FSServer

This section contains the test cases and the results for FSServer.

Daemonization

The following integration test have been conducted.

Action	Expected	Result
Open terminal		
Run `./fsdaemon start`	the prompt becomes immediately ready for input	success
Run `ps axo comm,ppid | grep fsdaemon`	returns `fsdaemon 1` to indicate that there is a running process with the name fsdaemon whos parent process is 1.	success

Custom logfile

The following integration test have been conducted.

Action	Expected	Result
Open terminal		
Run `du -sk test.log`	Try to determine the size of test.log. It will fail as there is no such file.	success
Run `./fsdaemon start -- --logfile=`pwd`/test.log`	Runs the server, touches the logfile and writes initial log messages.	success
Run `du -sk test.log`	Succeeds with a size greater than 1	success

Multiple tcp clients, lock mechanism, token timeout, no-response and broadcast

The following integration test have been conducted.

Action	Feature to test	Expected	Result
Open up three terminals Run `./fsserver` in terminal 1 Run `fsclient -i` in terminal 2 and 3	Multiple tcp clients	The server will log that both clients have connected	success
Run `lock` in terminal 2	lock	Lock will succeed	success
Run `lock` in terminal 3	lock	Lock will fail	success
Run `unlock` in terminal 2	lock	Unlock will succeed	success
Run `lock` in terminal 3	lock	Lock will succeed	success
Run `lock` in terminal 2	lock	Lock will fail	success

Wait for server to timeout token Run `flush_stdout` in terminal 2 and 3	Token timeout, broadcast	Will print a “server_unlocked” broadcast message and a “must lock server” response	success
Run `lock` in terminal 2 Run `health_status` Run `health_status --no-response`	no-response	First health_status will return the health data, second will just return an OK	success

Command parsing

Unit tests for the CommandParser class have been implemented in “tests/lib/command_parser_test.rb”. The unit tests verifies the following behaviour:

Description	**Request**	**Expected id, token, command and options**	**Result**
Should parse lock without token	{"id":"1", "data":"lock"}	“1”, nil, Commands::Lock	success
Should parse with token	{"id":"1", "token":"0123456789abcdef", "data":"reset"}	“1”, “0123456789abcdef”, Commands::Reset	success
Should not parse unknown commands	{"id":"1", "token":"0123456789abcdef", "data":"blast_venus"}	“1”, “0123456789abcdef”, Commands::Unknown	success
Should not parse with wrong number of arguments	{"id":"1", "token":"0123456789abcdef", "data":"reset invalid_argument"}	“1”, “0123456789abcdef”, Commands::WrongNumberOfArguments	success
Should extract options	{"id":"1", "token":"0123456789abcdef", "data":"reset --timeout=20 --no-response"}	“1”, “0123456789abcdef”, Commands::Reset, {"timeout"=>"20", "no-response"=>true}	success

Sequentially executed satellite commands

To verify that the satellite requests is executed sequentially the following integration test have been conducted.

Action	**Expected**	**Result**
Run fsserver		
Lock the server and execute a health_status command from two fsclients with the same token like this:	In the server log, we should see that both requests has been accepted simultaneously but that the writes to the datalink are executed sequentially	success

\`TOKEN=fsclient -d lock\` \`fsclient --token=$TOKEN health_status& fsclient --token=$TOKEN health_status&\`		

Command validation

Common validations are implemented in the AbstractCommand class. The validations use methods implemented in “lib/ext/string.rb” and the unit tests are implemented in “test/lib/ext/string_test.rb”

The most common validations are that an argument is of a certain length and that it is addressable. The unit tests verifies that following values are or are not addressable:

Value	**Expected**	**Result**
0x01234567	TRUE	success
0x89ABCDEF	TRUE	success
0x89abcdef	TRUE	success
0xff	TRUE	success
0x01234556789abcdef	FALSE	success
0x100000000	FALSE	success
-0x1	FALSE	success
0xfg	FALSE	success
0x0.2	FALSE	success
0x0,2	FALSE	success
0	TRUE	success
4294967295	TRUE	success
4294967296	FALSE	success
-1	FALSE	success
a	FALSE	success
0.1	FALSE	success
0,1	FALSE	success
address	FALSE	success

Spaced hex

When sending a value to the satellite it expects it to have a certain data length. To ensure that the decimal value 2 will be send to the satellite as 4 bytes the method “spaced_hex” have been implemented in “lib/ext/string.rb”. The following values have been tested:

Bytes	**Value**	**Expected**	**Result**
8	0x00000000	00 00 00 00	success
8	0x0	00 00 00 00	success
8	0xff	00 00 00 ff	success
8	0xff000000	ff 00 00 00	success
8	0xffeeddcc	ff ee dd cc	success

8	0xffffffff	ff ff ff ff	success
8	0x123456789	NotAddressableError	success
8	-0x1	NotAddressableError	success
8	0	00 00 00 00	success
8	255	00 00 00 ff	success
8	512	00 00 02 00	success
8	1024	00 00 04 00	success
8	2048	00 00 08 00	success
8	4294967295	Ff ff ff ff	success
8	4294967296	NotAddressableError	success
8	-1	NotAddressableError	success
4	0	00 00	success
4	255	00 ff	success
4	0x10000	NotAddressableError	success
16	0xffffffffffffffff	ff ff ff ff ff ff ff ff	success
16	0x0	00 00 00 00 00 00 00 00	success
3	0	NotDividableByTwo	success

Commands

The individual commands have been tested with the fsclient. It is possible to test for the expected debug message, but not always the data (checksum, ram_test etc). Expected data have been stated where it makes sense. The id and type attributes have been removed from the responses to keep it simple.

Calculate Check Sum (address, length)

Fsclient arguments	calculate_check_sum 0 128
Datalayer write	0a 00 08 00 00 00 00 00 80 00 00 00 CD
Expected	Return code = 0x0a
Datalayer read	0a ff 04 00 15 04 92 a7h
Response	{"status":10,"data":2811364373,"message":"ACK"}
Result	Test success

Call Function (address, parameter)

This one needs some explaining. The following command will timeout instead of returning with a response. What happens is, that the function located at address 0x00000000 happens to be the reset function. The reset function will fire and never return anything. The debug port tells us what is going on:

```
* Packet received
* Call function command received

Init SPS/SIB
```

```
      Error decrement SIB
      * Fail-safe startup
```

Fsclient arguments	call_function 0 0
Datalayer write	03 00 08 00 00 00 00 00 00 00 00 00 CD
Expected	Should timeout due to reset
Datalayer read	Timeout
Response	{"status":106,"data":null,"message":"Timeout"}
Result	Test success

Copy To Flash (from address, to address, length)

Fsclient arguments	copy_to_flash 0x40000000 128000 512
Datalayer write	06 00 0c 00 00 00 00 40 00 f4 01 00 00 02 00 00 CD
Expected	Response code = 0xff (No error)
Datalayer read	ff ff 00 00
Response	{"status":255,"data":"","message":"No error"}
Result	Test success

Copy To Ram (from address, to address, length)

Fsclient arguments	copy_to_ram 128000 0x40003000 512
Datalayer write	07 00 0c 00 00 f4 01 00 00 30 00 40 00 02 00 00 CD
Expected	Response code = 0xff (No error)
Datalayer read	ff ff 00 00
Response	{"status":255,"data":””,"message":"No error"}
Result	Test success

Delete Flash Block (address)

Fsclient arguments	delete_flash_block 128000
Datalayer write	0b 00 04 00 00 f4 01 00 CD
Expected	Response code = 0xff (No error)
Datalayer read	ff ff 00 00
Response	{"status":255,"data":””,"message":"No error"}
Result	Test success

Download (address, length)

Fsclient arguments	download 0x40000000 10
Datalayer write	09 00 08 00 01 30 00 40 00 02 00 00 CD
Expected	Response code = 0x09 and 10 bytes of data
Datalayer read	09 ff 0a 00 0f 1c 1f ee 24 e5 b9 ce 1c 41
Response	{"status":9,"data":[15,28,31,238,36,229,185,206,28,65],"message":"ACK"}

Result	Test success

Download Sib

Fsclient arguments	download_sib
Datalayer write	10 00 00 00 CD
Expected	Response code = 0x10 and 32 bytes of data
Datalayer read	10 ff 20 00 ef be ed fe 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 b1 e5 50 4b
Response	{"status":16,"data":[239,190,237,254,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,177,229,80,75],"message":"ACK"}
Result	Test success

Execute (address)

Calling execute with address 0x00000000 will reset the satellite. But instead of timing out like the call_function, execute does not need to wait for a function return and therefor will not timeout.

Fsclient arguments	execute 0
Datalayer write	10 00 00 00 CD
Expected	Response code = 0xff (no error)
Datalayer read	ff ff 00 00
Response	{"status":255,"data":"","message":"No error"}
Result	Test success

Flash Test (address)

Fsclient arguments	flash_test 128000
Datalayer write	0f 00 04 00 00 f4 01 00 CD
Expected	Response code = 0xff (no error), data should be on of the following: • 0x00000000 (Flash block OK) • 0xaa000000 (Flash prepare block error) • 0xbb000000 (Flash erase block command error) • 0xcczzzzzz (Flash erase block + offset of error) • 0xdd000000 (Flash write block command error) • 0xffzzzzzz (Flash write block error + offset of error)
Datalayer read	ff ff 00 00
Response	{"status":255,"data":[null],"message":"No error"}

Result	Test failure!
	
	The data is not what was expected. The debug output looks like this:
	`* FLASH test command received`
	`* Flash - OK`
	`* ACK  response`
	There is no indication of an error, so either the failsafe documentation is not up-to-date with the implementation or vice verca.

Health Status

Fsclient arguments	health_status
Datalayer write	13 00 00 00 CD
Expected	Response code = 0x13 and 16 bytes of data
Datalayer read	13 ff 14 00 01 00 ff 0e 71 00 71 00 72 00 6e 00 71 00 71 00 00 00 00 00
Response	{"status":19,"data":[1,0,255,14,113,113,114,110,113,113],"message":"ACK"}
Result	Test failure! 20 bytes is has been read instead of 16 bytes. Only the 16 first of these bytes are used in the response. There is no indication of an error, so either the failsafe documentation is not up-to-date with the implementation or vice verca.

List Scripts

Fsclient arguments	list_scripts
Datalayer write	N/A
Expected	Response code = 100 and any available scripts as data
Datalayer read	N/A
Response	{"status":100,"data":[{"help":"Usage: count\nDescription: Count to five\n","path":"count"},{"help":"Usage: upload_file token filepath address\nDescription: Upload a file to an address in the satellites memory\nArguments:\n\tfilepath (string)\n\taddress (hexadecimal)\n","path":"upload_file"}],"message":"OK"}
Result	Test success!

Lock

Fsclient arguments	lock
Datalayer write	N/A
Expected	Response code = 100 and the token as data
Datalayer read	N/A
Response	{"status":100,"data":"eoAeDC5NiBr1Ea5o","message":"OK"}
Result	Test success!

Ram Test (address, length)

Fsclient arguments	ram_test 0x40003000 512
Datalayer write	0e 00 08 00 00 30 00 40 00 02 00 00 CD
Expected	Response code = 0x0e and the data should be one of the following: • 0x00000000 – no error, Ram area OK • 0xaazzzzzz – data bus error + error pattern • 0xbbzzzzzz – address bus error + address offset • 0xcczzzzzz – memory area error + address offset

Datalayer read	0e ff 04 00 00 00 00 00
Response	{"status":14,"data":[0],"message":"ACK"}
Result	Test success!

Read register (address)

Read register reads 4 bytes starting from the address given. It does not check the that the address refers the an actial register so I have tested with an internal Ram address.

Fsclient arguments	read_register 0x40000000
Datalayer write	0c 00 04 00 00 00 00 40 CD
Expected	Response code = 0x0c and 4 bytes of data. The data should be interpreted as an 4 byte little endian
Datalayer read	0c ff 04 00 0f 1c 1f ee
Response	{"status":12,"data":3995016207,"message":"ACK"}
Result	Test success!

Read sensor (address)

Fsclient arguments	read_sensor 1
Datalayer write	14 00 04 00 01 00 00 00 CD
Expected	Response code = 0x14 and 4 bytes of data. The data should be interpreted as an 4 byte little endian
Datalayer read	14 ff 04 00 72 02 00 00
Response	{"status":20,"data":626,"message":"ACK"}
Result	Test success!

Reset

Fsclient arguments	reset
Datalayer write	01 00 00 00 CD
Expected	Should timeout due to reset
Datalayer read	Timeout
Response	{"status":106,"data":null,"message":"Timeout"}
Result	Test success!

Reset Sib

Fsclient arguments	reset_sib
Datalayer write	12 00 00 00 CD
Expected	Should timeout due reset sib
Datalayer read	Timeout
Response	{"status":106,"data":null,"message":"Timeout"}
Result	Test success!

Run Script

Fsclient arguments	run_script count
Datalayer write	N/A
Expected	Should receive 5 partial messages, one per second, and one last message indicating that the script is done.
Datalayer read	N/A

Response	{"status":100,"data":"Counted to 1\r\n","message":"OK","partial":true}
	{"status":100,"data":"Counted to 2\r\n","message":"OK","partial":true}
	{"status":100,"data":"Counted to 3\r\n","message":"OK","partial":true}
	{"status":100,"data":"Counted to 4\r\n","message":"OK","partial":true}
	{"status":100,"data":"Counted to 5\r\n","message":"OK","partial":true}
	{"status":100,"data":null,"message":"OK"}

Result	Test success!

Set Autoreset (value)

Fsclient arguments	set_autoreset 01
Datalayer write	04 00 01 00 01 CD
Expected	Response code = 0xff
Datalayer read	ff ff 00 00
Response	{"status":255,"data":"","message":"No error"}
Result	Test success!

Sleep (seconds)

Fsclient arguments	sleep 3
Datalayer write	N/A
Expected	Send a response after 3 seconds
Datalayer read	N/A
Response	{"status":100,"data":null,"message":"OK"}
Result	Test success!

Unlock

Fsclient arguments	unlock
Datalayer write	N/A
Expected	Send an unlock response
Datalayer read	N/A
Response	{"status":108,"data":null,"message":"Server has been unlocked"}
Result	Test success!

Unlock Flash

Fsclient arguments	unlock_flash

Datalayer write	05 00 00 00 CD
Expected	Response code = 0xff
Datalayer read	ff ff 00 00
Response	{"status":255,"data":"","message":"No error"}
Result	Test success!

Upload (address, data)

Fsclient arguments	upload 0x40003000 0xffeeffeeffeeff
Datalayer write	08 00 0b 00 00 30 00 40 ff ee ff ee ff ee ff CD
Expected	Response code = 0xff
Datalayer read	ff ff 00 00
Response	{"status":255,"data":"","message":"No error"}
Result	Test success!

Upload Sib (data)

Fsclient arguments	upload_sib 0xefbeedfe000000000000000000000000000000000000000000000000
Datalayer write	11 00 1c 00 ef be ed fe 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 CD
Expected	Response code = 0xff
Datalayer read	fc ff 00 00
Response	{"status":252,"data":"","message":"Flash write error"}
Result	Test failure! The sib being uploaded is the one retrieved with the download_sib command, but without the checksum (the last 4 bytes). Either the failsafe documentation is not up-to-date with the implementation or vice verca.

Write register (address)

Write register writes 4 bytes starting from the address given. It does not check the address so I have tested an address in the internal Ram.

Fsclient arguments	write_register 0x40003000 0xffffffff
Datalayer write	0d 00 08 00 00 30 00 40 ff ff ff ff CD
Expected	Response code = 0xff
Datalayer read	ff ff 00 00
Response	{"status":255,"data":””,"message":"No error"}
Result	Test success!

FSGui

To verify that the gui works as expected I have conducted the following integration test.

Action	**Feature to test**	**Expected**	**Result**
Open the gui, make sure the autlock option is checked, click “connect”	Connect, autolock	The gui connects and locks the server.	Success
Choose the server scripts tab, Click refresh list, Choose Count, Click execute	Server Script	5 messages should be displayed in the script panel	Success
Click on the command sequence tab, Click “Add command” twice	Add command	Two commands should be added to the sequence	Success
Choose “health_status” for the first command and “sleep” for the second.	change command,	The dropdown boxes changes to the commands. Arguments fields are added.	Success
Move sleep command up, move sleep command down	move up, move down remove command	Sleep commands moves up, sleep command moves down, sleep command gets remove	Success
Click “execute”	execute	The interface gets locked, the result gets printed in the command	Success
Click “save”. Save as “test.json”	save	A save dialog is opened, the file is saved	Success
Click “new”. Click “OK” to the confirmation dialog	new	A confirmation dialog is show, the sequence is cleared after confirmation.	Success
Set dir to the directory of “test.json”. Choose “test.json” from the filetree. Click “open”.	Set dir, load	A health command is added the sequence.	Success
Click “Export as Ruby”. Enter description, enter fsclient path, Save as “test”	export	A save dialog is opened, a description dialog is opened, a fsclient path dialog is opened, the exported file is saved.	Success
Choose “local scripts”, set the root dir to the directory of the exported script, choose the file, click “Execute”	Run exported script, local scripts	The health_status is printed in the command panel	Success
Choose the health status tab, and click update status	Health status	The health status data should be inserted in the picture	Success

FSClient

To verify that fsclient works as expected I have conducted the following integreation test. Before performing the test start the server and open a console.

Action	Feature to test	Expected	Result
TOKEN=`fsclient -d lock`; echo $TOKEN	Data only option	Fsclient should only return the data attribute and echo it in the stored bash variable TOKEN.	Success
fsclient –token=$TOKEN health_status	Single command	Should print the health_status from the satellite.	Success
fsclient –token=$TOKEN unlock	unlock	Should unlock the server	Success
fsclient -ia	Interactive mode, auto_lock	Should go into interactive mode and lock the server	Success
health_status	Interactive mode	Should print the health_status from the satellite.	Success
unlock	Interactive mode	Should unlock	Success
exit	Interactive mode	Should exit	Success

Upload File script

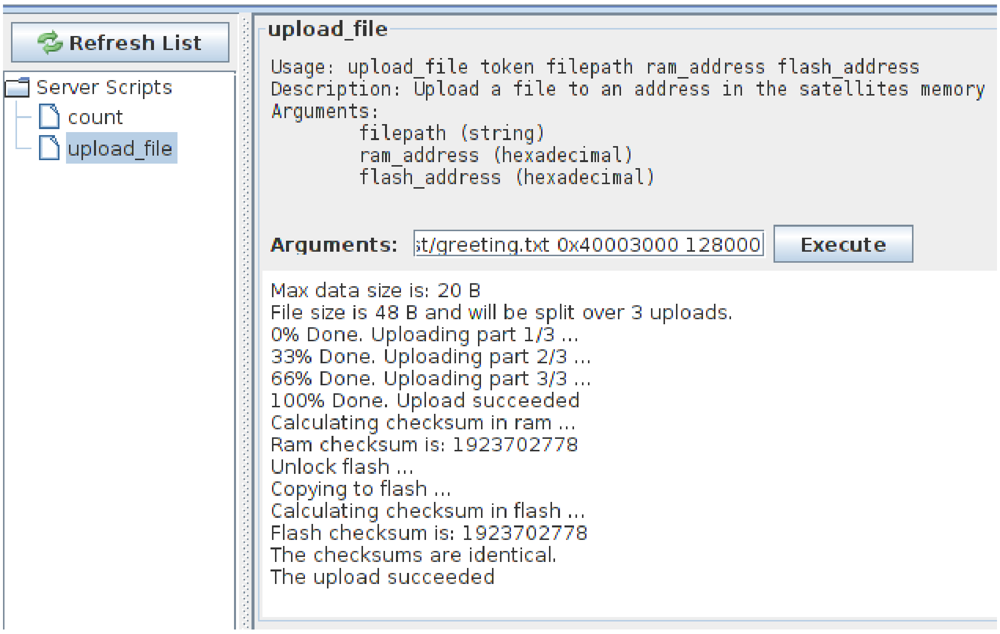
To test the upload_file script I have uploaded the file “test/greeting.txt” which contains the follow message:

Hi,

I am going to space ... and back again ...

Fsclient arguments	run_script upload_file test/greeting.txt 0x40003000 128000
Expected	The script should split up the file in appropriate sizes and upload the file chunk by chunk. To monitor the progress a message should be send whenever a part has been uploaded.

Responses	{"status":100,"data":"Max data size is: 20 B\r\n","message":"OK","partial":true}
	{"status":100,"data":"File size is 48 B and will be split over 3 uploads.\r\n","message":"OK","partial":true}
	{"status":100,"data":"0% Done. Uploading part 1/3 ...\r\n","message":"OK","partial":true}
	{"status":100,"data":"33% Done. Uploading part 2/3 ...\r\n","message":"OK","partial":true}
	{"status":100,"data":"66% Done. Uploading part 3/3 ...\r\n","message":"OK","partial":true}
	{"status":100,"data":"100% Done. Upload succeeded\r\n","message":"OK","partial":true}
	{"status":100,"data":"Calculating checksum in ram ... \r\n","message":"OK","partial":true}
	{"status":100,"data":"Ram checksum is: 1923702778\r\n","message":"OK","partial":true}
	{"status":100,"data":"Unlock flash ... \r\n","message":"OK","partial":true}
	{"status":100,"data":"Copying to flash ... \r\n","message":"OK","partial":true}
	{"status":100,"data":"Calculating checksum in flash ...

	\r\n","message":"OK","partial":true} {"status":100,"data":"Flash checksum is: 1923702778\r\n","message":"OK","partial":true} {"status":100,"data":"The checksums are identical.\r\n","message":"OK","partial":true} {"status":100,"data":"The upload succeeded\r\n","message":"OK","partial":true} {"status":100,"data":null,"message":"OK"} This is what it looks like in the GUI:  Refresh List Server Scripts count upload_file Execute upload_file Usage: upload_file token filepath ram_address flash_address Description: Upload a file to an address in the satellites memory Arguments: filepath (string) ram_address (hexadecimal) flash_address (hexadecimal) Arguments: st/greeting.txt 0x40003000 128000 Max data size is: 20 B File size is 48 B and will be split over 3 uploads. 0% Done. Uploading part 1/3 ... 33% Done. Uploading part 2/3 ... 66% Done. Uploading part 3/3 ... 100% Done. Upload succeeded Calculating checksum in ram ... Ram checksum is: 1923702778 Unlock flash ... Copying to flash ... Calculating checksum in flash ... Flash checksum is: 1923702778 The checksums are identical. The upload succeeded
Result	Test success!

To verify that the file actually has been uploaded we can download the data back again and interpret the bytes as characters. The ruby code is implemented in “test/download_chars.rb”:

Ruby code:	require 'rubygems' require 'json' token = JSON.parse(`fsclient lock`)["data"] bytes = JSON.parse(`fsclient --token=#{token} download 0x40003000 48`) ['data'] puts bytes.pack("c"*48) # Interpret as 48 characters
Expected	*Hi,* *I am going to space ... and back again ...*
Output	*Hi,* *I am going to space ... and back again ...*
Result	Test success!