERPASS
3617 terrBADPARAMETER
3618 terrBADENVIRONMENT
3619 terrNOINCOMING
361A terrLINKBUSY
361B terrNOLINKINTERFACE
361C terrNOLINKRESPONSE
361D terrNODNRPENDING
361E terrBADALIVEMINUTES
361F terrBUFFERTOOSMALL
3620 terrNOTSERVER
```

If you have an older version of `NList.Data` or are still using Apple's internal beta test `NList.AppleData` (which should no longer be used), then you will need to find the appropriate lines yourself.

Once the changes have been made, save them back to disk and reboot. You should now be able to issue Nifty List commands against the Marinetti tool calls and error codes. If issuing calls outside of your application, you will most likely need to use Nifty List to issue the `_LoadOneTool(36,200)` call first.

Porting from BSD UNIX

In order to ease porting from code using BSD socket interfaces, the following is a list of BSD system calls and library functions, and the closest, if any, Marinetti call which performs the same or a similar function.

BSD call/function	Marinetti equivalent	Comments
accept	TCPIPAcceptTCP	These calls are functionally equivalent
bind	TCPIPLogin TCPIPListenTCP	This function is duplicated by issuing the two Marinetti calls in order
close	TCPIPCloseTCP	These calls are functionally equivalent
connect	TCPIPSetNewDestination TCPIPOpenTCP	There is no direct way to duplicate this function
gethostbyaddr		There is no way to duplicate this function
gethostbyname	TCPIPDNRNameToIP	While not as detailed, these calls are functionally equivalent
gethostid	TCPIPGetMyIPAddress	These calls are functionally equivalent
gethostname	TCPIPGetHostName	These calls are functionally equivalent
getpeername		There is no direct way to duplicate this function
listen	TCPIPListenTCP	These calls are functionally equivalent
read	TCPIPReadTCP	These calls are functionally equivalent
recv, recvfrom, recvmsg	TCPIPGetNextDatagram	While functionally equivalent, the Marinetti call is more flexible, as it is a generic call for other transport layers as well as UDP
send, sendmsg, sendto	TCPIPSendUDP	The data passed by these BSD functions will need to be altered to match the format used for the Marinetti call
setsockopt		There is no way to duplicate this function
shutdown		There is no way to duplicate this function
socket	TCPIPLogin	The Marinetti call also provides a number of configuration parameters, which the BSD function requires other calls to duplicate
write	TCPIPWriteTCP	These calls are functionally equivalent

Constants and equates

Tool error codes

terrOK	$0000	
terrBADIPID	$3601	Bad IPID for this request
terrNOCONNECTION	$3602	Not connected to the network
terrNORECONDATA	$3603	No reconnect data
terrLINKERROR	$3604	Problem with the link layer
terrSCRIPTFAILED	$3605	The script failed / timed out
terrCONNECTED	$3606	Not while connected to the network
terrSOCKETOPEN	$3607	Cannot complete - socket still open
terrINITNOTFOUND	$3608	The Marinetti init is not loaded
terrVERSIONMISMATCH	$3609	The init and tool set have different versions
terrBADTUNETABLELEN	$360A	Tune table length in Marinetti 2.0 must be 10
terrIPIDTABLEFULL	$360B	Marinetti cannot handle any more ipids
terrNOICMPQUEUED	$360C	No ICMP datagrams in the queue
terrLOGINSPENDING	$360D	There are still ipids logged in
terrTCPIPNOTACTIVE	$360E	Not active - probably in P8 mode
terrNODNSERVERS	$360F	No servers registered with Marinetti
terrDNRBUSY	$3610	DNR is currently busy - try again later
terrNOLINKLAYER	$3611	Unable to load link layer module
terrBADLINKLAYER	$3612	Not a link layer module
terrENJOYCOKE	$3613	But not so close to the keyboard
terrNORECONSUPPORT	$3614	This module doesn't support reconnect
terrUSERABORTED	$3615	The user aborted the connect/disconnect
terrBADUSERPASS	$3616	Invalid username and/or password
terrBADPARAMETER	$3617	Invalid parameter for this call
terrBADENVIRONMENT	$3618	No desktop or tools not started
terrNOINCOMING	$3619	There is no pending incoming request
terrLINKBUSY	$361A	Modem or interface is busy
terrNOLINKINTERFACE	$361B	No dial tone or similar
terrNOLINKRESPONSE	$361C	No modem answer or similar
terrNODNRPENDING	$361D	No such entry in DNR list
terrBADALIVEMINUTES	$361E	Minutes value is invalid
terrBUFFERTOOSMALL	$361F	Buffer is too small
terrNOTSERVER	$3620	This ipid is not set up as a server

Connect methods

conEthernet	$0001	A generic ethernet card
conMacIP	$0002	IP gateway over AppleTalk/LocalTalk
conPPPCustom	$0003	Scriptable PPP
conSLIP	$0004	Scriptable SLIP
conTest	$0005	Developer test ID – not for public release
conPPP	$0006	Basic PPP
conDirectConnect	$0007	For connection of two IIGS' via a serial cable
conAppleEthernet	$0008	Apple's never released Apple II ethernet card

Protocols

`protocolAll`	$0000
`protocolICMP`	$0001
`protocolTCP`	$0006
`protocolUDP`	$0011

Domain Name Resolver status codes

`DNR_Pending`	$0000	Request is still being processed
`DNR_OK`	$0001	Your request completed successfully, and `dnrBuffer` contains the requested data
`DNR_Failed`	$0002	The request failed. Either the connection timed out, or some other network error
`DNR_NoDNSEntry`	$0003	Requested domain has no DNS entry
`DNR_Cancelled`	$0004	Cancelled by user

TCP logic errors

The following error codes are issued by Marinetti's TCP logic, and are standard TCP error codes from the RFC, they are not tool call error codes. A tool call error code indicates that the tool call failed, which in this instance is not the case. As such, these logic error codes will only be returned by TCP tool calls when the call succeeds, that is when the tool call error code is `terrOK`.

`tcperrOK`	$0000	"tcperr" error codes from TCP RFC
`tcperrDeafDestPort`	$0001	
`tcperrHostReset`	$0002	
`tcperrConExists`	$0003	"connection already exists"
`tcperrConIllegal`	$0004	"connection illegal for this process"
`tcperrNoResources`	$0005	"insufficient resources"
`tcperrNoSocket`	$0006	"foreign socket unspecified"
`tcperrBadPrec`	$0007	"precedence not allowed"
`tcperrBadSec`	$0008	"security/compartment not allowed"
`tcperrBadConnection`	$0009	"connection does not exist"
`tcperrConClosing`	$000A	"connection closing"
`tcperrClosing`	$000B	"closing"
`tcperrConReset`	$000C	"connection reset"
`tcperrUserTimeout`	$000D	"connection aborted due to user timeout"
`tcperrConRefused`	$000E	"connection refused"

TCP states

tcpsCLOSED	$0000
tcpsLISTEN	$0001
tcpsSYNSENT	$0002
tcpsSYNRCVD	$0003
tcpsESTABLISHED	$0004
tcpsFINWAIT1	$0005
tcpsFINWAIT2	$0006
tcpsCLOSEWAIT	$0007
tcpsLASTACK	$0008
tcpsCLOSING	$0009
tcpsTIMEWAIT	$000A