User Guide and Reference

ISC Script Uploader   
Demonstration for Aware360 hardware Setup March 2019

© 2019 Synapse Wireless® All rights reserved. Synapse products are patented or patents pending. Specifications are subject to change without notice.

# Overview

This document provides the necessary technical details for setting up and running the Script Uploader library demonstration prepared for ISC. Specifically, it covers:

* Setting up the demonstration hardware.
* Compiling the demonstration software.
* Executing the demonstration.

This document assumes that the zip file containing the ISC deliverables has been unpacked and the contents can be found in a directory that we will call <ISC\_BASE>. This directory will contain a ‘demonstration’ directory which has all of the files for the demonstration, and a ‘documentation’ directory (which contains this file).

Specifically not covered in this document is the SpyConverter program that was previously provided by Synapse to ISC as that program has not changed.

# Demonstration Hardware and Setup

The hardware used for the demonstration is the current version of the Aware360/ISC LENS gateway shown below:



The demonstration hardware is connected by performing the following steps:

1. Unpack the above hardware and leave powered off.
2. Remove the top cover for access to the UART – Using provided USB-to-Serial converter by Aware360, it is a FTDI device, wired to match the hardware.
3. Connect Ethernet to a DHCP server for remote IP connections (i.e. ssh)
4. Provide power to the hardware at 14V DC.
5. Log into the unit using correct username and password.
   1. For this test the user was always root, password left out of this document intentionally
6. Verify the IP address for copying files to the system (i.e. using ifconfig)

# Demonstration Software

There are two software components that take part in the demonstration:

* The binary form of the SNAPpy script that will be uploaded. (Previously converted by the SpyConverter program from a .spy file to the .bin file.)
* The built demo program to execute on the system performing the uploading action.

# SNAPpy Script Setup

The test files used for this were provided by ISC and are named TruLinkUploadTest.py TruLinkUploadTest2.py and TruLinkUploadTest3.py. These files are exported from Portal as .spy files and converted to binary by the converter. Prior to running the demonstration, we want to make sure that the test1 script is not currently loaded on the module. To confirm this:

1. Open Portal and connect to a bridge node.
2. Configure the bridge node for default communications:
   1. Network ID: 1C2C
   2. Channel: 4
   3. No encryption
   4. Feature bit 8 off (no second CRC)
3. Find the target module by its address (found on the label).
4. Refresh the node information for the target module.
5. If it is currently loaded with the testscript, erase the script from the module.

# Running the Demonstration

To upgrade the Synapse SM220UF1 Module on board, the demo program will need to be loaded onto the hardware with the binary image of the test scripts.

Before using the uploader program the Synapse SM220UF1 must be enabled by outputting a high value on GPIO123, please issue this command any time prior to using the program.

*echo 1 > /sys/class/gpio/gpio123/value*

The program for the demo is named “scriptUploaderDemo.x”. To run this command with the first test program, we will need to supply the proper arguments.

Run the program without any arguments to see the correct order and format

*>./scriptUploaderDemo.x*

*Usage: The usage is scriptUploader.x binary\_file\_name script\_name script\_crc*

*binary\_file\_name is the file generated by SpyConverter.exe*

*script\_name is the name of the script (without the .py extension)*

*script\_crc is the 4-digit hex script CRC shown in Portal*

The arguments are:

**binary\_file** – This is the actual binary file to be loaded

**script\_name** – This is the formal name of the script and is a required argument. It may or may not match the binary file name, but normally would

**script\_crc** – this is the known CRC value of the script as shown by compiling the script in Portal. A table is provided below for the test scripts.

Please Note that the provided scripts from ISC render the UART unusable after upgrade so the upgrade fails with NO RESPONSE TO SCRIPT NAME because there is no longer communication after the reboot. For this reason, a very simple script that contains no hardware interaction TruLinkUploadeTestJRA.py is provided for testing.

|  |  |
| --- | --- |
| **Script Name** | **Scrip CRC**  **(compiled for SM220UF1, firmware 2.5.6 with AES)** |
| TruLinkUploadTest.py | 0x26FC |
| TruLinkUploadTest2.py | 0xE037 |
| TruLinkUploadTest3.py | 0x0554 |
| TruLinkUploadTestJRA.py | 0x0333 |

To upgrade the Synapse Module with TruLinkUploadTest.py we would issue the following command:

*>./scriptUploaderDemo.x TruLinkUploadTest.bin TruLinkUploadTest 26FC*

*The current API version is: 1.0.0*

*Our FD is 3*

*Checking the current CRC*

*The received CRC value is 0*

*The upgrading CRC is 9980*

*................................................*

*DONE*

If the script is already loaded on the module the following output would be seen:

*>./scriptUploaderDemo.x TruLinkUploadTest.bin TruLinkUploadTest 26FC*

*The current API version is: 1.0.0*

*Our FD is 3*

*Checking the current CRC*

*The received CRC value is 9980*

*The upgrading CRC is 9980*

*Existing script CRC matches current CRC. No upload required.*

# Compiling the Demonstration Code

The demonstration software can be compiled using a virtual environment (Oracle VirtualBox, with Ubuntu 18.04 64 bit) installed. Primary files and environment setup were provided by Aware360 and are not included in this distribution.

We assume that the user is using the bash shell for all of these commands.

Run these setup commands after installing Ubuntu OS

*sudo apt install build-essential*

*sudo apt install tclsh*

*sudo dpkg --add-architecture i386*

*sudo apt-get update*

*sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386*

*sudo apt-get install net-tools*

Now unpack the build environment:

*sudo mkdir /usr/share/cross-compile*

*cd /usr/share/cross-compile*

*sudo tar –xzvf arm-sourcery-glibc.tar.gz*

After the build system is setup, the following command should be sourced to set up the environment variables for the compile.

*>source /usr/share/cross-compile/arm-sourcery-glibc/SOURCE\_ME*

Unpack the delivered tar image with software for building the demo program into a directory within the home user directory. For this document we will assume the name to be Synapse.

*>cd Synaspe*

*>tar -xvf ISC-Aware-Demo.tar*

The output should provide the demonstration and documentation directories. The demonstration directory contains the **license.txt** file, **README**, **main.c/cpp** program, the uart interface file **bbuart.cpp/h** and the Synapse Library **scriptuploader.a** and header files **script\_uploader.h** and **syn\_types.h** as well as the **makefile**. The documentation directory includes this document as well as the original instructions (slightly modified) that came from Aware360.

*>make -e*

The makefile is written to support only building for the arm architecture and the output can be found in the Synapse/bin/arm directory.

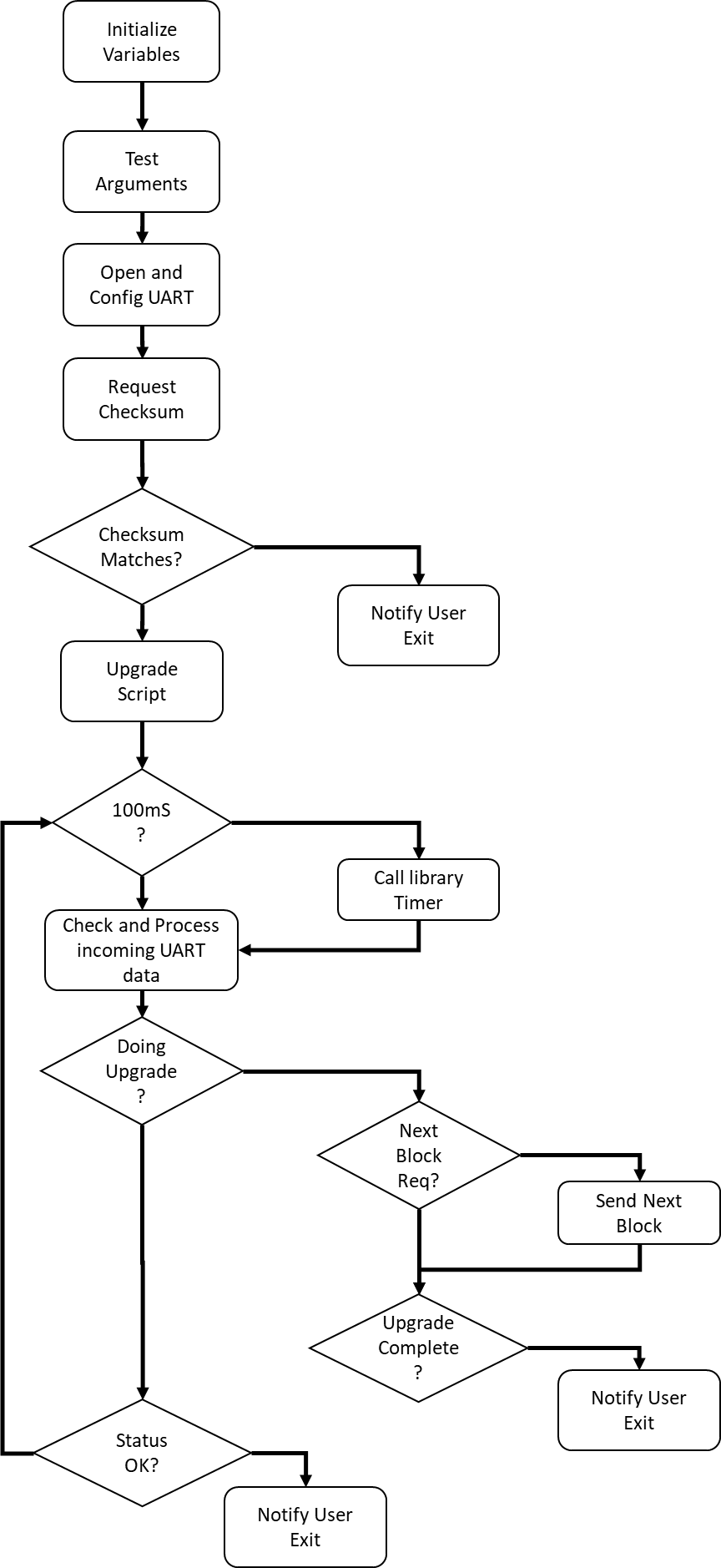
The makefile included does provide for a make clean that will remove the built binary file and directory.

The make file passes the compiler definition of STATIC\_HOOKS during the compile, this #define must be defined to work with the library as the library is compiled to only work with STATIC\_HOOKS. This means that the call back functions are static at link time and not passed in as an argument to a register function.

# The Demonstration program

The demonstration program has a very simple flow and few functions to interoperate with the library a timing mechanism, request next block, and UART RX and TX functions are provided.

The program operates with a super loop polling UART inbound to send data to the library, checking outbound data for the UART TX and if a new block has been requested. The overall flowchart of the demo program is shown below.



The program begins by verifying the number of arguments and printing the usage if not provided the correct number. The first action the program takes requesting the version of the API and prints that to the screen.

Next the UART is opened and configured via methods found in the bbuart.c/h files.

It is desirable to check the CRC of the existing script to verify if an upgrade is necessary or not. This is implemented using a small loop that is broken when the CRC is returned. Should communication with the module fail, this would result in an infinite loop requiring the program to be terminated by a control+C.

The timing control for both loops is provided by checking the current time via the gettimeofday method. This method is affected if the system clock is changed. Other methods that use a monatomic clock were attempted but did not seem to be available on this hardware or libc. The timing is largely addressed in the function testTime and can be modified as desired to provide better timing.

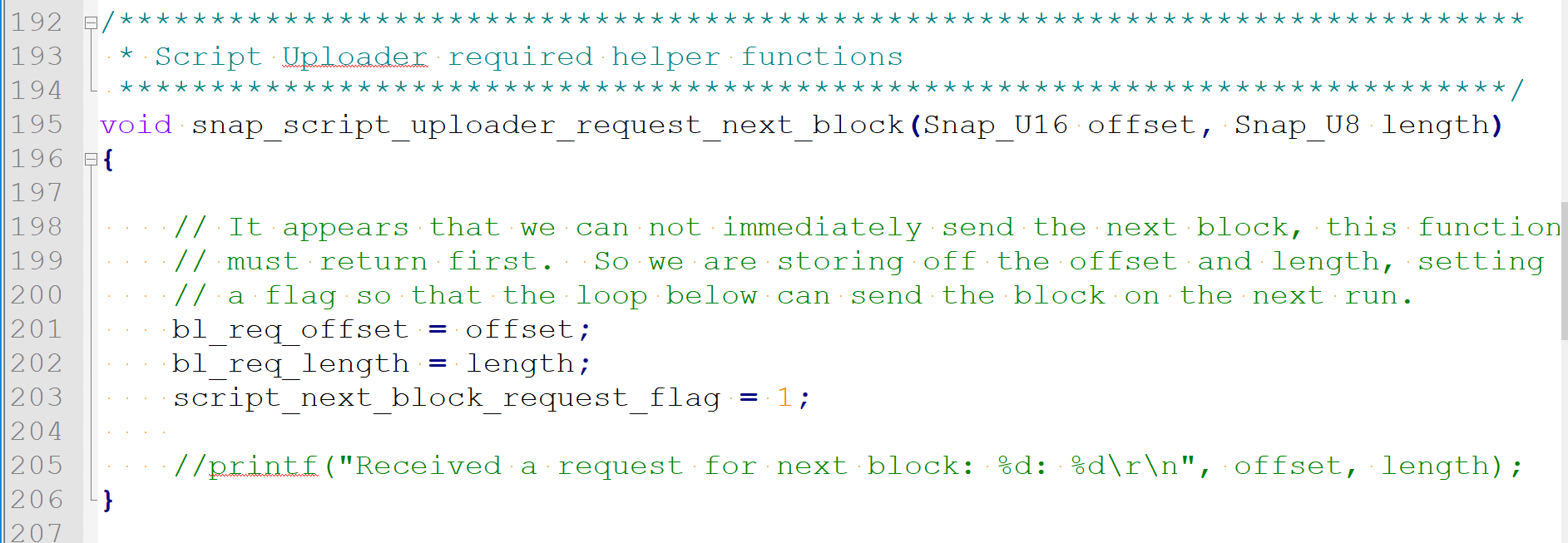
Once the CRC has returned, if it matches the upgrade script, the program exists as shown above.

If an upgrade is required, the upgrade is started and monitored for status. Once completed the system will either output an error or DONE. The valid error conditions are:

|  |  |
| --- | --- |
| **ERROR #** | **Description** |
| 1 | NO\_RESPONSE\_TO\_ERASE\_COMMAND |
| 2 | NO\_RESPONSE\_TO\_WRITE\_DATA\_BLOCK |
| 3 | DATA\_BLOCK\_TOO\_LONG |
| 4 | UNREQUESTED\_DATA\_BLOCK\_RECEIVED |
| 5 | NO\_DATA\_BLOCK\_RECEIVED |
| 6 | INVALID\_OFFSET\_IN\_WRITE\_ACK |
| 7 | NO\_RESPONSE\_REBOOT\_RPC |
| 8 | NO\_RESPONSE\_SCRIPT\_NAME\_RPC |
| 9 | INVALID\_SCRIPT\_NAME |
| 10 | NO\_RESPONSE\_SCRIPT\_CRC\_RPC |
| 11 | INVALID\_SCRIPT\_CRC |
| 12 | INVALID\_STATUS\_CODE |

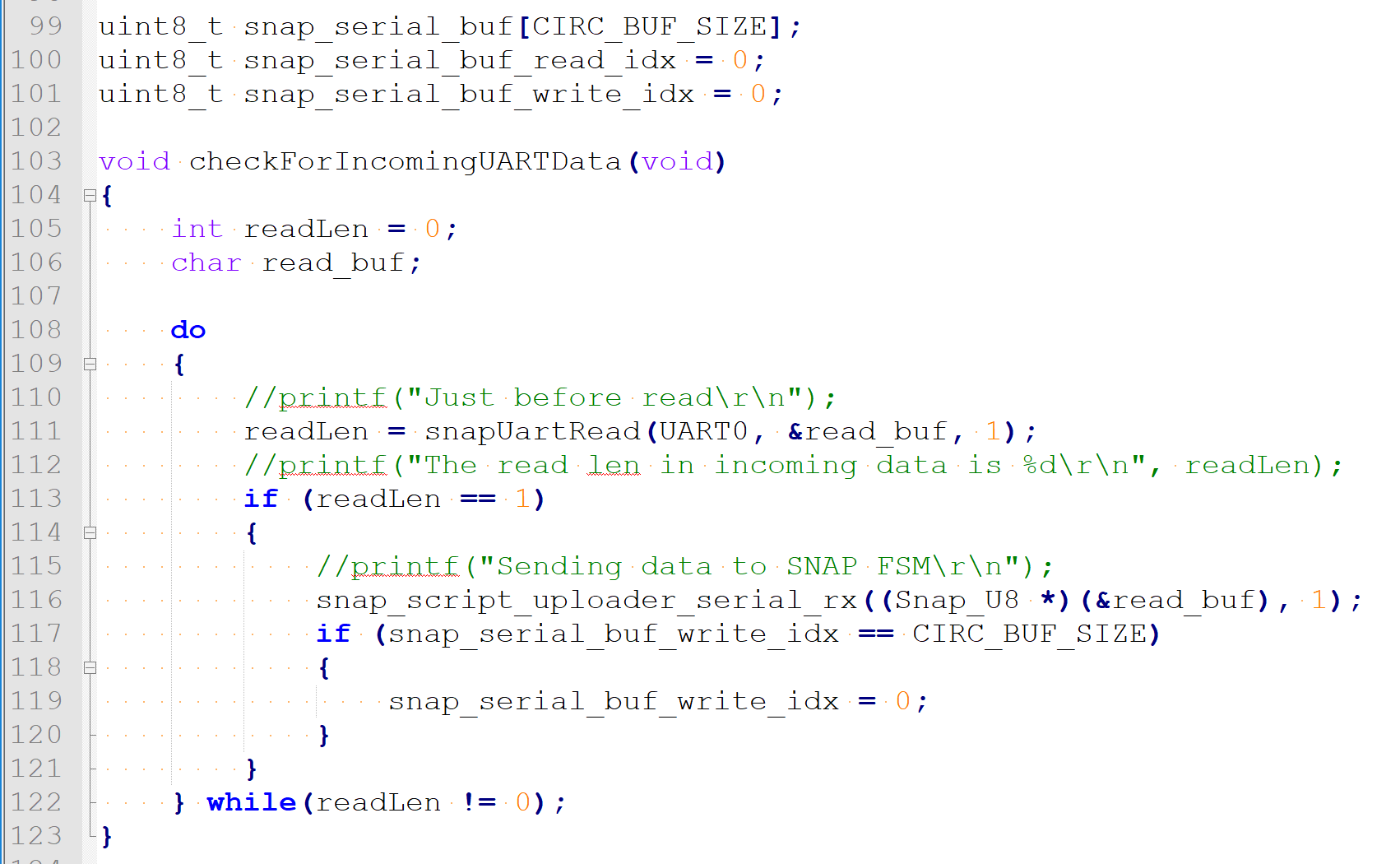
Below are some code excerpts highlighting portions that the end user may wish to change implementation of.

The request next block sets a state variable so that on the next loop iteration the block is read from the binary file using c’s binary file open, seek and read methods.

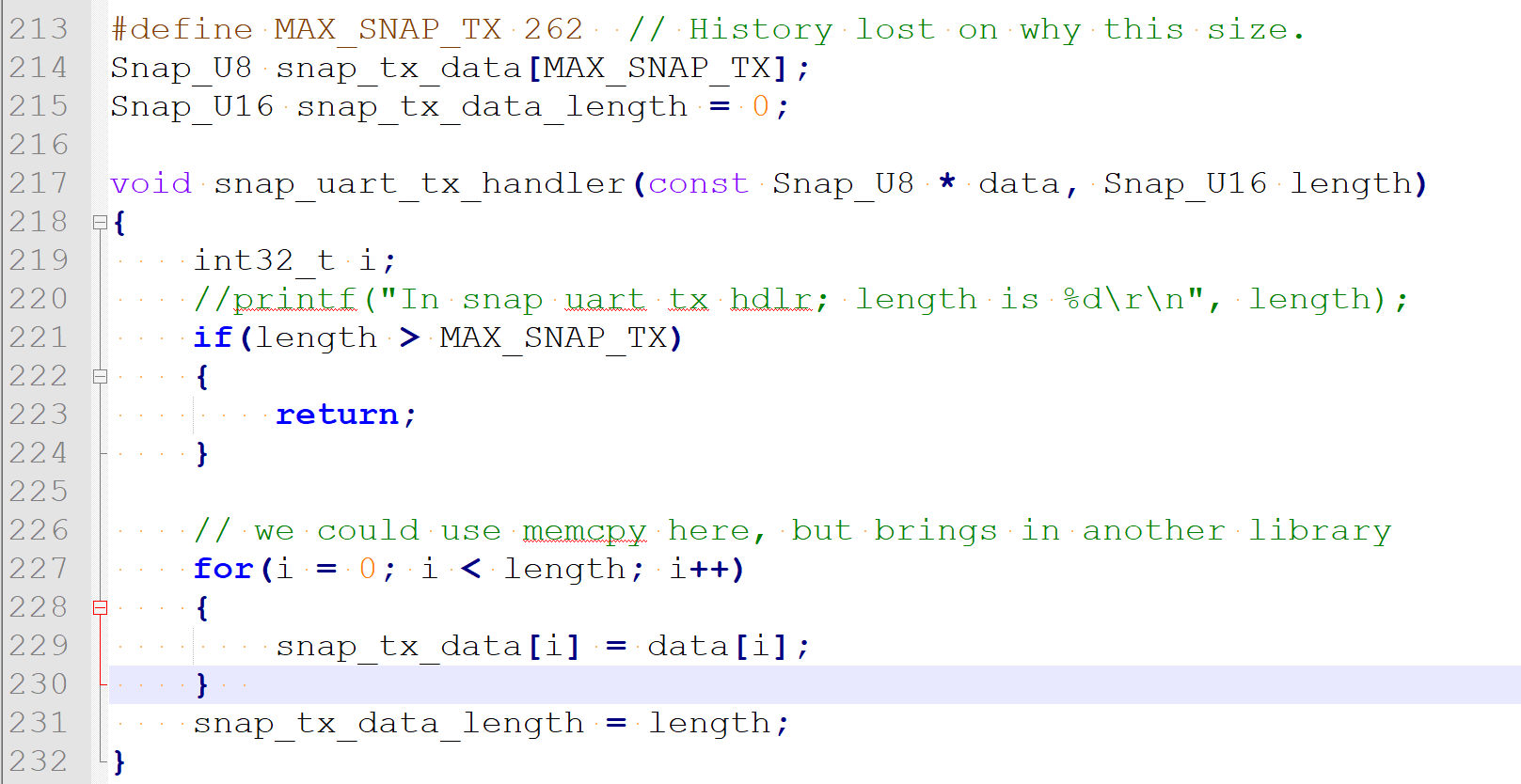


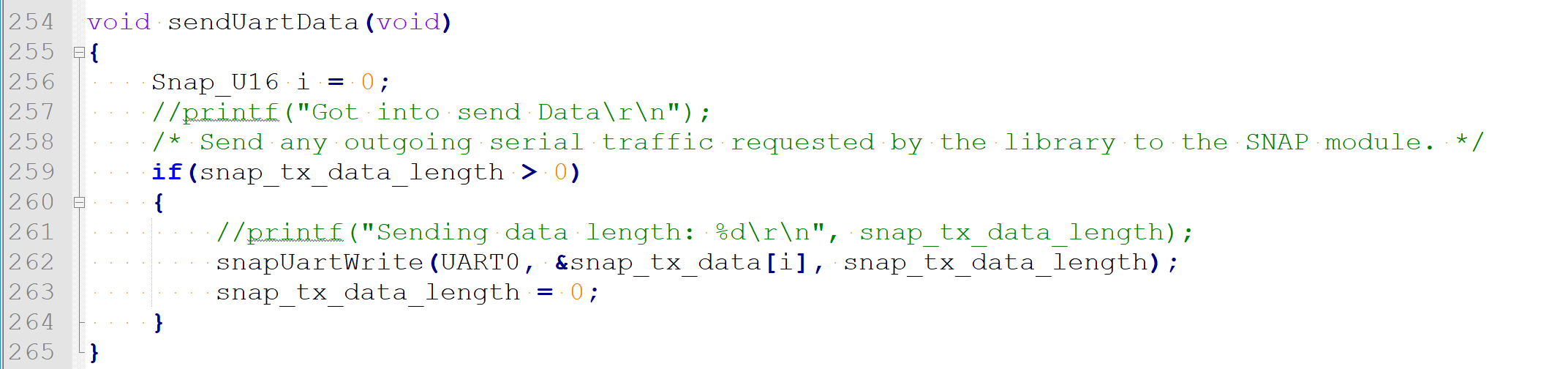
During the read the appropriate method is called in the library to provide the require information for next block or end of data.

The UART is read one character at a time and returns immediately if there are no characters to read but will continue to read as long as data is available. Each character is sent to the library for parsing as it is received. The serial read buffer is not used.



The UART TX handler does use a global buffer and length flag to indicate to the main loop to send the data. The main loop will send all the data to the UART at once.





While not shown the most likely place for modification is within the timing for the 100 millisecond tick. Give that the implementation is system clock dependent.

