Rapid Internet Device Development with Interpretive Languages

Michael Johnson
iReady Corporation
2903 Bunker Hill Lane, Suite 200
Santa Clara, CA 95054

mike@corp.iready.com

Abstract

In its quest for ubiquitous Internet computing, iReady has developed the Internet Tuner with Ethernet (ITE) a highly integrated Internet system-on-chip. Firmware called a "personality module" is designed to complement the ITE system providing high level Internet protocol functionality, vertical applications like Internet faxing, or to provide for the ITE to be a standalone device.

A personality module that is being developed is the NetScript module. This module allows the ITE to be used as a standalone programmable internet computer. Using the netbasic interpreter and its powerful internet functions it is possible to design, develop and debug simple internet devices in very little time. For more complex development the netbasic interpreter can run on a Personal Computer and emulate the network functionality of the ITE. This environment provides a higher level of debugging capabilities, which allows more complex designs to be created without being constrained by an embedded environment. Designs created in the emulated environment will run without modification on the ITE environment.

This paper will touch on the origins of NetScript, introduce the NetScript language, and show that rapid Internet device development has never been easier than with the NetScript interpreter and the ITE. Examples of a simple webserver and an email-reporting engine that interact with their environment will also be provided.

Introduction

The Internet is rapidly driving a new generation of computing technologies. Network computers, webpads, handheld PC's and PDA's are the first wave of the next generation. The second wave will be more ubiquitous, connecting appliances, industrial machinery, environmental control systems, alarm systems, and every other product that has the ability to be controlled or report data. Driving this second wave will be low cost, easy to use and integrated Internet hardware and it's associated software.

With this in mind, iReady has designed a hardware product called the *Internet Tuner with Ethernet* (ITE*)*. It is a highly integrated Internet system-on-chip (see figure 1) that provides all the protocols needed to connect and operate on the Internet via Ethernet or dialup PPP. The ITE lowers the barriers to connect devices to the Internet by lowering costs and reducing development and integration time.

The ITE has an 8-bit general-purpose processor onboard. This allows the system to provide higher level Internet protocols like DHCP, SMTP, POP, HTTP and to take advantage of the on-chip accelerators that handle Base64 and MIME encoding.

The ITE was designed to be a slave in an embedded system, accepting commands via a host CPU interface to provide internet functionality. The functionality that the ITE would provide could be generic UDP and TCP sockets with DNS and DHCP or it could provide the functionality of a more vertical application like a SOHO router or an Internet faxing device. The personality of the ITE can be selected by changing the firmware. Each set of firmware for the ITE is called a *personality module*.

A personality module that is being developed is the NetScript module. Using this module the ITE becomes a low cost, programmable, Internet speaking microcontroller. This module can be integrated into existing designs or used as the controller of new Internet appliance/controller designs.

This paper will describe the NetScript language, the embedded development environment, the emulation environment, and give two examples of internet appliances built with very little effort using the ITE + NetScript personality module.

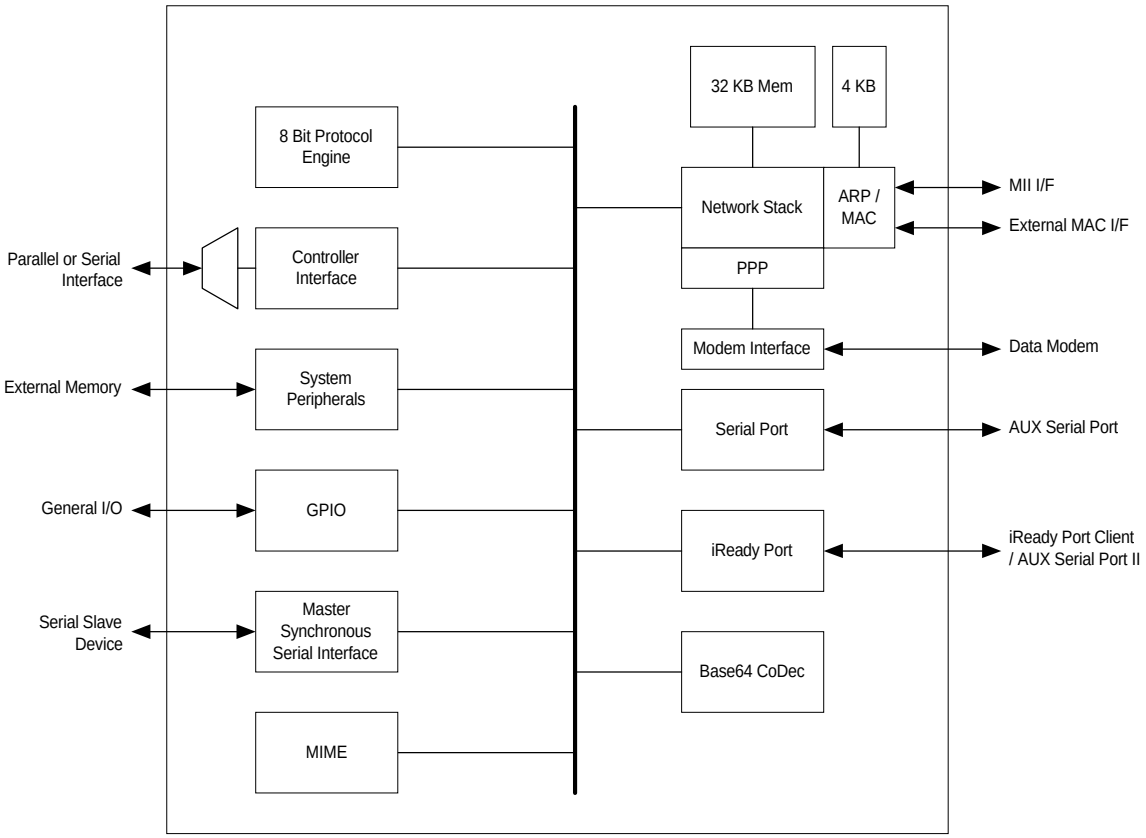

Figure 1. Internet Tuner with Ethernet Block Diagram

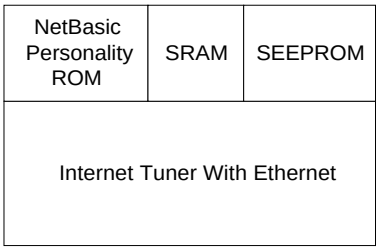

Figure 2. NetScript Module based on the ITE

Why NetScript?

NetScript was developed by iReady as a personality module to shield developers from the complexities of Internet protocols. It allows development of network applications with a minimum of support overhead and custom development tools. It also allows the underlying hardware to be modified and enhanced in further revisions without the need to port applications.

Netscript is quite simply, a basic language. It can be run on a very small footprint with powerful results. Not 20 years ago many complex applications were developed on the Apple II computer using the Applesoft version of BASIC. Most of these applications fit into 48K of program space, with the Applesoft Basic occupying 16K of the 64K memory map. Color graphics games, accounting and database software, word processors, spreadsheets, and other complex programs were developed using this resource-limited system. iReady has taken the BASIC language concept and added built-in Internet functions that allow developers to quickly code and test network applications. These Internet functions enable single line manipulation of IP addresses, socket connections and statuses, and other network functions.

A webserver on a sprinkler system, a notification unit for a mailbox or alarm system, or a temperature sensor in a freezer needs very little processing power. Many embedded systems have similar simple processing requirements. NetScript fits these processing requirements and has the added benefit that many companies that do not have software engineers or professional programmers on staff may still wish to develop internet connected products and would feel comfortable developing on the ITE with NetScript.

Even for professional developers, there may be no faster way to create simple Internet-enabled devices than with NetScript and the ITE. Connecting a serial port or opening a TELENT session to the ITE and typing a few lines of NetScript code can get an email reporting engine or webserver going. A few more lines of NetScript code and status can be read via the GPIO pins or the synchronous serial interface port. The possibilities are endless.

The NetScript Language

All statement lines are of the form

<Line> <Keyword> <Parameters> [: <Statement> [: ...]]

where *Line* is a statement line number, *Keyword* is a NetScript statement keyword, and *Parameters* are a set of parameters associated with that keyword.

The line number has two purposes: It serves as a label for statements that control execution flow, such as a goto statement, and it serves as a sorting tag for statements inserted into the program. As a sorting tag, the line number facilitates a line-editing environment in which editing and command processing are mixed in a single interactive session.

While simple in design, line numbers do give the interpreter environment the ability to update the program one statement at a time. This ability stems from the fact that a statement is a single parsed entity and can be linked in a data structure with line numbers. Without line numbers, often it is necessary to re-parse the entire program when a line changes.

The keyword identifies the NetScript statement. The NetScript interpreter will support a slightly extended set of BASIC keywords, including *goto, gosub, return, print, if, end, data, restore, read, on, rem, for, next, let, input, stop*, and *dim*.

Each keyword has a set of legal keyword parameters that can follow it. For example, the goto keyword must be followed by a line number, the if statement must be followed by a conditional expression as well as the keyword then -- and so on. The

Expressions and operators

A parameter specified in a statement is often an expression. NetScript supports all of the standard mathematical operations, logical operations, exponentiation, and a simple function library. The most important component of the expression grammar is the ability to call functions.

Variables and data types

Part of the reason BASIC is such a simple language is because it has only two data types: numbers and strings. Some scripting languages, such as PERL, don't even make this distinction between data types until they are used. But with BASIC, a simple syntax is used to identify data types.

Variable names in NetScript are strings of letters and numbers that always start with a letter. Variables are not case-sensitive. Thus *A, B, FOO,* and *FOO2* are all valid variable names. Furthermore, the variable *FOOBAR* is equivalent to *FooBar*. To identify strings, a dollar sign ($) is appended to the variable name; thus, the variable *FOO$* is a variable containing a string.

Finally, arrays are supported using the *dim* keyword and a variable syntax of the form *NAME*(*index1, index2,* ...) for up to four indices.

Program structure

Programs in NetScript start by default at the lowest numbered line and continue until there are either no more lines to process or the *stop* or *end* keywords are executed. A very simple NetScript program is shown below:

```
100 REM This is an Example NetScript program
110 REM Note that REM statements are ignored.
120 PRINT "This is an Example program."
130 PRINT "Summing the values between 1 and 10"
140 LET total = 0
150 FOR I = 1 TO 10
160 LET total = total + i
170 NEXT I
180 PRINT "The total of all digits between 1 and 10 is " total
190 END
```

The line numbers above indicate the lexical order of the statements. When they are run, lines 120 and 130 print messages to the output, line 140 initializes a variable, and the loop in lines 150 through 170 update the value of that variable. Finally, the results are printed out.

Program Construction

A NetScript program is constructed of numbered NetScript statements. Line numbers are integral numbers between 0 and 63999. A statement has the general form of :

[NUMBER] KEYWORD Statement Specific information [: KEYWORD Statement]

Variables in NetScript are either numeric or string variables. String variables are distinguished by a trailing $ character. Legal numeric variables are A - Z, A0 - A9,..., Z0 - Z9. The set of legal string variables are A$ - Z$.

String constants are expressed in NetScript by any characters surrounded by double quote (") characters.

"Internet Ready","a","123"

Are all examples of legal string constants.

Numeric constants are expressed as integer numbers.

Statements

The NetScript interpreter accepts several statements in the construction of programs, statements are preceded by a keyword and followed by statement specific information. Statements that begin with a line number are stored for later execution, statements that do not have a line number are executed immediately. The statements that NetScript implement are shown below.

LET	LET a=b+c	The LET statement is optional, any statement that starts with a variable is assumed to be a LET statement.
DATA	DATA 1,3,5,7	Specifies an expression list, which may be sequentially retrieved by READ
DIM	DIM a(50)	Creates an array a of size 50.
END	END	Halts the currently running program.
FOR	FOR a=1 TO 5	For look a=1 to 5
GOSUB	GOSUB 1000	Calls a subroutine at line 1000
GOTO	GOTO 2000	jump to line 2000
IF	IF x=1 THEN GOTO 100	If test of x=1 is true then following THEN is executed.
INPUT	INPUT x	Input numeric and string data from the console during execution.
LIST	LIST 10-50	List Lines 10 through 50
NEXT	NEXT a	specifies end of for loop, index variable is optional
NEW	NEW	Clears program and variable space.
ON	ON x GOTO 1,2	conditionally branch depending on value of x.
PRINT	PRINT “hello”	Sends expressions and strings to output device.
READ	READ A	Reads a value from a DATA list an incerments the DATA pointer.
REM	REM	Allows remarks in a program
RESTORE	RESTORE	Resets the READ instruction pointr to the beginning of DATA list.
RETURN	RETURN	Returns from subroutine call
STOP	STOP	Stops program execution
SAVE	SAVE	Saves program to non volatile memory

Functions

NetScript has many built in functions. These are used for generating random values and dissecting strings. The complete list as of this writing are as follows:

NetScript Numeric Functions

TICK	x=TICK();	Returns the current TICK count (Each Tick = 100ms)
RND	x=RND(n)	Return a Random Number between 1 and n
MAX	x=MAX(a,5*b)	Returns the value of the expression that is greater.
MIN	x=MIN(a,5*b)	Returns the value of the expression that is less.
ABS	x=ABS(-5)	Returns the absolute value of the expression.

NetScript String Functions

VAL	x=VAL(n$)	Converts a numeric constant stored in a string into a integer.
LEN	x=LEN(n$)	Returns the number of characters in a string.
LEFT$	x$=LEFT$(a$,n)	Returns the n leftmost characters in a string.
RIGHT$	x$=RIGHT$(a$,n)	Returns the n rightmost characters in a string.
MID$	x$=MID$(a$, n, m)	Returns all the characters between the nth and mth positions in the string
STRSTR$	x=STRSTR(a$,b$)	Returns the index in a$ of occurrence of b$
CHR$	x$=CHR$(20)	Returns the ASCII character of expression.
STR$	x$=STR$(n)	Returns a string containing the numeric value of expression.

NetScript Internet Functions

IPSTR$	x$=IPSTR$(n$)	Returns a ASCII string representation of an IP address from a 4 byte binary string.
IPVAL$	x$=IPVAL$(n$)	Returns a 4 byte binary string for a ASCII string representation of an IP address.
SETIP	x$=SETIP(n$)	Sets system's IP addresses.
GETIP	x$=GETIP()	Gets system's IP address.
SETIPM	x$=SETIPM(n$)	Sets system's IP address mask.
SETGW	x$=SETGW(n$)	Sets system's Gateway.
DHCP()	x=DHCP()	Configure the ITE via Ethernet using DHCP.
RESOLVE$()	x$=RESOLVE$(n$)	Resolve name into an IP address.
SLISTEN()	x=SLISTEN(p)	TCP listen on port p
SOPEN()	x=SOPEN(i$,p)	TCP connect to host i on port p
SBIND()	x=SBIND(p)	Connect Socket to UDP port p
SSTATE()	x=SSTATE(s)	Returns the state of socket s
SCLOSE()	SCLOSE(s)	Closes a TCP socket or removes a SBIND command
SPEEK()	x=SPEEK(s)	Returns the number of bytes waiting to be read from a socket.
SREAD$()	x$=SREAD$(s,len)	Reads socket s up to len bytes
SWRITE$()	SWRITE(s,n$)	Writes socket s string n$

NetScript Misc. Funtions

NVREAD$()	x$=NVREAD$(n)	Reads byte n from non volital memory.
NVWRITE$()	NVWRITE(n,x$)	Writes first character in x$ to byte n of non volatile memory.
IOREAD()	x$=IOREAD()	Read the GPIO pins
IOWRITE()	IOWRITE(x$)	Write the GPIO pins
IODIR()	IODIR(x$)	Sent the GPIO direction of each of the GPIO pins
ASIN()	a$=ASIN()	Reads a byte for the AUX serial port.
ASOUT()	ASOUT(a$)	Writes a byte to the AUX serial port.

Embedded Development Environment

Shown in Figure 3 is a typical development environment using a NetScript module. The NetScript module's console is an interactive input/output device. Typically a PC running serial communications software is connected to the NetScript module's console. All interactive NetScript program editing and debugging takes place on the console. Error messages are output to the console. Program listings and PRINT output are sent to the console.

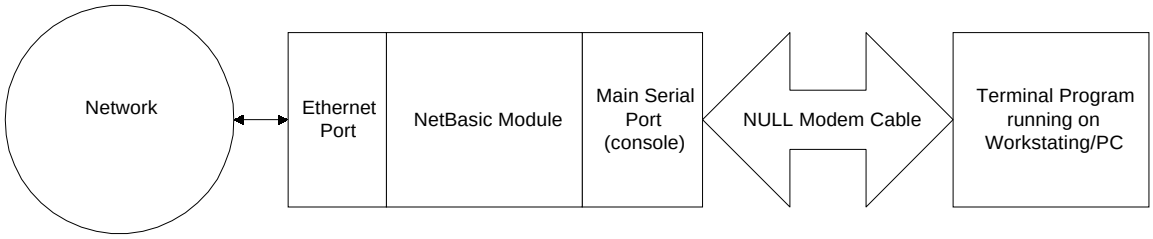

Figure 3. Embedded Development Environment

The console is required to write and debug NetScript programs. It is not required to RUN NetScript programs.

Emulation Development Environment

With the emulation development environment a user develops a program using an interactive console window that is part of a software program that runs under Linux or MS Windows. This emulation environment uses the exact same NetScript interpreter that runs in the embedded environment but with some enhancements. These enhancements include a user interface that allows easy access to an advanced debugger and ITE emulator that allows the host operating system to provide all of the Network functionality that the ITE would normally provide. A block diagram of this environment is shown below in figure 4.

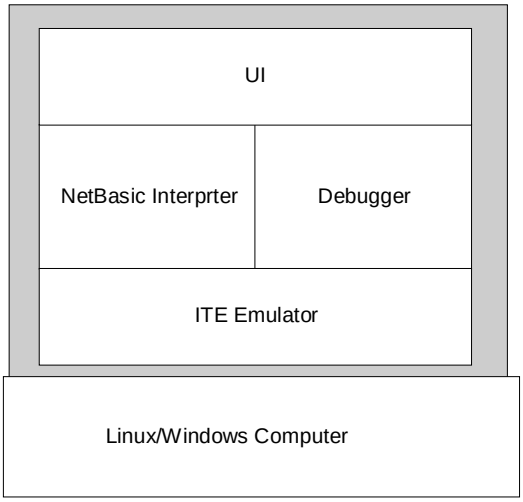

Figure 4. Emulation Development Environment

This environment may also be used as an evaluation platform, allowing developers to create Internet applications on a personal computer system, thereby allowing the developer to experience the simple yet powerful NetBasic environment. Applications could then run on the personal computer system or be transferred to a NetScript Module.

Example NetScript Designs

The main goal of the NetScript module it to lower the effort to develop simple Internet enabled devices. Most developers should be able to get simple Internet devices up and running in less than an hour.

The following two sections show example designs developed with the NetScript Module.

Email Reporting Engine

This example illustrates how the NetScript Module can be used to monitor the state of sensors connected to the GPIO pins of the ITE and send an email whenever the state of the sensors change. Figure 5 below, depicts how the NetScript module is used in this example.

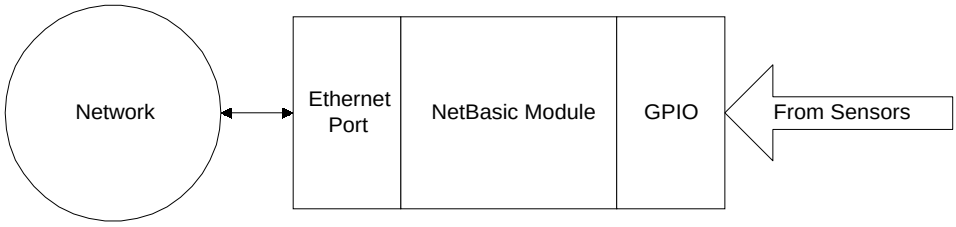

Figure 5. Email Reporting Engine Hookup

The example is less than 50 lines of code to initialize the system using DHCP, check the sensors for change, do a DNS server lookup of the mailserver, and connecti and sendithe message to the email server. A state transition diagram of the software is shown in figure 6.

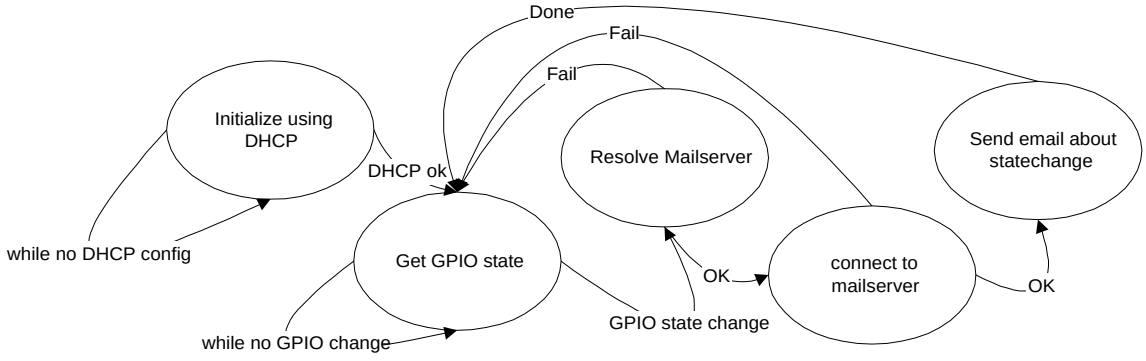

Figure 6. Email Reporting Engine Software State Diagram

This example illustrates how to develop a simple system that waits for a GPIO to enter an active state, then using this state as a trigger, send an email to an administrator to log this state.

```
10 REM Setup Some Global Buffers and constants
20 DIM buffer$(80)
30 DIM ip$(16)
40 DIM mailserver$(80)
50 crlf$=CHR$(10)+CHR$13
```

```
60 mailserver$=”mail.iready.org”
```

The following code example uses DHCP to configure the system. It will keep trying until the DHCP function returns a success code.

```
100 REM Setup the Ethernet parameters via DHCP
110 x=DCHP();
120 if x=1 then goto 150
130 PRINT “DHCP failed, trying again”
140 goto 110

150 REM Read current state of port 0 GPIO pins0
160 a = IOREAD(0)
170 REM Wait for one of the GPIO pins to go high.
180 b=IOREAD(0)
190 if b != a THEN GOTO 300
200 goto 170

300 REM we have an I/O pin change, send an email to report it.
310 ip$=RESOLVE$(mailserver$)
320 IF ip$!=”” THEN GOTO 350
330 PRINT “Failed DNS lookup of” + mailserver$
340 GOTO 150

350 REM open a socket to the mailserver
360 socket=SOPEN(ip$,25)
370 REM wait for socket to open
380 x=SSTATE(socket);
390 ON x GOTO 370,370,400,700,700,700,700

400 REM socket is open send the email message
410 buffer$=”HELO remote.iready.org”+crlf$
420 SWRITE(socket,buffer$)
430 buffer$=”MAIL FROM:<remote@iready.org>"+crlf$
440 SWRITE(socket,buffer$)
450 buffer$=”RCPT TO:<server@iready.org>"+crlf$
460 SWRITE(socket,buffer$)
470 buffer$=”DATA”+crlf$
480 SWRITE(socket,buffer$)
450 buffer$=”device reporting state change”+crlf$
460 SWRITE(socket,buffer$)
470 buffer$=”original state value = ”+STR$(a)+crlf$
480 SWRITE(socket,buffer$)
490 buffer$=”new state value = “+STR$(b)+crlf$
500 SWRITE(socket,buffer$)
510 buffer$=”.”+crlf$
520 SWRITE(socket,buffer$)
530 SCLOSE(socket)
540 goto 150

700 REM had a problem opening the connection to the server
710 PRINT “error connecting to the mail server”
720 goto 150
```

Device Webserver

This next example illustrates how the NetScript Module can be used as a webserver to monitor and control a device connected to the GPIO pins of the ITE. A web browser can connect to this module and read the current status of the device and change its settings. Figure 7 depicts how the NetScript module is used in this example.

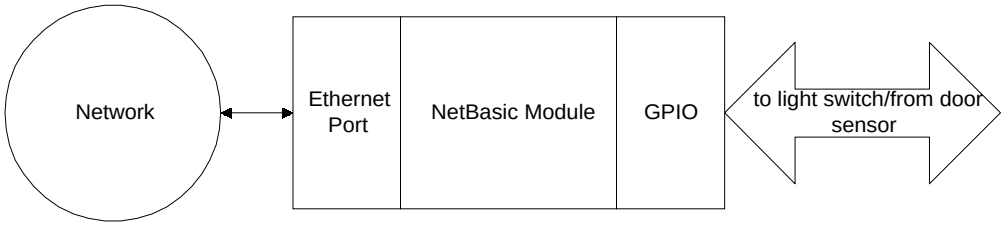

Figure 7. Webserver Block Diagram

The example is less than 100 lines of code to perform system initialization, listen for connections from Web Browsers, parse HTTP GET requests, read sensors and display their values via html, change switches depending on user requests, and send html data across the network. A state transition diagram of the software is shown in figure 8.

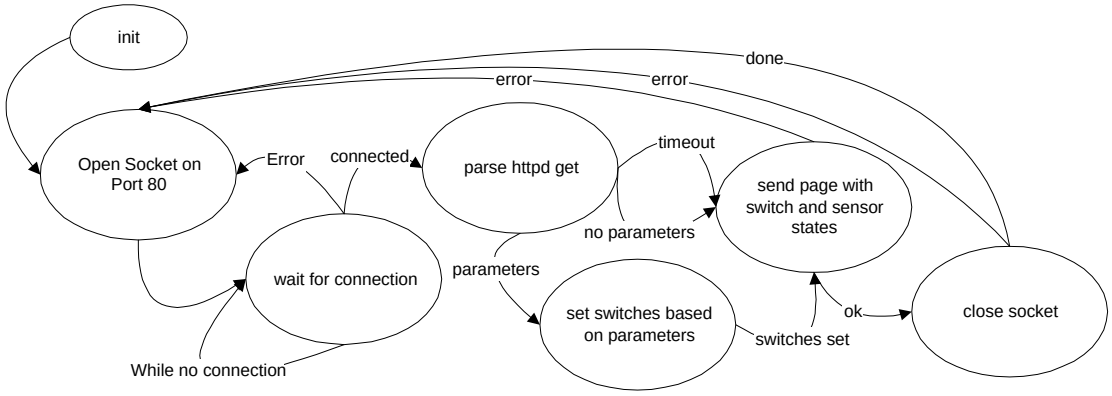

Figure 8. Webserver software state diagram.

The following codes shows hot to serve up a simple WebPages that will report the state of all the GPIO pins.

```
50 REM Setup Some Global Buffers
60 DIM x$(25)
70 DIM buffer$(200)
50 crlf$=CHR$(10)+CHR$13
```

The webserver design uses manual configuration of the network instead of using DHCP.

```
100 REM Setup the ethernet IP address, Netmask, Gateway and DNS server
110 x$=”192.168.1.10”:SETIP(x$)
120 x$=”255.255.255.0”:SETIPM(x$)
130 x$=”192.168.1.1”:SETGW(x$)
```

```
140 x$=”192.168.1.1”:SETDNS(x$)

150 REM Setup the GPIO for 7 inputs and 1 output
160 x$=0feh:IODIR(x$)
```

Start the webserver by opening a listen socket on port 80.

```
200 REM Start listening to for tcp connections on port 80
210 s=SLISTEN(80)
220 REM wait for a connection on port 80
230 x=SSTATE(s)
240 IF x=3 THEN GOTO 300
250 IF x>10 THEN GOTO 800
250 GOTO 230

300 REM
301 REM There is a waiting connection, service it.
302 REM Wait for an incoming GET on the socket
320
330 GOSUB readlnsocket
340 if error != 0 then goto 800

350 REM
351 REM parse incoming request
352 REM
360 bit=1
370 x=STRSTR(buffer$,”\o”);
380 IF x>0 THEN GOTO 410
390 x=STRSTR(buffer$,”\f”);
400 bit=0;
410 IOWRITE(bit)

500 REM
501 REM dump the WebPages
502 REM
510 buffer$=crlf$+crlf$
520 SWRITE(s,buffer$)
530 buffer$=”<HTML><TITLE>iReady NetBasic Webserver</TITLE>”
540 SWRITE(s,buffer$)
550 buffer$=”<body bgcolor=””#ddFFdd””> <FONT SIZE=6><B><CENTER>”
560 SWRITE(s,buffer$)
570 buffer$=”iReady NetBasic Webserver</B></FONT>”
580 SWRITE(s,buffer$)
590 buffer$=”<br><br>Door is ”
600 SWRITE(s,buffer$)
610 x=IOREAD()
620 IF (x & 02h) THEN GOTO 640
630 buffer$=”CLOSED “:GOTO 650
640 buffer$=”OPEN “
650 buffer$=buffer$+”<br><br>”
660 SWRITE(s,buffer$)
```

```
670 buffer$=”<br><br>Light is ”
680 SWRITE(s,buffer$)
690 IF x & 1 THEN GOTO 710
700 buffer$=”OFF <a href=”\n”>(turn on)</a>”:GOTO 720
710 buffer$=”ON <a href=”\f”>(turn off)</a>”
720 SWRITE(s,buffer$)
730 buffer$=”<HTML>”
740 SWRITE(s,buffer$)

800 REM close socket and restart
810 SCLOSE(s)
820 GOTO 200

900 REM
901 REM readlnsocket
902 REM
910 t=TICK();
920 IF ABS(TICK()-t) > 50) THEN GOTO 980
930 IF SPEEK(s) = 0 THEN GOTO 920
940 buffer$=SREAD(s,200)
950 error=0
960 RETURN
980 error=1
995 RETURN
```

Conclusion

The ITE with the NetScipt personality module is a very powerful programmable Internet computer. It is a great replacement for many much more expensive Internet connectivity solutions or a solid starting point for creating new Internet device solutions.

The external interfacing of the ITE is very powerful and flexible. The fully programmable GP I/O pins can be used to directly interface to TTL-level devices, such as buttons, LEDs, speakers, potentiometers, and shift registers. With just a few extra components, these I/O pins can also be connected to non-TTL devices, such as solenoids, relays, and other high current/voltage devices. A powerful array of synchronous and asynchronous serial ports allows direct interaction and control of many types of serial devices.

Adding the programmability of NetScript and solid Internet + Ethernet connectivity to the I/O capabilities of the ITE make it the first choice for any Internet device development effort. There is no quicker, easier, or cost effective way to develop simple Internet devices than with the ITE and the NetScript personality module.