KwikNet®
Web Server

User's Guide

Version 3

First Printing: November 27, 1998
Last Printing: September 15, 2005

Manual Order Number: PN303-9W

KADAK Products Ltd.
206 - 1847 West Broadway Avenue
Vancouver, BC, Canada, V6J 1Y5
Phone: (604) 734-2796
Fax: (604) 734-8114

TECHNICAL SUPPORT

KADAK Products Ltd. is committed to technical support for its software products. Our programs are designed to be easily incorporated in your systems and every effort has been made to eliminate errors.

Engineering Change Notices (ECNs) are provided periodically to repair faults or to improve performance. You will automatically receive these updates during the product's initial support period. For technical support beyond the initial period, you must purchase a Technical Support Subscription. Contact KADAK for details. Please keep us informed of the primary user in your company to whom update notices and other pertinent information should be directed.

Should you require direct technical assistance in your use of this KADAK software product, engineering support is available by telephone, fax or e-mail. KADAK reserves the right to charge for technical support services which it deems to be beyond the normal scope of technical support.

We would be pleased to receive your comments and suggestions concerning this product and its documentation. Your feedback helps in the continuing product evolution.

KADAK Products Ltd.
206 - 1847 West Broadway Avenue
Vancouver, BC, Canada, V6J 1Y5

Phone:	(604) 734-2796
Fax:	(604) 734-8114
e-mail:	amxtech@kadak.com

DISCLAIMER

TRADEMARKS

KwikNet Web Server User's Guide

Table of Contents

KwikNet Web Server User's Guide

Table of Figures

1. KwikNet Web Server Overview

1.1 Introduction

The HyperText Transfer Protocol (HTTP) is a standard protocol used for transferring documents between machines over TCP/IP based networks, the most common being the Internet. HTTP is often referred to as the World Wide Web (WWW) protocol since it is used to manipulate interconnected documents around the globe. Each document is ultimately retrieved from a web server operating on some computer with access to the document. The term document is used loosely to reference any type of data, be it text, graphics, sound or video.

The KwikNet™ Web Server implements the HTTP protocol on top of the KwikNet TCP/IP Stack, a compact, reliable, high performance TCP/IP stack, well suited for use in embedded networking applications.

The KwikNet Web Server is best used with a real-time operating system (RTOS) such as KADAK's AMX™ Real-Time Multitasking Kernel. However, the KwikNet Web Server can also be used in a single threaded environment without an RTOS. The KwikNet Porting Kit User's Guide describes the use of KwikNet with your choice of RT/OS. Note that throughout this manual, the term RT/OS is used to refer to any operating system, be it a multitasking RTOS or a single threaded OS.

You can readily tailor the KwikNet stack to accommodate your web server HTTP needs by using the KwikNet Configuration Builder, a Windows® utility which makes configuring KwikNet a snap. Your KwikNet stack will only include the HTTP features required by your application.

This manual makes no attempt to describe the HyperText Transfer Protocol (HTTP), what it is or how it operates. It is assumed that you have a working knowledge of the HTTP protocol as it applies to your needs. Reference materials are provided in Appendix A of the KwikNet TCP/IP Stack User's Guide.

Note

The KwikNet HTTP option is founded upon the HTTP web server from Treck Inc. Hence you must become familiar with the HTTP application programming interface (API) described in the Treck Web Server User Manual.

The purpose of this manual is to provide the system designer and applications programmer with the information required to properly configure and implement a networking system using the KwikNet TCP/IP Stack and HTTP. It is assumed that you are familiar with the architecture of the target processor.

KwikNet and its options are available in C source format to ensure that regardless of your development environment, your ability to use and support KwikNet is uninhibited. The source program may also include code fragments programmed in the assembly language of the target processor to improve execution speed.

The C programming language, commonly used in real-time systems, is used throughout this manual to illustrate the features of KwikNet and its Web Server.

1.2 General Operation

The HyperText Transfer Protocol (HTTP) is a standard protocol used for transferring documents between machines over TCP/IP based networks such as the Internet. HTTP version 1.1 is formally defined by the IETF document RFC-2616. The KwikNet Web Server is compliant with that specification. The RFC should be consulted for any detailed questions concerning the HTTP protocol. The KwikNet Web Server implements the subset of HTTP features typically required for use in embedded applications.

HTTP is a client-server protocol. One machine, the client, initiates a document transfer by contacting another machine, the server. The most common HTTP client is the familiar web browser. The term web browser will be used throughout this document instead of the more cumbersome term HTTP client. The web browser issues an HTTP request to the web server in order to access a document. The server must be operating before the browser initiates its requests. Generally, a browser communicates with one server at a time while most servers are designed to work concurrently with multiple browsers.

The KwikNet Library provides all of the services necessary to implement one or more KwikNet Web Servers. Although multiple servers can coexist and operate concurrently, most applications will implement only a single KwikNet Web Server.

HTTP Requests

When a browser contacts a web server, a TCP connection is established between the two machines as follows. The web server listens on a TCP socket (usually port 80) for connection requests from potential browsers. The browser attempts to connect its own TCP socket to the server's socket. If the browser requests that HTTP version 1.0 be used, the connection, once established, is used for only one browser request. The connection is broken by the server once the requested document has been sent to the browser.

To improve network performance, HTTP version 1.1 permits a persistent connection to be established for as long as the browser requires access to the server, provided both the web server and browser support this feature. The KwikNet Web Server supports persistent connections when HTTP version 1.1 is used.

An HTTP request is an ASCII string which always includes a request line followed by zero or more lines of text which provide additional information about the request. All of these text lines are terminated by a two character, end of line sequence consisting of a carriage return (ASCII 13, *0x0D*, *'\r'*) followed by a linefeed (ASCII 10, *0x0A*, *'\n'*).

The request line includes a Universal Resource Identifier (URI) which identifies the document of interest. The request line also identifies the operation (called a method) to be performed using that document. Figure 1.2-1 lists the HTTP method strings which the KwikNet Web Server supports. The request line also indicates the HTTP version being used by the browser making the request.

HTTP Replies

The web server reply is an ASCII string which always includes a status line followed by zero or more lines of text which provide additional information about the reply. The requested document, if any, then follows as the body of the reply. Note that some requests can be handled by the web server without any document data being returned to the browser.

The status line indicates the HTTP version being used by the server in its response. A 3 digit decimal status code followed by some explanatory text completes the status line. Generally, status codes in the 200 range indicate success and codes in the 500 range indicate failures. See the RFC for a complete guide to status codes.

Request		Purpose
OPTIONS	*	Determine features supported by the web server
GET		Retrieve a document or other identified entity
HEAD		Respond as though to a *GET* request but do not transfer the actual document or other identified entity
TRACE	*	Echo the HTTP request in the body of the response
POST		Provide server with additional document information
PUT	*	Provide server with a new document or other entity
DELETE	*	Delete a document or other identified entity

Note * Request not supported by KwikNet Web Server

Figure 1.2-1 KwikNet HTTP Requests

HTTP Documents

The documents presented by a web server to a browser are usually just files which reside on the same computer as the web server. These files, once delivered to the browser, are then rendered by the browser as screen images for viewing by the browser's user.

The most common documents are text files written using the HyperText Markup Language (HTML). Each HTML page specifies what is to appear on the screen image which the browser will produce. Often this is just text contained in the HTML document.

An HTML page can specify that another document actually defines what is to be presented at a particular point within the screen image described by the HTML page. For example, a graphic image can be inserted on the page simply by referencing the name and location of the file containing the image data. As the browser processes the HTML page, it retrieves the graphic image document from the web server which has access to that document.

The real power of the web browser comes from its interpretation of hyperlinks within the HTML page. A hyperlink is a method of connecting one document to another. When the browser detects that its user has selected a hyperlinked region within the screen image being generated by the browser, it fetches the document referenced by the hyperlink and shows its content to the user.

Executable (CGI) Documents

Not all HTTP documents are files. Recall that a document is simply the "thing" referenced by a Universal Resource Identifier (URI) in an HTTP request. The interpretation of the URI is up to the web server. It is this characteristic of HTTP that is of special significance in embedded applications.

The web server can interpret a URI to be anything, including some application or function which can execute on the same processor on which the web server operates. Such a function is called a Common Gateway Interface (CGI) function, sometimes referred to as a CGI script.

If a URI references a CGI function, what does the browser get in response to its request? The answer depends on the CGI function executed by the web server. In some cases, the browser gets nothing other than a reply indicating that the requested operation was performed. In other cases, the browser will get document data that is created dynamically by the function when it executes. It is also possible for the browser to get a real HTTP document, the particular document being determined by the CGI function according to its analysis of parameters provided in the browser's request.

Common Gateway Interface (CGI)

Applications which need to adapt their web server's response to a browser request can do so by way of the Common Gateway Interface (CGI). This interface is also commonly used to present an embedded application with configuration information or operating parameters from an HTML form.

Recall that the web server can interpret a Universal Resource Identifier (URI) to be anything. In particular, a URI can identify a function which the web server must execute. The KwikNet Web Server treats any URI which resembles a file name with a specific file path as a CGI reference to a function to be executed whenever a browser tries to access that URI. The special file path is called the CGI directory. The default CGI directory is *"/cgi-bin"*. You can specify your own CGI directory when you start your web server.

When the KwikNet Web Server receives an HTTP request which references a file in the CGI directory, it calls an application HTTP event handler to service the request. The event handler is a function which you must provide and register with a call to service procedure *tfHttpdUserSetEventHandler()*. It is this function which must interpret the CGI request in an application dependent manner and generate the appropriate response.

Your HTTP event handler must be written according to the specifications provided in the Treck Web Server User Manual. Function *tfHttpdUserGetConInfo()* can be called to get the full path and name of the CGI file and the parameters provided by the browser in its request. The event handler can use the file name to further qualify or identify the specific action which it must perform.

The HTTP event handler can call function *tfHttpdUserSendBuffer()* to present data for delivery to the browser. Alternatively, it can call function *tfHttpdUserSendFile()* to send a file to the browser.

The web server generates its HTTP reply to the browser and the file or data, if any, provided by the HTTP event handler is returned in the body of the reply.

Server Side Includes (SSI)

Some applications may have a number of HTML documents, each of which repeats common information. For example, every document may have a common banner, title or copyright statement. In an embedded system in which all documents are kept in memory as virtual files, the memory used to replicate such information in each file may be significant.

The KwikNet Web Server implements a server dependent HTML enhancement which addresses this issue. The feature, called the Server Side Include (SSI), permits one HTML document to include another. The feature is analogous to the use of the `#include` directive within C files to include other C header files.

A special form of HTML tag is used to identify an SSI document. You must register your SSI tag by calling function `tfHttpdUserRegisterSsiTag()` to specify the tag string and the application function to be called to service each occurrence of the tag in documents presented by the web server to the browser.

It is recommended that a variant of the HTML comment tag be used to identify the document to be included. For example, assume that the following HTML comment tag is present in a document being delivered by the web server.

```
<!--#include file="filename" -->
```

If you have registered the string `"!--#include file="` as your SSI tag, then any HTML tag beginning with the string `"<!--#include file="` will be presented in its entirety to your SSI handler function for service. Your SSI handler will receive a pointer to the full tag string exactly as it appears above. The handler can decode the text in the SSI tag string to further qualify or identify the specific action which it must perform. In this example, your handler would have to parse the tag string to isolate the name of the file to be sent to the browser in place of the SSI tag.

Your SSI handler must be written according to the specifications provided in the Treck Web Server User Manual. Function `tfHttpdUserGetConInfo()` can be called to get access to the HTTP connection information. The SSI handler must ignore CGI related parameters which this function may return.

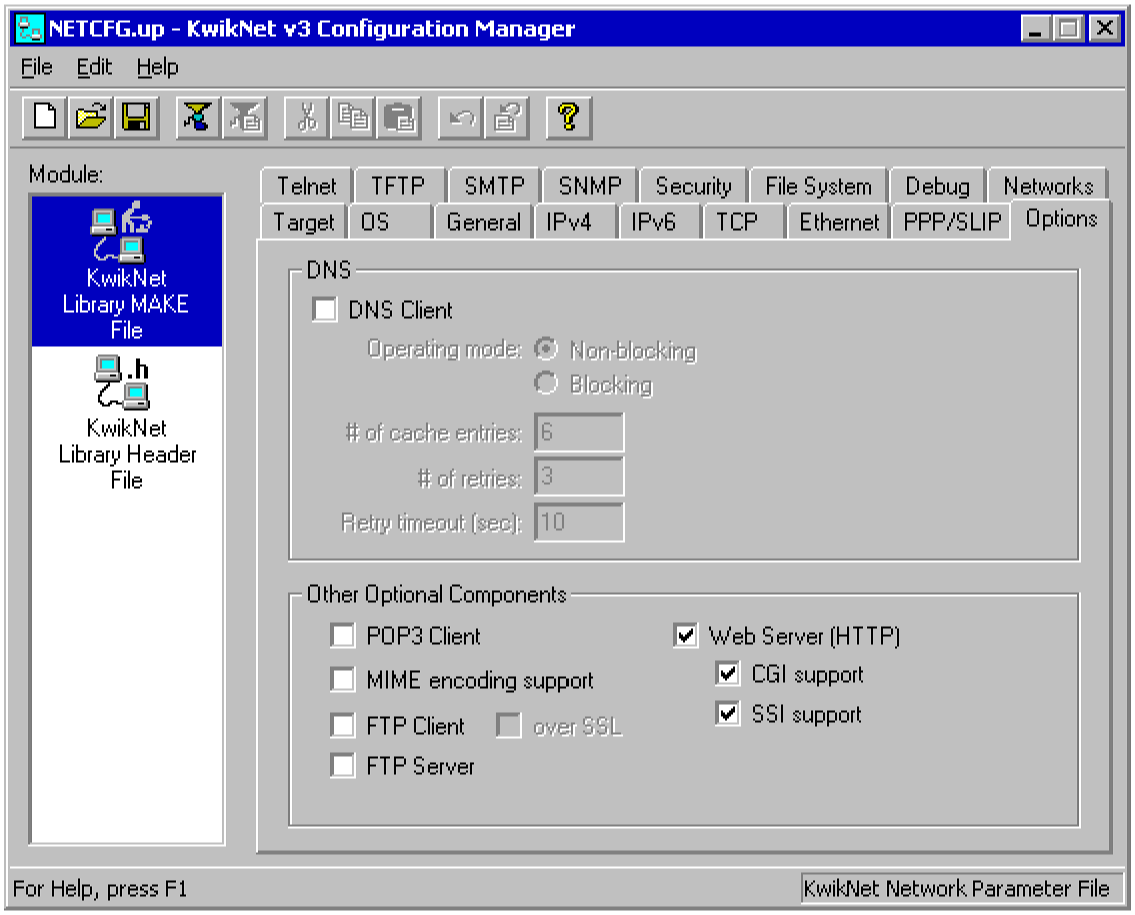
The SSI handler can call function `tfHttpdUserSendBuffer()` to present data for delivery to the browser. Alternatively, it can call function `tfHttpdUserSendFile()` to send a file to the browser.

When the KwikNet Web Server returns a file to a browser, it must scan the file for SSI tags in order to replace an SSI tag with the document generated by your SSI handler for that tag. To optimize performance, the web server only scans files which match the file selection criteria which you specify with calls to `tfHttpdUserSetSsiFileFilter()`. For example, you can specify that all files with extension `STM` or in a directory named `SSITAG` are to be subject to SSI tag replacement.

1.3 KwikNet Web Server Configuration

You can readily tailor the KwikNet stack to support an HTTP web server by using the KwikNet Configuration Builder to edit your KwikNet Network Parameter File. The KwikNet Library parameters are edited on the Options property page. The layout of the window is shown below.

Note that the TCP protocol and a file system are prerequisites for the HTTP web server. You must include these components in your KwikNet Network Parameter File in order to use the web server.



DNS

☐ DNS Client

Operating mode: [✓ Non-blocking] Blocking

# of cache entries: 6

# of retries: 3

Retry timeout (sec): 10

Other Optional Components

☐ POP3 Client

☐ MIME encoding support

☐ FTP Client ☐ over SSL

☐ FTP Server

☑ Web Server (HTTP)

☑ CGI support

☑ SSI support

Web Server Parameters (continued)

Web Server (HTTP)

Check this box if your application will include a web server to provide network access to your target system. Otherwise, leave this box unchecked.

CGI Support

Check this box if your HTTP server application will include support for dynamically generated web pages. Otherwise, leave this box unchecked.

SSI Support

Check this box if your HTTP server application will include support for web pages with dynamically generated embedded content. Otherwise, leave this box unchecked. CGI support must be enabled before SSI support can be enabled.

1.4 File Services

The HyperText Transfer Protocol (HTTP) is a standard protocol used for transferring documents between a web browser and a web server. In most cases, documents are simply files sent from the web server to the browser. Consequently, the web server requires access to a file system for the actual retrieval and storage of files.

KwikNet includes a file system interface which provides the KwikNet Web Server with access to one of the supported file systems: the AMX/FS File System, the KwikNet Virtual File System, standard C file I/O, a user defined file system, the Treck RAM file system or the Treck ROM file system. This file system interface is described in Appendix C of the KwikNet TCP/IP Stack User's Guide.

KADAK's AMX/FS File System is ready for use with the AMX Real-Time Multitasking Kernel. Since AMX/FS includes a RAM disk driver, it is well suited for use with the KwikNet Web Server in embedded applications.

The standard C file I/O library from the C compiler vendor can be used if it is available for the target processor. Unfortunately, few C libraries provide file services for embedded targets.

Special considerations apply when using AMX 86 which can coexist with MS-DOS®. AMX 86 includes a component called the PC Supervisor which permits tasks running under AMX 86 to concurrently access the MS-DOS file system using standard C file I/O calls. Hence, when using AMX 86, the KwikNet Web Server can connect to the standard C file I/O library.

Virtual File System (VFS)

For embedded systems which may not need a full-featured file system, KwikNet offers a very simple Virtual File System (VFS) which can provide access to a limited set of read-only files built into the application. The Virtual File System can be used with or without a real file system. It will forward file requests which it is not equipped to handle, to the underlying file system, if one exists.

Using the KwikNet VFS Generator, you can create compressed HTML files for use with the KwikNet Web Server. These files will be automatically decompressed by the KwikNet Virtual File System prior to use by the web server.

The KwikNet Virtual File System and its VFS Generator are described in Chapter 7 of the KwikNet TCP/IP Stack User's Guide.

Note

The file system must be ready for use before the KwikNet Web Server is started. See Appendix C of the KwikNet TCP/IP Stack User's Guide for details.

User Administration

If you are using the KwikNet Virtual File System, the AMX/FS File System or a custom file system that is accessed via the KwikNet Universal File System interface, then the web server must log in with a valid user name and password to gain access to the underlying file system services.

Most operating systems used in embedded applications are not like UNIX; they do not require or support user names and passwords. The KwikNet Administration interface resolves this dilemma by allowing you to define the users which will be permitted access to your system. Follow the instructions presented in Appendix D of the KwikNet TCP/IP Stack User's Guide.

Once the file system is initialized and ready for use, your web server must call function *`tfFSUserLogin()`* to log in for file system access. Unless altered by you, the user name *`treck`* and password *`treck`* defined in the KwikNet Administration interface will allow the web server full access to the underlying file system services. The log in provides a user handle, an identifier which the web server must present when accessing any file system service. Once the web server has started, it must call *`tfHttpdUserSetFSparam()`* to register the user handle.

Directory Access by Web Server

The web server "user" can be given a unique base directory which will be the default current working directory established when the web server logs in. The definition of the user base directory must be a full path, including drive if required by the file system.

The extent to which the web server user is permitted to traverse directories is determined by the visibility access right granted in the user's definition. Unless the user has been granted full visibility, the user will only be allowed to traverse directories forward from the user's default directory.

Note

User names, passwords and access rights are handled by the KwikNet Administration interface which is described in Appendix D of the KwikNet TCP/IP Stack User's Guide.

1.5 Web Server Task

The KwikNet Web Server option includes a set of services which can be used to implement a web server. The web server is an application program which makes use of these services to communicate with one or more web browsers.

Before your web server can use any KwikNet services, the underlying file system must be initialized and ready for use. All disk drives which the web server will be permitted to access must be mounted and ready for use. If you are using the KwikNet Virtual File System, KwikNet will automatically initialize the VFS when KwikNet is first started.

Your web server can provide a specific IPv4 address and port which browsers must use to connect to the web server. Alternatively, the web server can accept all HTTP requests received on any of the operational (open) network interfaces which KwikNet services, provided that the requests are directed to the well known HTTP port number 80.

The WEB Sample Program provided with the KwikNet Web Server includes a fully functional web server. It is described in Chapter 2.1.

Multitasking Operation

When used with a real-time operating system (RTOS) such as KADAK's AMX Real-Time Multitasking Kernel, the web server operates as an application task. Such a task is referred to as a web server task. Although more than one web server task is allowed, rarely is there such a need.

Before any web server task can operate, KwikNet must have been successfully started and your application must have called function *`tfHttpdUserInit()`* once, and only once, to initialize KwikNet's HTTP services.

A web server task is created and started just like any other application task. When ready to begin operation, the server task simply calls procedure *`tfHttpdUserStart()`* to establish a web server session. If you are using a file system that requires a user log in, the web server task must call function *`tfFSUserLogin()`* to log in for file system access. In this case, the web server must then call *`tfHttpdUserSetFSparam()`* to register the user handle.

Finally, the web server task calls *`tfHttpdUserExecute()`* to begin service. Although a web server session can be created to operate in non-blocking mode, it is more usual to operate in blocking mode. In that case, there is no return from procedure *`tfHttpdUserExecute()`* until the server is forced to stop.

Note

In multitasking systems, the KwikNet Web Server task MUST execute at a priority below that of the KwikNet Task.

The web server task will operate until some unrecoverable error condition is detected or until some other application task calls procedure *tfHttpdUserStop()* requesting the server to stop. The web server task will abort all of its active HTTP sessions and resume execution following the initial call to *tfHttpdUserExecute()*.

Single Threaded Operation

When used with a single threaded operating system, the web server operates in the KwikNet domain in the context of the KwikNet Task as described in Chapter 1.2 of the KwikNet TCP/IP Stack User's Guide.

Before any web server session can operate, KwikNet must have been successfully started and your application must have called function *tfHttpdUserInit()* once, and only once, to initialize KwikNet's HTTP services.

When your App-Task calls KwikNet procedure *tfHttpdUserStart()*, a web server session is established. The web server must be started to operate in non-blocking mode.

If you are using a file system that requires a user log in, the web server must call function *tfFSUserLogin()* to log in for file system access. In this case, the web server must then call *tfHttpdUserSetFSparam()* to register the user handle.

Once the web server has been started, you must periodically call procedure *tfHttpdUserExecute()* to allow the server to service its clients. The easiest way to do this is to use KwikNet procedure *kn_addserver()* to add a server function to the KwikNet server queue. Thereafter, the KwikNet Task will call your server function at the periodic interval specified by you in your call to *kn_addserver()*.

When a web server function is added to the KwikNet server queue, the function is referred to as a web server task. Although more than one web server task is allowed, rarely is there such a need.

The WEB Sample Program implements server function *webs_service()* which calls *tfHttpdUserExecute()* at the specified service interval until the server is stopped.

Once the web server is operational, your App-Task must regularly call KwikNet procedure *kn_yield()* to let KwikNet and all server tasks, including your web server task, operate.

The web server task will operate until some unrecoverable error condition is detected or until your App-Task calls procedure *tfHttpdUserStop()* requesting the server to stop. The web server task will abort all of its active HTTP connections and remove itself from the KwikNet server queue as illustrated by service function *webs_service()* in the WEB Sample Program.

This page left blank intentionally.

2. KwikNet Web Server Applications

2.1 Web Server Sample Program

A WEB Sample Program is provided with the KwikNet Web Server option to illustrate the use of a primitive web browser and the KwikNet Web Server. The sample program is ready for use with the AMX Real-Time Multitasking Kernel and the Treck RAM File System. The sample program can also be used with any of the porting examples provided with the KwikNet Porting Kit.

With simple modifications to the configuration and link process, the KwikNet WEB Sample Program can also be adapted to use the Treck ROM File System, the KwikNet Virtual File System, the AMX/FS File System or your own custom file system. The AMX 86 sample can be adapted to use the PC Supervisor to access MS-DOS® file services.

The sample configuration supports a single network interface. The network uses the KwikNet Ethernet Network Driver. Because the sample must operate on all supported target processors without any specific Ethernet device dependence, KwikNet's Ethernet Loopback Driver is used. Use of this driver allows the sample's web browser and the KwikNet Web Server to be tested even if network hardware is not available. Once the WEB Sample Program has been tested in loopback fashion, you can replace the Ethernet Loopback Driver with your own network device driver. Then real web browsers will be able to access the KwikNet Web Server.

The KwikNet TCP/IP Stack requires a clock for proper network timing. The examples provided with the KwikNet Porting Kit all illustrate the clock interface. If you are using KwikNet with AMX, you must provide an AMX clock driver. If you have ported the AMX Sample Program to your hardware platform, you can use its AMX Clock Driver.

The sample includes a web server task and a primitive web browser task. The browser uses the KwikNet console driver to provide a command line interface with a user. The console driver can be configured as described in Chapter 1.8 of the KwikNet TCP/IP Stack User's Guide to use any of several possible terminal devices as an interactive terminal. If you are using KwikNet with AMX and have ported the AMX Sample Program to your hardware platform, you can use its serial UART driver for console I/O.

The sample also uses the KwikNet data logging and message recording services to record messages generated by the KwikNet TCP/IP Stack as it operates. These services are described in Chapters 1.6 and 1.7 of the KwikNet TCP/IP Stack User's Guide. The messages are recorded into an array of strings in memory. The web browser's interactive `dump` command can be used to list these messages on the console device and empty the recording array.

Startup

The manner in which the KwikNet WEB Sample Program starts and operates is completely dependent upon the underlying operating system with which KwikNet is being used. All sample programs provided with KwikNet and its optional components share a common implementation methodology which is described in Appendix E of the KwikNet TCP/IP Stack User's Guide. Both multitasking and single threaded operation are described.

When used with AMX, the sample program operates as follows. AMX is launched from the *`main()`* program. Restart Procedure *`rrproc()`* starts the print task, creates the web server task and then creates and starts the web browser task. The web server task remains idle until started by the browser task as will be described.

Once the AMX initialization is complete, the high priority print task executes and waits for the arrival of AMX messages in its private mailbox. Each AMX message includes a pointer to a log buffer containing a KwikNet message to be recorded.

Once the print task is ready and waiting, the web browser task finally begins to execute. It starts KwikNet at its entry point *`kn_enter()`*. KwikNet self starts and forces the KwikNet Task to execute. Because the KwikNet Task operates at a priority above all tasks which use its services, it temporarily preempts the browser task. The KwikNet Task initializes the network and its associated loopback driver and prepares the IP and TCP protocol stacks for use by the sample program.

Once the KwikNet initialization is complete, the web browser task resumes execution.

Web Browser Operation

Once the KwikNet initialization is complete, the web browser task resumes execution. It initializes the KwikNet console driver and generates a signon message on the console device.

The web browser generates a command line prompt *"KwikNet HTTP>"* and waits for a user to enter a lower case directive and any parameters required by that directive. The directive is terminated by the Enter key (*'\r'* character).

The web browser decodes the directive and performs the requested action. Since every web browser HTTP request is exercised by the web browser task, its code can serve as an excellent programming model for your own web browser software.

Each HTTP reply is echoed by the web browser to its terminal device. The HTTP response line, header information and entity body, if any, is echoed. Binary data, if present, will be echoed without graphical rendering and will usually appear as gibberish on the terminal device.

The following complete list of directives will be presented if either *help* or *?* is entered as the command line directive.

```
Commands:
   help - Display this text.
   exit - Terminate this sample program.
   lit  - Send a literal request.
   get <remote file path>   - Get a file from the server.
   head <remote file path>  - Sense a file on the server.
          The file content will NOT be retrieved.
   form <remote file path>  - Post a form to the server.
   query <remote file path> - Post a query to the server.

   serv - Start the KwikNet Web Server on this machine.
   stop - Stop  the KwikNet Web Server.
   open  <server IPv4 address> <port>
   open6 <server IPv6 address>[%<scope id>] <port>
          Identify a web server.
          If port is 0, web server port 80 will be used.
   user <username> - Specify user name.
          For no authorization, omit user name.
   ver  - Toggle HTTP version.
   dump [stat] - Dump KwikNet recorded log [and statistics].

glossary:
   <text> - String you must provide.
   [optional] - Parameter(s) within [] can be omitted.
            (omit the <,>,[ and ] characters).
```

Web Server Operation

The web browser cannot connect to a web server unless such a server exists on the network. Unless you have replaced the Ethernet Loopback Driver with a real device driver, the WEB Sample Program has no direct network connection. Hence no web server is accessible.

If you give the web browser the *serv* directive, it will start the KwikNet Web Server task which will immediately begin operation since it is of higher priority than the browser task. After starting the server task, the browser task pauses briefly and then fetches the web server task's IP address and port number. Armed with this information, the web browser is then able to converse with the KwikNet Web Server *across the network* even though both are executing on the same processor.

The web browser task displays the server's IP address and port number giving you, the user, the chance to see that a server now exists. If you have modified the sample program to include a real network interface, you will be able to use any web browser on your network to access the KwikNet Web Server using this IP address.

By default the web browser uses HTTP version 1.1 in its requests. You can use the *ver* directive to toggle between version 1.1 and 1.0.

At any time, you can enter the *stop* directive which causes the web browser to call KwikNet procedure *tfHttpdUserStop()* requesting the web server task to stop execution.

Test Features

The *open* directive can be used to alter the IP address and port number which the web browser uses to identify the web server to which its HTTP requests are to be directed. This directive is intended for KADAK use only.

The *user* directive can be used to specify a user name for the web browser. When prompted, you will also have to provide a password for the web browser. The web browser will send this user name and password in the header of every HTTP request that it sends to a web server. This directive is intended for KADAK use only.

Shutdown

When the web browser task decodes the *exit* directive, it prepares to shut down. If the web server task is still operating, the web browser task requests it to stop.

Next, the web browser task generates a signoff message and relinquishes use of the KwikNet console driver. It then calls procedure *kn_exit()* to stop operation of the KwikNet TCP/IP Stack.

Finally, after pausing briefly, it initiates a shutdown of the underlying operating system (if possible) and a return to the *main()* procedure.

KwikNet and Web Server Logging

The WEB Sample Program uses the simple KwikNet message recording service to log text messages. The recorder saves the recorded text strings in a 30,000 byte memory buffer until either 500 strings have been recorded or the memory buffer capacity is reached.

The WEB Sample Program directs messages to this recorder by calling the KwikNet log procedure *kn_dprintf()*. This procedure operates similarly to the C *printf()* function except that an extra integer parameter of value *0* must precede the format string. The web browser task uses this feature to record a shutdown message. The web server task also uses this feature to record errors as they are detected.

KwikNet formats the message into a log buffer and passes the buffer to an application log function for printing. Log function *sam_record()* in the KwikNet Application OS Interface serves this purpose.

In a multitasking system the log buffer is delivered as part of an RTOS dependent message to a print task. The print task calls *kn_logmsg()* in the KwikNet message recording module to record the message and release the log buffer.

In a single threaded system, the log function *sam_record()* can usually call *kn_logmsg()* to record the message and release the log buffer. However, if the message is being generated while executing in the interrupt domain, the log buffer must be passed to the KwikNet Task to be logged. The sample programs provided with the KwikNet Porting Kit illustrate this process.

Since the recorded strings are just stored in memory, they are not readily visible. To overcome this restriction, you can use the web browser's interactive *dump* command to list all of the recorded messages on the console device and empty the recording array.

Alternatively, if a debugger is used to control execution of the WEB Sample Program, the program can be stopped and the strings can be viewed in text form in a display window by viewing the array variable *kn_recordlist[]* in module *KNRECORD.C*.

Web Browser Logging

The WEB Sample Program's browser task logs messages directly to the console terminal which it uses for its command line user interface.

Files for Browsing

The WEB Sample Program is ready for use with the Treck RAM File System. The web browser task in the WEB Sample Program dynamically creates the following collection of files, placing them in the root directory of the RAM drive.

INDEX.HTM	Main HTML document with hyperlinks to *SSITEST.STM*, *COUNTER.STM* and *PARAMS.CGI*
SSITEST.STM	HTML document which tests the SSI feature
SSITXT.SSI	Text inserted into the *SSITEST.STM* page using SSI
COUNTER.STM	HTML document which tests the SSI feature

If the KwikNet Library is configured to use the AMX/FS File System, the sample's web browser task dynamically creates the same collection of files in the root directory of the AMX/FS RAM drive.

If the KwikNet Virtual File System is used, an identical set of **VFS files** will be used. The virtual files are located in the VFS Data File *KNWEBVFS.C* provided with the WEB Sample Program. This C file is compiled to create the VFS Data File object module which is linked with the WEB Sample Program. The virtual files are in directory *VFS*.

The web server task registers function *wsam_eventcb()* as the HTTP **event handler** which services each browser connection as it is opened and closed. The function also decodes and services CGI related events.

The event handler function *wsam_eventcb()* decodes CGI requests using its CGI lookup array *path_to_cgi_array[]*. For example, the event handler translates any reference to file */cgi-bin/params.cgi* as a request to execute the CGI function *wsam_params()*.

The WEB Sample Program configures the web server to handle any request for a file in directory *"/cgi-bin"* as a **CGI reference**.

File *INDEX.HTM* uses a reference to file *"/cgi-bin/params.cgi"* to invoke the execution of CGI function *wsam_params()* to generate a text page which displays the query parameters specified by the CGI hyperlink.

The WEB Sample Program configures the web server to allow **SSI tags** in any file with extension *STM*. The SSI tag is defined to be *"!--#ssi"*. The SSI tag is serviced by function *wsam_ssiproc()* which uses the array *tag_to_ssi[]* to identify the SSI handler function to be called for a particular use of the tag.

File *SSITEST.STM* uses an SSI tag to include file *SSITXT.SSI* in place of the tag.
File *COUNTER.STM* uses an SSI tag to replace the tag with the current value of a counter.

Viewing the Files

The web browser provided with the WEB Sample Program cannot render an HTML page as would a conventional browser. It can only display the HTML page in source form as raw text. Hence, you can use the sample's *`get`* command to actually view the source code of any of the files listed above.

To view the sample HTML pages and actually exercise the features of the KwikNet Web Server, you must use a real browser and for that, you will have to replace the KwikNet Ethernet Loopback Driver with a real network connection.

To view the VFS files, browse address *`http://192.168.0.3/VFS/index.htm`*.
To view the RAM drive files, browse address *`http://192.168.0.3/index.htm`*.

Replace the IP address *`192.168.0.3`* with the actual IPv4 address assigned to the KwikNet Web Server and reported by the sample's web browser.

Note

An alternate set of VFS files is provided with the KwikNet Virtual File System. For a description of these files and their use, browse the HTML file *`VFS_USE.HTM`* in the KwikNet installation directory *`VFSGEN\VFS_INFO`*.

Running the Sample Program

The KwikNet WEB Sample Program requires that your target hardware include an interface to a console terminal. The web browser task will use its command line interface to interact with you at that terminal. You are therefore the real user behind the web browser task.

Each action which you initiate using a command line directive will generate a response on the terminal as the browser task handles your request. The web server task will only run if you use the *`serv`* directive to start it. The WEB Sample Program runs until you issue the *`exit`* directive to shut it down.

KwikNet includes a number of debug features (see Chapter 1.9 of the KwikNet TCP/IP Stack User's Guide) which can assist you in using the WEB Sample Program. With KwikNet's debug features enabled, you can place a breakpoint on procedure *`kn_bphit()`* to trap all errors detected by KwikNet. Of course, if you are using AMX, it is always wise to execute with a breakpoint on the AMX fatal exit procedure *`cjksfatal`* (*`ajfatl`* for AMX 86).

The web server task uses the KwikNet message recording service to log messages concerning its operation. KwikNet also records selected debug and trace information if any of these features are enabled. Unless you have modified the KwikNet recording method, these messages are simply saved in memory and are therefore not visible. However, you can use the web browser's interactive *`dump`* command to list all of the recorded messages on its console device and empty the recording array.

Note

To use a real browser to view the files provided with the WEB Sample Program, you must replace the KwikNet Ethernet Loopback Driver with your own network device driver. Use the KwikNet Configuration Manager to edit the Network Parameter File *`KNWEBLIB.UP`* to use your driver.

2.2 Making the WEB Sample Program

The sheer volume of detail necessary to understand and use HTTP with TCP/IP may at first be daunting. However, constructing the KwikNet WEB Sample Program is actually a fairly simple process made even simpler by the KwikNet Configuration Manager, a Windows® utility provided with KwikNet.

The WEB Sample Program includes all of the components needed to build the sample application for a particular target processor and file system. You can take these components and, with minor modifications, adapt them for your particular target processor and development environment.

As delivered, the KwikNet WEB Sample Program uses the Treck RAM File System. Alternatively, the sample can be configured to use the Treck ROM File System or the KwikNet Virtual File System (VFS).

The sample can also be used with AMX or AMX 86 and the AMX/FS File System, with or without the VFS. For PC targets with AMX 86, it can be used with MS-DOS® file services, with or without the VFS.

Use the KwikNet Configuration Manager to edit the sample Network Parameter File to select the alternate file system and, if necessary, link the file system modules with the sample.

Note

The KwikNet WEB Sample Program for a particular target processor family is provided ready for use on one of the development boards used at KADAK for testing.

WEB Sample Program Directories

When KwikNet and its Web Server Option are installed, the following subdirectories on which the sample program construction process depends are created within directory *KNTnnn*.

TCPIP	KwikNet header and source files, Ethernet Network Driver Ethernet and Serial Loopback Drivers
CFGBLDW	KwikNet Configuration Builder; template files
ERR	Construction error summary
MAKE	KwikNet Library make directory
TOOLXXX	Toolset specific files
TOOLXXX\DRIVERS	KwikNet device drivers and board driver
TOOLXXX\LIB	Toolset specific KwikNet Library will be built here
TOOLXXX\SAM_MAKE	Sample program make directory
TOOLXXX\SAM_WEB	KwikNet WEB Sample Program directory
TOOLXXX\SAM_COMN	Common sample program source files

One or more toolset specific directories *TOOLXXX* will be present. There will be one such directory for each of the software development toolsets which KADAK supports. Each toolset vendor is identified by a unique two or three character mnemonic, *XXX*. The mnemonic *UU* identifies the toolset vendor used with the KwikNet Porting Kit.

WEB Sample Program Files

To build the KwikNet WEB Sample Program using make file *KNWEBSAM.MAK*, each of the following source files must be present in the indicated destination directory.

Source File	Destination Directory	File Purpose	
.	*CFGBLDW*	KwikNet Configuration Builder; template files	
	KwikNet source directories containing:		
KN_API.H	*TCPIP*	KwikNet Application Interface definitions	
KN_OSIF.H	*TCPIP*	KwikNet OS Interface definitions	
KN_VFS.H	*TCPIP*	KwikNet Virtual File System definitions	
KN_FILES.H	*TCPIP*	KwikNet Universal File System definitions	
KNFSUSER.H	*TCPIP*	KwikNet User File System definitions	
KN_SOCK.H	*TCPIP*	KwikNet Socket Interface definitions	
	Toolset root directory containing:		
KN_OSIF.INC	*TOOLXXX*	OS Interface Make Specification	
KNZZZCC.INC	*TOOLXXX*	Tailoring File (for use with make utility)	
KNZZZCC.H	*TOOLXXX*	Compiler Configuration Header File	
	KwikNet WEB Sample Program directory containing:		
KNWEBSAM.MAK	*TOOLXXX\SAM_WEB*	WEB Sample Program make file	
KNWEBSAM.C	*TOOLXXX\SAM_WEB*	WEB Sample Program	
KNWEBVFS.C	*TOOLXXX\SAM_WEB*	WEB Sample Program VFS Data File	
KNZZZAPP.H	*TOOLXXX\SAM_WEB*	WEB Sample Program Application Header	
KNWEBLIB.UP	*TOOLXXX\SAM_WEB*	Network Parameter File	
KNWEBSAM.LKS	*TOOLXXX\SAM_WEB*	Link Specification File	(toolset dependent)
		Other toolset dependent files may be present.	
KNWEBSCF.UP	*TOOLXXX\SAM_WEB*	User Parameter File	(for use with AMX)
KNWEBTCF.UP	*TOOLXXX\SAM_WEB*	Target Parameter File	(for use with AMX)
	Common sample program source files:		
KNSAMOS.C	*TOOLXXX\SAM_COMN*	Application OS Interface	
KNSAMOS.H	*TOOLXXX\SAM_COMN*	Application OS Interface header file	
KNRECORD.C	*TOOLXXX\SAM_COMN*	Message recording services	
KNCONSOL.C	*TOOLXXX\SAM_COMN*	Console driver	
KNCONSOL.H	*TOOLXXX\SAM_COMN*	Console driver header	
		Console driver serial I/O support:	
KN8250S.C	*TOOLXXX\SAM_COMN*	INS8250 (NS16550) UART driver	
KN_BOARD.C	*TOOLXXX\DRIVERS*	Board driver for target hardware	

WEB Sample Program Parameter File

The Network Parameter File *KNWEBLIB.UP* describes the KwikNet and HTTP options and features illustrated by the sample program. This file is used to construct the KwikNet Library for the WEB Sample Program.

The Network Parameter File *KNWEBLIB.UP* also describes the network interfaces and the associated device drivers that the sample program needs to operate.

WEB Sample Program KwikNet Library

Before you can construct the KwikNet WEB Sample Program, you must first build the associated KwikNet Library.

Use the KwikNet Configuration Builder to edit the sample program Network Parameter File *KNWEBLIB.UP*. Use the Configuration Builder to generate the Network Library Make File *KNWEBLIB.MAK*.

Look for any KwikNet Library Header File *KN_LIB.H* in your toolset library directory *TOOLXXX\LIB*. If the file exists, delete it to ensure that the KwikNet Library is rebuilt to match the needs of the WEB Sample Program.

Then copy files *KNWEBLIB.UP* and *KNWEBLIB.MAK* into the *MAKE* directory in the KwikNet installation directory *KNTnnn*. Use the Microsoft make utility and your C compiler and librarian to generate the KwikNet Library. Follow the guidelines presented in Chapter 3.2 of the KwikNet TCP/IP Stack User's Guide.

Note

The KwikNet Library must be built before the WEB Sample Program can be made. If file *KN_LIB.H* exists in your toolset library directory *TOOLXXX\LIB*, delete it to force the make process to rebuild the KwikNet Library.

The WEB Sample Program Make Process

Each KwikNet sample program must be constructed from within its own directory in the KwikNet toolset directory. Hence, the KwikNet WEB Sample Program must be built in directory *TOOLXXX\SAM_WEB*.

All of the compilers and librarians used at KADAK were tested on a Windows® workstation running Windows NT, 2000 and XP. However, you can build each KwikNet sample program using any recent version of Windows, provided that your software development tools operate on that platform.

To create the KwikNet WEB Sample Program, proceed as follows. From the Windows Start menu, choose the MS-DOS Command Prompt from the Programs folder. Make the KwikNet toolset *TOOLXXX\SAM_WEB* directory the current directory.

To use Microsoft's *NMAKE* utility, issue the following command.

```
NMAKE -fKNWEBSAM.MAK "TOOLSET=XXX" "TRKPATH=treckpath"
     "OSPATH=yourospath" "TPATH=toolpath" "FSPATH=afspath"
```

The make symbol *TOOLSET* is defined to be the toolset mnemonic *XXX* used by KADAK to identify the software tools which you are using.

The symbol *TRKPATH* is defined to be the string *treckpath*, the full path (or the path relative to directory *TOOLXXX\SAM_WEB*) to your Turbo Treck TCP/IP installation directory.

The symbol *OSPATH* is defined to be the string *yourospath*, the full path (or the path relative to directory *TOOLXXX\SAM_WEB*) to the directory containing your RT/OS components (header files, libraries and/or object modules). When using AMX, string *yourospath* is the path to your AMX installation directory.

The symbol *TPATH* is defined to be the string *toolpath*, the full path to the directory in which your software development tools have been installed. For some toolsets, *TPATH* is not required. The symbol is only required if it is referenced in file *KNZZZCC.INC*.

If you are using KwikNet with AMX and the AMX/FS File System, you must define the symbol *FSPATH* to be the string *afspath*, the full path (or the path relative to directory *TOOLXXX\SAM_WEB*) to the directory in which you installed AMX/FS. If you are not using the AMX/FS File System, omit the definition of symbol *FSPATH*.

The KwikNet WEB Sample Program load module *KNWEBSAM.xxx* is created in toolset directory *TOOLXXX\SAM_WEB*. The file extension of the load module will be dictated by the toolset you are using. The extension, such as *OMF*, *ABS*, *EXE*, *EXP* or *HEX*, will match the definition of macro *XEXT* in the tailoring file.

The final step is to use your debugger to load and execute the KwikNet WEB Sample Program load module *KNWEBSAM.xxx*.

2.3 Adding the Web Server to Your Application

Before you can add the KwikNet Web Server to your application, there are a number of prerequisites which your application must include. You must have a working KwikNet TCP/IP stack operating with your RT/OS and a file system. It is imperative that you start with a tested TCP/IP stack and file system with functioning device drivers before you add the web server. If these components are not operational, the KwikNet Web Server cannot operate correctly.

KwikNet Library

Begin by deciding which HTTP features must be supported. Review the Web property page described in Chapter 1.3. In particular, omit CGI and SSI support unless you actually have such a need. The memory savings are significant.

Read Appendix C of the KwikNet TCP/IP Stack User's Guide to see how the KwikNet file system interface will have to be configured to operate with the file system you have chosen.

If you use a file system that is accessed via the KwikNet Universal File System (UFS), your web server will require a user name and password to gain access to the file system services. Read Appendix D of the KwikNet TCP/IP Stack User's Guide to see how the KwikNet Administration interface must be adapted to define such a user name and password. If necessary, edit the administration file *KN_ADMIN.C* to revise the default user name and password provided for web server use.

Use the KwikNet Configuration Manager to edit your application's KwikNet Network Parameter File to support a file system and to include the KwikNet Web Server. Armed with your modified files (if any), rebuild your KwikNet Library. The library extension may be *.A* or *.LIB* or some other extension dictated by the toolset which you are using.

Memory Allocation

Each web server task requires one socket for listening for requests from potential web browsers (clients). Each browser session will require an additional socket for control (commands) and data transfer. Additional memory is allocated for each browser session managed by your web server.

To meet these requirements, you may have to edit your KwikNet Network Parameter File to increase the memory available for allocation.

Web Server Task

You must provide one task for each web server which you wish to incorporate into your application. Usually only one web server task is required. Rarely are two or more web servers required.

In a multitasking system, you may have to increase the total number of tasks allowed by your RTOS in order to add the web server task.

A stack size of 4K to 8K bytes is considered adequate for the web server task when used with most file systems and device drivers. The stack size can be trimmed after your web server task has been tested and actual stack usage observed using your debugger.

In a multitasking system, each web server task must be of lower execution priority than the KwikNet Task. If your application also includes a web browser task, it is usual to make the web server task of higher priority than the web browser task.

To incorporate a web server task, you need only create a web server task procedure which begins operation as described in Chapter 1.5.

The web server task C source module must be compiled just like any other KwikNet application module. The compilation procedure is described in Chapter 3.4 of the KwikNet TCP/IP Stack User's Guide.

Reconstructing Your KwikNet Application

Since you are adding a web server to an existing KwikNet application, there is little to be done.

To meet the memory demands of your web server, you may have to edit your KwikNet Network Parameter File to increase the memory available for allocation. If you do so, you must then rebuild your KwikNet Library.

Your application link and/or locate specification file must include the KwikNet Library which you built with support for a web server. The object module for your web server task and any support modules that it might require must also be included in your link specification together with your other application object modules.

With these changes in place, you can link and create an updated KwikNet application with web server support included.

AMX Considerations

When reconstructing a KwikNet application which uses the AMX Real-Time Multitasking Kernel, adapt the procedure just described to include the following considerations.

You may have to edit your AMX User Parameter File to increase the maximum number of tasks allowed in order to add a web server task.

A web server task can be predefined in your AMX User Parameter File or it can be created dynamically at run-time as is done in the KwikNet WEB Sample Program. Such a task is a simple AMX trigger task without message queues.

A stack size of 4K to 8K bytes is considered adequate for use with most file systems and device drivers. The stack size can be trimmed after your web server task has been tested and actual stack usage observed using your debugger.

The web server task priority must be lower than that of the KwikNet Task. If your application also includes a web browser task, it is usual to make the web server task of higher priority than the web browser task. If you are using AMX 86 to access MS-DOS® file services, the PC Supervisor Task should be below the web server task in priority.

If you edit your AMX User Parameter File, you must then rebuild and compile your AMX System Configuration Module. If you are using the AMX/FS File System, you should also rebuild and compile your AMX/FS File System Configuration Module.

No changes to your AMX Target Configuration Module are required to support HTTP unless your web server task requires special device support which is not already part of your application.

Performance Considerations

A meaningful discussion of all of the issues which affect the performance of a web server or browser are beyond the scope of this document. Factors affecting the performance of the KwikNet Web Server include the following:

processor speed
memory access speed and caching effects
file system performance and disk access times
competing disk accesses for different users
network type (Ethernet, SLIP, PPP)
network device driver implementation (buffering, polling, DMA support, etc.)
TCP protocol effects (window size adaptations)
IP packet fragmentation
network hops required for connection
operation of the remote (foreign) connected browser
KwikNet TCP/IP Stack configuration (clock, memory availability, sockets, etc.)

Of all these factors, only the last one can be easily adjusted. Increasing the fundamental clock rate for the KwikNet TCP/IP Stack beyond 50Hz will have little effect and will adversely affect systems with slow processors or memory. Increasing the memory available for use by the TCP/IP stack will help if high speed Ethernet devices are in use and the processor is fast enough to keep up.

This page left blank intentionally.