



rt.buffer User Manual

2017-10-11
Firmware Version 1.00

Table of Contents

| | | | |
|--|----|---|----|
| 1. Overview..... | 1 | 9. Update Mechanism..... | 24 |
| 1.1. Data Collection..... | 1 | 9.1. update.txt Keywords..... | 24 |
| 1.2. Data Delivery..... | 1 | 10. Configuration Variables..... | 26 |
| 1.3. Configuration and Setup..... | 2 | 11. Config 'lua' : Lua Configuration..... | 27 |
| 1.4. Remote Management and Updates..... | 2 | 12. Config 'c.cell' : Cellular..... | 28 |
| 1.5. Magnet Trigger..... | 3 | 13. Config 'c.site' : Site Deployment..... | 29 |
| 2. Getting Started..... | 4 | 13.1. Suggestions..... | 29 |
| 2.1. Initial Steps..... | 4 | 14. Config 'c.iot' : IoT Parameters..... | 30 |
| 2.2. Required Configuration Values..... | 5 | 15. Config 'c.tls' : Security Settings..... | 32 |
| 2.3. Schedule Strings..... | 7 | 16. Config 'term' : Terminal Controls..... | 33 |
| 2.3.1. Schedule Examples..... | 7 | 17. Config 'arc' : Archive Parameters..... | 34 |
| 2.4. Expanded Values..... | 8 | 18. Config 'alm' : Alarm..... | 35 |
| 3. Using rtbFileAccess..... | 9 | 19. Config 'user' : User IoT settings..... | 36 |
| 3.1. Upgrading Firmware..... | 9 | 20. Information Virtual Lua Table..... | 37 |
| 3.2. Loading and Viewing Applications..... | 11 | 20.1. i.smp.XXXX..... | 37 |
| 3.3. Viewing, Saving, and Editing Files..... | 12 | 20.2. i.app.XXXX..... | 38 |
| 4. Folder Structure..... | 13 | 21. Info 'cell' : Cellular Information..... | 39 |
| 5. Viewing real-time logs..... | 14 | 22. Info 'rt' : rt.buffer Information..... | 41 |
| 6. Terminal Commands..... | 15 | 23. Info 'fw' : Firmware Information..... | 42 |
| 6.1. General Functions..... | 15 | 24. Info 'iot' : IoT Information..... | 43 |
| 6.2. Modem..... | 15 | 25. Info 'pwr' : Power Information..... | 44 |
| 6.3. External Devices..... | 16 | 26. Info 'lua' : Lua Information..... | 45 |
| 6.4. Lua Core..... | 17 | 27. LED Sequences..... | 46 |
| 6.5. NAND Flash..... | 19 | 28. Alphabetical Index..... | 47 |
| 6.6. Internal Debug Commands..... | 21 | | |
| 7. BootLoader Terminal Commands..... | 22 | | |
| 8. User Process..... | 23 | | |
| 8.1. Description..... | 23 | | |
| 8.2. Configuration Options..... | 23 | | |
| 8.3. Stopping the User Process..... | 23 | | |

1. Overview

The rt.buffer is effectively a "platform" that provides fully scriptable operations for collecting, manipulating, calculating, and sending data from a variety of sensor types.

Out of the box, it effectively does nothing. It needs to be taught how to collect data, and told what to do with it. A number of Lua "App" scripts can be stored in the rt.buffer's flash file system, and one of those Apps can be configured to run - giving the rt.buffer its "personality".

The rt.buffer is powered by a sophisticated and efficient real-time-operating system (RTOS) that uses the Lua scripting core to provide user applications and extensions. Power consumption is kept to an absolute minimum, and can allow 5 year run-time from internal LSH-20 style batteries¹.

1.1. Data Collection

Data can be collected from sensors and devices such as:

- Digital pulse inputs (x2)
- RS-232 devices
- RS-485 devices
- ModBus RTU sensors
- Analog (24-bit ADC differential) suitable for pressure transducers etc

Complex second-resolution schedules for data collection can be easily setup using plain-text strings and the Lua scripting.

Alternatively, when connected to devices that "push" serial data, the rt.buffer can be set to wake up from ultra-low power mode to catch the data and store it.

1.2. Data Delivery

Data that has been collected can be retrieved through the USB connection manually, but the real power of the rt.buffer is over the 3G embedded modem. Collected data files can be pushed over FTP or FTP+SSL/TLS² using gzip compression.

¹ See the rt.buffer Hardware Manual for specifications and details

² HTTP and HTTPS to a web-server or node.js server will be available with a firmware update

By default the rt.buffer will push the data into the "/Send" folder of the IoT³ server. However, the choice of server URLs, what data to send, and whether to use gzip compression is all completely programmable using Lua.

Data can be scheduled for delivery, or the Lua App can be programmed to push data when "interesting" data values or sequences appear.

1.3. Configuration and Setup

Initial configuration is normally performed through the USB connection, using Scannex's rtbFileAccess application, or through the rtbAPI.dll (collectively referred to as the "USB-HID"). This provides full visibility of the internal file-structure of the rt.buffer's 128Mbyte flash storage⁴.

Initially, one or more Lua apps must be loaded into the /Lua folder. Only one Lua App will be executed, but each rt.buffer can be deployed with multiple applications (so the App can be chosen immediately).

Individual configuration settings can be made over USB-HID, or a whole configuration file can be read or written with the '/Config/config.txt' file. The config.txt file is a compact human-readable ASCII format that allows for easy cloning or modification of settings.

Commands can be used over USB-HID to find out the internal state of the rt.buffer.

Additionally, the rt.buffer(s) can be remotely controlled by the Internet-Of-Things (IoT) server over the cellular Internet connection.

1.4. Remote Management and Updates

Nearly everything in the rt.buffer can be controlled remotely through the IoT server.

When the rt.buffer connects to the IoT server it will ask for updates. Those updates can include any of the following:

- Firmware upgrade
- Lua App upgrades
- Cellular survey requests
- Diagnostic Dump requests
- Configuration changes
- Executing arbitrary Lua code (e.g. for controlling connected devices)

³ IoT = Internet Of Things server. e.g. FTP or HTTP server.

⁴ Not replacable. The flash is manufactured with chip-on-board for maximum reliability

1.5. Magnet Trigger

The rt.buffer includes a magnet sensor on its side. This starts the "User Process" that is designed to cover the needs of, say, an engineer that needs to get some live data from the rt.buffer in the field.

Once triggered, it will connect to cellular Internet, contact the IoT server, and push a block of information at intervals.

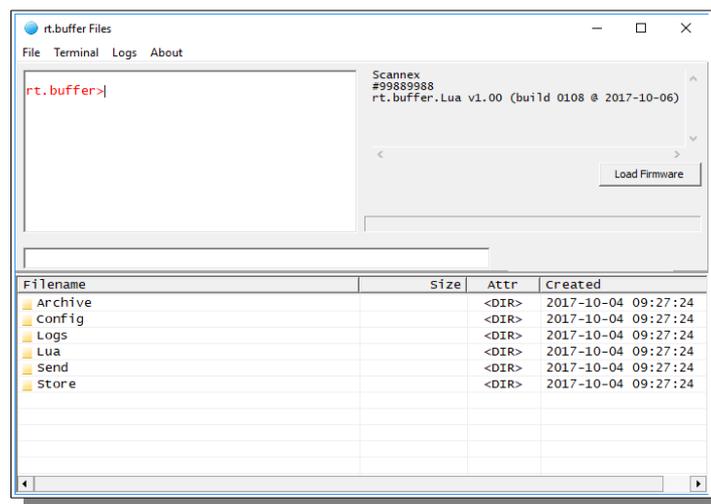
Back-end processes can then analyse the data block and present it as HTML so the engineer can view on their smart phone. In fact, such data could be viewed anywhere in the world (assuming the web service is made available publicly).

The timing, intervals, and specific action of the User Process can be overridden by writing Lua functions in the App.

2. Getting Started

2.1. Initial Steps

- Connect to the rt.buffer Engineer port with the USB cable.
 - No drivers are required, but Windows may show a pop-up from the system tray as it registers the USB-HID device.
- Run the rtbFileAccess.exe application on Windows.
 - This application does not need to be installed, and does not require any DLLs or other libraries.
- Power up the rt.buffer (if not already powered)
- You should see the rt.buffer appear immediately - the serial number and name of the buffer will be shown in the top right window.
- Click on the terminal window and hit the ESC key. You should see a red prompt appear:
 - "**rt.buffer>**" shows the rt.buffer is running its main firmware. You should be able to see the folder structure, and can navigate to folders.
 - "**bootloader>**" shows that the rt.buffer is in the recovery boot loader mode. From this mode you can reboot, run the application, or load new firmware (either bootloader firmware or application firmware).
 - "**ate>**" shows the automated test firmware is running.



- You will need to install a valid SIM card

- There are a set of configuration values that must be set before the rt.buffer can access the Internet and deliver data.
 - You can type the configuration values directly into the Terminal window
 - You can copy from another Windows application and use Ctrl-V to paste the changes into the Terminal window
 - You can edit the /Config/config.txt file

2.2. Required Configuration Values

• `c.lua_app='name'`

- Sets the Lua App name. The 'name' should be just the base-name (i.e. no extension).
- e.g. `c.lua_app='NMEA'`

• `c.site_name='AcmeSite'`

- The name of the site. This is not absolutely required, but is helpful for the scripting and naming of URLs.

• `c.cell_apn='Internet'`

- The APN name required by the SIM contract

• `c.cell_user=''`

- The username for the SIM Internet access (if required)

• `c.cell_pass=''`

- The password for the SIM Internet access (if required)

• `c.cell_pin=''`

- The PIN code for the SIM card (if set)

• `c.iot_url='ftp://user:pass@ftp.scannex.com/data'`

- The base URL for accessing the IoT server
- (Warning: don't use the one that's printed here!! Use your own.)

• `c.iot_job='@23:00'`

- The job schedule for data and updates to the IoT server.

- `c.iot_upd='Update/{c.site_name}'`

- Sets the relative path of the Update directory (i.e. the update.txt file).
- Can be a full URL if you need a completely different server to the c.iot_url base.

● The Lua App you have selected may also require additional configuration settings.

2.3. Schedule Strings

The rt.buffer has a flexible job scheduler that is used throughout the firmware. the rt.buffer has up to twelve jobs that can be used for different purposes (e.g. deliver data; contact update server). Each job can have eight different time slots (e.g. for different times throughout the week).

A schedule is programmed with a string that has the following structure:

- The primary value is the interval (default is 24 hours)
 - 00:00:15 = every 15 seconds
- The @ specifies a time range (default start time is 00:00:00, and default end time is 24:00:00)
 - @23:00 = at (or from) 11 pm
 - @6:00-23:30 = between 6:00am (inclusive) and 11:30pm⁵ (exclusive)
- The # specifies the day bit-field
 - Where: 1 = Sunday, 2 = Monday, 4 = Tuesday, 8 = Wednesday, 16 = Thursday, 32 = Friday, 64 = Saturday
 - #62 = weekdays
 - #65 = weekends
 - #d = weekdays, Mon-Fri
 - #e = weekends, Sat+Sun
 - #a = all days (#127)
- A comma (,) separates the entries

2.3.1. Schedule Examples

- '00:00:30@8-18, 00:05:00'
 - every 30s between 8am and 6pm, and 5 minutes outside those times
- '00:00:30@8-18#2, 00:00:45@8-18#60, 00:05:00'
 - every 30s between 8am and 6pm on Monday; every 45s between 8am and 6pm on Tue-Fri, and 5 minutes otherwise.
- '4@8-18, 12'
 - every 4 hours between 8am and 8pm, and every 12hrs otherwise (00:00:00 & 12:00:00)

⁵ The end time is *exclusive*. So, for example, '2:00@12:00-18:00' will fire at 12:00, 14:00, and 16:00 only. This method allows for clarity when changing the interval, say to 0:00:01.

- '@12:30'
 - an event that occurs each day at 12:30pm

2.4. Expanded Values

When a string supports expansion, e.g. the c.iot_url values, then each pair of curly-braces {..} are evaluated as a Lua value.

e.g.

```
'Data-{c.site_name}-{i.rt_sn}'
```

Will create a string that is built from the Site Name (c.site_name) and Serial Number (i.rt_sn) values.

e.g.

```
'Data-{i.rt_sn * 12345 + 34.56}'
```

Will create a string that is based on a calculation of the serial number (!).

3. Using rtbFileAccess

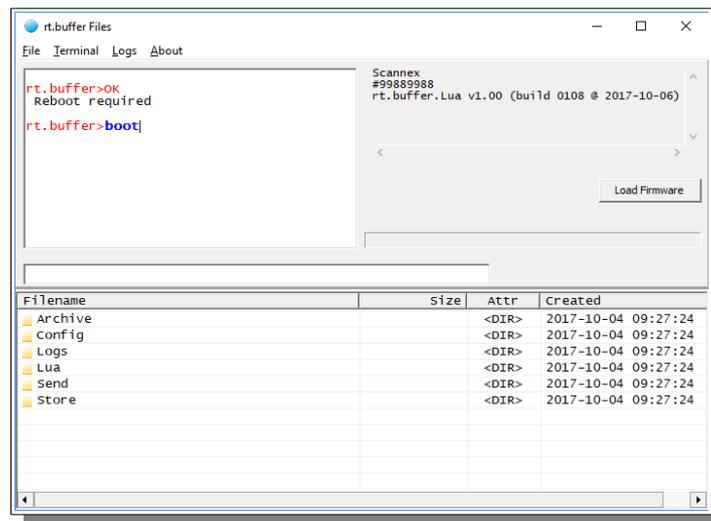
3.1. Upgrading Firmware

Firmware can be upgraded from either the BootLoader or from the main application.

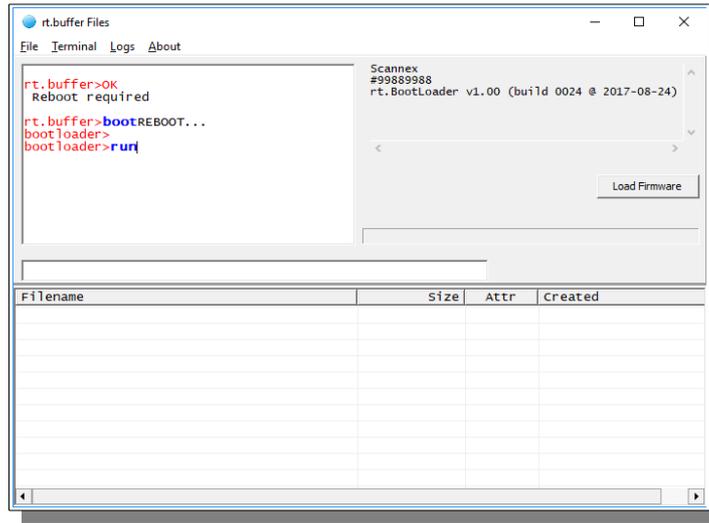
- Changing firmware does not affect the NAND flash storage necessarily. You can upgrade bootloader or main firmware without affecting data.

Using the rtbFileAccess tool, you can use the [Load Firmware] button.

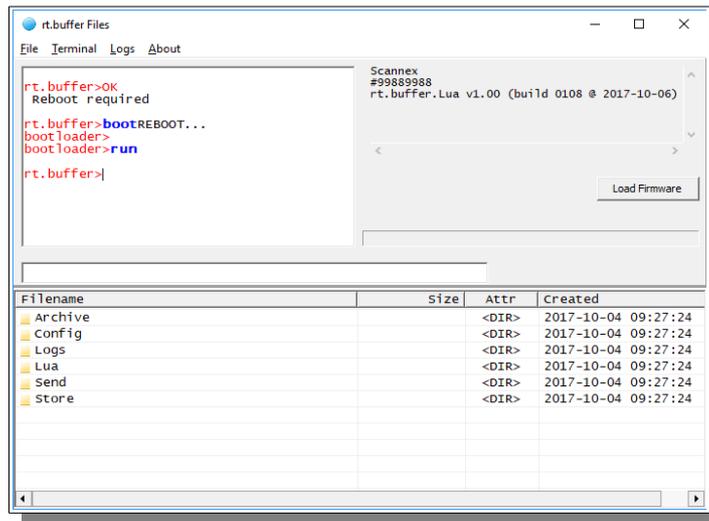
- Verify that you have the right rt.buffer connected by checking its serial number in the top right panel.
- Click the [Load Firmware] button
- Navigate to a .BLF file (Binary Loader Format)
- Select the file, and click "Open"
- In a few seconds the display should show "Reboot required"...
- Type in "boot" [Return] into the terminal window...



- When the "**bootloader>**" prompt appears, type "**run**" [Return]



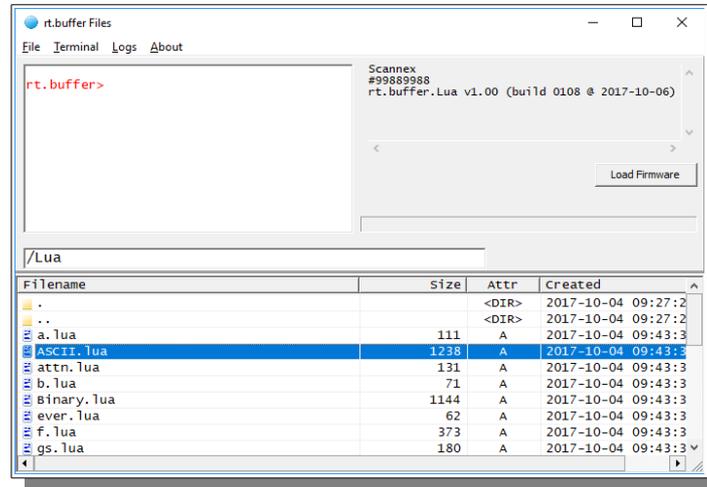
- The new firmware should be running.



• You can check which version of firmware and bootloader are installed by typing "**info**" [Return] in the terminal window.

3.2. Loading and Viewing Applications

- Navigate to the /Lua folder
 - Double click on the yellow folder icons (".." will go up a directory level)
 - Or type in "/Lua" into the path bar and click [Return]
- You should see a list of .lua files



- Install new or updated Lua Apps:
 - Drag-and-drop from a file from a Windows Explorer window onto the file list.

● You cannot drop onto a yellow folder with the tool. You can only drop into the current directory.

- View or edit a Lua App:
 - You can just double click the file to get a convenient file editor/viewer.
- Save a file/files/directory to the PC:
 - Select the items you want to save to the PC
 - Right click and select "Save to PC"
 - Navigate to the required folder on the PC and click [OK].

3.3. Viewing, Saving, and Editing Files

Use the file list to navigate, and double click to allow viewing or editing of any file in the rt.buffer.

The right-mouse click context menu provides options for:

- Deleting files
- Renaming files
- Making directories
- Making empty files

● If the file has binary characters (e.g. NULL) and you hit the [Cancel] or the [Esc] key, the rtbFileAccess tool may tell you that the file has changed - even if you did nothing to it. Just click to lose the changes.

4. Folder Structure

The following directories are used by the firmware (although other directories can be created and used within the Lua scripting):

/Archive

Files that are sent to the IoT server from the /Send folder will be temporarily saved in the Archive folder.

By default, the last 32 files or 4MB are kept in this folder.

/Lua

Folder for all Lua scripts.

/Config

Configuration folder. Config.txt is the primary file.

/Logs

The log files. System.log is the main log file.

When system.log reaches 64k, it is renamed system.log.2. Log files .2, .3, and .4 are kept.

/Store

The folder where incoming data is built up in temporary files.

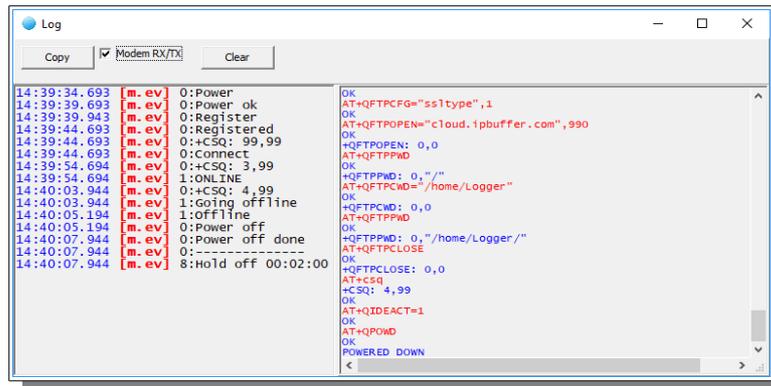
/Send

The folder where outgoing data is saved, ready to be sent over 3G. When it is time to transfer a file, or the /Store file is getting large, the file will be renamed and placed in the /Send folder.

5. Viewing real-time logs

The rtbFileAccess tool can also show what is happening inside the rt.buffer.

- Click the "Logs" menu item
- A new window will appear that shows rt.buffer events in real-time over USB
- Checking the "Modem RX/TX" option will split the panel and also show real-time receive and transmit between the rt.buffer and internal 3G modem.
 - This is helpful if there are carrier or Internet issues.
- Pressing the [Copy] button will put onto the Clipboard:
 - The Terminal window
 - The Event log window
 - The Modem log window (if visible)
 - This data can be pasted into a word-processor or email to send back for support issues.



The screenshot shows a window titled "Log" with a "Copy" button, a checked "Modem RX/TX" checkbox, and a "Clear" button. The window is split into two panes. The left pane displays a list of events with timestamps, status codes, and descriptions. The right pane displays the corresponding AT command responses from the modem.

| Time | Status | Event | Modem Response |
|--------------|--------|---------------------|--------------------------------------|
| 14:39:34.693 | [m.ev] | 0:Power | OK |
| 14:39:39.693 | [m.ev] | 0:Power ok | AT+QFTPCFG="ss1type",1 |
| 14:39:39.943 | [m.ev] | 0:Register | OK |
| 14:39:44.693 | [m.ev] | 0:Registered | AT+QFTPOPEN="cloud.1pbuffer.com",990 |
| 14:39:44.693 | [m.ev] | 0:+CSQ: 99,99 | +QFTPOPEN: 0,0 |
| 14:39:44.693 | [m.ev] | 0:connect | AT+QFTPPWD |
| 14:39:54.694 | [m.ev] | 0:+CSQ: 3,99 | OK |
| 14:39:54.694 | [m.ev] | 1:ONLINE | +QFTPPWD: 0,""/" |
| 14:40:03.944 | [m.ev] | 0:+CSQ: 4,99 | AT+QFTPCMD="/home/Logger" |
| 14:40:03.944 | [m.ev] | 1:Going offline | OK |
| 14:40:03.944 | [m.ev] | 1:offline | +QFTPCMD: 0,0 |
| 14:40:05.194 | [m.ev] | 0:Power off | AT+QFTPPWD |
| 14:40:07.944 | [m.ev] | 0:Power off done | +QFTPPWD: 0,""/home/Logger/" |
| 14:40:07.944 | [m.ev] | 0:----- | AT+QFTPCLOSE |
| 14:40:07.944 | [m.ev] | 8:Hold off 00:02:00 | OK |
| | | | +QFTPCLOSE: 0,0 |
| | | | AT+CSQ |
| | | | +CSQ: 4,99 |
| | | | OK |
| | | | AT+QIDEACT=1 |
| | | | OK |
| | | | AT+QPOWD |
| | | | OK |
| | | | POWERED DOWN |

6. Terminal Commands

These commands apply to "`rt.buffer>`" prompt through `rtbFileAccess`.

6.1. General Functions

info

Show information about the firmware.

boot

reboot

Reboot the `rt.buffer` immediately. If USB is connected, the `rt.buffer` will remain in the Boot Loader for 5 seconds (unless you type a boot loader command).`

time

```
time YYYY-MM-DD HH:MM:SS [TZ]
```

```
YYYY-MM-DD HH:MM:SS:   UTC time  
TZ:                   Optional timezone offset, in minutes
```

Show the current time information, or set and show the time.

temp

Show temperature.

deepsleep

Turn 'off' the `rt.buffer`.

You can only wake up by reconnecting the Engineer serial port or USB for more than 1 second.

6.2. Modem

cho

Clear modem holdoff. If the modem was pausing between attempts, this command will make it finish the pause immediately.

modem

Enter modem loop-back mode. Useful for diagnostics via USB.

6.3. External Devices

adc

Show the ADC readings.

pcr

Pulse Counter Reset.

pcv

Show Pulse Counter Values.

pt[@]d [STOP]

| | |
|-------|--|
| @: | Include the '@' symbol to pass the characters directly to the terminal ⁶ . Without the '@' then control characters are escaped in Javascript/Lua/C style ⁷ . |
| STOP: | Optional Lua expression to alter the stop sequence ⁸ . Normally the [ESC] key quits the pass through mode. |

Enter pass-through mode for the Device COM port.

```
ptd
pt@d
ptd '!'
ptd '\x1b'
```

pt[@]e [STOP]

(as for ptd above)

Enter pass-through mode for the Engineer COM port.

⁶ Use the 'pt@d' format when using software to communicate directly through rtbAPI.dll or USB-HID control.

⁷ e.g. a NULL will be escaped in ASCII as '\x00', an ACK character as '\x06' etc. See https://en.wikipedia.org/wiki/Escape_sequences_in_C - but '\r', '\n', and '\t' are sent as real characters CR, LF, TAB.

⁸ Although the STOP can be a string sequence, the terminal is not fully buffered. If you are using a keyboard, then use a single character!

6.4. Lua Core

`lboot`

Reboot Lua.

`lm`

`lm0`

Show Lua memory stats.

"lm0" will show and reset the statistics.

`lgc`

`lgc0`

Garbage collect Lua memory and show stats.

"lgc0" will show and reset the statistics.

`lv LIST`

`?LIST`

1.00

LIST: comma separated list of values

Show Lua value tree(s), or specific values.

```
lv
lv i
lv i.smp,i.app,c.iot
?c.iot
```

`lx CMD`

`/CMD`

1.00

`=CMD9`

1.00

CMD: Lua command

Execute Lua expression, and show the Lua stack.

```
lx iot.go()
lx return iot.flg()
/return rt.ms()
```

⁹ Short-cut to "lx return CMD"

c.KEY = VALUE

| | |
|--------|------------------------|
| KEY: | Configuration key name |
| VALUE: | Value in Lua format. |

Set a configuration value.

```
c.iot_url='ftp://ftp.scannex.com/Path/To/Success'
```

Can also be used to wipe out a setting, or a tree of settings, using the Lua shorthand "_" for NIL:

```
c.iot_url=_  
c.iot=_
```

lf FILE

| | |
|-------|-----------------|
| FILE: | Filename to run |
|-------|-----------------|

Execute a Lua file.

If FILE does not include a '/' character, then '/Lua/' is assumed.

If FILE does not include a '.' character, then '.lua' is assumed.

6.5. NAND Flash

ninfo

Show information about the flash drive.

nfcheck

Check the flash disk.

nfformat

Erase EVERYTHING.

● CAUTION: There is no way to undo this command! You WILL lose everything in the rt.buffer

nftype FILE

FILE: Full filename

Shows the contents of a file.

nfdir PATH

PATH: Path of the folder to view.

Show the folder listing.

● The path separator character is forward-slash '/' (Like UNIX/Linux, not Windows!)

nfdel FILE

FILE: Full path and filename

Delete a file from flash disk.

nfmkdir PATH

PATH: Full path

Make a folder, and intermediaries.

```
nfmkdir /Temp/Dir/Here
```

nfrmdir PATH

PATH: Full path

Remove a folder. Will fail if the folder is not empty

nfwipedir PATH

PATH: Full path

Wipe a folder, including all its contents.

 **CAUTION:** There is no way to undo this!
May take a long while to execute, especially if there are lots of files.

nfrename SOURCE TARGET

SOURCE: Source path + filename
TARGET: Target filename (no path!)

Rename a file.

nfmove SOURCE TARGET

SOURCE: Source path + filename
TARGET: Target path + filename

Moves a file or folder.

nfwrite FILE

FILE: Path and filename

Write to a file. (Used internally by the rtbFileAccess and rtbAPI tools)

6.6. Internal Debug Commands

- These commands are not guaranteed to remain, and the format may change.
They are primarily for Scannex development and testing.

pmm

Show power manager modes.

wde

wde0

Show watchdog entries.

'wde0' will reset the maximum counters.

led

Show LED status.

ninv

Show the sense inputs on the COM ports.

mv

Show the millivolt readings for power, etc.

work-

work+

Decrement or increment the counter for hard-work.

If the counter is >0 then the CPU runs at 96MHz and will not enter low power mode.

7. BootLoader Terminal Commands

These USB-HID terminal commands apply to the "`bootloader>`" prompt.

`run`

Leave the bootloader and run the install firmware.

`erasefw`

Erase the current application firmware. Only the bootloader will remain.

This will not affect any stored NAND flash data.

`eraseall`

Erase the whole NAND flash - erasing all Lua Apps, settings, and stored data.

The firmware and bootloader are not affected.

- You will lose everything that was stored. Each block of NAND flash is physically erased. This is useful if the data was sensitive and the device needs to be shipped.

`info`

Show information about the rt.buffer, bootloader, and application firmware.

`boot`

`reboot`

Reboot the rt.buffer.

`deepsleep`

Turn 'off' the rt.buffer.

You can only wake up by reconnecting the Engineer serial port or USB for more than 1 second.

8. User Process

8.1. Description

The user process is typically triggered by the magnet.

The rt.buffer will connect to the IoT server and repeatedly send the config and info sets. This should enable a back-end process to present the data to a live web-page, so any engineer can view the current information on a phone, tablet or PC.

The process will pause, by default 15s, and send again. This will continue for a default time of 10 minutes. The settings `c.user_ins` & `c.user_onm` override the values.

8.2. Configuration Options

The process will post the data to a file named `"user.{i.rt_sn}.txt.gz"` in the folder specified by `c.iot_url + c.iot_user`.

If the `c.iot_user` value is not present, then `"Update/{c.site_name}"` is used (same logic as the `c.iot_upd` value).

The function `"doUser"` will be called on each loop, allowing the Lua App to include additional values (like live ADC readings) in the information tree.

8.3. Stopping the User Process

The back-end process can abort the user mechanism by leaving a file named:

```
"user.{i.rt_sn}.stop"
```

When the rt.buffer detects this file, it deletes it and quits the User Process.

9. Update Mechanism

The update mechanism works by pulling an 'update.txt' file from the IoT server, and parsing the lines of the ASCII file.

When completed, the "update.txt" file will be renamed on the server as "update.YYYYMMDDHHMMSS.txt" (using the rt.buffer's UTC time that the update was completed).

The commands and available options are listed below.

9.1. update.txt Keywords

- If the `c.iot_gz=0`, then the following transfers will not use gzip compression, and will not use the `.gz` extension.

cellinfo

Perform a cellular survey and post the results to the IoT server.

The file will be named "cellinfo.{i.rt_sn}.txt.gz"

diag

Post a diagnostics dump file onto the IoT server.

The diag file will be named "diag.{i.rt_sn}.txt.gz"

lx COMMAND

COMMAND: Lua expression

Execute a Lua command. This is syntactically the same as the USB command "lx". However, the results of the Lua command will be thrown away.

Complex expressions can be executed, with a line length up to 128 characters.

```
lx job.utc(os.time()+6000, function() iot.go('user') end)
```

If an App has utility functions, for example to make configuration changes to the connected device, you can call these too. They will be executed from the context of the modem task.

```
lx CallMyFunction(10.34, 34.56)
```

getfile SOURCE, TARGET

| | |
|---------|---|
| SOURCE: | The full path and name of the file on the rt.buffer |
| TARGET: | The (path and) filename of where to save the file on the IoT server |

Post a file from the flash file system onto the IoT server.

.gz will be added to the target filename, if appropriate.

```
getfile /Logs/system.log,/temp/system.log
```

setfile SOURCE, TARGET

| | |
|---------|--|
| SOURCE: | Source filename to ask from the IoT server |
| TARGET: | The full path of where to save the file on the rt.buffer |

Pull a file from the IoT server and save into the flash file system.

● NOTE: The source file cannot be gzip or zlib compressed. The rt.buffer does not have the RAM to decompress data.

```
setfile /temp/config.mine.txt,/Config/config.txt
```

● WARNING! If updating the /Config/config.txt file be **very** careful you do not change the IoT settings, or break the cellular settings... otherwise you could lose control of the rt.buffer!

NAME.blf

| | |
|-------|---|
| NAME: | Filename (and path) on the IoT server to pull |
|-------|---|

Pulls a firmware image from the IoT server and burns.

```
../rt.buffer.Lua.1.02.blf
```

c.KEY = VALUE

1.00

| | |
|--------|------------------|
| c.KEY: | Config parameter |
| VALUE: | Config value |

Make a configuration change

10. Configuration Variables

The rt.buffer stores the configuration variables inside an efficient C++ "key=value" structure that has about 6k of space.

The contents of this store are saved and read from the file /Config/config.txt

However, to make life much easier inside Lua, the values can be queried and modified directly as a Lua global table, called 'c'.

Within the C++ structure, the keys are organised in ASCII order, with the underscore "_" character representing the branches of a tree. This method enables simple 'wildcard' style erasing and querying of sets of values, and is also compatible with Lua's syntax.

```
c.my_value = 123
c.my_other = 'A string'
c.my_more = 'A complex \r\n string\twith\x00controlcodes'10
c.mynot = 'Not the same tree'

c.my_other=_ -- erases just one key-value
c.my=_ -- erases all c.my* key-values. c.mynot remains
```

A handful of configuration parameters are fixed, and used within the firmware. The Lua App can choose to make use of additional configuration parameters as needed.

The following sections list the firmware-fixed configuration settings.

¹⁰ See https://en.wikipedia.org/wiki/Escape_sequences_in_C for examples of escaped characters. e.g. '\t' = TAB, '\r' = CR, '\x00' = NULL, etc

11. Config 'lua' : Lua Configuration

```
c.lua_app = ''
```

The app name to run.

- This configuration value is REQUIRED for the rt.buffer to work! And the corresponding .lua app must already be in the /Lua directory.

```
c.lua_app = 'Test' -- uses /Lua/Test.lua
```

12. Config 'c.cell' : Cellular

```
c.cell_pin = ''
```

(optional) PIN number for the SIM card

```
c.cell_apn = 'Internet'
```

APN name for the Internet connection

```
c.cell_user = ''
```

(optional) Username for the Internet connection

```
c.cell_pass = ''
```

(optional) Password for the Internet connection

```
c.cell_mto = 15
```

(optional) Maximum time online, minutes.

```
c.cell_hos = 2
```

(optional) hold off success time, in minutes.

```
c.cell_hof = 10
```

(optional) hold off failure time, in minutes.

13. Config 'c.site' : Site Deployment

```
c.site_name = ''
```

(optional) rt.buffer Site Name.

Used by the smp library for naming the files.

13.1. Suggestions

```
c.site_loc = ''
```

Site location details.

14. Config 'c.iot' : IoT Parameters

```
c.iot_url = ''
```

(required) Base URL for IOT.

● Required!

This uses Lua expansions.

```
c.iot_url = 'ftp://mp:pm@ftp.scannex.com/Upload/{c.site_name}-  
{i.rt_sn}'
```

```
c.iot_job = '@23:00'
```

(optional) Job string for Data delivery (and Update).

● (Developers) Sets both onJob11 for Update, and onJob12 for Data delivery, unless c.iot_ujob is set – in which case c.iot_job does just data.

```
c.iot_ujob = _
```

(optional) Update job string. If not present, then c.iot_job sets both update and data job strings.

```
c.iot_var = 3600
```

Server variance (seconds).

Jobs that are scheduled can be skewed by a combination of c.iot_var and the serial number of the rt.buffer. This ensures that your server does not get 'hit' by many rt.buffer all at the same time, but are spread over time.

```
c.iot_ntp = 'time.apple.com'
```

(optional) NTP server name.

```
c.iot_tfr = 1
```

(optional) Use Temp File and Rename for FTP.

Set to 0 to disable this method and send the file directly.

- c.iot_tfr=1 requires the user FTP account support renaming. However, this approach is **STRONGLY** recommended as it ensures files are completely transferred. Just make sure your back-end processes ignore "*.tmp" files¹¹.

```
c.iot_gz = 1
```

(optional) Use GZIP compression and naming.

```
c.iot_upd = 'Update/{c.site_name}'
```

(optional) Update URL or path for the update mechanism, relative to c.iot_url (although can be an absolute URL too!)

Uses Lua expansions - see rt.exp.

```
c.iot_user = ''
```

(optional) User process URL or path.

Uses Lua expansions.

```
c.iot_data = ''
```

(optional) Data process URL or path.

Uses Lua expansions.

¹¹It is also a good idea to purge very old *.tmp files, e.g. over 1 month old.

15. Config 'c.tls' : Security Settings

`c.tls_csc = 0xffff`

1.00

(optional) The modem cipher suite code for IoT connections¹².

- 0X003D¹³ = TLS_RSA_WITH_AES_256_CBC_SHA256
- **0X0035 = TLS_RSA_WITH_AES_256_CBC_SHA1**
- **0X002F = TLS_RSA_WITH_AES_128_CBC_SHA1**
- 0X0005 = TLS_RSA_WITH_RC4_128_SHA1
- 0X0004 = TLS_RSA_WITH_RC4_128_MD5
- 0X000A = TLS_RSA_WITH_3DES_EDE_CBC_SHA1
- 0XFFFF = Support all ciphersuites above (default)

`c.tls_ver = 3`

1.00

(optional) TLS Version

- 0 = SSLv3 (not recommended)
- 1 = TLSv1.0 (not recommended)
- 2 = TLSv1.1
- **3 = TLSv1.2**

`c.tls_ilt = 1`

1.00

(optional) Ignore Local Time¹⁴

- 0 = Care about time checks for certificates
- **1 = Ignore time checks**

¹² Recommended either 0x0035 or 0x002f as a balance between security and server compatibility.

¹³ This setting is only possible when `c.tls_ver=3`
(AES-256/SHA-256 is only available in TLSv1.2)

¹⁴ Checking the time against the certificate requires the local rt.buffer time be valid too!

16. Config 'term' : Terminal Controls

```
c.term_pass = ''
```

(optional) Terminal password.

- Do NOT forget this password - there is no backdoor.
(Though it is possible, but complex, to erase the whole rt.buffer)

```
c.term_epw = 1
```

1.00

(optional) Whether Engineer Serial Port needs a password.

If there is a dedicated MCU connected to the Engineer Serial Port, then this can be set to 0 so that no password is required (but it will be required from the USB still if c.term_pass is set).

17. Config 'arc' : Archive Parameters

The /Archive/ folder saves copies of the files that are sent. When the file count is exceeded, or the total bytes is exceeded, the oldest files are removed from the folder.

- The folder is 'flat' - any files that do not begin with '/Send/' are mapped so that the '/' characters are replaced by '^' symbols.
e.g. '/MyFolder/myfile.txt' is archived as
'/Archive/^MyFolder^myfile.txt'

If you create sub-folders within /Archive/ then these folders will not be scanned, nor curated (i.e. they will stay there and not be pruned).

```
c.arc_fc = 32
```

1.00

Maximum number of files to keep in the /Archive/ directory.

```
c.arc_kb = 4096
```

1.00

Maximum number of kilo-bytes to keep in the /Archive/ directory.

18. Config 'alm' : Alarm

```
c.alm_txt01 = ''  
c.alm_txt02 = '' ...
```

Text names for the alarms 01-32.

● (Developers) Can be retrieved in Lua with alm.txt(...) function.

19. Config 'user' : User IoT settings

See section 8 - the user process is typically triggered by the magnet.

```
c.user_onm = 2
```

The online time, in minutes, for the default User mode (by default: how many minutes to keep pushing the diagnostics data).

```
c.user_ins = 15
```

The interval, in seconds, for the default User mode (by default: how many seconds to wait after sending the diagnostics data before sending again).

20. Information Virtual Lua Table

Like the Configuration table, the global table "i" provides a window into some internal C++ values (without consuming Lua memory).

However, there are two 'real' Lua tables available too - that can be used for passing back application-specific information.

20.1. *i.smp.XXXX*

This is a real Lua table that is used for the activities of the Lua App's Sample (smp) library. By default, any record that is saved by the Lua App will update these info entries.

These parameters are only updated when the default smp table (ft) is used. When other tables are used - e.g. when handling multiple sample files - these info variables are not updated.

`i.smp.utc`

The UTC time that smp.save was last called

`i.smp.txt`

The last text that was written with smp.save

`i.smp.cnt`

The number of times that smp.save has been called.

This counter is reset if the rt.buffer is cold-booted, watchdog restarted, or Lua reboots.

20.2. *i.app.xxxx*

- For developers.

This is also a real Lua table that can be used by the application to hand back information through the USB console, or through the diagnostic dumps, etc.

The "i.app" table can contain additional tables within it, and these will be handled correctly.

```
-- within App
i.app.mine = {} -- blank table

-- later in the App
i.app.mine.v = 123
i.app.mine.too = 'Hello'
```

```
-- Within the USB terminal or via Update.txt:
lv i.app.mine
```

21. Info 'cell' : Cellular Information

`i.cell_lsq`

Last signal quality value.

`i.cell_lto`

Last time online. Start time in UTC format.

`i.cell_ldo`

Last duration online, seconds.

`i.cell_csq`

Current signal quality value (if online).

`i.cell_cto`

Current time online. Start time in UTC format.

`i.cell_cdo`

Current duration online, seconds.

`i.cell pha`

The current, or last, phase of the cellular activity. (If `i.cell_sta` is 'off' then this is the last phase).

- `_ / NIL` = idle
- `'update'` = update process
- `'time'` = time sync
- `'pass'` = pass thru
- `'term'` = Terminal
- `'user'` = User Process
- `'data'` = Data Upload
- `'loop'` = Loopback control
- `'hold'` = Hold-off

`i.cell_pwr`

Total number of seconds the modem has been powered.

i.cell_sta

String containing the state of the modem:

- 'off' = powered off
- 'pwr' = powering on
- 'reg' = registered
- 'online' = PDP context
- 'fail' = serious error
- '?' = unknown error

22. Info 'rt' : rt.buffer Information

`i.rt_sn`

Serial number string

`i.rt_cbf`

Number of configuration store bytes free.

`i.rt_utc`

The current time.

23. Info 'fw' : Firmware Information

`i.fw_ver`

Firmware version

`i.fw_date`

Firmware date

`i.fw_desc`

Firmware name

`i.fw_blv`

BootLoader version

24. Info 'iot' : IoT Information

`i.iot_var`

The actual number of seconds this rt.buffer will have variance-enabled schedules delayed.

25. Info 'pwr' : Power Information

`i.pwr_alv`

Seconds alive.

`i.pwr_ccb`

Battery capacity, in C.

`i.pwr_ccu`

Estimated charge used

`i.pwr_pct`

1.00

Estimated percentage power remaining (0-99%)

`i.pwr_slp`

Number of seconds spent asleep (in ultra-low power mode).

`i.pwr_tob`

Number of seconds running on battery.

26. Info 'lua' : Lua Information

i.lua_run

The seconds-alive value when the Lua App was started.

Zero means there is no Lua loop running - either onLoop has quit, or has not been defined.

27. LED Sequences

The single, ultra-bright, amber LED gives an indication of the state of the rt.buffer. LED sequences consist of pairs of flashes.

● CAUTION: Do not stare directly into the LED - it is very bright (especially if the top lid is off)!

In the list below, e.g. "12" = one flash + pause + two flashes

- **Status**
 - 11 = Alive (every 10 seconds)
 - If the rt.buffer is in ultra-low power mode the LED will be very dim, otherwise you'll see the '11' sequence but at normal brightness.
 - 12 = Data has been stored
- **Engineer Port**
 - 22 = Engineer USB connected
 - 23 = Engineer USB in use
 - 25 = Magnet triggered
- **Modem**
 - 31 = Modem powering up
 - 32 = Modem registered
 - 33 = Modem online
- **Boot Loader**
 - 41 = Boot Loader active
 - 42 = USB running
 - 44 = Upgrading firmware
 - 45 = No Application BLF
- **Faults**
 - 54 = Debug
 - 55 = Fault

28. Alphabetical Index

Alphabetical Index

| | | |
|---------------------------------|---------------------------------|--------------------------|
| Bootloader Commands..... | c.site_name.....29 | nfinfo.....19 |
| boot.....22 | c.term_epw.....33 | nfmkdir.....19 |
| deepsleep.....22 | c.term_pass.....33 | nfmmove.....20 |
| eraseall.....22 | c.tls_csc.....32 | nfrename.....20 |
| erasefw.....22 | c.tls_ilt.....32 | nfrmdir.....20 |
| info.....22 | c.tls_ver.....32 | nftype.....19 |
| reboot.....22 | c.user_ins.....36 | nfwipedir.....20 |
| run.....22 | c.user_onm.....36 | nfwrite.....20 |
| Config..... | LED Sequences.....46 | ninv.....21 |
| c.alm_txt##.....35 | Terminal Commands.....15 | pcr.....16 |
| c.cell_apn.....28 | adc.....16 | pcv.....16 |
| c.cell_hof.....28 | boot.....15 | pmm.....21 |
| c.cell_hos.....28 | c.KEY = VALUE.....18 | pt (Pass Through).....16 |
| c.cell_mto.....28 | cho.....15 | reboot.....15 |
| c.cell_pass.....28 | deepsleep.....15 | temp.....15 |
| c.cell_pin.....28 | info.....15 | time.....15 |
| c.cell_user.....28 | lboot.....17 | wde.....21 |
| c.iot_data.....31 | led.....21 | work.....21 |
| c.iot_gz.....31 | lf FILE.....18 | ?LIST.....17 |
| c.iot_job.....30 | lgc / lgc0.....17 | /CMD.....17 |
| c.iot_ntp.....30 | lm / lm0.....17 | =CMD.....17 |
| c.iot_tfr.....31 | lv LIST.....17 | update.txt..... |
| c.iot_ujob.....30 | lx CMD.....17 | c.KEY = VALUE.....25 |
| c.iot_upd.....31 | modem.....15 | cellinfo.....24 |
| c.iot_url.....30 | mv.....21 | diag.....24 |
| c.iot_user.....31 | nfcheck.....19 | Firmware.....25 |
| c.iot_var.....30 | nfdel.....19 | getfile.....25 |
| c.lua_app.....27 | nfdir.....19 | lx COMMAND.....24 |
| c.site_loc.....29 | nfformat.....19 | setfile.....25 |