Failsafe Control for DTUSat2

Kasper Bjørn Nielsen (s052808)

Kongens Lyngby 2010
IMM-BSC-2010-26

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

The DTUSat-2 project is a student satellite project at DTU. Any software being run on the satellite could potentially make the satellite unresponsive if a serious error occured. Therefore, a failsafe mode has been developed which should catch theses failures so that the DTUSat-2 staff can investigate errors and prevent them from happening again by uploading new software. To operate the failsafe mode the staff needs both a console tool and a graphical user interface. It is the purpose of this project to investigate possible solutions and produce the software necesarry for the staff to be able to operate the satellite when in failsafe mode.

Preface

This Bachelor thesis was accomplished at the Department of Informatics and Mathematical Modelling (IMM), at the Technical University of Denmark (DTU), during the period 1st of February 2010 to the 30th of June 2010. The project was supervised by Associate Professor Hans Henrik Løvengreen of the IMM department at DTU.

This thesis documents the Failsafe Control for DTUSat-2 software. It is aimed at the DTUSat-2 staff and should give any staff member a thorough overview of the requirements, design decisions, implementation specifics, test results, installation instructions and operating instructions of the software produced.

I would like to thank Hans Henrik Løvengreen for his advice, guidance and enthusiasm throughout the project. Also a big thanks to my friends and my family for their support and advice.

Lyngby, June 2010

Kasper Bjørn Nielsen, s052808

Contents

Chapter 1

Introduction

The DTUSat-2 is a student satellite project of several departments at DTU. The goal of the project is to implement a spaceborne radio-tracking system capable of locating birds on intercontinental migration routes. When the satellite has been launched and is orbiting earth, it is practically impossible to press a reset button in case of a hardware or software failure. Therefore, unforseen failures that make the satellite unresponsive, must be handled. This is achieved by constantly monitoring for failures and unresponsiveness.

There are two main programs on the satellite. The nominal mode, which is the full functioning autonomous program that normally runs on the satellite. This mode performs the tasks necessary for tracking birds.

If a failure is detected the software will reboot into the unautonomous failsafe mode which is a small program consisting of just 20 commands. In this mode it is possible to investigate what went wrong and to upload new software.

Operating software to the satellite's nominal mode have already been developed. There is a console program called fsterm which was used in DTUSat-1 project to operate the failsafe mode, but this program is no longer sufficient as new commands have been added and a graphical user interface (GUI) has become a requirement.

1.1 Thesis Statement

The purpose of this project is to implement a software solution that makes it possible for the DTUSat-2 staff to operate the satellite when in failsafe mode. Especially it should provide:

- Interaction via a console.
- Interaction via a desktop computer via a GUI. The GUI must be easy to get up and running.
- Flexible tools for specific tasks, such as uploading new software, running tests and performing diagnostics
- A graphical representation of the state of the satellite and its subsystems.

Conditions for the project:

- The final software must run on the Ground Station which is the computer responsible for the radio communication with the satellite.
- Communication with the satellite must go through an already given protocol layer written in C.
- Strive for platform independence but favor UNIX.

1.2 Approach

There are many people involved with the DTUSat-2 project, working on different parts at the same time. The failsafe software was not implemented at the start of this project, so no detailed documentation of communication interfaces, byteordeing etc. was available. Furthermore, some feature requirements was added during the implementation process.

These conditions make it cumbersome to achieve a satisfying solution with a rigid classical analysis-implement-test approach. Instead the process has been of a more iterative and agile nature. An iteration was typically one week in length and started with a supervisor meeting where we agreed on the next week's workload based on an evaluation of the previous week's work.

1.3 Outline of Chapters

The report consists of nine chapters and four appendixes. Here is an outline of the individual chapters.

Chapter 2 **Requirements Specification**, states the requirements of the project.

Chapter 3 **General analysis and design**, analyses the requirements of the project, discusses alternative designs and presents the chosen design which is divided in four key parts: FSServer, FSClient, FSGui and user scripts.

Chapter 4 **FSServer**, analyses the requirements to the server part, discusses alternative designs and presents the chosen design. Explains non-trivial part of the implementation.

Chapter 6 **FSClient**, analyses the requirements to the client part, discusses alternative designs and presents the chosen design. Explains non-trivial part of the implementation.

Chapter 5 **FSGui**, analyses the requirements to the gui part, discusses alternative designs and presents the chosen design. Explains non-trivial part of the implementation.

Chapter 7 **Upload File script - an example of a user script**, analyse the requirements to the upload script part, discuss alternative designs and presents the chosen design. Explains the implementation.

Chapter 8 **Tests and Results**, describes the test stragedy, the test cases and the test results

Chapter 9 **Conclusion**, summarizes what has been done, which goals have been met and gives a perspective for the future of the project.

Appendix A **User Guide**, contains installation and operation instructions for the individual subsystems.

Appendix B **Failsafe Commands**, contains a list of the 20 failsafe commands

Appendix C **Test Cases**, contains all the test cases

Appendix D **Source Code**, contains the entire source code as well as information to the svn repository

Chapter 2

Requirements Specification

A good way to evaluate the success of a project is to state some measurable requirements that can be tested for when the system has been been implemented. This chapter deals with the requirements of the project.

Before stating the requirements they must first be identified. The requirements are directly dictated by or based on meetings with my supervisor who is a member of the DTUSat-2 staff.

The requirements to this project are:

- The user must be able to operate the satellite in failsafe mode from a desktop computer via a console program or a GUI.
- Command combinations
  - must be able to combine several commands in conditionals and loops.
  - must be able to save, load and execute these combinations
  - must be flexible to create and change
- Console program
  - must take a command and its arguments as parameters on the command line

  - must have an interactive mode. This mode must prompt the user for a command, execute it, print the result and prompt again.

- GUI
  - must show a graphical representation of the satellites health status
  - must create, save, load, export and execute command combinations
  - must provide a graphical tool to put together command sequences (without conditionals and loops)
  - must be easy to install and run

- Upload script
  - must take a file and a memory address as parameters and upload to the satellite at the given memory address.
  - must notify the user with the progress

- Protocol layer
  - must use the protocol layer so that the datalink implementation can be changed later on

Chapter 3

General Analysis and Design

Before implementing the system a good design must be chosen. The design should be based on thorough analysis and discussion of alternative solutions.

This chapter deals with the analysis and design of the system. First the context of the final system is stated. Then the requirements specification is broken down and an analysis mixed with a design discussion is given for each requirement.

The chapter concludes with a summary of the chosen design and an overview of its main components.

3.1 Context

To design the system it is crucial to know the context of it.

The satellites Failsafe Mode (FS)
The failsafe program on the satellite is basically a request-response loop. Requests and responses are simple binary formatted streams of data. The failsafe commands and their formats are listed in appendix B.

Ground Station (GS)
When orbiting earth, communication with the satellite goes through a radio. This radio is controlled by a computer called the Ground Station. GS runs a UNIX variant as operation system. To operate the failsafe mode communication must therefore go through the Ground Station computer. There is a protocol layer for the datalink that must be used for communication with the radio.

The staffs desktop computers
The software must be operational from the staffs desktop computers.

3.2 Analysis and design discussion

In the follow each requirement is analysed. Challenges and possible solutions are discussed along the way.

The analysis is of an iterative nature. For each iteration the overall design will be enhanced to incorporate the requirement.

3.2.1 Console program - FSClient

We could solve this requirement with a console program named fsclient running on the Ground Station. Fsclient could take a satellite command and its arguments as parameters and send it via the radio interface of the Ground Station to the statellite. This is basically how fsclient for the DTUSat-1 works.

As fsclient would be installed on the Ground Station the staff members must use the physical computer or have access to it via SSH for instance.

3.2.2 GUI

There are many posibilities when it comes to implementing a GUI but regardless of the choice there must be communication with the satellite over the radio.

One way to solve this is to let the GUI connect to the Ground Station via SSH and use fsclient each time it must send a failsafe request.

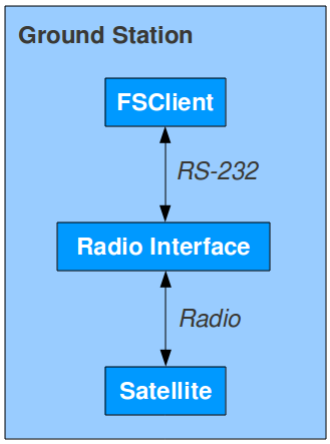


Figure 3.1: FSClient on the Ground Station

A drawback to this solution is that not all GUI frameworks and operating systems have support for SSH out of the box.

3.2.3 TCP Interface to the radio

Alternatively there could be a TCP server on the Ground Station that listens for failsafe commands from connected TCP clients and have them forwarded to the radio. With this TCP server, called fsserver, we effectively have a TCP interface to the satellite and the advantage here is that all major GUI frameworks and all major operating systems have TCP support making this solution far more flexible than the SSH solution.

3.2.4 Command Combinations

The failsafe software has a set of basic commands which by themselves are of limited use but becomes powerful when combined. As we cannot know which failures might occur in the future, it is critical that there is a flexible way of combining these commands in conditionals and loops.

One solution is to combine the commands in the GUI. Then we would need to construct individual GUI elements for concepts like a command, a command sequence, an if statement and its branches, a loop statement and its body and

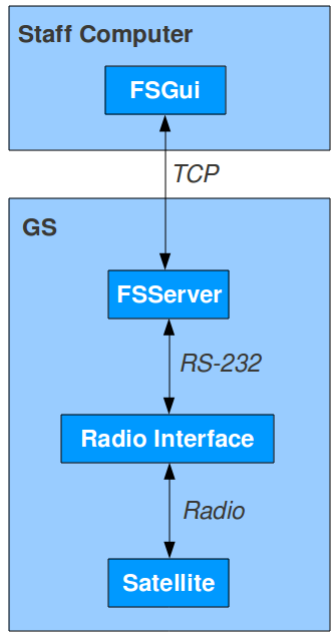


Figure 3.2: TCP Interface to the radio

so forth. This solution is hard to extend with a switch-statement in two years time when someone needs it - and impossible to do without the source code.

The use of script languages to combine commands Alternatively there is a far more flexible solution to this challenge when it is recognized that conditionals and loops already have been implemented numerous times in script languages such as Ruby, Python or TCL and that they can be used to achieve the required flexibility.

Send failsafe commands from a script using UNIX It is necessary that the scripts can communicate with the Ground Station. To solve this each script could implement a TCP client, but that would mean two things. One, a TCP client for each specific language would be needed before they could be used to write failsafe scripts. This is inflexible. Two, there would exist several implementation of the TCP client which is error-prone.

Is there anything all scripting languages have in common? Well, if we assume that the scripts are being run on a UNIX system, they will have exactly that in common. All languages have built-in libraries for running a UNIX console command and manipulating the output. So, we could use fsclient to pass failsafe

commands through. Fsclient will just handle the TCP connection to the Ground Station and forward the commands.

FSServer must be aware of multiple TCP clients and ensure that commands from various TCP clients are not executed at the same time. This challenge is covered the chapter 4.

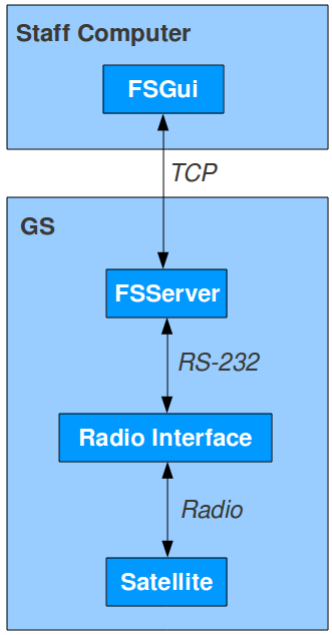


Figure 3.3: FSClient and executing scripts

3.2.5 Graphical representation of the health status

The failsafe software has a command called HEALTH_STATUS which will return a set of numbers representing the state of the subsystems. These numbers will be interpreted as temperature, current and voltage values in the GUI. The GUI must provide a graphical representation of the subsystems and their health status.

3.3 Chosen design

Here is a wrap up of the final design that consists of four key elements:

FSServer

- Installed on GS
- A TCP server
- Send failsafe commands via the datalink
- Send failsafe response back to TCP clients
- Ensure atomicity of failsafe commands

FSClient

- Installed on the staff's desktop computers
- Unix program
- A TCP client to FSServer
- Forward failsafe commands from other unix programs to the server
- Interactive mode

FSGui

- Installed on the staff's desktop computers
- Runs on unix and windows systems
- A TCP client to FSServer
- No installation required, just download and double-clik
- Create, save, load, export command sequences
- Execute failsafe scritps
- View graphical representation of the health status

Scripts

- Any script or programming language that can execute unix commands
- Combine failsafe commands in conditionals and loops

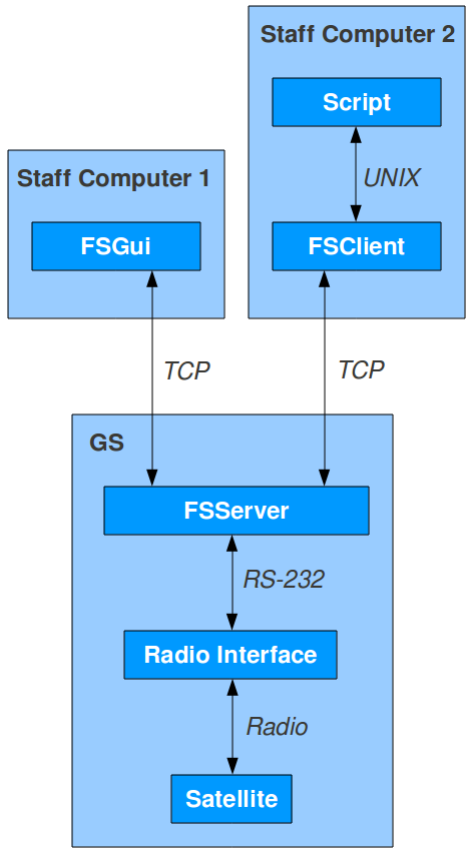


Figure 3.4: The chosen design

Now that there is an overall design, lets proceed with an analysis, design and implementation part of the individual parts.

Chapter 4

FSServer

This chapter deals with the requirements specification, analysis, design and non-trivial implementation details of FSServer.

4.1 Requirements Specification

The requirements for FSServer are:

- must be installed on GS
- must accept TCP connection
- must send failsafe commands via the datalink
- must send failsafe responses back to TCP clients
- must ensure atomicity of failsafe commands

In addition to these requirements, the following requirements was identified when the overall design was agreed upon:

- must run as a daemon and log all activity to a file
- must execute scripts on GS and from a TCP client and immediately send any response back to the TCP client
- must validate failsafe commands

4.2 Analysis and design

Lets go through the requirements one by one. For each requirement we will enhance the design to incorporate the requirement.

Must be installed on GS
Trivial constraint.

Must accept TCP connections
Clients can connect to the server with the TCP protocol over an internet socket. The server could accept one or multiple connections at a time. We want to ensure that only one command is send to the satellite at a time, so lets accept only one connection for now.

Must send failsafe commands via the datalink and send response back to clients
We could implement pull-behaviour where each request is answered with exactly one response or we could implement push-behaviour where data can be pushed to the client without being requested. It is not obvious which of these behaviours we should implement so lets choose the pull-behaviour for now.

Must ensure atomicity of failsafe commands
Only one satellite command can be executed at a time. If we use a request queue with atomic enqueue and dequeue operations we can implement this behaviour without race conditions.

Must run as a daemon and log all activity to a file
A unix daemon is a process with a parent process id of 1. When starting the server, the staff must be able to decide whether to run it as a daemon or to run it as a normal process. The logfile could be a predefined file, but we should let the staff decide by providing an option when starting the server.

Must execute scripts on GS and from a TCP client and send any response back to the TCP client
Some tasks, such as uploading a binary file to the satellite, are so common that

they should be available for every client. Instead of implementing these tasks in every client, the server should implement them and execute them when requested by the client. The scripts can be written in any language that is supported by the server environment. To capture the output of a script, the server will open a pseudo terminal that has control of standard output and input. The script is executed in the pseudo terminal and any output can be send to the client.

One or multiple TCP connection
Now consider the following scenario. A client connects to the server and requests to execute script X. Because of the pull behaviour we decided upon earlier the client will now wait for the script to finish with the response. If script X needs to interact with the satellite, it must connect to the server and do some requests but the server only accepts one TCP connection at a time, and we have a deadlock situation.

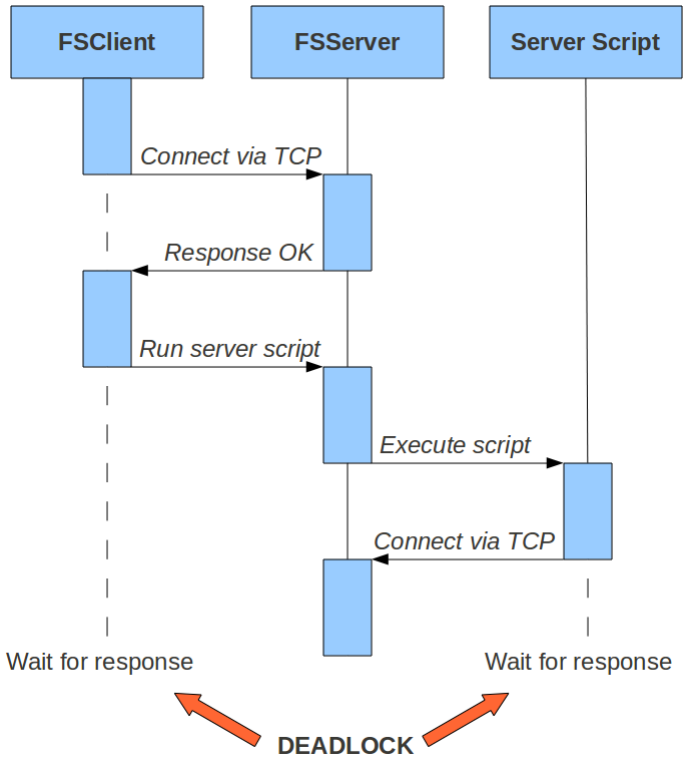


Figure 4.1: Deadlock when executing script

So there is a need for handling multiple TCP clients and their requests simultaneously. Satellite commands over the serial connection must not be simultaneous

however!

Blocking vs. non-blocking execution
In a program a non-blocking execution of a statement will not wait for the result of that statement before proceeding with the execution of the next statement. A blocking execution of a statement will wait for the result before proceeding. In order to handle multiple TCP clients concurrently the server must execute the requests in a non-blocking fashion.

Lock mechanism
With multiple clients new concerns must be dealt with. Consider the scenario where two clients execute a server script each. The script consists of 2 satellite commands and although we can ensure that only one satellite command are being executed at a time, we can not ensure that the 2 commands are executed sequentially.

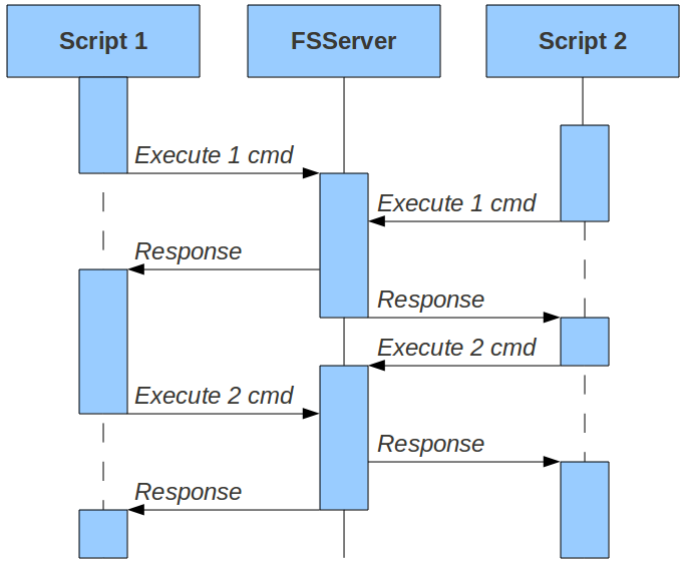


Figure 4.2: Two scripts executing without locking

We need a way to lock the server to prevent multiple executing scripts from stepping on each others toes. So after connecting to the server a client must first issue a lock request and will receive a session token if the lock succeeded or a message indicating that the server is already locked.

When finished an unlock request can be sent to unlock the server. Things can go wrong however and leave the server in a locked state, so there will be

implemented a timeout on the token. After each request this timeout will be reset. If the token gets timed out, a broadcast message will be sent to all connected clients. It is important to note that two scripts requested by clients using the same token will be executed simultaneously!

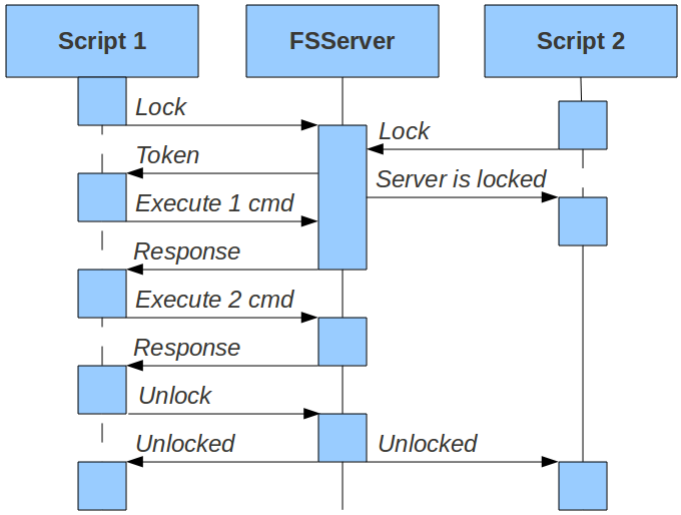


Figure 4.3: Two scripts executing with locking

Partial responses
Another thing to consider is the output of the server scripts. Imagine for instance an upload script that prints to standard out each time 10% of the upload is done. The client would like to follow this progress in real time as opposed to seeing the total output of a script when execution has finished. So in the case of the upload script we would at least be sending 10 partial messages in response to one request. This along with the timeout broadcast leads to the need for pushing data to clients and we must therefore implement the push-behaviour in favor of the pull-behaviour.

Must validate failsafe commands
Failsafe commands can have arguments such as ”address”, ”length” etc. These arguments must be validated before being send to the satellite. Arguments are typically memory addresses and other hexadecimal values. It would be convenient to use both hexadecimal (”0xff”) and decimal (”255”) values as arguments. If an argument is invalid or missing the client must be informed and the request must not be executed.

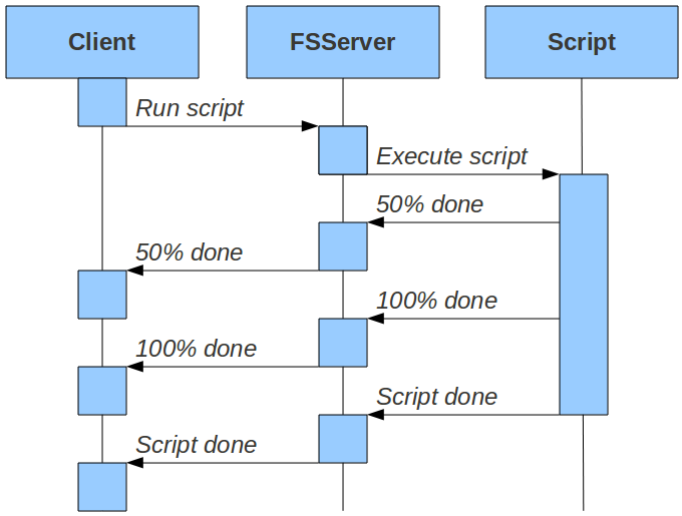


Figure 4.4: Partial response

4.2.1 Data format

Now that we have a design that covers all features, lets look at the data format. We have several choices here. We could go with a simple text format, XML or JSON.

The client must be able to send a message to the server and react to the received response. There must be a way of determining whether incoming data is a response to some outgoing data. As there are no such information in the order of the outgoing and the incoming data, a request must be stamped with a unique identification key that can be used by the server to stamp the response. The data format must at least have two parameters then, a stamp and the data.

If we use a simple text format we would need a separator to distinguish the parameters. Then we would have to care about escaping certain characters etc. Instead we should consider using the semantically equal XML or JSON formats. We will be using JSON.

- Request: {`"id":KEY, "data":REQUEST`}
- Response: {`"id":KEY, "data":RESPONSE`}

Example of a ”lock” request:

- Request: {"id":"1","data":"lock"}
- Response: {"id":"1","data":"7KFdnNXBYi8nmuWV"}

If the client has locked the server, the token is sent along the request like this:

- Request: {"id":"1","data":"reset","token":"7KFd"}

Besides from being able to respond to incoming requests, the server also needs to push data to the client independently of any requests. The client must be able to determine the data type to handle it correctly. The format must at least look like:

- Message: {"type":TYPE}

Example of an "token timout" message:

- Message: {"type":"server_unlocked"}

For format consistency and faster processing on the client side, the response format will also have the type parameter:

- Response: {"type":"response", "id":KEY,"data":RESPONSE}

The server must also be able to send more than one response to a given request. This can be solved with the following format:

1. Req: {"id":"2", "data":"run_script upload filepath", "token":"7KFd"}
2. Res: {"type":"response", "id":"2", "data":"50% done", "partial":"true"}
3. Res: {"type":"response", "id":"2", "data":"100% done", "partial":"true"}
4. Res: {"type":"response", "id":"2", "data":""}

4.3 Implementation

This section deals with the non-trivial implementation details.

Language and libraries
There are no contraints to the programming language and as the server will neither be cpu- or io intensive the server has been implemented in Ruby. It could have been implemented in C or Java, and probably should have if there where any requirements to speed.

Ruby has a standard library with support for basic things like sockets, threads, system etc. There is a packaging tool called rubygems with which one can install third party libraries called gems.

Instead of reinventing a TCP server, a serialport wrapper and a JSON parser we will be using production ready ruby gems that has been tested over long periods of time by many people and have been accepted within the ruby community.

Third-party gems used in the implementation:

- **Eventmachine** - Instead of the relatively slow socket support in the standard library we will be using the evetnmachine gem. Eventmachine is a library for implementing non-blocking TCP servers and clients.
- **JSON** - provides a JSON parser class.
- **Serialport** - provides a class for using RS-232 serial ports.
- **Daemons** - provides an easy way to wrap existing ruby scripts to be run as a daemon and to be controlled by simple start/stop/restart commands.

Safe script execution
When executing server scripts from a client we must be very careful about the parameters. Consider the scenario were a client sends this request:

```
run_script scriptname param1;rm -rf}
```

If we are not careful and just pass along the parameters without escaping the semicolon, the result of this command will delete all files on the server, that the currently running user has permissions for.

Ruby has a wrapper for the UNIX command execve. Execve can create new UNIX processes and will safely pass arguments along. Ruby's pseudo tty library uses the execve wrapper to execute console commands.

Failsafe Commands
A failsafe command is a Ruby class that can have an initialize, a validate and an execute method. Initialize takes the arguments for the command, validate will validate the arguments and execute will execute the command.

All commands inherits common code from the AbstractCommand class. This class implements some common validations, a default initialize method, a default validate method and a default execute method.

Parsing
When the server gets a new request, it is parsed with the JSON gem. The id and token are easily looked up in the resulting Ruby hash. The data field is first split on spaces to separate the command from its arguments. The command is then camelized which means that the first character and any characters immediately after an underscore is capitalized. The underscores are then removed. Each command has a corresponding camelized Ruby class.

For example:
The server receives this request:

```
{"id":"1","token":"abcdefg", "data":"run_script filepath arg1 arg2"}
```

First it is parsed as JSON. This results in a Ruby hash where we immediately lookup the id and the token. The data is then split on spaces:

```
cmd, *arguments = parsed_request["data"].split
```

Ruby has a function called eval that takes a string as a parameter and executes that string as Ruby code. To get an instance of the camelized class to a run_script command we can execute the following:

```
eval(cmd.camelize+".new(*arguments)")
```

which is equivalent to:

```
eval("RunScript.new(*arguments)")
```

When this string is executed by eval, the initialization code for RunScript will be called with all the arguments.

Options are also extracted. If an argument begins with a dash (”-”) then it will be split on equal-signs and stored in an options hash.

Again we must be very careful when executing a string from an unknown place. In this way a user could create an instance of any Ruby class and pass any arguments to it. To deal with this, we encapsulate all valid commands in a module called Commands. So now the eval looks like:

```
eval("Commands::#{cmd.camelize}.new(*arguments)")
```

Validation
When the command has been parsed the server will call the validate method before calling the execute method. If the validate method fails the execute method will never be called, instead a failed validation message is sent to the client.

Common validations are implemented in the AbstractCommand class and uses some extensions to the String class. The most common validation is to ensure that the argument is addressable. To be addressable the value must be given as a hex or a decimal value. The validation ensures that the value is not greater than 0xffffffff.

Satellite IO
The server will have to communicate with the radio link on GS when communicating with the satellite. During development however a development board has been used. This board has a RS-232 interface, so the serialport gem has been used to write an serialport implementation that adheres to the protocol layer interface.

Request queue with atomic enqueue and dequeue operations
It is important that only one command is being sent at any given time to the satellite. Therefore, a ProcessingQueue module has been implemented. A processing queue has an array of requests, a mutex and a boolean value to indicate that the processing has already started. When enqueueing, the request is put into the queue when the mutex becomes available. Then the processesing loop is started unless it has already been started.

The queue is when the mutex becomes available and the request is processed. As soon as the processing is done the mutex is released and the processing is started again until there is no more requests left in the queue.

Options for satellite commands:

- **–timeout=SEC** - There is a timeout for each executing satellite command. It is set to 5 seconds by default but can be changed per request with the timeout options e.g. ”reset” and ”reset –timeout=20” are both valid commands. The former will timeout after 5 seconds and the latter will timout after 20 seconds. Alternatively each command could have had its own default timeout.
- **–no-response** - Sometimes we do not care about the response for a command and would like to skip it in order to save execution time and batterylife. The option looks like this: ”reset -no-response”.

Broadcasting
Broadcasting can be done by maintaining a global array of connected clients. A message can then be sent to each client.

Token handling and timeout
The TokenHandler class is responsible for timing out the token. It has a token variable, which is just a string. The server is unlocked if this value is nil. When setting the token variable a timer will be started. If the timer runs out the token variable will be set to nil and a broadcast message is sent to all connected clients.

Logging
When starting the server one can indicate to use a logfile for the server output. If none is indicated the server will print its output to standard out.

Chapter 5

FSGui

This chapter will deal with the requirements specification, analysis, design and non-trivial implementation details of FSGui.

5.1 Requirements Specification

The requirements for FSGui are:

- must be installed on the staff's desktop computers
- must run on unix and windows systems
- must be a TCP client to FSServer
- must not need installation, just download and double-clik
- must create, save, load, export command sequences
- must execute failsafe scritps
- must view graphical representation of the health status

In addition to these requirements, the following requirement was identified when the overall design was agreed upon:

- must be able to retrieve a list of server scripts and execute them with custom arguments
- must be able to execute local scripts
- must have an auto_lock option
- must indicate connection and lock state
- must indicate when the health status was last updated

5.2 Analysis and design

Lets go through the requirements one by one. For each requirement we will enhance the design to incorporate the requirement.

Must be installed on the staff's desktop computers
Trivial constraint

Must run on unix and windows systems
There exists a number of cross-platform GUI frameworks we could use like Java swing or Xulrunner.

Xulrunner is Mozilla's cross-platform and open source framework. Xulrunner has the advantage that it is very easy to build complex and appealing GUIs in no time. The disadvantage is that the framework takes up 20 MB of harddisk space and is not included on UNIX og Windows by default. Furthermore, it lacks some systemwise features like spawning new processes and executing console commands.

The advantage with Java is that it runs on most systems and once you have the java virtual machine installed it is fairly easy to just pack an executable jar file and run it with a double click. The FSGui will be implemented in JAVA using the Swing Framework.

Must be a TCP client to FSServer
When the program starts the user must indicate the address and port of the FSServer. With a 30 sec token timeout the user will experience lock errors often enough to be annoying. Therefore, it must be possible to set an autolock option.

It must also be clear to the user whether the server is locked or not and whether a connection has been made to FSServer or not.

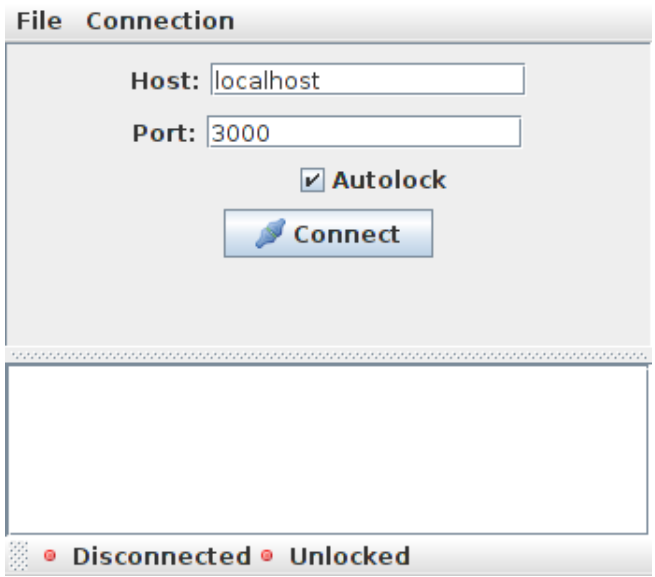


Figure 5.1: The connect panel

Must not need installation, just download and double-clik
The FSGui will be packed as an executable jar file. The only requirement to the staff computers is that they have a Java Virtual Machine installed.

Must send failsafe commands
Java has built-in support for sockets and TCP.

Must create, save, load and export command sequences
For easy and fast testing, simple command sequences can be composed in the GUI. There are no conditionals or loops and the sequence will stop executing if one of the commands fails for one reason or another. It is possible to save and load sequences and to export a sequence to a Ruby script.

Sequences can be saved, loaded and exported to scripts.

Must view graphical representation of the health status
The satellite can read various health information from the individual subsystems and send them back to earth. Which data the subsystems will actually yield and how that data should be interpreted is not agreed upon as of this writing.

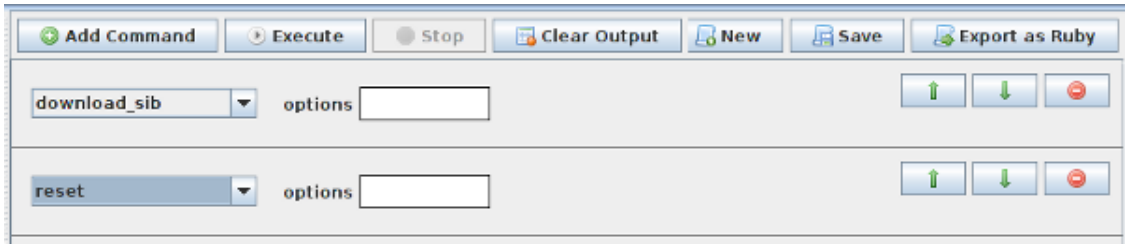


Figure 5.2: Command Sequences

Instead the development board, that was used in place of the real satellite, simulates this data by reading several voltages and currents. When the real data is available the staff should be able to modify the source code of this project and fairly quickly replace the development data with the real data.

Must be able to execute local scripts
In addition to running a script in the console it should also be possible to run and see the output of that script from within the GUI. The scripts are organized in a filetree and the user should be able to set the root of the filetree, navigate through the filetree, selecting a script, passing arguments to it and executing it.

Must be able to retrieve a list of server scripts and execute them with custom arguments
FSGui can retrieve a list of available server scripts that the user can choose from. FSServer will implement the available_scripts command to retreive this information. Once retrieved the GUI builds a tree of the scripts for easy navigation.

When a script has been clicked on in the tree, a script description and a textfield for passing arguments are presented to the user. An execute button will execute the script on the server, any response data will be appended just below the script.

Must have an auto_lock option
An auto_lock option is available in the connect panel and from the menubar.

Must indicate connection and lock state
There is a statusbar in bottom of the window indicating whether there is a connection or not and whether the server is locked by the gui or not.

must indicate when the health status was last updated
In the health panel there is an indication of when the health status was last

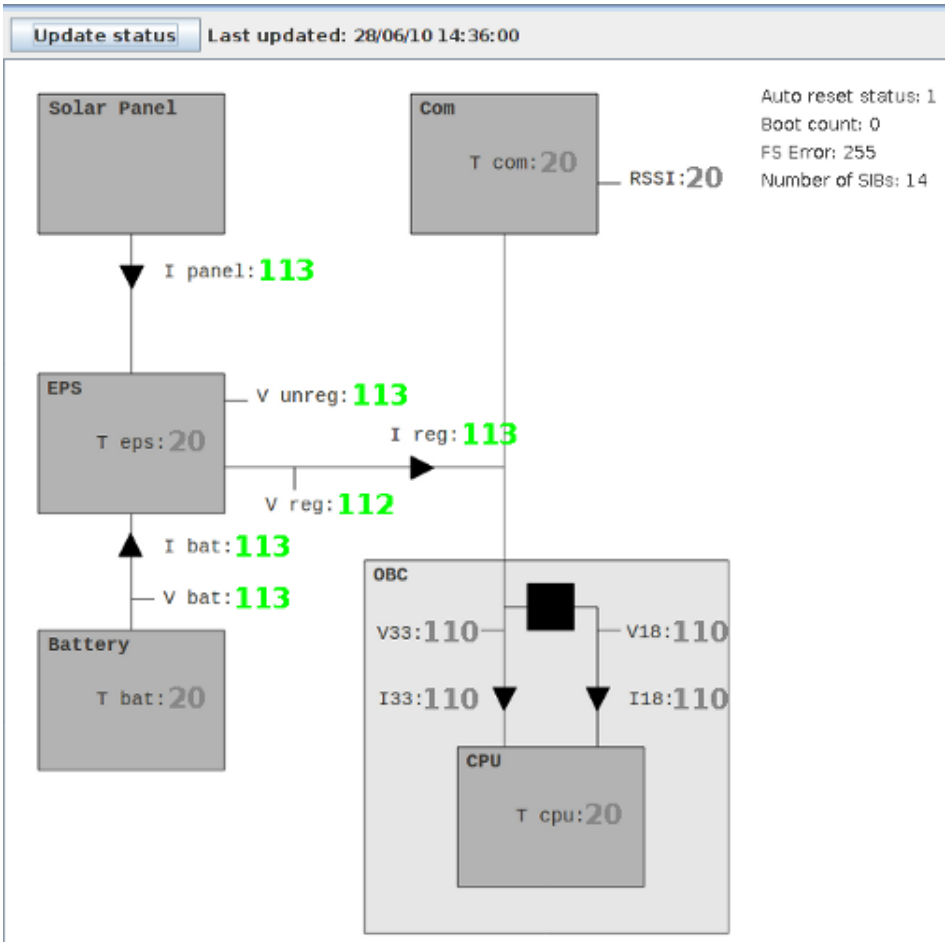


Figure 5.3: The Health Status with last updated stamp

updated.

5.3 Implementation

Lets look at the non-trivial part of the implementation.

FSController
At the heart of the FSGui is the FSController singleton class which is responsible

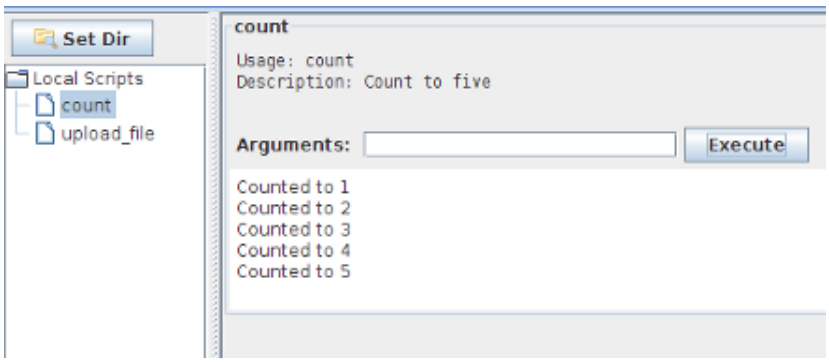


Figure 5.4: Local Scripts

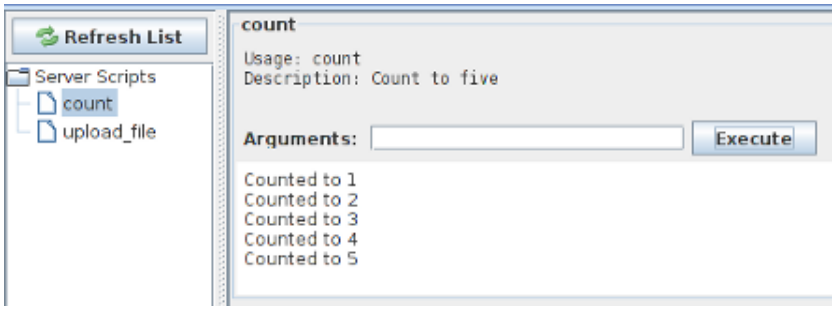


Figure 5.5: Server Scripts

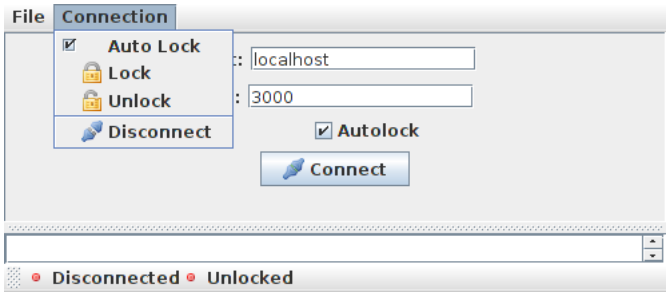


Figure 5.6: Auto Lock, connection and lock status

for creating the TCP socket, building the swing gui and setting up the data handlers. Because of its bigbrother like nature it is also extensively used in other classes to reference other parts of the program.

Local Scripts
Java has a built-in class called ProcessBuilder that can create a process given a list of the command and its arguments. It will execute and capture the output of the process. We will use this class to execute local scripts.

Socket and Socket Callbacks
The built-in class Socket will be used to create the TCP connection and to read/write on the socket. To prevent cluttering in FSController the Socket will be wrapped in a class called FSSocket that will perform callbacks to a class that implements the FSSocketObserver interface when interesting events happen, such as on connection, on disconnection, on incomming data etc. The FSController implements this interface and will therefore be able to act on these events.

Data Handlers and request callbacks
When we send a request to the server we will often expect some kind of response. As the solution is push driven we cannot know when the response will come. We cannot send the request and then immediately wait for the response on the socket, potentially throwing away responses to other requests. We have to send the request and just hope that the response will come.

When the response finally do come, we won't know what to do with it. So along with sending a request we most associate it with a callback that should be executed upon a received response.

This is done with a FSCallback which is an interface consisting of one method called onResponse that takes a FSResponse as its parameter. The callback is registered in the requestCallbacks Hashtable where an entriy consists of the requests id as key and the callback as a value.

A class called FSSocketReader will constantly be reading the socket and whenever a message has been read it is first sent to the appropriate data handler and if it is a response, the response handler will look for the id in the requestCallbacks and execute the callback.

When the client gets an incoming message FSGui determines the type of the message and dispatches it to the appropriate data handler. Data handlers must therefore be setup before connecting to the server.

The response handler
When sending a request one specifies what piece of code to run when the response is received. The response handler looks at the id parameter and matches it to the corresponding request and dispatches the data to that piece of code.

The token timeout handler
This handler just notifies the user by changing the "locked"-icon in the GUI.

JSON Parsing
Java does not have built-in support for JSON, so a third-party library called JSONObject has been used.

Generating unique ids for the requests
Unique id generation is handled by FSSocket and is appended the the request before sending it off to the server. The id should just be an increasing integer as the id-space is limited to each connecting tcp client. The server increments the id in a syncronized method to ensure that only one thread can increment it at a time.

If the id-space was limited to each client, a solution could be to use random numbers. UUIDS are very good to such situations [1].

Command Sequences
The command sequences are saved in a simple json format. A sequence of the following commands:

```
health_status
download 0x40000000 512
```

Will be saved as the following JSON data:

```
[
{"command":"health_status", "arguments":[""]},
{"command":"download","arguments":["0x40000000","512",""]}
]
```

NOTE: The empty string in the end of each argument is the options textfield.

[1] A good introduction to UUID's can be found at wikipedia: <http://en.wikipedia.org/wiki/Universally_Unique_Identifier>

Chapter 6

FSClient

This chapter will deal with the requirements specification, analysis, design and non-trivial implementation details of FSClient.

6.1 Requirements Specification

The requirements for FSClient are:

- must be installed on the staff's desktop computers
- must be a unix program
- must be a TCP client to FSServer
- must forward failsafe commands from other unix scripts to the server
- must have an interactive mode

In addition to these requirements, the following requirements was identified when the overall design was agreed upon:

- must have a data_only mode
- must have an auto_lock mode

6.1.1 Analysis and design

Lets go through the requirements one by one. For each requirement we will enhance the design to incorporate the requirement.

Must be installed on the staff's desktop computers
Trivial constraint.

Must be a unix program
Trivial constraint.

Must be a TCP client to FSServer
Trivial constraint

Must forward failsafe commands from other unix scripts to the server
Fsclient will take a failsafe command and its arguments as parameters, send it over TCP to the server and print the response to standard out.

Must have an interactive mode
In interactive mode, it prompts the user for a failsafe command, sends it to the server, retreives the response, prints the response and prompts the user again until the user enters exit. The –interactive option will do this.

Must have a data_only option
Sometimes we don't want to see the entire response message but just the data field. The `--data-only` option will do this.

Must have an auto_lock option
There should also be an option for auto locking when starting the interactive mode. The `--auto-lock` option will do this.

6.1.2 Implementation

This section deals with the non-trivial implementation details.

There are no contraints to program speed so the FSClient has been implemented

in ruby. Fsclient uses rubys standard command "$stdin.gets" to prompt the user for input. The TCP connection has been implemented with eventmachine.

Chapter 7

Upload File Script - an example of a user script

This chapter analyses the requirements and discuss various design alternatives of the upload script. The implementation details of the chosen design will follow.

7.1 Analysis and Design

The upload file is an example of a user script. It must take a filepath as parameters and upload it to the flash on the satellite. There are no failsafe command to upload a file to the flash, so a script must written that uses several failsafe commands to achieve to overall goal.

Firstly the script must be divide the file in parts of the maximum data size and upload them to the satellite ram memory. After uploading to ram it must copy the data from the ram to the flash and lastly calculate the checksums to ensure that everything got copied correctly.

The staff typically wants to upload a new version of the nominal mode. There are different ways of storing the nominal mode in a file. It can be stored as binary data in a file or as hex formatted data in a file.

This script will assume that the data is stored as a binary file. Therefore addresses to the ram and flash memory must also be given as arguments.

During upload the user must be notified with the overall progress.

7.2 Implementation

The script is an executable file written in ruby with the following path `"scripts/upload_file"`.

It takes three arguments: token, filepath and ram_address flash_address.

It validates that these three arguments have been given, that the file exists and that the address is a valid address.

Then it determines the size of the file, and how many individual uploads there is needed to upload the entire file.

For each upload the bytes to be uploaded are read from the file and uploaded via fsclient.

The progress is printed.

If the upload went well the next part of the file will be uploaded.

If the upload went bad the script is stopped and the user is notified.

When the upload has finished it will start the copying from ram to flash.

When the copying is done the checksum will be calculated to ensure that everything is OK.

Maximum data size of 20 bytes
The upload script has a maximum data size of 20 B instead of the allowed 1020 B. This is because the current implementation of the failsafe software has some problems with data being send too fast. The current workaround is to sleep for 0.2 seconds inbetween each byte written. It will takes 204 seconds to upload 1020 bytes and in that time space the satellite will reset and the command will fail. In contrast it takes 4 seconds to upload 20 bytes.

The maxumum data size should be changed when the issue has been fixed in the failsafe software.

Chapter 8

Tests and Results

Now that the implementation is complete, we should test that it meets the requirements of the system and ultimately give us an idea of the success of the project.

This chapter will start by stating the test stragedy. Then the test setup is described along with an example of a test case. Then all test cases are outlined. The chapter ends with a summary of the test results.

All test cases can be found in Appendix C.

8.1 Test Stragedy

The stragedy has been to perform functional tests of all requirements. The tests are combined in integration tests as some requirements span over the individual subsystems.

Unit tests have been performed on the command validations.

Lastly all failsafe commands have been tested with fsclient.

8.2 Test Setup

The setup has consisted of two, sometimes three, terminals. FSServer ran in the first terminal logging to standard out. Commands was then send via FSClient in the other terminals and the outcome of the test was determined based on the response and log messages of FSServer.

Unit tests was implemented in ruby with the built-in UnitTest library. To run the validation tests for example type `ruby test/lib/string_test.rb`

8.3 List of Test Cases

The test cases all follow the same format:

ACTION, EXPECTED, RESULT

If it makes sense a test case will have more data. For example, a failsafe command test case also states what has been written and read on the datalink:

FSClient args	`calculate_check_sum 0 128`
Datalyer 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 a7`
Response	`{"status":10,"data":2811364373,"message":"ACK"}`
Result	`success`

Here is a list of all the test cases:

- FSServer
  - Daemonization
  - Custom logfile
  - Multiple tcp clients, lock mechanism, token timeout, no-response and broadcast
  - Command parsing
  - Sequentially executed satellite commands

  - Command validation
  - Spaced hex

- Commands
  - Calculate Check Sum
  - Call Function
  - Copy To Flash
  - Copy To Ram
  - Delete Flash Block
  - Download
  - Download Sib
  - Execute
  - Flash Test
  - Health Status
  - List Scripts
  - Lock
  - Ram Test
  - Read register
  - Read sensor
  - Reset
  - Reset Sib
  - Run Script
  - Set Autoreset
  - Sleep
  - Unlock
  - Unlock Flash
  - Upload
  - Upload Sib
  - Write Register
- FSGui
- FSClient
- Upload File Script

8.4 Test summary

This section will summarize the test results.

FSServer test results
All integration tests passed.
All unit tests passed.
All but 3 command tests passed:

- **Flash Test** - Does not return a test result in the data.
- **Health Status** - 20 bytes is read from the datalink instead of 16 bytes.
- **Upload Sib** - Flash Write Error when uploading a new sib

FSServer works as expected and according to the requirements, the failsafe commands flash_test, health_status and upload_sib do not however.

FSGUI test results
All tests passed.
FSGui works as expected and according to the requirements

FSClient test results
All tests passed.
Fsclient works as expected and according to the requirements

Upload File Script test results
All tests passed.
The upload scripts works as expected and according to the requirements

Overall test results
All but 3 tests passed. The 3 tests are concerned with the response of some failsafe commands and does not deal with the implementation of this project.

The result is that the overall system works as expected and according to the requirements, but that the documentation of 3 failsafe commands is not uptodate with the implementation or vice verca.

Chapter 9

Conclusion

9.1 Achievements

The requirements of the project have been dictated by or based on meetings with the DTUSat-2 staff. Based on these requirements an overall design was chosen among various alternatives. The overall design was broken into four parts, a server part, a command client part, a GUI part and a custom scripts part. Each part was further analysed, discussed and designed before being implemented.

The system was testet with test cases that covered all requirements. All but 3 tests passed. The 3 tests was concerned with failsafe commands responses. The result is that the overall system works as expected and according to the requirements, but that the documentation of 3 failsafe commands is not uptodate with the implementation or vice verca. Therefore, this project can be considered a success.

The greatest achievement in this project is the flexibility of the user scripts. The script author can take advantage of all the features and libraries of his or her favorite programming language and will be able to quickly write scripts if the satellite goes into failsafe mode.

9.2 Further work

Authentication
This projects has not dealt with user access to the server. There is no access constraints to who can log on to the server and send commands to the satellite. This should be considered.

Encryption
The data between the FSServer and the TCP clients are not encrypted. This could fairly easy be achieved by using SSL or TLS. However, it is not expected that the failsafe mode will be use often, so it might be overkill to do more than absolutely necessary.

Upload hex formatted file script
The upload script in this report uploads a binary file to the satellite. The staff will likely have a hex-formatted version of the file, so this could be one of the first user scripts to implement.

State of the software
Although the software meet all requirements, some modifications needs to made before the software can be considered production ready. The FSServer currently only communicates with the development board via a serialport implementation of the protocol layer. This should be substituted with a radio implementation.

There is an issue when uploading data to the development board that are currently being worked-around with a sleep of 0.2 seconds inbetween each byte being written. This may not be a problem with the radio implementation but should be tested and the sleep should be removed when fixed.

9.3 Conclusion

Ultimately the staff of the DTUSat-2 project is now able to operate the satellite in failsafe mode from their desktop computers via a command program og a GUI. Using the GUI they can monitor the health status in a graphical overview. Lastly and perhaps most importantly they are now able to write custom scripts in any programming language and have them executed on their local machines or on the Ground Station.

Appendix A

User Guide

Contains installation and operation instructions for the individual subsystems.

A.1 FSServer

Installation instructions on Ubuntu
FSServer needs ruby and some additional rubygems to run. Here are the commands to install the server.

```
sudo apt-get install ruby1.8 ruby1.8-dev rubygems1.8
sudo gems install eventmachine json daemons
```

Operation instructions
To run FSServer as a normal process, listen on `localhost:3000` and print the log messages to standard out run this command:

```
./fsserver
```

You can pass the following options to fsserver:

```
    --host=HOST (default is '0.0.0.0')
    --port=PORT (default is 3000)
    --timeout=TIMEOUT  (default is 30)
    --logfile=LOGFILE  (default is stdout)
```

To run the server as a daemon run this command:

```
./fsdaemon start
```

options to daemon mode are given after an extra double dash like this:

```
./fsdaemon start -- --option1=VALUE1 --options2=VALUE
```

A.2 FSClient

'Installation instructions on Ubuntu
The fsclient needs ruby, the eventmachine gem and the JSON gem. To install run this command:

```
sudo apt-get install ruby1.8 ruby1.8-dev rubygems1.8
sudo gems install eventmachine json
```

It is convenient to add the path of the fsclient executable to the PATH variable and a requirement to do so on the Ground Station in order for the upload script to work properly.

Operation instructions
Fsclient's help:

```
Usage: fsclient [options] <command [command_args ... ]>
    --host=HOST                  Server host (default is 0.0.0.0)
    --port=PORT                  Server port (default is 3000)
    --token=TOKEN                Token
    --timeout=SEC                Timeout option to command
    -i, --interactive            Interactive mode
    -d, --data-only              Only print data parameter
    -a, --auto-lock              Auto lock in interactive mode
    -n, --no-response            No-response option to command
```

A.3 FSClient

Installation instructions
Install a Java Virtual Machine for your system.

Operation instructions
Double click the executable jarfile

A.4 Upload script

Installation instructions
Needs ruby, the JSON gem and a working fsclient that has been added to the PATH environment variable.

Operation instructions
upload_file's help:

```
Usage: upload_file token filepath address
Description: Upload a file to an address in the satellites memory
Arguments:
    filepath (string)
    address (hexadecimal)
```

Appendix B

Failsafe Commands

A list of the 20 failsafe commands

- `calculate_check_sum (address, length)`
- `call_function (address,parameter)`
- `copy_to_flash (from,to,length)`
- `copy_to_ram (from,to,length)`
- `delete_flash_block (address)`
- `download (address, length)`
- `download_sib`
- `execute (address)`
- `flash_test (address)`
- `health_status`
- `ram_test (address, length)`
- `read_register (address)`

- `read_sensor (address)`
- `reset`
- `reset_sib`
- `set_autoreset (value)`
- `unlock_flash`
- `upload (address, data)`
- `upload_sib (data)`
- `write_register (address, data)`

Appendix C

Test Cases

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 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 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 0xefbeedfe0000000000000000000000000000000000000000000000000000
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 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

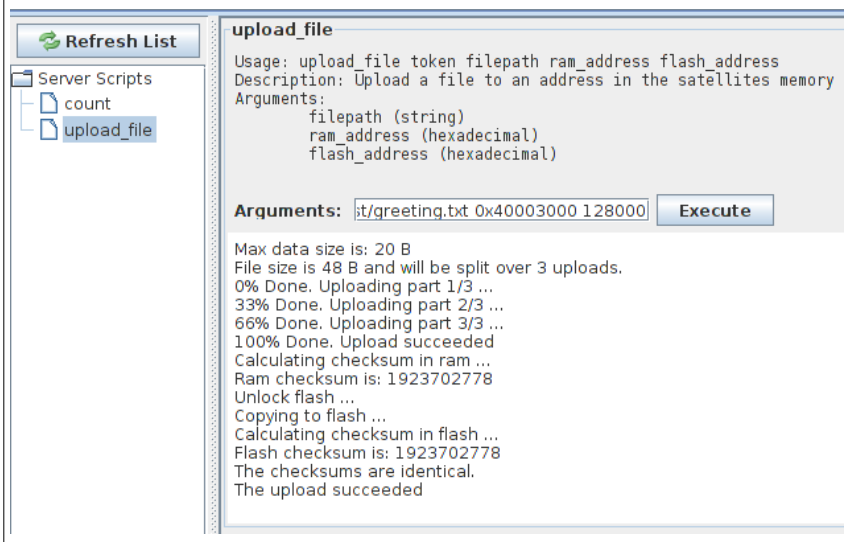
Upload File script

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: 
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!

Appendix D

Source Code

D.1 FSServer

fsdaemon

```
1 #!/usr/bin/ruby
2 require 'rubygems'
3 require 'daemons'
4
5 Daemons.run(File.dirname(__FILE__)+"/fsserver")
```

fsserver

```
1 #!/usr/bin/ruby
2 require 'optparse'
3 require 'rubygems'
4 require 'serialport'
5
6 options = {
7   :host => '0.0.0.0',
8   :port => 3000,
9   :logfile => STDOUT,
```

```
10    :timeout => 30
11  }
12
13  op = OptionParser.new do |opts|
14    opts.banner = "Usage: fsserver [options]"
15    opts.on("--host=HOST", String, "('0.0.0.0')")
                  {|o| options[:host] = o}
16    opts.on("--port=PORT", Integer, "(3000)")
                  {|o| options[:port] = o}
17    opts.on("--timeout=TIMEOUT", Integer, "(30)")
                  {|o| options[:timeout] = o}
18    opts.on("--logfile=LOGFILE", String, "(stdout)")
                  {|o| options[:logfile] = o}
19
20    opts.separator ""
21    opts.separator "Common options:"
22
23    opts.on_tail("-h", "--help", "Show this message") do
24      puts opts
25      exit
26    end
27
28    opts.on_tail("--version", "Show version") do
29      puts "FSServer Version 1.0"
30      puts "Kasper BjÃ¸rn Nielsen (s052808@student.dtu.dk)"
31      exit
32    end
33  end
34  op.parse!
35
36  # Setup root dir
37  ROOT_DIR = File.dirname(__FILE__)
38
39  # Setup log
40  require ROOT_DIR+"/lib/logger"
41  $LOG = FSLogger.new(options[:logfile])
42
43  # Setup thread abort
44  Thread.abort_on_exception = true
45
46  # Run server
47  require ROOT_DIR+"/lib/server"
48  Server.instance.start(options)
```

lib/command_parser.rb

```
require ROOT_DIR+'/lib/ext/string'
require ROOT_DIR+'/lib/commands/abstract_command'
require ROOT_DIR+'/lib/response_helpers'
require ROOT_DIR+'/lib/constants'
Dir.glob(ROOT_DIR+"/lib/commands/*.rb").each {|f| require f }

class CommandParser
  include ResponseHelpers
  include Constants

  def parse(raw)
    id = nil
    token = nil
    command = Commands::Unknown.new

    begin
      request = JSON.parse(raw)
      id = request["id"]
      token = request["token"]
      cmd_string, *arguments = *request["data"].split(" ")

      opts = arguments.map {|a| a[0].chr == "-" ? a : nil};opts.delete(nil)
      options = {}
      opts.each do |o|
        key,val = o.split("=")
        options[key[2..-1]] = val || true
      end
      arguments = arguments.delete_if {|a| a[0].chr == "-"}

      if cmd_string[0].chr.match(/[a-zA-Z]/)
        command = eval("Commands::#{cmd_string.camelize}.new(*arguments)")
      end
    rescue ArgumentError => e
      command = Commands::WrongNumberOfArguments.new
    rescue => e
    end

```

```
38      # Set defaults
39      options["timeout"] ||= DEFAULT_TIMEOUT
40      options["no-response"] ||= false
41      command.options = options
42      command.id = id
43
44      return id, token, command
45    end
46
47  end
```

lib/commands/abstract_command.rb

```
1   require ROOT_DIR+'/lib/response_helpers'
2   require ROOT_DIR+'/lib/constants'
3   require ROOT_DIR+'/lib/ext/string'
4   require ROOT_DIR+"/lib/ext/fixnum"
5
6   class AbstractCommand
7     include ResponseHelpers
8     include Constants
9
10    attr_accessor :validation_errors, :timeout, :client, :
        id, :options
11
12    def execute
13      caller.send response(@id, STATUS_OK)
14    end
15
16    def unpack(data)
17      data
18    end
19
20    def satellite_command(cmd)
21      SerialRequestHandler.instance.request(cmd, @options)
          do |return_code,length,data|
22        if return_code == STATUS_SERIALPORT_NOT_CONNECTED
23          @client.send response(@id, return_code, data)
24        else
25          if block_given?
26            yield(return_code,length,data)
27          else
28            data = unpack(data)
```

```
29            @client.send_response(@id, return_code, data)
30          end
31        end
32      end
33    end
34
35    def validate
36    end
37
38    def valid?
39      @validation_errors = []
40      validate_positive_integer "Timeout", @options["
          timeout"]
41      validate
42      @validation_errors.empty?
43    end
44
45    def validate_addressable(name, var, bytes=4)
46      @validation_errors << "#{name} must be addressable
          (<= #{bytes} bytes)" unless !var.nil? && var.
          addressable?(bytes)
47    end
48
49    def validate_positive(name, var)
50      @validation_errors << "#{name} must be a positive
          number" unless !var.nil? && var.positive?
51    end
52
53    def validate_positive_integer(name, var)
54      @validation_errors << "#{name} must be a positive
          integer" unless !var.nil? && var.positive_integer?
55    end
56
57    def validate_positive_hex(name, var)
58      @validation_errors << "#{name} must be a positive hex
          " unless !var.nil? && var.positive_hex?
59    end
60
61    def validate_byte_length(name, var, max=4)
62      @validation_errors << "#{name} is too many bytes (<=
          #{max})" if var.nil? || var.byte_length > max
63    end
64
65    def validate_max_value(name, var, max)
```

```
66      @validation_errors << "#{name} is too long (<= #{max
            })" if var.nil? || var.int_or_hex > max
67    end
68
69  end
```

lib/commands/calculate_check_sum.rb

```
1   module Commands
2     class CalculateCheckSum < AbstractCommand
3       def initialize(address, length)
4         @address = address
5         @length = length
6       end
7
8       def validate
9         validate_addressable "Address", @address
10        validate_positive "Length", @length
11        validate_byte_length "Length", @length, 4
12      end
13
14      def execute
15        input  = [
16          "0a",                    # cmd
17          "00",                    # uplink
18          "08 00",                 # data length
19          @address.spaced_hex.split.reverse,  # address
20          @length.spaced_hex.split.reverse,   # length
21          "CD"
22        ]
23
24        satellite_command(input) do |return_code,length,
              data|
25          if return_code == FS_CALCULATE_CHECK_SUM
26            data = data.unpack("V").first # Unpack as 4
                  bytes little-endian
27          end
28
29          @client.send response(@id, return_code, data)
30        end
31      end
32    end
33  end
```

lib/commands/call_function.rb

```
1  module Commands
2    class CallFunction < AbstractCommand
3      def initialize(address, parameter)
4        @address = address
5        @parameter = parameter
6      end
7
8      def validate
9        validate_addressable "Address", @address
10       validate_positive "Parameter", @parameter
11       validate_byte_length "Parameter", @parameter, 4
12     end
13
14     def execute
15       input  = [
16         "03",                      # cmd
17         "00",                      # uplink
18         "08␣00",                   # data length
19         @address.spaced_hex.split.reverse,     # address
20         @parameter.spaced_hex.split.reverse,   # parameter
21         "CD"
22       ]
23
24       satellite_command(input) do |return_code,length,
             data|
25         if return_code == FS_CALL_FUNCTION
26           # Unpack as 4 bytes little-endian
27           data = data.unpack("V").first
28         end
29
30         @client.send response(@id, return_code, data)
31       end
32     end
33   end
34 end
```

lib/commands/copy_to_flash.rb

```
1  module Commands
2    class CopyToFlash < AbstractCommand
3      EXTERNAL_FLASH_RANGE = (0x80000000..0x801e8480) # 2
           MB
```

```
4       INTERNAL_RAM_RANGE    = (0x40000000..0x40003e80) # 16
            K
5       INTERNAL_FLASH_RANGE = (0..256000)               # 256
            K
6       INTERNAL_LENGTHS = ["512","1024","2048","4096"]
7
8       def initialize(from,to,length)
9         @from = from
10        @to = to
11        @length = length
12      end
13
14      def validate
15        validate_addressable "From address", @from
16        validate_addressable "To address", @to
17        validate_positive "Length", @length
18        validate_byte_length "Length", @length, 4
19
20        if EXTERNAL_FLASH_RANGE.include?(@to.int_or_hex)
21          @validation_errors << "For external flash, length
                must be a multipler of 2" unless (@length.
                int_or_hex % 2 == 0)
22        elsif INTERNAL_FLASH_RANGE.include?(@to.int_or_hex)
23          @validation_errors << "For internal flash, source
                address must be within internal RAM" unless
                INTERNAL_RAM_RANGE.include?(@from.int_or_hex)
24          @validation_errors << "For internal flash, length
                must be one of #{INTERNAL_LENGTHS.join(", ")}
                " unless INTERNAL_LENGTHS.include?(@length)
25        else
26          @validation_errors << "To address must be within
                the external of interal flash range"
27        end
28      end
29
30      def execute
31        input  = [
32          "06",           # cmd
33          "00",           # uplink
34          "0c 00",        # data length
35          @from.spaced_hex.split.reverse,
36          @to.spaced_hex.split.reverse,
37          @length.spaced_hex.split.reverse,
38          "CD"
```

```
39        ]
40
41        satellite_command(input)
42      end
43    end
44  end
```

lib/commands/copy_to_ram.rb

```
1  module Commands
2    class CopyToRam < AbstractCommand
3
4      def initialize(from,to,length)
5        @from = from
6        @to = to
7        @length = length
8      end
9
10     def validate
11       validate_addressable "From address", @from
12       validate_addressable "To address", @to
13       validate_positive "Length", @length
14       validate_byte_length "Length", @length, 4
15     end
16
17     def execute
18       input  = [
19         "07",            # cmd
20         "00",            # uplink
21         "0c 00",         # data length
22         @from.spaced_hex.split.reverse,
23         @to.spaced_hex.split.reverse,
24         @length.spaced_hex.split.reverse,
25         "CD"
26       ]
27
28       satellite_command(input)
29     end
30   end
31 end
```

lib/commands/delete_flash_block.rb

```
1  module Commands
2    class DeleteFlashBlock < AbstractCommand
3      def initialize(address)
4        @address = address
5      end
6
7      def validate
8        validate_addressable "Address", @address
9      end
10
11     def execute
12       input  = [
13         "0b",                    # cmd
14         "00",                    # uplink
15         "04 00",                 # data length
16         @address.spaced_hex.split.reverse,  # address
17         "CD"
18       ]
19
20       satellite_command(input)
21     end
22   end
23 end
```

lib/commands/download.rb

```
1  module Commands
2    class Download < AbstractCommand
3
4      def initialize(address,length)
5        @address = address
6        @length = length
7      end
8
9      def validate
10       validate_addressable "Address", @address
11       validate_positive "Length", @length
12       validate_max_value "Length", @length,
             FS_MAX_DATA_SIZE
13     end
14
15     def execute
16       input  = [
```

```
17          "09",           # cmd
18          "00",           # uplink
19          "08 00",        # data length
20          @address.spaced_hex.split.reverse,
21          @length.spaced_hex.split.reverse,
22          "CD"
23        ]
24
25        satellite_command(input) do |return_code,length,
              data|
26          if return_code == FS_DOWNLOAD
27            data = data.unpack("C"*length) # Unpack as 1
                  byte chars
28          end
29          @client.send_response(@id, return_code, data)
30        end
31      end
32    end
33  end
```

lib/commands/download_sib.rb

```
1  module Commands
2    class DownloadSib < AbstractCommand
3      def execute
4        satellite_command("10 00 00 00 CD")
5      end
6
7      def unpack(data)
8        data.unpack("C"*32) # Unpack as 32 unsigned chars
9      end
10   end
11 end
```

lib/commands/execute.rb

```
1  module Commands
2    class Execute < AbstractCommand
3
4      def initialize(address)
5        @address = address
6      end
7
```

```
8       def validate
9         validate_addressable "Address", @address
10      end
11
12      def execute
13        input  = [
14          "02",                   # cmd
15          "00",                   # uplink
16          "04␣00",                # data length
17          @address.spaced_hex.split.reverse,  # address
18          "CD"
19        ]
20
21        satellite_command(input)
22      end
23    end
24  end
```

lib/commands/flash_test.rb

```
1   module Commands
2     class FlashTest < AbstractCommand
3       def initialize(address)
4         @address = address
5       end
6
7       def validate
8         validate_addressable "Address", @address
9       end
10
11      def execute
12        input  = [
13          "0f",                   # cmd
14          "00",                   # uplink
15          "04␣00",                # data length
16          @address.spaced_hex.split.reverse,  # address
17          "CD"
18        ]
19
20        satellite_command(input) do |return_code,length,
              data|
21          # Unpack as 4 bytes little-endian
22          data = data.unpack("V")
```

```
23          @client.send_response(@id, return_code, data)
24        end
25      end
26    end
27  end
```

lib/commands/health_status.rb

```
1   module Commands
2     class HealthStatus < AbstractCommand
3
4       def execute
5         input = "13 00 00 00 CD"
6
7         satellite_command(input) do |return_code,length,
              data|
8           if return_code == FS_HEALTH_STATUS
9             # Unpack as 4 chars and 6 little-endian shorts
10            data = data.unpack("CCCCvvvvvv")
11          end
12
13          @client.send_response(@id, return_code, data)
14        end
15      end
16    end
17  end
```

lib/commands/list_scripts.rb

```
1   module Commands
2     class ListScripts < AbstractCommand
3       def execute
4         list = Dir.glob("scripts/**/*").select do |f|
5           File.executable?(f) && !File.directory?(f)
6         end.sort.map do |f|
7           {
8             :path => f[8..-1], # Remove scripts/
9             :help => `#{File.expand_path(f)} --help`
10          }
11        end
12        @client.send_response(@id, STATUS_OK, list)
13      end
14    end
```

```
15 end
```

lib/commands/lock.rb

```
1  module Commands
2    class Lock < AbstractCommand
3      def execute
4        TokenHandler.instance.token = generate_token
5        @client.send response(@id,STATUS_OK,TokenHandler.
             instance.token)
6      end
7
8      private
9      def generate_token(len=16)
10       chars = ("a"..."z").to_a + ("A"..."Z").to_a + ("0"..."
             9").to_a
11       newpass = ""
12       1.upto(len) { |i| newpass << chars[rand(chars.size
             -1)] }
13       newpass
14     end
15   end
16 end
```

lib/commands/ram_test.rb

```
1  module Commands
2    class RamTest < AbstractCommand
3      def initialize(address, length)
4        @address = address
5        @length = length
6      end
7
8      def validate
9        validate_addressable "Address", @address
10       validate_byte_length "Length", @length, 4
11     end
12
13     def execute
14       input  = [
15         "0e",                  # cmd
16         "00",                  # uplink
17         "08␣00",               # data length
```

```
18          @address.spaced_hex.split.reverse,   # address
19          @length.spaced_hex.split.reverse,    # data
20          "CD"
21        ]
22
23        satellite_command(input) do |return_code,length,
              data|
24          # Unpack as 4 bytes little-endian
25          data = data.unpack("V")
26          @client.send response(@id, return_code, data)
27        end
28      end
29    end
30  end
```

lib/commands/read_register.rb

```
1   module Commands
2     class ReadRegister < AbstractCommand
3       def initialize(address)
4         @address = address
5       end
6
7       def validate
8         validate_addressable "Address", @address
9       end
10
11      def execute
12        input  = [
13          "0c",                    # cmd
14          "00",                    # uplink
15          "04␣00",                 # data length
16          @address.spaced_hex.split.reverse,   # address
17          "CD"
18        ]
19
20        satellite_command(input) do |return_code,length,
              data|
21          if return_code == FS_READ_REGISTER
22            # Unpack as 4 byte little endian
23            data = data.unpack("V").first
24          end
25          @client.send response(@id, return_code, data)
```

```
26        end
27      end
28    end
29  end
```

lib/commands/read_sensor.rb

```
1  module Commands
2    class ReadSensor < AbstractCommand
3      def initialize(address)
4        @address = address
5      end
6
7      def validate
8        validate_addressable "Address", @address
9      end
10
11     def execute
12       input  = [
13         "14",                    # cmd
14         "00",                    # uplink
15         "04 00",                 # data length
16         @address.spaced_hex.split.reverse,  # address
17         "CD"
18       ]
19
20       satellite_command(input) do |return_code,length,
             data|
21         if return_code == FS_READ_SENSOR
22           # Unpack as one 4 char little-endian long
23           data = data.unpack("V").first
24         end
25
26         @client.send response(@id, return_code, data)
27       end
28     end
29   end
30 end
```

lib/commands/reset.rb

```
1  module Commands
2    class Reset < AbstractCommand
```

```
3       def execute
4         satellite_command("01␣00␣00␣00␣CD")
5       end
6     end
7   end
```

lib/commands/reset_sib.rb

```
1   module Commands
2     class ResetSib < AbstractCommand
3       def execute
4         satellite_command("12␣00␣00␣00␣CD")
5       end
6     end
7   end
```

lib/commands/run_script.rb

```
1   require 'pty'
2   require 'expect'
3
4   module Commands
5     class RunScript < AbstractCommand
6       attr_accessor :token
7
8       def initialize(script, *args)
9         @script = script
10        @args = args
11        @token = TokenHandler.instance.token
12      end
13
14      def validate
15        script_exists = false
16        Dir.glob(ROOT_DIR+"/scripts/**/*").each do |f|
17          cmd = File.expand_path(f)
18          if cmd == File.expand_path(File.join("scripts/",
                @script)) # Does the script exists?
19            @cmd = cmd
20            script_exists = true
21          end
22        end
23        @validation_errors << "Unknown␣script" unless
              script_exists
```

```
24      end
25
26      def execute
27        begin
28          PTY.spawn(@cmd, @token, *@args) do |r, w, pid|
29            loop {
30              out = r.expect(%r/^.+\n$/io)
31              @client.send response(@id,STATUS_OK,out[0],:
                    partial => true) unless out.nil?
32            }
33          end
34        rescue PTY::ChildExited => e
35          status = (e.status.to_i == 0) ? 100 : 101
36          @client.send response(@id,status)
37          return;
38        end
39      end
40    end
41  end
```

lib/commands/set_autoreset.rb

```
1   module Commands
2     class SetAutoreset < AbstractCommand
3       def initialize(value)
4         @value = value
5       end
6
7       def validate
8         @validation_errors << "Value must be either 01 (
              enable) or 00 (disable)" unless ["00","01"].
              include?(@value)
9       end
10
11      def execute
12        input  = [
13              "04",           # cmd
14              "00",           # uplink
15              "01 00",        # data length
16              @value,         # value
17              "CD"
18        ]
19
```

```
20          satellite_command(input)
21        end
22      end
23    end
```

lib/commands/sleep.rb

```
1   module Commands
2     class Sleep < AbstractCommand
3       def initialize(seconds)
4         @seconds = seconds
5       end
6
7       def validate
8         validate_positive_integer "Seconds", @seconds
9       end
10
11      def execute
12        TokenHandler.instance.stop_timer
13        sleep(@seconds.to_i)
14        TokenHandler.instance.start_timer
15        @client.send response(@id,STATUS_OK, nil)
16      end
17    end
18  end
```

lib/commands/unknown.rb

```
1   module Commands
2     class Unknown < AbstractCommand
3       def execute
4         @client.send response(@id, STATUS_UNKNOWN_COMMAND)
5       end
6     end
7   end
```

lib/commands/unlock.rb

```
1   module Commands
2     class Unlock < AbstractCommand
3       def execute
4         TokenHandler.instance.token = nil
5         @client.send response(@id, STATUS_SERVER_UNLOCKED)
```

```
6         @client.broadcast_message("server_unlocked",
              STATUS_SERVER_UNLOCKED)
7       end
8     end
9   end
```

lib/commands/unlock_flash.rb

```
1   module Commands
2     class UnlockFlash < AbstractCommand
3       def execute
4         satellite_command("05 00 00 00 CD")
5       end
6     end
7   end
```

lib/commands/upload.rb

```
1   module Commands
2     class Upload < AbstractCommand
3       def initialize(address,data)
4         @address = address
5         @data = data
6       end
7
8       def validate
9         validate_addressable "Address", @address
10        validate_positive_hex "Data", @data
11        validate_byte_length "Data", @data, (
              FS_MAX_DATA_SIZE - 4)
12      end
13
14      def execute
15        input  = [
16              "08",           # cmd
17              "00",           # uplink
18              (@data.byte_length+4).spaced_hex(2).split.
                    reverse, # data length
19              @address.spaced_hex.split.reverse,
20              @data.spaced_hex(@data.byte_length).split,
21              "CD"
22        ]
23
```

```
24        satellite_command(input)
25      end
26    end
27  end
```

lib/commands/upload_sib.rb

```
1  module Commands
2    class UploadSib < AbstractCommand
3
4      def initialize(data)
5        @data = data
6      end
7
8      def validate
9        validate_positive_hex "Data", @data
10       validate_byte_length "Data", @data, 28
11     end
12
13     def execute
14       input = [
15         "11",
16         "00",
17         "1c␣00",
18         @data.spaced_hex(28).split.reverse,
19         "CD"
20       ]
21       satellite_command(input)
22     end
23   end
24 end
```

lib/commands/write_register.rb

```
1  module Commands
2    class WriteRegister < AbstractCommand
3      def initialize(address, data)
4        @address = address
5        @data = data
6      end
7
8      def validate
9        validate_addressable "Address", @address
```

```
10        validate_byte_length "Data", @data, 4
11      end
12
13      def execute
14        input = [
15          "0d",                                  # cmd
16          "00",                                  # uplink
17          "08 00",                               # data length
18          @address.spaced_hex.split.reverse,     # address
19          @data.spaced_hex.split.reverse,        # data
20          "CD"
21        ]
22
23        satellite_command(input)
24      end
25    end
26  end
```

lib/commands/wrong_number_of_arguments.rb

```
1  module Commands
2    class WrongNumberOfArguments < AbstractCommand
3      def execute
4        @client.send response(@id,
             STATUS_WRONG_NUMBER_OF_ARGUMENTS)
5      end
6    end
7  end
```

lib/constants.rb

```
1  module Constants
2
3    # Default timeout in seconds for a serialport request
4    DEFAULT_TIMEOUT              = "500"
5
6    # Server Status Codes
7    STATUS_OK                    = 100
8    STATUS_ERROR                 = 101
9    STATUS_IS_LOCKED             = 102
10   STATUS_MUST_LOCK             = 103
11   STATUS_WRONG_NUMBER_OF_ARGUMENTS     = 104
12   STATUS_UNKNOWN_COMMAND       = 105
```

```
13  STATUS_TIMEOUT                = 106
14  STATUS_VALIDATION_ERROR       = 107
15  STATUS_SERVER_UNLOCKED        = 108
16  STATUS_UNKNOWN_SCRIPT         = 109
17  STATUS_SERIALPORT_NOT_CONNECTED = 110
18  STATUS_JSON_PARSE_ERROR       = 111
19
20  # FS
21  FS_MAX_DATA_SIZE              = 1024
22  FS_CMD_SIZE                   = 1
23  FS_DIRECTION                  = 1
24  FS_DATA_LENGTH_SIZE           = 2
25
26  # Packet header size
27  FS_PACKET_HEADER_SIZE         = FS_CMD_SIZE + FS_DIRECTION
        + FS_DATA_LENGTH_SIZE
28
29  # Maximal packet size
30  FS_MAX_PACKET_SIZE            = FS_PACKET_HEADER_SIZE +
        FS_MAX_DATA_SIZE
31
32  # Size of the SIB block
33  FS_SIB_SIZE                   = 32
34
35  # DIRECTION CODES
36  FS_UP_LINK                    = 0
37  FS_DOWN_LINK                  = 255
38
39  # Command Codes
40  FS_RESET                  = 1
41  FS_EXECUTE                = 2
42  FS_CALL_FUNCTION          = 3
43  FS_SET_AUTORESET          = 4
44  FS_UNLOCK_FLASH           = 5
45  FS_COPY_TO_FLASH          = 6
46  FS_COPY_TO_RAM            = 7
47  FS_UPLOAD                 = 8
48  FS_DOWNLOAD               = 9
49  FS_CALCULATE_CHECK_SUM    = 10
50  FS_DELETE_FLASH_BLOCK     = 11
51  FS_READ_REGISTER          = 12
52  FS_WRITE_REGISTER         = 13
53  FS_RAM_TEST               = 14
54  FS_FLASH_TEST             = 15
```

```
55  FS_DOWNLOAD_SIB              = 16
56  FS_UPLOAD_SIB                = 17
57  FS_RESET_SIB                 = 18
58  FS_HEALTH_STATUS             = 19
59  FS_READ_SENSOR               = 20
60
61
62  # Return Codes
63  FS_NO_ERROR                    = 0xFF
64  FS_UNDEFINED_CMD               = 0xFE
65  FS_RAM_ERROR                   = 0xFD
66  FS_FLASH_WRITE_ERROR           = 0xFC
67  FS_FLASH_DELETE_ERROR          = 0xFB
68  FS_PACKET_LENGTH_ERROR         = 0xFA
69  FS_RAM_ADDRESS_ERROR           = 0xF9
70  FS_FLASH_ADDRESS_ERROR         = 0xF8
71  FS_FLASH_OPERATION_LOCKED      = 0xF7
72  FS_ADDRESS_ERROR               = 0xF6
73  FS_UNKONWN_SENSOR              = 0xF5
74
75  # Protocol Codes
76  PRO_OK=                         0
77  PRO_FAILURE=                   -1
78  PRO_TIMEOUT=                   -2
79  PRO_TOOLARGE=                  -3
80  PRO_NOROOM=                    -4
81  PRO_FAILSAFE=                  -5
82  PRO_NOT_FAILSAFE=              -6
83  PRO_NO_SIGNAL=                 -7
84  PRO_NOT_READY=                 -8
85
86  # Messages
87  MESSAGES = {
88    STATUS_OK                       => "OK",
89    STATUS_ERROR                    => "Error",
90    STATUS_IS_LOCKED                => "Server␣is␣locked",
91    STATUS_MUST_LOCK                => "You␣must␣lock␣the␣
          server␣before␣executing␣commands",
92    STATUS_WRONG_NUMBER_OF_ARGUMENTS     => "Wrong␣number␣
          of␣arguments",
93    STATUS_UNKNOWN_COMMAND          => "Unknown␣command",
94    STATUS_TIMEOUT                  => "Timeout",
95    STATUS_VALIDATION_ERROR         => "Validation␣error",
```

```
96      STATUS_SERVER_UNLOCKED      => "Server has been
            unlocked",
97      STATUS_UNKNOWN_SCRIPT       => "Unknown script",
98      STATUS_SERIALPORT_NOT_CONNECTED => "Serial port not
            connected",
99      STATUS_JSON_PARSE_ERROR => "JSON parse error",
100
101     FS_RESET                    => "ACK",
102     FS_EXECUTE                  => "ACK",
103     FS_CALL_FUNCTION            => "ACK",
104     FS_SET_AUTORESET            => "ACK",
105     FS_UNLOCK_FLASH             => "ACK",
106     FS_COPY_TO_FLASH            => "ACK",
107     FS_COPY_TO_RAM              => "ACK",
108     FS_UPLOAD                   => "ACK",
109     FS_DOWNLOAD                 => "ACK",
110     FS_CALCULATE_CHECK_SUM      => "ACK",
111     FS_DELETE_FLASH_BLOCK       => "ACK",
112     FS_READ_REGISTER            => "ACK",
113     FS_WRITE_REGISTER           => "ACK",
114     FS_RAM_TEST                 => "ACK",
115     FS_FLASH_TEST               => "ACK",
116     FS_DOWNLOAD_SIB             => "ACK",
117     FS_UPLOAD_SIB               => "ACK",
118     FS_RESET_SIB                => "ACK",
119     FS_HEALTH_STATUS            => "ACK",
120     FS_READ_SENSOR              => "ACK",
121
122     FS_NO_ERROR                   => "No error",
123     FS_UNDEFINED_CMD              => "Undefined command",
124     FS_RAM_ERROR                  => "Ram error",
125     FS_FLASH_WRITE_ERROR          => "Flash write error",
126     FS_FLASH_DELETE_ERROR         => "Flash delete error",
127     FS_PACKET_LENGTH_ERROR        => "Packet length error",
128     FS_RAM_ADDRESS_ERROR          => "Ram address error",
129     FS_FLASH_ADDRESS_ERROR        => "Flash address error",
130     FS_FLASH_OPERATION_LOCKED => "Flash operation locked"
            ,
131     FS_ADDRESS_ERROR              => "Address error",
132     FS_UNKONWN_SENSOR             => "Unknown sensor"
133   }
134 end
```

lib/ext/fixnum.rb

```
1  class Fixnum
2    def spaced_hex(bytes=4)
3      "0x#{self.to_s(16)}".spaced_hex(bytes)
4    end
5  end
```

lib/ext/string.rb

```
1  class NotAddressableError < StandardError; end
2  class NotANumberError < StandardError; end
3
4  class String
5    def camelize
6      self.split(/[^a-z0-9]/i).map{|w| w.capitalize}.join
7    end
8
9    def positive?
10     self.positive_integer? || self.positive_hex?
11   end
12
13   def positive_integer?
14     self.match(/^\d+$/) != nil
15   end
16
17   def positive_hex?
18     self.match(/^(0x(\d|[a-fA-F])+)$/) != nil
19   end
20
21   def addressable?(bytes=4)
22     addressable_int?(bytes) || addressable_hex?(bytes)
23   end
24
25   def addressable_int?(bytes=4)
26     self.positive_integer? and self.to_i.to_s(16).size <=
         bytes*2
27   end
28
29   def addressable_hex?(bytes=4)
30     self.positive_hex? and self.hex.to_s(16).size <=
         bytes*2
31   end
32
```

```
33    def int_or_hex(bytes=4)
34      if self.addressable_hex?(bytes)
35        self.hex
36      elsif self.addressable_int?(bytes)
37        self.to_i
38      else
39        raise NotANumberError
40      end
41    end
42
43    def byte_length
44      val = if self.positive_hex?
45        self.hex
46      elsif self.positive_integer?
47        self.to_i
48      else
49        raise NotANumberError
50      end
51      l = val.to_s(16).size
52      l += 1 unless l%2==0
53      l/2
54    end
55
56    def spaced_hex(bytes=4)
57      length = bytes*2
58      raise NotAddressableError unless self.addressable?(
            bytes)
59
60      # Prepend zeroes
61      s = int_or_hex(bytes).to_s(16)
62      s = "0"*(length-s.size)+s
63
64      # Divide after each two
65      r = []
66      bytes.times {|n| r << s[n*2..n*2+1]}
67      r.join("␣")
68    end
69
70  end
```

lib/logger.rb

```
1  require 'fileutils'
```

```
2  class FSLogger
3
4    attr_accessor :filename, :logformat, :timeformat
5
6    def initialize(filename, logformat = "[%time] %message"
       , timeformat = "%d/%m/%Y %H:%M:%S")
7      @filename = filename
8      @logformat = logformat
9      @timeformat = timeformat
10   end
11
12   def log(s)
13     time = Time.now.strftime(@timeformat)
14     msg = @logformat.gsub("%time",time).gsub("%message",s
         )
15
16     if @filename == STDOUT
17       $stdout.puts(msg)
18     else
19       FileUtils.touch(@filename) unless File.exists?(
           @filename)
20       File.open(@filename, 'a') do |f|
21         f.puts(msg)
22       end
23     end
24   end
25 end
26
27 module Loggable
28   def log(s) $LOG.log(s); end
29 end
```

lib/processing_queue.rb

```
1  require "thread"
2
3  class ProcessingError < StandardError; end
4
5  module ProcessingQueue
6
7    def setup_processing_queue
8      @process_queue = []
9      @process_queue_mutex = Mutex.new
```

```
10      @process_queue_started = false
11    end
12
13    def enqueue(request, &callback)
14      if ready?
15        @process_queue_mutex.synchronize {
16          @process_queue << [request, callback]
17        }
18        process_next unless @process_queue_started
19      else
20        not_ready(request, &callback)
21      end
22    end
23
24    private
25    def process_next
26      @process_queue_mutex.synchronize {
27        item = @process_queue.shift
28        @process_queue_started = !item.nil?
29        if @process_queue_started
30          begin
31            process(item[0], &item[1])
32          rescue ProcessingError => e
33            @process_queue_started = false
34          end
35        end
36      }
37      process_next if @process_queue_started
38    end
39
40  end
```

lib/response_helpers.rb

```
1   require 'rubygems'
2   require 'json'
3   require ROOT_DIR + "/lib/constants"
4
5   module ResponseHelpers
6     include Constants
7
8     def message(type,status,data=nil,options = {})
```

```
9       {:type => type, :status => status, :message =>
            MESSAGES[status], :data => data}.merge(options)
10    end
11
12    def response(id, status, data=nil, options = {})
13      message('response', status, data, {:id => id}.merge(
            options))
14    end
15
16  end
```

lib/serial_request_handler.rb

```
1   require 'singleton'
2
3   # TODO: REPLACE WITH C EXTENSION TO REAL DATALINK
4   require ROOT_DIR + '/link_fs/link_fs_devel'
5
6   require ROOT_DIR + "/lib/constants"
7   require ROOT_DIR + "/lib/logger"
8   require ROOT_DIR + "/lib/response_helpers"
9   require ROOT_DIR + "/lib/processing_queue"
10
11  class SerialRequestHandler
12    include Singleton
13    include Constants
14    include Loggable
15    include ResponseHelpers
16    include ProcessingQueue
17
18    def initialize
19      setup_processing_queue
20      status = Link_fs.link_fs_init("w")
21      if status == PRO_OK
22        @connected = true
23        log "Initialized the datalink"
24      else
25        log "Could not initialize the datalink"
26      end
27    end
28
29    def ready?
30      @connected
```

```
31    end
32
33    def not_ready(request, &callback)
34      callback.call(STATUS_SERIALPORT_NOT_CONNECTED,0,nil)
35    end
36
37    def request(request, options, &callback)
38      req = request.to_a.flatten.join("␣").split("␣")
39      enqueue({"command" => req, "timeout" => options["
          timeout"].to_i, "no-response" => options["no-
          response"]},&callback)
40    end
41
42    def process(request, &callback)
43      begin
44        write(request)
45        if request["no-response"]
46          return_code, length, data = STATUS_OK, nil, nil
47        else
48          return_code, length, data = read(request)
49        end
50        Thread.new do
51          callback.call(return_code, length, data)
52        end
53      rescue Errno::EIO => e
54        @connected = false
55        log "The␣datalink␣has␣been␣disconnected"
56        not_ready(request, &callback)
57        raise ProcessingError
58      end
59    end
60
61    def write(request)
62      data = request["command"].map{|h| h.hex}
63      log "Datalink␣writing␣...␣"
64      status = Link_fs.link_fs_send_packet(data, data.
          length)
65      if status == PRO_OK
66        log "Datalink␣wrote:␣#{request["command"].join('␣')
            }"
67      else
68        log "An␣failure␣occured␣during␣write␣to␣the␣
            datalink"
69      end
```

```
70    end
71
72    def read(request)
73      buffer = ""
74      log "Datalink reading ..."
75      buffer, result = Link_fs.link_fs_receive_packet(
            buffer, FS_MAX_PACKET_SIZE, request["timeout"])
76
77      if result == PRO_OK
78        return_code     = buffer[0]
79        downlink        = buffer[1]
80        data_length_raw = buffer[2..3]
81        data_length     = data_length_raw.unpack("v").first
82        data            = buffer[4..(4+data_length-1)]
83        log "Datalink read: #{return_code.spaced_hex(1)} #{
              downlink.spaced_hex(1)} #{data_length_raw.bytes.
              map{|s| s.to_i.spaced_hex(1)}.join(" ")} #{data.
              bytes.map{|s| s.to_i.spaced_hex(1)}.join(" ")}"
84        return return_code, data_length, data
85      else
86        log 'Datalink read timeout'
87        return STATUS_TIMEOUT, nil, nil
88      end
89    end
90
91  end
```

lib/server.rb

```
1  require 'rubygems'
2  require 'eventmachine'
3  require 'singleton'
4  require 'socket'
5
6  require ROOT_DIR+"/lib/logger"
7  require ROOT_DIR+"/lib/ext/string"
8  require ROOT_DIR+"/lib/ext/fixnum"
9  require ROOT_DIR+"/lib/command_parser"
10 require ROOT_DIR+'/lib/response_helpers'
11 require ROOT_DIR+'/lib/constants'
12 require ROOT_DIR+'/lib/token_handler'
13 require ROOT_DIR+'/lib/serial_request_handler'
14
```

```
15 class Server
16   include Loggable
17   include Singleton
18
19   VERSION = '1.0'
20   LISTENING_ON = "Listening␣on␣$0"
21
22   def start(options)
23     EM.run do
24       # Set token timeout
25       TokenHandler.instance.timeout = options[:timeout]
26
27       # Initialize datalink layer
28       SerialRequestHandler.instance
29
30       EM.start_server options[:host], options[:port],
             EMServer
31       log "Listening␣on␣#{options[:host]}:#{options[:port
             ]}"
32     end
33   end
34 end
35
36 module EMServer
37   include Loggable
38   include ResponseHelpers
39   include Constants
40
41   def post_init
42     # Maintain list of all connected clients
43     $clients_list ||= {}
44     @identifier = self.object_id
45     $clients_list.merge!({ @identifier => self})
46
47     # Setup token_reset broadcast
48     TokenHandler.instance.reset_callback ||= Proc.new {
49       broadcast message("server_unlocked",
             STATUS_SERVER_UNLOCKED)
50     }
51
52     port, ip = Socket.unpack_sockaddr_in(get_peername)
53     @client = "<#{ip}:#{port}>"
54     log "#{@client}␣logged␣on"
55   end
```

```
56
57    def receive_data(data)
58      log "#{@client} requests: #{data}"
59      id, token, command = CommandParser.new.parse(data)
60
61      if TokenHandler.instance.taken?
62        if TokenHandler.instance.token != token
63          send(response(id, STATUS_IS_LOCKED))
64          return;
65        end
66        TokenHandler.instance.reset_timer
67      else
68        unless command.is_a?(Commands::Lock)
69          send(response(id,STATUS_MUST_LOCK))
70          return;
71        end
72      end
73
74      # Command is already formatted as an parse error
75      if(command.is_a?(Hash))
76        send(command)
77        return;
78      end
79
80      if command.valid?
81        command.client = self
82        operation = proc {command.execute}
83        EventMachine.defer(operation)
84      else
85        send(response(id,STATUS_VALIDATION_ERROR, command.
              validation_errors))
86      end
87    end
88
89    def send(data)
90      begin
91        data = data.to_json if data.is_a?(Hash)
92      rescue
93        data = "{\"status\": #{STATUS_JSON_PARSE_ERROR}, \"
              raw\": #{data}, \"message\":#{MESSAGES[
              STATUS_JSON_PARSE_ERROR]}}"
94      end
95      log "#{@client} response: #{data}"
96      send_data(data+"\0")
```

```
97    end
98
99    def unbind
100     $clients_list.delete(@identifier)
101     log "#{@client} logged off"
102   end
103
104   def broadcast(data)
105     data = data.to_json if data.is_a?(Hash)
106     log "Broadcasting: #{data}"
107     $clients_list.values.each do |client|
108       client.send_data(data+"\0")
109     end
110   end
111 end
```

lib/token_handler.rb

```
1  require 'singleton'
2
3  require ROOT_DIR+"/lib/logger"
4
5  class TokenHandler
6    include Singleton
7    include Loggable
8
9    attr_reader :token
10   attr_accessor :timeout, :reset_callback
11
12   def initialize
13     @token = nil
14     @mutex = Mutex.new
15     @started = false # Shared variable. Must protect
           reads and writes.
16   end
17
18   def token=(token)
19     @token = token
20     (free?) ? stop_timer : start_timer
21   end
22
23   def free?
24     @token.nil?
```

```
25    end
26
27    def taken?
28      !free?
29    end
30
31    def stop_timer
32      @mutex.synchronize {
33        @started = false
34        @timer_thread.kill
35      }
36    end
37
38    def reset_timer
39      @timer = @timeout
40    end
41
42    def start_timer
43      @mutex.synchronize {
44        unless @started
45          @started = true
46          reset_timer
47          @timer_thread = Thread.new do
48            loop do
49              sleep(1)
50              @timer -= 1
51              if @timer < 1
52                log "Token has been reset after #{@timeout}
                      seconds."
53                @token = nil
54                @reset_callback.call
55                stop_timer
56              end
57            end
58          end
59        end
60      }
61    end
62
63  end
```

link_fs/link_fs_devel.rb

```
1  require 'serialport'
2
3  require ROOT_DIR + "/lib/logger"
4  require ROOT_DIR + "/lib/constants"
5
6  module Link_fs
7
8    class << self
9      include Constants
10     include Loggable
11
12     DEVICE    = "/dev/ttyUSB0"
13     BAUD      = 9600
14     DATA_BITS = 8
15     STOP_BITS = 1
16     PARITY    = SerialPort::NONE
17
18     def link_fs_init(rw)
19       begin
20         $sp = SerialPort.new(DEVICE, BAUD, DATA_BITS,
               STOP_BITS, PARITY)
21         return PRO_OK
22       rescue => e
23         log "SerialPort init error: #{e}"
24         return PRO_FAILURE
25       end
26     end
27
28     def link_fs_send_packet(data, length)
29       begin
30         data.each do |s|
31           sleep(0.2)
32           $sp.putc(s)
33         end
34         return PRO_OK
35       rescue => e
36         log "SerialPort write error: #{e}"
37         return PRO_FAILURE
38       end
39     end
40
41     def link_fs_receive_packet(buffer, maxlength, timeout
         )
42       done_reading = false
```

```
43        status = ""
44
45        # Read thread
46        r = Thread.new do
47          begin
48            return_code     = $sp.getc
49            downlink        = $sp.getc
50            data_length_raw = $sp.read(2)
51            data_length     = data_length_raw.unpack("v").
                  first
52            data            = $sp.read(data_length)
53
54            done_reading    = true
55            buffer          << return_code << downlink <<
                  data_length_raw << data
56            status          = PRO_OK
57          rescue => e
58            log "SerialPort read error: #{e}"
59            status          = PRO_FAILURE
60          end
61        end
62
63        # Timeout thread
64        sleep_step = 0.5
65        t = Thread.new do
66          (0..(timeout/(100*sleep_step))).each do
67            sleep(sleep_step)
68            Thread.current.kill! if done_reading
69          end
70          r.kill!
71          status = PRO_TIMEOUT
72        end
73
74        # Wait for read thread and timeout
75        r.join
76        t.join
77
78        return buffer, status
79      end
80
81    end
82  end
```

scripts/count

```
#!/usr/bin/ruby
require 'optparse'

op = OptionParser.new do |opts|
  opts.banner = "Usage: count"
  opts.separator "Description: Count to five"
end
op.parse!

5.times do |n|
  sleep(1)
  puts "Counted to #{(n+1)}"
end

exit(0)
```

scripts/upload_file

```
#!/usr/bin/ruby
require 'optparse'
require 'rubygems'
require 'json'
require 'open3'
require 'ftools'

ROOT_DIR = File.dirname(__FILE__) + "/.."
require ROOT_DIR + "/lib/ext/string"
require ROOT_DIR + "/lib/ext/fixnum"
require ROOT_DIR + "/lib/constants"

include Constants

op = OptionParser.new do |opts|
  opts.banner = "Usage: upload_file token filepath
      ram_address flash_address"
  opts.separator "Description: Upload a file to an
      address in the satellites memory"
  opts.separator "Arguments:"
  opts.separator "\tfilepath (string)"
  opts.separator "\tram_address (hexadecimal)"
  opts.separator "\tflash_address (hexadecimal)"
end
```

```
23 op.parse!
24
25 if ARGV.length != 4
26   puts "Not enough arguments"
27   exit(1)
28 end
29
30 $token = ARGV[0]
31 filepath = ARGV[1]
32 ram_address = ARGV[2]
33 flash_address = ARGV[3]
34
35 unless File.exists?(filepath)
36   puts "File not found"; exit(1)
37 end
38
39 unless ram_address.addressable?
40   puts "Ram address is not addressable"; exit(1)
41 end
42
43 unless flash_address.addressable?
44   puts "Flash address is not addressable"; exit(1)
45 end
46
47 # Helper method
48 $out = ""
49 $err = ""
50 def fsclient(*args)
51   stdin, stdout, stderr = Open3.popen3("fsclient", "--
       token=#{$token}", *args)
52   $out = stdout.readlines.join
53   $err = stdout.readlines.join
54   if $err == ""
55     begin
56       return JSON.parse($out);
57     rescue => e
58       puts "JSON Parse error: #{e}"
59       puts "Raw: #{$out}"
60       exit(1)
61     end
62   else
63     puts "Fsclient stderror: #{$err}"
64     exit(1)
65   end
```

```
66 end
67 def error_exit
68   puts "Error: #{$out}"
69   exit(1)
70 end
71
72
73 file = File.new(filepath, "r")
74 size = File.size(filepath)
75 max_data_size = 20#FS_MAX_DATA_SIZE - 4
76 total_uploads = size / max_data_size
77 total_uploads += 1 unless size % max_data_size == 0
78 puts "Max data size is: #{max_data_size} B"
79 puts "File size is #{size} B and will be split over #{
       total_uploads} uploads."
80
81 # Upload
82 address = ram_address.int_or_hex
83 total_uploads.times do |i|
84   data = "0x"
85   file.read(max_data_size).each_byte {|b| data << b.
       spaced_hex(1) }
86   puts "#{i*100/total_uploads}% Done. Uploading part #{i
       +1}/#{total_uploads} ..."
87   response = fsclient("upload", address.to_s, data)
88   error_exit unless response['status'] == 255
89   address += (max_data_size)
90 end
91 file.close
92 puts "100% Done. Upload succeeded"
93
94
95 # Ram Checksum
96 puts "Calculating checksum in ram ... "
97 response = fsclient("calculate_check_sum", ram_address,
       size.to_s)
98 error_exit unless response['status'] == 10
99 ram_checksum = response["data"]
100 puts "Ram checksum is: #{ram_checksum}"
101
102 # Copy to Flash
103 puts "Unlock flash ... "
104 error_exit unless fsclient("unlock_flash")['status'] ==
       255
```

```
105
106
107 puts "Copying␣to␣flash␣...␣"
108
109 flash_size = 512
110 flash_size = 1024 if size > 512
111 flash_size = 2048 if size > 1024
112 flash_size = 4096 if size > 4096
113
114 max_length = 4096
115
116 unless flash_size > max_length
117
118   error_exit unless fsclient("copy_to_flash", ram_address
        , flash_address, flash_size.to_s)['status'] == 255
119
120 else
121   total_uploads = size / max_length
122   last_length = size % max_length
123   total_uploads += 1 unless last_length == 0
124   last_length = max_length if last_length == 0
125
126   puts "Must␣split␣copy␣in␣#{total_uploads}␣parts."
127
128   current_ram_address   = ram_address.int_or_hex
129   current_flash_address = flash_address.int_or_hex
130   total_uploads.times do |i|
131     puts "#{i*100/total_uploads}%␣Done.␣Copying␣part␣#{i
          +1}/#{total_uploads}␣..."
132
133     length = if(1+i == total_uploads) # Is it the last
          part
134       last_length
135     else
136       max_length
137     end
138
139     response = fsclient("copy_to_flash",
          current_ram_address.to_s, current_flash_address.
          to_s, length)
140     error_exit unless response['status'] == 255
141
142     current_ram_address += max_length
143     current_flash_address += max_length
```

```
144   end
145
146 end
147
148 # Flash Checksum
149 puts "Calculating␣checksum␣in␣flash␣...␣"
150 response = fsclient("calculate_check_sum", flash_address,
        size.to_s)
151 error_exit unless response['status'] == 10
152 flash_checksum = response["data"]
153 puts "Flash␣checksum␣is:␣#{ram_checksum}"
154
155 # Check Checksums
156 if ram_checksum == flash_checksum
157   puts "The␣checksums␣are␣identical."
158   puts "The␣upload␣succeeded"
159 else
160   puts "The␣checksums␣are␣not␣identical"
161   puts "The␣upload␣failed"
162 end
163
164 exit(0)
```

test/download_chars.rb

```
1 require 'rubygems'
2 require 'json'
3
4 token = JSON.parse(`fsclient lock`)["data"]
5 bytes = JSON.parse(`fsclient --token=#{token} download 0
      x40003000 48`)['data']
6
7 puts bytes.pack("c"*48) # Interpret as 48 characters
```

test/greeting.txt

```
1 Hi,
2
3 I am going to space ... and back again ...
```

test/lib/command_parser_test.rb

```
1 require File.dirname(__FILE__) + "/../test_helpers"
```

```
2
3  require ROOT_DIR + "/lib/command_parser"
4
5  class CommandParserTest < Test::Unit::TestCase
6
7    def test_should_parse_lock_without_token
8      request = '{"id":"1",␣"data":"lock"}'
9      id, token, command = CommandParser.new.parse(request)
10
11     assert_equal "1", id
12     assert_equal nil, token
13     assert_equal Commands::Lock, command.class
14   end
15
16   def test_should_parse_with_token
17     request = '{"id":"1",␣"token":"0123456789abcdef",␣"
           data":"reset"}'
18     id, token, command = CommandParser.new.parse(request)
19
20     assert_equal "1", id
21     assert_equal "0123456789abcdef", token
22     assert_equal Commands::Reset, command.class
23   end
24
25   def test_should_not_parse_unknown_commands
26     request = '{"id":"1",␣"token":"0123456789abcdef",␣"
           data":"blast_venus"}'
27     id, token, command = CommandParser.new.parse(request)
28
29     assert_equal "1", id
30     assert_equal "0123456789abcdef", token
31     assert_equal Commands::Unknown, command.class
32   end
33
34   def
         test_should_not_parse_with_wrong_number_of_arguments
35     request = '{"id":"1",␣"token":"0123456789abcdef",␣"
           data":"reset␣invalid_argument"}'
36     id, token, command = CommandParser.new.parse(request)
37
38     assert_equal "1", id
39     assert_equal "0123456789abcdef", token
40     assert_equal Commands::WrongNumberOfArguments,
           command.class
```

```
41    end
42
43    def test_should_extract_options
44      request = '{"id":"1", "token":"0123456789abcdef", "
            data":"reset --timeout=20 --no-response"}'
45      id, token, command = CommandParser.new.parse(request)
46
47      assert_equal "1", id
48      assert_equal "0123456789abcdef", token
49      assert_equal ({"timeout"=>"20", "no-response"=>true},
            command.options)
50    end
51
52  end
```

test/lib/ext/string_test.rb

```
1   require 'test/unit'
2   require File.dirname(__FILE__) + "/../../../lib/ext/
        string"
3
4   class StringTest < Test::Unit::TestCase
5
6     def test_should_be_addressable?
7       assert_equal true, "0x01234567".addressable?
8       assert_equal true, "0x89ABCDEF".addressable?
9       assert_equal true, "0x89abcdef".addressable?
10      assert_equal true, "0xff".addressable?
11
12      assert_equal false, "0x01234556789abcdef".addressable
            ?
13      assert_equal true,  "0x01234556789abcdef".addressable
            ?(8)
14
15      assert_equal false, "0x100000000".addressable?
16      assert_equal false, "-0x1".addressable?
17      assert_equal false, "0xfg".addressable?
18      assert_equal false, "0x0.2".addressable?
19      assert_equal false, "0x0,2".addressable?
20
21      assert_equal true, "0".addressable?
22      assert_equal true, "4294967295".addressable?
23
```

```
24      assert_equal false, "4294967296".addressable?
25      assert_equal false, "-1".addressable?
26      assert_equal false, "a".addressable?
27      assert_equal false, "0.1".addressable?
28      assert_equal false, "0,1".addressable?
29
30      assert_equal false, "address".addressable?
31    end
32
33    def test_byte_length
34      assert_equal 1, "0x0".byte_length
35      assert_equal 1, "0x00000000000000000000000000000000".
            byte_length
36      assert_equal 1, "0xf".byte_length
37      assert_equal 1, "0xff".byte_length
38      assert_equal 4, "0xffffffff".byte_length
39      assert_equal 8, "0xffffffffffffffff".byte_length
40
41      assert_equal 1, "0".byte_length
42      assert_equal 1, "15".byte_length
43      assert_equal 1, "255".byte_length
44      assert_equal 4, "4294967295".byte_length
45      assert_equal 8, "18446744073709551615".byte_length
46
47      assert_raise NotANumberError do "asd".byte_length end
48    end
49
50    def test_should_be_spaced_hex
51      assert_equal "00␣00␣00␣00", "0x00000000".spaced_hex
52      assert_equal "00␣00␣00␣00", "0x0".spaced_hex
53      assert_equal "00␣00␣00␣ff", "0xff".spaced_hex
54      assert_equal "ff␣00␣00␣00", "0xff000000".spaced_hex
55      assert_equal "ff␣ee␣dd␣cc", "0xffeeddcc".spaced_hex
56      assert_equal "ff␣ff␣ff␣ff", "0xffffffff".spaced_hex
57
58      assert_raise NotAddressableError do "0x123456789".
            spaced_hex end
59      assert_raise NotAddressableError do "-0x1".spaced_hex
            end
60
61      assert_equal "00␣00␣00␣00", "0".spaced_hex
62      assert_equal "00␣00␣00␣ff", "255".spaced_hex
63      assert_equal "00␣00␣02␣00", "512".spaced_hex
64      assert_equal "00␣00␣04␣00", "1024".spaced_hex
```

```
65      assert_equal "00␣00␣08␣00", "2048".spaced_hex
66      assert_equal "ff␣ff␣ff␣ff", "4294967295".spaced_hex
67
68      assert_raise NotAddressableError do "4294967296".
            spaced_hex end
69      assert_raise NotAddressableError do "-1".spaced_hex
            end
70
71      assert_equal "00␣00", "0".spaced_hex(2)
72      assert_equal "00␣ff", "255".spaced_hex(2)
73      assert_raise NotAddressableError do "0x10000".
            spaced_hex(2) end
74      assert_equal "00␣00␣00␣00␣00␣00␣00␣00", "0x0".
            spaced_hex(8)
75      assert_equal "ff␣ff␣ff␣ff␣ff␣ff␣ff␣ff", "0
            xffffffffffffffff".spaced_hex(8)
76
77    end
78
79  end
```

test/test_helpers.rb

```
1  require 'test/unit'
2  ROOT_DIR = File.dirname(__FILE__)+"/.."
```

D.2 FSClient

fsclient

```
1  #!/usr/bin/ruby
2  require 'optparse'
3  options = {:host => '0.0.0.0', :port => 3000, :
       interactive => false, :auto_lock => false, :data_only
       => false, :token => nil, :command_options => []}
4  op = OptionParser.new do |opts|
5    opts.banner = "Usage:␣fsclient␣[options]␣<command␣[
         command_args␣...␣]>"
6    opts.on("--host=HOST", String, "Server␣host␣(default␣is
         ␣0.0.0.0)")  {|h| options[:host] = h}
```

```
7    opts.on("--port=PORT", Integer, "Server port (default is 3000)")     {|p| options[:port] = p}
8    opts.on("--token=TOKEN", String, "Token") {|t| options[:token] = t}
9    opts.on("--timeout=SEC", String, "Timeout option to command") {|t| options[:command_options] << "--timeout=#{t}"}
10   opts.on("-i","--interactive", "Interactive mode") {|i| options[:interactive] = true}
11   opts.on("-d","--data-only", "Only print data parameter") {|i| options[:data_only] = true}
12   opts.on("-a","--auto-lock", "Auto lock in interactive mode") {|i| options[:auto_lock] = true}
13   opts.on("-n","--no-response", "No-response option to command") {|i| options[:command_options] << "--no-response"}
14
15   opts.separator ""
16   opts.separator "Common options:"
17
18   opts.on_tail("-h", "--help", "Show this message") do
19     puts opts
20     exit
21   end
22 end
23 op.parse!
24
25 if !options[:interactive] && ARGV.size < 1
26   puts op
27   exit
28 end
29
30 # Setup root dir
31 ROOT_DIR = File.dirname(__FILE__)
32
33 # Start the client
34 require ROOT_DIR + '/lib/client'
35 Client.instance.start(options)
```

lib/client.rb

```
1 require 'rubygems'
2 require 'eventmachine'
```

```
3  require 'singleton'
4  require 'json'
5
6  class Client
7    include Singleton
8
9    attr_accessor :options
10
11   def start(options)
12     @options = options
13     EventMachine::run do
14       EventMachine::connect options[:host], options[:port
             ], EMClient
15     end
16   end
17 end
18
19 module EMClient
20   def post_init
21     @auto_lock = Client.instance.options[:auto_lock]
22     @interactive = Client.instance.options[:interactive]
23     @data_only = Client.instance.options[:data_only]
24     @token = Client.instance.options[:token]
25     @command_options = Client.instance.options[:
           command_options]
26     @id_mutex = Mutex.new
27     @last_id = 0
28
29     unless @interactive
30       @exit = true
31       execute((ARGV << @command_options).join(' '), true)
32     else
33       @token = nil
34       puts "FSClient - Interactive Mode"
35
36       # Autolock
37       if @auto_lock
38         @request = 'lock'
39         execute(@request, false)
40       else
41         wait_for_user_request
42       end
43     end
44   end
```

```
45
46    def receive_data(data)
47      data.split("\0").each do |raw| # Split in case of
            broadcast messages
48        @response = JSON.parse(raw)
49
50        if @verbose
51          if @data_only
52            puts @response["data"]
53          else
54            puts raw
55          end
56        end
57
58        if @response["type"] == "response"
59          if @response['partial'].nil?
60            @token = remember_or_forget_token(@request,
                  @response)
61
62            if @exit
63              unbind
64            else
65              wait_for_user_request
66            end
67          end
68        end
69      end
70    end
71
72    def safe_exit
73      # Unlock and stop EM
74      if @auto_lock
75        @exit = true
76        execute("#{@token} unlock", false)
77      else
78        unbind
79      end
80    end
81
82    def unbind
83      EventMachine.stop_event_loop
84    end
85
86    def wait_for_user_request
```

```
87      putc ">"
88      putc "␣"
89      @request = $stdin.gets.gsub(/\n/,'')  # Remove
            newline
90
91      if @request =~ /^exit/
92        safe_exit
93      elsif @request == ""
94        wait_for_user_request
95      else
96        execute(@request)
97      end
98    end
99
100   def remember_or_forget_token(request, response)
101     if(request =~ /lock/ && response['status'] == 100)
102       puts "Token␣remembered␣and␣will␣be␣send␣
              automatically␣before␣any␣command."
103       response['data']
104     elsif response['status'] =~ 108
105       nil
106     else
107       @token
108     end
109   end
110
111   def execute(request,verbose=true)
112     @verbose = verbose
113     send_data({:id => next_id, :data => request, :token
            => @token}.to_json)
114   end
115
116   def next_id
117     @id_mutex.synchronize {
118       @last_id += 1
119     }
120     @last_id
121   end
122 end
```

D.3 FSGui

dtusat/Driver.java

```
1  package dtusat;
2
3  public class Driver {
4    public static void main(String[] args) {
5      FSController controller = FSController.getInstance();
6      controller.start();
7    }
8  }
```

dtusat/FSCallback.java

```
1  package dtusat;
2
3  public interface FSCallback {
4    public void onResponse(FSResponse response);
5  }
```

dtusat/FSController.java

```
1  package dtusat;
2
3  import java.awt.Toolkit;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import java.text.SimpleDateFormat;
7  import java.util.Calendar;
8  import java.util.Hashtable;
9
10 import javax.swing.ImageIcon;
11 import javax.swing.JFrame;
12 import javax.swing.JMenu;
13 import javax.swing.JMenuBar;
14 import javax.swing.JMenuItem;
15 import javax.swing.JSeparator;
16
17 import dtusat.components.ConnectPanel;
18 import dtusat.components.FSMenu;
19 import dtusat.components.MainPanel;
20 import dtusat.components.MainTabs;
```

```
21
22 public class FSController implements Logger,
       FSSocketObserver {
23
24   // --- Singleton Pattern
         ------------------------------------
25
26   private FSController() {}
27   private static FSController instance = new FSController
         ();
28   public static synchronized FSController getInstance() {
         return instance; }
29   public Object clone() throws CloneNotSupportedException
         { throw new CloneNotSupportedException(); }
30
31   // ---- STATUS
         ------------------------------------------------
32
33   public final static int STATUS_OK                = 100;
34   public final static int STATUS_ERROR             = 101;
35   public final static int STATUS_IS_LOCKED         = 102;
36   public final static int STATUS_MUST_LOCK         = 103;
37   public final static int STATUS_WRONG_ARGUMENTS   =
         104;
38   public final static int STATUS_UNKNOWN_COMMAND   =
         105;
39   public final static int STATUS_TIMEOUT           = 106;
40   public final static int STATUS_VALIDATION_ERROR  =
         107;
41   public final static int STATUS_SERVER_UNLOCKED   =
         108;
42   public final static int STATUS_UNKNOWN_SCRIPT    =
         109;
43   public final static int STATUS_SERIALPORT_NOT_CONNECTED
         = 110;
44
45   // Command Codes
46   public final static int FS_HEALTH_STATUS         = 19;
47
48
49   // ---- Fields
         ------------------------------------------------
50
51   private FSSocket socket;
```

```
52    public JFrame frame;
53    public MainPanel mainPanel;
54    public ConnectPanel connectPanel;
55    public MainTabs mainTabs;
56    public FSMenu menu;
57
58    Hashtable<String, IncomingDataHandler>
          incomingDataHandlers;
59    Hashtable<String, FSCallback> requestCallbacks;
60
61    public boolean isConnected, isLocked, isAutoLocked,
          isServerLocked;
62
63    // ---- Intitialization
          ----------------------------------------
64
65    public void start() {
66      isAutoLocked = true;
67      isConnected = false;
68      isServerLocked = false;
69
70      socket = new FSSocket();
71      mainPanel = new MainPanel();
72      mainTabs = new MainTabs();
73      connectPanel = new ConnectPanel();
74      menu = new FSMenu();
75
76      incomingDataHandlers = new Hashtable<String,
            IncomingDataHandler>();
77      requestCallbacks = new Hashtable<String, FSCallback
            >();
78
79      setupIncomingDataHandlers();
80      setupWindow();
81      showConnectPanel();
82    }
83
84    // --- Incoming Data Handlers
          ---------------------------------
85
86    private void setupIncomingDataHandlers() {
87
88      // Response Handler
```

```
89      incomingDataHandlers.put("response", new
            IncomingDataHandler() {
90        public void onData(FSResponse response) {
91
92          try {
93
94            // Check for lock errors
95            if(response.status == STATUS_MUST_LOCK) {
96              setIsLocked(false);
97            } else if(response.status == STATUS_IS_LOCKED)
                  {
98              isServerLocked = true;
99              setIsLocked(false);
100           }
101
102           requestCallbacks.get(response.id).onResponse(
                  response);
103
104           // Forget callback unless partial
105           if(!response.isPartial())
106             requestCallbacks.remove(response.id);
107
108         } catch(NullPointerException e) {
109           // No callback for response
110         }
111       }
112     });
113
114     // Server Unlocked Handler
115     incomingDataHandlers.put("server_unlocked", new
            IncomingDataHandler() {
116       public void onData(FSResponse response) {
117         isServerLocked = false;
118         setIsLocked(false);
119       }
120     });
121   }
122
123   public void setupWindow() {
124     frame = new JFrame("Failsafe Control GUI");
125     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
126     frame.setSize(Toolkit.getDefaultToolkit().
            getScreenSize());
127     frame.setContentPane(mainPanel);
```

```
128     frame.setJMenuBar(menu);
129     frame.setVisible(true);
130   }
131
132   // --- Views ---------------------------------
133
134   public void showConnectPanel() {
135     mainPanel.setTopPanel(connectPanel);
136   }
137
138   public void showMainPanel() {
139     mainPanel.setTopPanel(mainTabs);
140   }
141
142   // --- Log -----------------------------------
143
144   public void log(String msg) {
145     Calendar cal = Calendar.getInstance();
146       SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/
              yy HH:mm:ss");
147       String time = "["+sdf.format(cal.getTime())+"]";
148
149     mainPanel.log(time+" "+msg);
150   }
151
152
153   // --- FSSocket Callbacks
          ------------------------------
154
155   public void onConnected() {
156     setIsConnected(true);
157     showMainPanel();
158     if(!isLocked && isAutoLocked)
159       lock();
160   }
161
162   public void onConnectionRefused() {
163     setIsConnected(false);
164     log("Connection refused to "+socket.host+":"+socket.
          port);
165   }
166
167   public void onDisconnected() {
168     if(isConnected) {
```

```
169         setIsConnected(false);
170         showConnectPanel();
171       }
172     }
173
174     public void onIncomingData(String s) {
175       log("< "+s);
176       FSResponse response = new FSResponse(s);
177       incomingDataHandlers.get(response.type).onData(
              response);
178     }
179
180     public void onRegisterCallback(String id, FSCallback
          callback) {
181       requestCallbacks.put(id, callback);
182     }
183
184     // --- FS Commands -----------------------------------
185
186     public void lock() {
187       socket.request("lock", new FSCallback() {
188         public void onResponse(FSResponse response) {
189           if(response.isSuccess()) {
190             socket.token = response.dataAsString();
191             setIsLocked(true);
192           }
193         }
194       });
195     }
196
197     public void unlock() {
198       socket.request("unlock", new FSCallback() {
199         public void onResponse(FSResponse response) {
200           if(response.isSuccess()) {
201             socket.token = "";
202             setIsLocked(false);
203           }
204         }
205       });
206     }
207
208     // --- Setters & Getters
            ---------------------------------
209
```

```
210   public FSSocket getSocket() {
211     return socket;
212   }
213
214   public void setAutoLocked(boolean selected) {
215     isAutoLocked = selected;
216
217     connectPanel.autoLockCheckBox.setSelected(selected);
218     menu.autoLockItem.setSelected(selected);
219
220     if(!isLocked && isAutoLocked && isConnected)
221       lock();
222   }
223
224   private void setIsConnected(boolean b) {
225     isConnected = b;
226     if(b)
227       mainPanel.showConnectedStatus();
228     else
229       mainPanel.showDisconnectedStatus();
230   }
231
232   private void setIsLocked(boolean b) {
233     isLocked = b;
234     if(b) {
235       mainPanel.showLockedStatus();
236     } else {
237       mainPanel.showUnLockedStatus();
238       if(isAutoLocked && isConnected && !isServerLocked)
239         lock();
240     }
241   }
242 }
```

dtusat/FSResponse.java

```
1  package dtusat;
2
3  import org.json.JSONArray;
4  import org.json.JSONException;
5  import org.json.JSONObject;
6
7  public class FSResponse {
```

```
8
9   public String type;
10  public String id;
11  public int status;
12  private JSONObject parsed;
13  private boolean partial;
14
15  public FSResponse(String raw) {
16    try {
17      parsed = new JSONObject(raw);
18      this.type = parsed.getString("type");
19
20      if(type.equals("response")) {
21        this.status = parsed.getInt("status");
22        this.id = parsed.getString("id");
23      }
24
25    } catch (JSONException e) {
26      e.printStackTrace();
27    }
28
29    try {
30      this.partial = parsed.getBoolean("partial");
31    } catch (JSONException e1) {
32      this.partial = false;
33    }
34  }
35
36  public String dataAsString() {
37    String r = null;
38    try {
39      r = parsed.getString("data");
40    } catch (JSONException e) {
41      e.printStackTrace();
42    }
43    return r;
44  }
45
46  public JSONArray dataAsArray() {
47    JSONArray res = null;
48    try {
49      res = parsed.getJSONArray("data");
50    } catch (JSONException e) {
51      e.printStackTrace();
```

```
52      }
53      return res;
54    }
55
56    public boolean isSuccess() {
57      return (status == FSController.STATUS_OK);
58    }
59
60    public boolean isPartial() {
61      return partial;
62    }
63
64    public String messageAsString() {
65      String r = null;
66      try {
67        r = parsed.getString("message");
68      } catch (JSONException e) {
69        e.printStackTrace();
70      }
71      return r;
72    }
73
74  }
```

dtusat/FSSocket.java

```
1   package dtusat;
2
3   import java.io.*;
4   import java.math.BigInteger;
5   import java.net.*;
6   import java.security.SecureRandom;
7
8   public class FSSocket {
9
10    public String host, port;
11    public String token;
12    private Socket socket;
13    private BufferedReader in;
14    public FSSocketObserver observer;
15    public FSSocketReader socketReader;
16    private Thread inThread;
17    private int lastId;
```

```
18
19    public FSSocket() {
20      observer = FSController.getInstance();
21      token = "";
22      lastId = 0;
23    }
24
25    public void connect(String host, String port) {
26      this.host = host;
27      this.port = port;
28
29      try {
30        socket = new Socket(host, Integer.parseInt(port));
31        in = new BufferedReader(new InputStreamReader(
              socket.getInputStream()));
32
33        socketReader = new FSSocketReader(in, observer);
34        inThread = new Thread(socketReader);
35        inThread.start();
36
37        observer.onConnected();
38      } catch (UnknownHostException e) {
39        e.printStackTrace();
40        observer.onConnectionRefused();
41      } catch (ConnectException e) {
42        observer.onConnectionRefused();
43      } catch (IOException e) {
44        e.printStackTrace();
45        observer.onConnectionRefused();
46      }
47    }
48
49    public void disconnect() {
50      try {
51        socket.close();
52        observer.onDisconnected();
53        socketReader.stop();
54      } catch (IOException e) {
55        e.printStackTrace();
56      }
57    }
58
59    public void send(String command) {
60      request(command, null);
```

```
61    }
62
63    public void request(String command, FSCallback callback
          ) {
64      String id = nextId();
65      String request = "{\"id\":\""+id+"\",␣\"data\":\""+
            command+"\",␣\"token\":\""+token+"\"}";
66
67      if(callback != null)
68        observer.onRegisterCallback(id, callback);
69
70      try {
71        observer.log(">␣"+request);
72        byte[] bytes = request.getBytes();
73        socket.getOutputStream().write(bytes);
74      } catch (SocketException e) {
75        observer.onDisconnected();
76      } catch (IOException e) {
77        e.printStackTrace();
78      } catch (NullPointerException e) {
79        // Not connected yet
80      }
81
82    }
83
84    synchronized private String nextId() {
85      return ""+(lastId++);
86    }
87
88  }
```

dtusat/FSSocketObserver.java

```
1   package dtusat;
2
3   public interface FSSocketObserver {
4     public void onConnected();
5
6     public void onConnectionRefused();
7
8     public void onDisconnected();
9
10    public void log(String string);
```

```
11
12    public void onRegisterCallback(String id, FSCallback
          callback);
13
14    public void onIncomingData(String s);
15
16  }
```

dtusat/FSSocketReader.java

```
1   package dtusat;
2
3   import java.io.BufferedReader;
4   import java.io.IOException;
5   import java.io.PrintWriter;
6   import java.net.SocketException;
7
8   public class FSSocketReader implements Runnable {
9
10    private BufferedReader in;
11    public FSSocketObserver observer;
12    public boolean stop;
13
14    public FSSocketReader(BufferedReader in,
          FSSocketObserver observer) {
15      this.in = in;
16      this.observer = observer;
17      stop = false;
18    }
19
20    public void run() {
21      String s;
22      try {
23        while(!stop) {
24          s = "";
25          // Read until zero-char
26          int next = in.read();
27          while(next != 0) {
28            s += (char) next;
29            next = in.read();
30          }
31          observer.onIncomingData(s);
32        }
```

```
33      } catch(SocketException e) {
34        observer.onDisconnected();
35      } catch (IOException e) {
36        e.printStackTrace();
37      }
38    }
39
40    public void stop() {
41      stop = true;
42    }
43
44  }
```

dtusat/IncomingDataHandler.java

```
1  package dtusat;
2
3  public interface IncomingDataHandler {
4    public void onData(FSResponse response);
5  }
```

dtusat/Logger.java

```
1  package dtusat;
2
3  public interface Logger {
4    public void log(String s);
5  }
```

dtusat/components/CommandPanel.java

```
1  package dtusat.components;
2
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.Component;
6  import java.awt.FlowLayout;
7  import java.awt.event.ActionEvent;
8  import java.awt.event.ActionListener;
9
10 import javax.swing.AbstractButton;
11 import javax.swing.BorderFactory;
12 import javax.swing.ImageIcon;
```

```
13 import javax.swing.JButton;
14 import javax.swing.JComboBox;
15 import javax.swing.JLabel;
16 import javax.swing.JPanel;
17 import javax.swing.JScrollBar;
18 import javax.swing.JScrollPane;
19 import javax.swing.JTextArea;
20 import javax.swing.JTextField;
21
22 import org.json.JSONArray;
23 import org.json.JSONException;
24 import org.json.JSONObject;
25
26 import dtusat.FSController;
27
28 public class CommandPanel extends JPanel implements
       ActionListener {
29
30   String[][] commandsAndArguments = {
31         {"calculate_check_sum", "address", "length", "
              options"},
32         {"call_function", "address", "parameter", "
              options"},
33         {"copy_to_flash", "from", "to", "length", "
              options"},
34         {"copy_to_ram", "from", "to", "length", "options"
              },
35         {"delete_flash_block", "address", "options"},
36         {"download", "address", "length", "options"},
37         {"download_sib", "options"},
38         {"execute", "address", "options"},
39         {"flash_test", "address", "options"},
40         {"health_status", "options"},
41         {"list_scripts", "options"},
42         {"lock", "options"},
43         {"ram_test", "address", "length", "options"},
44         {"read_register", "address", "options"},
45         {"read_sensor", "address", "options"},
46         {"reset", "options"},
47         {"reset_sib", "options"},
48         {"run_script","path","arguments", "options"},
49         {"set_autoreset","value", "options"},
50         {"sleep","seconds", "options"},
51         {"unlock", "options"},
```

```
52          {"unlock_flash", "options"},
53          {"upload", "address", "data", "options"},
54          {"upload_sib", "sib", "options"},
55          {"write_register", "address", "data", "options"}
56        };
57    public JComboBox commandList;
58    private JButton removeButton;
59    private AbstractButton upButton;
60    private JButton downButton;
61    public JPanel argumentsPanel;
62    private int index;
63    public JTextArea outputArea;
64
65    public CommandPanel(int index) {
66      this.index = index;
67
68      setLayout(new BorderLayout());
69      setBorder(BorderFactory.createCompoundBorder(
            BorderFactory.createMatteBorder(0, 0, 1, 0, Color.
            GRAY),BorderFactory.createEmptyBorder(5,5,5,5)));
70
71      // West
72      JPanel west = new JPanel(new FlowLayout(FlowLayout.
            LEFT));
73      add(west, BorderLayout.WEST);
74
75        String[] commands = new String[commandsAndArguments
              .length];
76        for(int i=0;i<commandsAndArguments.length ; i++) {
77          commands[i] = commandsAndArguments[i][0];
78        }
79
80        commandList = new JComboBox(commands);
81        commandList.setBorder(BorderFactory.
              createEmptyBorder(5,5,5,5));
82        commandList.addActionListener(this);
83        west.add(commandList);
84
85        argumentsPanel = new JPanel();
86        updateArguments();
87        west.add(argumentsPanel);
88
89      // East
```

```
90      JPanel east = new JPanel(new FlowLayout(FlowLayout.
            LEFT));
91      add(east, BorderLayout.EAST);
92
93        upButton = new JButton(new ImageIcon("src/dtusat/
              icons/arrow_up.png"));
94        upButton.addActionListener(this);
95        east.add(upButton);
96
97        downButton = new JButton(new ImageIcon("src/dtusat/
              icons/arrow_down.png"));
98        downButton.addActionListener(this);
99        east.add(downButton);
100
101       removeButton = new JButton(new ImageIcon("src/
              dtusat/icons/remove.png"));
102       removeButton.addActionListener(this);
103       east.add(removeButton);
104
105     // South
106     JPanel south = new JPanel(new BorderLayout());
107     add(south, BorderLayout.SOUTH);
108
109       outputArea = new JTextArea();
110       outputArea.setBorder(BorderFactory.
              createEmptyBorder(5, 5, 5, 5));
111       outputArea.setWrapStyleWord(true);
112       outputArea.setVisible(false);
113       south.add(outputArea, BorderLayout.CENTER);
114   }
115
116   public CommandPanel(int index, JSONObject entry) {
117     this(index);
118     try {
119       String command = entry.getString("command");
120
121       String[] caa = null;
122
123       for(int i=0; i < commandsAndArguments.length;i++) {
124         if(commandsAndArguments[i][0].equals(command)) {
125           caa = commandsAndArguments[i];
126           commandList.setSelectedIndex(i);
127           break;
128         }
```

```
129         }
130
131         argumentsPanel.removeAll();
132         JSONArray arguments = entry.getJSONArray("arguments
                ");
133         for(int i=0;i<arguments.length();i++) {
134           argumentsPanel.add(newArgumentField(caa[i+1],
                  arguments.getString(i)));
135         }
136
137         FSController.getInstance().mainPanel.repaint();
138       } catch (JSONException e) {
139         e.printStackTrace();
140       }
141     }
142
143     private JPanel newArgumentField(String name) {
144       return newArgumentField(name, "");
145     }
146
147     private JPanel newArgumentField(String name, String
          value) {
148       JPanel p = new JPanel();
149
150       JLabel l = new JLabel(name);
151
152       JTextField a = new JTextField(value);
153       a.setColumns(8);
154       //a.setText(name);
155       a.setBorder(BorderFactory.createCompoundBorder(
              BorderFactory.createLineBorder(Color.BLACK, 1),
              BorderFactory.createEmptyBorder(5,5,5,5)));
156       a.setToolTipText(name);
157
158       p.add(l);
159       p.add(a);
160
161       return p;
162     }
163
164     private void updateArguments() {
165       argumentsPanel.removeAll();
166       String[] caa = commandsAndArguments[commandList.
              getSelectedIndex()];
```

```
167       for(int i=1;i<caa.length;i++) {
168         argumentsPanel.add(newArgumentField(caa[i]));
169       }
170
171       FSController.getInstance().mainPanel.repaint();
172     }
173
174     public String getFullCommand() {
175       String cmd = commandList.getSelectedItem().toString()
              ;
176
177       for(Component p : argumentsPanel.getComponents()) {
178         JTextField a = (JTextField) ((JPanel) p).
              getComponent(1);
179         cmd += "␣"+ a.getText();
180       }
181
182       return cmd;
183     }
184
185     public void setGUIEnabled(boolean b) {
186       commandList.setEnabled(b);
187       upButton.setEnabled(b);
188       downButton.setEnabled(b);
189       removeButton.setEnabled(b);
190       for(Component c : argumentsPanel.getComponents())
191         c.setEnabled(b);
192
193     }
194
195     public void actionPerformed(ActionEvent e) {
196       if(e.getSource() == commandList) {
197         updateArguments();
198       }
199
200       if(e.getSource() == removeButton) {
201         getParent().remove(this);
202         FSController.getInstance().mainPanel.repaint();
203       }
204
205       if(e.getSource() == downButton) {
206         if(index < getParent().getComponentCount()-1) {
207           index++;
```

```
208             ((CommandPanel) getParent().getComponent(index)).
                    index--;
209             getParent().add(this, index);
210             FSController.getInstance().mainPanel.repaint();
211           }
212         }
213
214         if(e.getSource() == upButton) {
215           if(index > 0) {
216             index--;
217             ((CommandPanel) getParent().getComponent(index)).
                    index++;
218             getParent().add(this, index);
219             FSController.getInstance().mainPanel.repaint();
220           }
221         }
222       }
223     }
```

dtusat/components/CommandSequencesPanel.java

```
1   package dtusat.components;
2
3   import java.awt.BorderLayout;
4   import java.awt.Color;
5   import java.awt.Component;
6   import java.awt.FlowLayout;
7   import java.awt.event.ActionEvent;
8   import java.awt.event.ActionListener;
9   import java.io.BufferedReader;
10  import java.io.BufferedWriter;
11  import java.io.File;
12  import java.io.FileReader;
13  import java.io.FileWriter;
14  import java.io.IOException;
15  import java.util.ArrayList;
16
17  import javax.swing.BorderFactory;
18  import javax.swing.BoxLayout;
19  import javax.swing.ImageIcon;
20  import javax.swing.JButton;
21  import javax.swing.JComboBox;
22  import javax.swing.JFileChooser;
```

```
23 import javax.swing.JOptionPane;
24 import javax.swing.JPanel;
25 import javax.swing.JScrollPane;
26 import javax.swing.JSeparator;
27 import javax.swing.JSplitPane;
28 import javax.swing.JTextArea;
29 import javax.swing.JTextField;
30 import javax.swing.JTree;
31
32 import org.json.JSONArray;
33 import org.json.JSONException;
34 import org.json.JSONObject;
35
36 import dtusat.FSController;
37 import dtusat.FSResponse;
38 import dtusat.FSCallback;
39
40 public class CommandSequencesPanel extends JPanel
       implements ActionListener {
41   private JPanel commandList;
42   private JPanel commandListContainer;
43   private FSController controller;
44   private JSplitPane splitPane;
45   private JButton addButton;
46   private JButton executeButton;
47   private JPanel leftPanel;
48   private FileTree tree;
49   private JPanel leftNorthPanel;
50   private JButton loadButton;
51   private JButton saveButton;
52   private JButton setDirButton;
53   private JPanel rightPanel;
54   private JButton exportButton;
55   private JButton newButton;
56   private JPanel rightEastPanel;
57   private String currentDir;
58   private CommandPanel[] commandsToExecute;
59   private int currentCommandIndex;
60   private JButton clearButton;
61   private JButton stopButton;
62   private boolean stop;
63
64   public CommandSequencesPanel() {
65     controller = FSController.getInstance();
```

```
66
67      setLayout(new BorderLayout());
68
69      // Splitpane
70      splitPane = new JSplitPane();
71      add(splitPane, BorderLayout.CENTER);
72
73        // Left Panel
74        leftPanel = new JPanel(new BorderLayout());
75        splitPane.setLeftComponent(leftPanel);
76
77          // North
78          leftNorthPanel = new JPanel();
79          leftPanel.add(leftNorthPanel, BorderLayout.NORTH)
                ;
80
81            setDirButton = new JButton("Set Dir", new
                  ImageIcon("src/dtusat/icons/folder_explore.
                  png"));
82            setDirButton.addActionListener(this);
83            leftNorthPanel.add(setDirButton);
84
85            loadButton = new JButton("Load", new ImageIcon(
                  "src/dtusat/icons/page_edit.png"));
86            loadButton.addActionListener(this);
87            leftNorthPanel.add(loadButton);
88
89            saveButton = new JButton("Save", new ImageIcon(
                  "src/dtusat/icons/script_save.png"));
90            saveButton.addActionListener(this);
91            leftNorthPanel.add(saveButton);
92
93
94        // Right Panel
95        rightPanel = new JPanel(new BorderLayout());
96        splitPane.setRightComponent(rightPanel);
97
98          // North Panel - Button Panel
99          JPanel rightNorthPanel = new JPanel(new
                BorderLayout());
100         rightPanel.add(rightNorthPanel, BorderLayout.
              NORTH);
101
102           JPanel rightNorthWestPanel = new JPanel();
```

```
103             rightNorthPanel.add(rightNorthWestPanel,
                    BorderLayout.WEST);
104
105               addButton = new JButton("Add Command", new
                      ImageIcon("src/dtusat/icons/add.png"));
106               addButton.addActionListener(this);
107               rightNorthWestPanel.add(addButton);
108
109               executeButton = new JButton("Execute", new
                      ImageIcon("src/dtusat/icons/execute.png"))
                      ;
110               executeButton.addActionListener(this);
111               rightNorthWestPanel.add(executeButton);
112
113               stopButton = new JButton("Stop", new
                      ImageIcon("src/dtusat/icons/stop.png"));
114               stopButton.setEnabled(false);
115               stopButton.addActionListener(this);
116               rightNorthWestPanel.add(stopButton);
117
118               clearButton = new JButton("Clear Output", new
                       ImageIcon("src/dtusat/icons/table_delete.
                      png"));
119               clearButton.addActionListener(this);
120               rightNorthWestPanel.add(clearButton);
121
122               newButton = new JButton("New", new ImageIcon(
                      "src/dtusat/icons/script_add.png"));
123               newButton.addActionListener(this);
124               rightNorthWestPanel.add(newButton);
125
126
127             JPanel rightNorthEastPanel = new JPanel();
128             rightNorthPanel.add(rightNorthEastPanel,
                    BorderLayout.EAST);
129
130               exportButton = new JButton("Export as Ruby",
                      new ImageIcon("src/dtusat/icons/script_go.
                      png"));
131               exportButton.addActionListener(this);
132               rightNorthEastPanel.add(exportButton);
133
134           // Right split
135           JSplitPane rightSplit = new JSplitPane();
```

```
136             rightSplit.setDividerLocation(800);
137             //rightPanel.add(rightSplit, BorderLayout.CENTER)
                    ;
138
139               // Left - Command list
140               commandListContainer = new JPanel(new
                      BorderLayout());
141
142                 commandList = new JPanel();
143                 commandList.setLayout(new BoxLayout(
                        commandList, BoxLayout.Y_AXIS));
144                 commandListContainer.add(commandList,
                        BorderLayout.NORTH);
145                 //rightSplit.setLeftComponent(new JScrollPane
                        (commandListContainer));
146                 rightPanel.add(new JScrollPane(
                        commandListContainer), BorderLayout.CENTER
                        );
147
148               // Right - Manual
149               rightEastPanel = new JPanel(new BorderLayout())
                      ;
150               //rightPanel.add(rightEastPanel, BorderLayout.
                      EAST);
151               rightSplit.setRightComponent(rightEastPanel);
152
153                 String[] commands = {"list_scripts","reset"};
154                 JComboBox commandsComboBox = new JComboBox(
                        commands);
155                 rightEastPanel.add(commandsComboBox,
                        BorderLayout.NORTH);
156
157                 JTextArea manualText = new JTextArea("Blah␣
                        blah␣blah");
158                 manualText.setBorder(BorderFactory.
                        createEmptyBorder(10, 10, 10, 10));
159                 rightEastPanel.add(new JScrollPane(manualText
                        ), BorderLayout.CENTER);
160
161     }
162
163
164     @Override
165     public void actionPerformed(ActionEvent e) {
```

```
166       if(e.getSource() == loadButton)
167         open();
168       else if(e.getSource() == setDirButton)
169         setDir();
170       else if(e.getSource() == addButton)
171         add();
172       else if(e.getSource() == executeButton)
173         startExecution();
174       else if(e.getSource() == stopButton)
175         stopExecution();
176       else if(e.getSource() == clearButton)
177         clearOutput();
178       else if(e.getSource() == newButton)
179         newSequence();
180       else if(e.getSource() == saveButton)
181         save();
182       else if(e.getSource() == exportButton)
183         export();
184     }
185
186     private void open() {
187       try {
188         String path = currentDir;
189         Object[] paths = tree.getSelectionPath().getPath();
190         for(int i=1;i<paths.length;i++) { // Skip first
191           path += File.separator+paths[i].toString();
192         }
193         BufferedReader in = new BufferedReader(new
              FileReader(path));
194         String json = "";
195         String str;
196         while ((str = in.readLine()) != null) {
197             json += str;
198         }
199         in.close();
200         JSONToSequence(json);
201       } catch (IOException e1) {
202
203         e1.printStackTrace();
204       }
205     }
206
207     private void setDir() {
208       final JFileChooser fc = new JFileChooser();
```

```
209       fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY
              );
210       if(fc.showOpenDialog(this) == JFileChooser.
              APPROVE_OPTION) {
211         currentDir = fc.getSelectedFile().getAbsolutePath()
                ;
212         updateFileTree();
213       }
214     }
215
216     private void add() {
217       commandList.add(new CommandPanel(commandList.
              getComponentCount()));
218       controller.mainPanel.repaint();
219     }
220
221     protected void startExecution() {
222       stop = false;
223       commandsToExecute = new CommandPanel[commandList.
              getComponentCount()];
224       for(int i=0;i<commandList.getComponentCount();i++)
225         commandsToExecute[i] = (CommandPanel) commandList.
                getComponent(i);
226
227       if(commandsToExecute.length > 0) {
228         setGUIEnabled(false);
229         clearOutput();
230         currentCommandIndex = 0;
231         executeNextCommand();
232       }
233     }
234
235     public void setGUIEnabled(boolean b) {
236       loadButton.setEnabled(b);
237       setDirButton.setEnabled(b);
238       addButton.setEnabled(b);
239       executeButton.setEnabled(b);
240       stopButton.setEnabled(!b);
241       clearButton.setEnabled(b);
242       newButton.setEnabled(b);
243       saveButton.setEnabled(b);
244       exportButton.setEnabled(b);
245       for(Component c : commandList.getComponents()) {
246         CommandPanel cp = (CommandPanel) c;
```

```
247         cp.setGUIEnabled(b);
248       }
249     }
250
251     private void clearOutput() {
252       for(Component c : commandList.getComponents()) {
253         CommandPanel cp = (CommandPanel) c;
254         cp.setBackground(this.getBackground());
255         cp.outputArea.setText("");
256         cp.outputArea.setVisible(false);
257       }
258     }
259
260     public void executeNextCommand() {
261       if(!stop) {
262         getCurrentCommandPanel().outputArea.setVisible(true
              );
263         getCurrentCommandPanel().outputArea.setBackground(
              Color.YELLOW);
264         controller.getSocket().request(commandsToExecute[
              currentCommandIndex].getFullCommand(), new
              FSCallback() {
265           public void onResponse(FSResponse response) {
266             CommandPanel cp = getCurrentCommandPanel();
267             cp.outputArea.append(response.messageAsString()
                  +":␣"+response.dataAsString());
268
269             if(response.status == FSController.STATUS_ERROR
                  ) {
270               setGUIEnabled(true);
271               cp.outputArea.setBackground(Color.RED);
272             } else {
273               if(!response.isPartial()) {
274                 cp.outputArea.setBackground(Color.GREEN);
275                 currentCommandIndex++;
276                 if(currentCommandIndex < commandsToExecute.
                      length)
277                   executeNextCommand();
278                 else
279                   setGUIEnabled(true);
280               }
281             }
282           }
283         });
```

```
284       }
285     }
286
287     public void stopExecution() {
288       stop = true;
289       setGUIEnabled(true);
290     }
291
292     private void newSequence() {
293       int n = JOptionPane.showConfirmDialog(
294             FSController.getInstance().frame,
295             "Are you sure? Any unsaved data will be lost.",
296             "Are you sure?",
297             JOptionPane.YES_NO_OPTION);
298
299       if(n == JOptionPane.YES_OPTION) {
300         commandList.removeAll();
301         FSController.getInstance().mainPanel.repaint();
302       }
303     }
304
305     private void save() {
306       final JFileChooser fc = new JFileChooser();
307       if(fc.showSaveDialog(this) == JFileChooser.
            APPROVE_OPTION) {
308         File f = new File(fc.getSelectedFile().
               getAbsolutePath());
309         doSave(f, sequenceToJSON());
310       }
311     }
312
313     private void export() {
314
315       final JFileChooser fc = new JFileChooser(new File(
            currentDir));
316       if(fc.showSaveDialog(this) == JFileChooser.
            APPROVE_OPTION) {
317         File f = new File(fc.getSelectedFile().
               getAbsolutePath());
318
319         String description = JOptionPane.showInputDialog(
               null,
320               "Description",
321               "Enter the description of the script",
```

```
322             JOptionPane.QUESTION_MESSAGE);
323
324         String fsclient_path = JOptionPane.showInputDialog(
                null,
325             "Path to fsclient",
326             "Enter the absolute path to fsclient",
327             JOptionPane.QUESTION_MESSAGE);
328
329         String data = sequenceToRuby(fc.getSelectedFile().
                getName(), description, fsclient_path);
330         doSave(f, data);
331         f.setExecutable(true);
332       }
333
334     }
335
336     private void doSave(File f, String data) {
337       try {
338         FileWriter fstream = new FileWriter(f);
339             BufferedWriter out = new BufferedWriter(fstream
                    );
340           out.write(data);
341           out.close();
342       } catch (IOException e1) {
343         e1.printStackTrace();
344       }
345     }
346
347     private CommandPanel getCurrentCommandPanel() {
348       return commandsToExecute[currentCommandIndex];
349     }
350
351     private void updateFileTree() {
352       try {
353         leftPanel.remove(tree);
354       } catch(NullPointerException noe) {
355         // First add
356       }
357       tree = new FileTree(currentDir);
358       leftPanel.add(tree, BorderLayout.CENTER);
359       FSController.getInstance().mainPanel.repaint();
360     }
361
362     private String sequenceToJSON() {
```

```
363     JSONArray data = new JSONArray();
364     try {
365       for(Component c : commandList.getComponents()) {
366         CommandPanel cp = (CommandPanel) c;
367         JSONObject entry = new JSONObject();
368         // Command
369         entry.put("command", cp.commandList.
              getSelectedItem().toString());
370         // Arguments
371         JSONArray arguments = new JSONArray();
372         for(Component p : cp.argumentsPanel.getComponents
              ()) {
373           JTextField a = (JTextField) ((JPanel) p).
                getComponent(1);
374           arguments.put(a.getText());
375         }
376
377         entry.put("arguments", arguments);
378
379         data.put(entry);
380       }
381     } catch (JSONException e1) {
382       e1.printStackTrace();
383     }
384     return data.toString();
385   }
386
387   private String sequenceToRuby(String filename, String
        description, String fsclient_path) {
388     String ruby = "";
389     ruby += "#!/usr/bin/ruby\n";
390     ruby += "\n";
391     ruby += "require␣'optparse'\n";
392     ruby += "require␣'pty'\n";
393     ruby += "require␣'expect'\n";
394     ruby += "\n";
395     ruby += "op␣=␣OptionParser.new␣do␣|opts|\n";
396     ruby += "␣␣opts.banner␣=␣\"Usage:␣"+filename+"␣token
          \"\n";
397     ruby += "␣␣opts.separator␣\"Description:␣"+
          description+"\"\n";
398     ruby += "end\n";
399     ruby += "op.parse!\n";
400     ruby += "\n";
```

```
401       ruby += "$token␣=␣ARGV[0]\n";
402       ruby += "\n";
403       ruby += "def␣fsclient(*args)\n";
404       ruby += "␣begin\n";
405       ruby += "␣␣␣PTY.spawn('"+fsclient_path+"',␣\"--token
              =#{$token}\",␣*args)␣do␣|r,␣w,␣pid|\n";
406       ruby += "␣␣␣␣␣␣loop␣{\n";
407       ruby += "␣␣␣␣␣␣␣␣out␣=␣r.expect(%r/^.+\\n$/io)\n";
408       ruby += "␣␣␣␣␣␣␣␣puts␣out␣unless␣out.nil?\n";
409       ruby += "␣␣␣␣␣␣}\n";
410       ruby += "␣␣␣end\n";
411       ruby += "␣rescue␣PTY::ChildExited␣=>␣e\n";
412       ruby += "␣end\n";
413       ruby += "end\n";
414       ruby += "\n";
415
416       for(Component c : commandList.getComponents()) {
417         CommandPanel cp = (CommandPanel) c;
418         ruby += "fsclient('"+cp.commandList.getSelectedItem
                ().toString()+"'";
419
420         for(Component p : cp.argumentsPanel.getComponents()
                ) {
421           JTextField a = (JTextField) ((JPanel) p).
                  getComponent(1);
422           ruby += ",␣'"+a.getText()+"'";
423         }
424
425         ruby += ")\n";
426       }
427
428       return ruby;
429     }
430
431     private void JSONToSequence(String raw) {
432       JPanel newSequence = new JPanel();
433       newSequence.setLayout(new BoxLayout(newSequence,
              BoxLayout.Y_AXIS));
434       try {
435         JSONArray parsed = new JSONArray(raw);
436
437         for(int i=0;i<parsed.length();i++) {
438           JSONObject entry = (JSONObject) parsed.get(i);
```

```
439             newSequence.add(new CommandPanel(newSequence.
                    getComponentCount(), entry));
440         }
441
442         commandListContainer.remove(commandList);
443         commandList = newSequence;
444         commandListContainer.add(commandList, BorderLayout.
                NORTH);
445         controller.mainPanel.repaint();
446       } catch (JSONException e) {
447         JOptionPane.showMessageDialog(controller.frame, "
                Wrong format.");
448       }
449     }
450
451 }
```

dtusat/components/ConnectPanel.java

```
1  package dtusat.components;
2
3  import dtusat.*;
4  import java.awt.FlowLayout;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7
8  import javax.swing.BoxLayout;
9  import javax.swing.ImageIcon;
10 import javax.swing.JButton;
11 import javax.swing.JCheckBox;
12 import javax.swing.JLabel;
13 import javax.swing.JPanel;
14 import javax.swing.JTextField;
15 import javax.swing.event.ChangeEvent;
16 import javax.swing.event.ChangeListener;
17
18 public class ConnectPanel extends JPanel {
19
20   public FSController controller;
21   public JTextField hostTextField, portTextField;
22   public JCheckBox autoLockCheckBox;
23   public JButton connectButton;
24
```

```
25  public ConnectPanel() {
26
27      controller = FSController.getInstance();
28
29      setLayout(new FlowLayout());
30
31      JPanel boxPanel = new JPanel();
32      boxPanel.setLayout(new BoxLayout(boxPanel, BoxLayout.
            Y_AXIS));
33      JPanel hostPanel = new JPanel();
34      JPanel portPanel = new JPanel();
35      JPanel buttonPanel = new JPanel();
36
37      JLabel hostLabel = new JLabel("Host:");
38      JLabel portLabel = new JLabel("Port:");
39
40      hostTextField = new JTextField("localhost", 15);
41      portTextField = new JTextField("3000", 15);
42
43      autoLockCheckBox = new JCheckBox("Autolock",
            controller.isAutoLocked);
44      autoLockCheckBox.addChangeListener(new ChangeListener
            () {
45        public void stateChanged(ChangeEvent arg0) {
46          controller.setAutoLocked(autoLockCheckBox.
                isSelected());
47        }
48      });
49
50      connectButton = new JButton("Connect", new ImageIcon(
            "src/dtusat/icons/connect.png"));
51      connectButton.addActionListener(new ActionListener()
            {
52        public void actionPerformed(ActionEvent arg0) {
53          controller.getSocket().connect(hostTextField.
                getText(), portTextField.getText());
54        }
55      });
56
57      hostPanel.add(hostLabel);
58      hostPanel.add(hostTextField);
59      portPanel.add(portLabel);
60      portPanel.add(portTextField);
61      buttonPanel.add(connectButton);
```

```
62      boxPanel.add(hostPanel);
63      boxPanel.add(portPanel);
64      boxPanel.add(autoLockCheckBox);
65      boxPanel.add(buttonPanel);
66      add(boxPanel);
67    }
68
69  }
```

dtusat/components/FSMenu.java

```
1   package dtusat.components;
2
3   import java.awt.event.ActionEvent;
4   import java.awt.event.ActionListener;
5
6   import javax.swing.ImageIcon;
7   import javax.swing.JCheckBoxMenuItem;
8   import javax.swing.JMenu;
9   import javax.swing.JMenuBar;
10  import javax.swing.JMenuItem;
11  import javax.swing.JSeparator;
12
13  import dtusat.FSController;
14
15  public class FSMenu extends JMenuBar implements
        ActionListener {
16
17    private JMenuItem disconnectMenuItem;
18    private JMenuItem quitMenuItem;
19    public JCheckBoxMenuItem autoLockItem;
20    private JMenuItem lockItem;
21    private JMenuItem unlockItem;
22    private FSController controller;
23
24    public FSMenu() {
25      controller = FSController.getInstance();
26
27      JMenu fileMenu = new JMenu("File");
28      quitMenuItem = new JMenuItem("Quit", new ImageIcon("
            src/dtusat/icons/door_out.png"));
29      quitMenuItem.addActionListener(this);
30      fileMenu.add(quitMenuItem);
```

```
31      add(fileMenu);
32
33      // Connection Menu
34      JMenu connectionMenu = new JMenu("Connection");
35
36      autoLockItem = new JCheckBoxMenuItem("Auto Lock",
            controller.isAutoLocked);
37      autoLockItem.addActionListener(this);
38      connectionMenu.add(autoLockItem);
39
40      lockItem = new JMenuItem("Lock", new ImageIcon("src/
            dtusat/icons/lock.png"));
41      lockItem.addActionListener(this);
42      connectionMenu.add(lockItem);
43
44      unlockItem = new JMenuItem("Unlock", new ImageIcon("
            src/dtusat/icons/unlock.png"));
45      unlockItem.addActionListener(this);
46      connectionMenu.add(unlockItem);
47
48      connectionMenu.add(new JSeparator());
49
50      disconnectMenuItem = new JMenuItem("Disconnect", new
            ImageIcon("src/dtusat/icons/disconnect.png"));
51      disconnectMenuItem.addActionListener(this);
52      connectionMenu.add(disconnectMenuItem);
53
54      add(connectionMenu);
55
56    }
57
58    public void actionPerformed(ActionEvent e) {
59      if(e.getSource() == disconnectMenuItem) {
60        controller.getSocket().disconnect();
61      }
62
63      if(e.getSource() == quitMenuItem) {
64        System.exit(0);
65      }
66
67      if(e.getSource() == lockItem) {
68        controller.lock();
69      }
70
```

```
71      if(e.getSource() == unlockItem) {
72        controller.unlock();
73      }
74
75      if(e.getSource() == autoLockItem) {
76        controller.setAutoLocked(autoLockItem.isSelected())
              ;
77      }
78    }
79
80  }
```

dtusat/components/FileTree.java

```
1   package dtusat.components;
2
3   import java.awt.BorderLayout;
4   import java.awt.event.ActionEvent;
5   import java.awt.event.ActionListener;
6   import java.io.File;
7
8   import javax.swing.JButton;
9   import javax.swing.JPanel;
10  import javax.swing.JTree;
11  import javax.swing.tree.DefaultMutableTreeNode;
12
13  public class FileTree extends JTree {
14
15    public FileTree(String root) {
16      super(createTree(root));
17    }
18
19    public static DefaultMutableTreeNode createTree(String
        dirname) {
20      File f = new File(dirname);
21      DefaultMutableTreeNode top = new
            DefaultMutableTreeNode();
22
23      top.setUserObject(f.getName());
24      if(f.isDirectory()) {
25        File fls[] = f.listFiles();
26        for(int i=0;i<fls.length;i++) {
27          top.insert(createTree(fls[i].getPath()),i);
```

```
28          }
29      }
30      return(top);
31    }
32
33  }
```

dtusat/components/HealthImagePanel.java

```
1   package dtusat.components;
2
3   import java.awt.Color;
4   import java.awt.Dimension;
5   import java.awt.Font;
6   import java.awt.Graphics;
7   import java.awt.Image;
8
9   import javax.swing.ImageIcon;
10  import javax.swing.JPanel;
11
12  public class HealthImagePanel extends JPanel {
13
14    private Image img;
15    private int margin;
16
17    public Integer[] data;
18
19    // [lower yellow, upper yellow, lower red, upper red]
20    public int[][] intervals = {
21        {},                 // Auto reset
22        {},             // Boot count
23        {},             // FS Error
24        {},             // Number of SIBs
25        {105, 115, 100, 120},    // I panel
26        {105, 115, 100, 120},    // I bat
27        {105, 115, 100, 120},    // V bat
28        {105, 115, 100, 120},    // V unreg
29        {105, 115, 100, 120},    // V reg
30        {105, 115, 100, 120}     // I reg
31    };
32
33    private Dimension size;
34
```

```
35    public HealthImagePanel() {
36      super();
37      img = new ImageIcon("src/dtusat/img/health_panel.png"
            ).getImage();
38      margin = 20;
39
40      setBackground(Color.white);
41      size = new Dimension(img.getWidth(null)+2*margin+200,
            img.getHeight(null)+2*margin);
42        setPreferredSize(size);
43        setMinimumSize(size);
44        setMaximumSize(size);
45        setSize(size);
46        setLayout(null);
47    }
48
49    public void paintComponent(Graphics g) {
50      g.drawImage(img, margin, margin, null);
51
52      int x_list = (int) (size.getWidth()-200);
53
54      try {
55        g.drawString("Auto reset status: "+data[0], x_list,
              10+margin);
56        g.drawString("Boot count: "+data[1], x_list, 30+
              margin);
57        g.drawString("FS Error: "+data[2], x_list, 50+
              margin);
58        g.drawString("Number of SIBs: "+data[3], x_list,
              70+margin);
59
60        g.setFont(new Font("Mono", Font.BOLD, 20));
61        g.setColor(Color.green);
62
63        Color old = g.getColor();
64
65        setHealthColor(g,4);g.drawString(""+data[4], 160+
              margin, 137+margin); // I panel
66        setHealthColor(g,5);g.drawString(""+data[5], 143+
              margin, 334+margin); // I bat
67        setHealthColor(g,6);g.drawString(""+data[6], 143+
              margin, 371+margin); // V bat
68        setHealthColor(g,7);g.drawString(""+data[7], 227+
              margin, 226+margin); // V Unreg
```

```
69          setHealthColor(g,8);g.drawString(""+data[8], 216+
                margin, 304+margin); // V reg
70          setHealthColor(g,9);g.drawString(""+data[9], 305+
                margin, 254+margin); // I reg
71
72          g.setColor(old);
73          // ---------- TODO Replace with real data
                ----------
74
75          g.setColor(new Color(0x888888));
76          g.drawString("20", 96+margin, 259+margin);   // T
                eps
77          g.drawString("20", 96+margin, 442+margin);   // T
                bat
78          g.drawString("20", 361+margin, 60+margin);   // T
                com
79          g.drawString("20", 393+margin, 525+margin); // T
                cpu
80          g.drawString("20", 465+margin, 71+margin);   // RSSI
81          g.drawString("110", 277+margin, 395+margin);// V 33
82          g.drawString("110", 277+margin, 443+margin);// I 33
83          g.drawString("110", 455+margin, 395+margin);// V 18
84          g.drawString("110", 455+margin, 443+margin);// I 18
85
86        } catch (NullPointerException e) {
87          // Data not set yet
88        }
89      }
90
91      public void setHealthColor(Graphics g, int i) {
92        Color color = Color.green;
93        int v = data[i];
94
95        if(v < intervals[i][0] || v > intervals[i][1])
96          color = Color.yellow;
97        if(v < intervals[i][2] || v > intervals[i][3])
98          color = Color.red;
99
100       g.setColor(color);
101     }
102 }
```

dtusat/components/HealthPanel.java

```
1  package dtusat.components;
2
3  import java.awt.BorderLayout;
4  import java.awt.Color;
5  import java.awt.FlowLayout;
6  import java.awt.Graphics;
7  import java.awt.Image;
8  import java.awt.event.ActionEvent;
9  import java.awt.event.ActionListener;
10 import java.text.SimpleDateFormat;
11 import java.util.Calendar;
12
13 import javax.net.ssl.SSLEngineResult.Status;
14 import javax.swing.BoxLayout;
15 import javax.swing.ImageIcon;
16 import javax.swing.JButton;
17 import javax.swing.JLabel;
18 import javax.swing.JPanel;
19 import javax.swing.JScrollPane;
20
21 import org.json.JSONArray;
22 import org.json.JSONException;
23
24 import dtusat.FSCallback;
25 import dtusat.FSController;
26 import dtusat.FSResponse;
27
28 public class HealthPanel extends JPanel {
29
30   private JPanel systemsPanel;
31   private JScrollPane scrollPane;
32   private HealthImagePanel healthImagePanel;
33   private JLabel lastUpdated;
34
35   public HealthPanel() {
36     setLayout(new BorderLayout());
37
38     JPanel northPanel = new JPanel(new FlowLayout(
         FlowLayout.LEFT));
39     add(northPanel, BorderLayout.NORTH);
40
41       JButton refreshButton = new JButton("Update␣status"
           );
```

```
42        refreshButton.addActionListener(new ActionListener
              () { public void actionPerformed(ActionEvent e)
              {refreshList();}});
43        northPanel.add(refreshButton);
44
45        lastUpdated = new JLabel();
46        northPanel.add(lastUpdated);
47
48      healthImagePanel = new HealthImagePanel();
49      scrollPane = new JScrollPane(healthImagePanel);
50      scrollPane.setBackground(Color.white);
51
52      add(scrollPane, BorderLayout.CENTER);
53
54
55    }
56
57    protected void refreshList() {
58      FSController.getInstance().getSocket().request("
            health_status", new FSCallback() {
59        public void onResponse(FSResponse response) {
60          if(response.status == FSController.
                FS_HEALTH_STATUS) {
61            JSONArray json_data = response.dataAsArray();
62            Integer[] data = new Integer[json_data.length()
                  ];
63            for(int i=0;i<json_data.length();i++) {
64              try {
65                data[i] = json_data.getInt(i);
66              } catch (JSONException e) {
67                e.printStackTrace();
68              }
69            }
70            healthImagePanel.data = data;
71
72
73            Calendar cal = Calendar.getInstance();
74              SimpleDateFormat sdf = new SimpleDateFormat("
                    dd/MM/yy␣HH:mm:ss");
75              String time = sdf.format(cal.getTime());
76
77            lastUpdated.setText("Last␣updated:␣"+time);
78            FSController.getInstance().mainPanel.repaint();
79          }
```

```
80          }
81        });
82    }
83 }
```

dtusat/components/LocalScriptPanel.java

```
1  package dtusat.components;
2
3  import java.io.BufferedReader;
4  import java.io.BufferedWriter;
5  import java.io.IOException;
6  import java.io.InputStreamReader;
7  import java.io.OutputStreamWriter;
8
9  import dtusat.FSController;
10 import dtusat.Logger;
11
12 public class LocalScriptPanel extends ScriptPanel {
13
14   Logger logger;
15
16   public LocalScriptPanel(String name, String path,
         String help) {
17     super(name, path, help);
18     logger = FSController.getInstance();
19   }
20
21   public void execute() {
22     outputArea.setText("");
23     try {
24       String token = FSController.getInstance().getSocket
             ().token;
25
26       String[] args = getArguments().split("␣");
27       String[] argList = new String[2+args.length];
28       argList[0] = path;
29       argList[1] = token;
30       for(int i=0;i<args.length;i++)
31         argList[2+i] = args[i];
32
33       Process child = new ProcessBuilder(argList).start()
             ;
```

```
34          BufferedReader out = new BufferedReader(new
                InputStreamReader(child.getInputStream()));
35          child.waitFor();
36
37          String line;
38          while ((line = out.readLine()) != null) {
39            outputArea.append(line+"\n");
40          }
41
42        } catch (IOException e) {
43          e.printStackTrace();
44        } catch (InterruptedException e) {
45          e.printStackTrace();
46        }
47      }
48  }
```

dtusat/components/LocalScriptsPanel.java

```
1   package dtusat.components;
2
3   import java.awt.BorderLayout;
4   import java.awt.FlowLayout;
5   import java.awt.event.ActionEvent;
6   import java.awt.event.ActionListener;
7   import java.io.BufferedReader;
8   import java.io.File;
9   import java.io.IOException;
10  import java.io.InputStreamReader;
11  import java.util.Enumeration;
12
13  import javax.swing.BoxLayout;
14  import javax.swing.ImageIcon;
15  import javax.swing.JButton;
16  import javax.swing.JFileChooser;
17  import javax.swing.JPanel;
18  import javax.swing.JScrollPane;
19  import javax.swing.JSplitPane;
20  import javax.swing.JTree;
21  import javax.swing.event.TreeSelectionEvent;
22  import javax.swing.event.TreeSelectionListener;
23  import javax.swing.tree.DefaultMutableTreeNode;
24
```

```
25 import org.json.JSONArray;
26 import org.json.JSONException;
27 import org.json.JSONObject;
28
29 import dtusat.FSController;
30 import dtusat.FSResponse;
31 import dtusat.FSSocket;
32
33 public class LocalScriptsPanel extends JPanel implements
       TreeSelectionListener {
34
35   FSSocket socket;
36   JScrollPane scriptView;
37   JPanel refreshPanel, scriptPanel;
38   JButton refreshButton;
39   private DefaultMutableTreeNode treeTop;
40   private JTree fileTree;
41   private String scriptsDir;
42   private JSplitPane splitPane;
43   private JPanel leftPanel;
44   private JPanel leftNorthPanel;
45   private JPanel rightPanel;
46
47   public LocalScriptsPanel() {
48     setLayout(new BorderLayout());
49
50     // Splitpane
51     splitPane = new JSplitPane();
52     add(splitPane, BorderLayout.CENTER);
53
54     // Left
55     leftPanel = new JPanel(new BorderLayout());
56     splitPane.setLeftComponent(leftPanel);
57
58       // North
59       leftNorthPanel = new JPanel(new FlowLayout(
             FlowLayout.LEFT));
60       leftPanel.add(leftNorthPanel, BorderLayout.NORTH);
61
62         refreshButton = new JButton("Set␣Dir",new
               ImageIcon("src/dtusat/icons/folder_explore.png
               "));
63         refreshButton.addActionListener(new
               ActionListener() { public void actionPerformed
```

```
            (ActionEvent e) {refreshList();}});
64          leftNorthPanel.add(refreshButton);
65
66          // Center
67          treeTop = new DefaultMutableTreeNode("Local Scripts
                ");
68          fileTree = new JTree(treeTop);
69          fileTree.addTreeSelectionListener(this);
70          leftPanel.add(new JScrollPane(fileTree),
                BorderLayout.CENTER);
71
72        // Right
73        rightPanel = new JPanel(new BorderLayout());
74        splitPane.setRightComponent(rightPanel);
75
76          // Center
77          scriptPanel = new JPanel();
78          scriptPanel.setLayout(new BorderLayout());
79          scriptView = new JScrollPane(scriptPanel);
80          rightPanel.add(scriptView, BorderLayout.CENTER);
81
82    }
83
84    private void refreshList() {
85      final JFileChooser fc = new JFileChooser();
86      fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY
            );
87
88      if(fc.showOpenDialog(this) == JFileChooser.
            APPROVE_OPTION) {
89
90          treeTop.removeAllChildren();
91
92          scriptsDir = fc.getSelectedFile().getAbsolutePath()
                ;
93          walkDir(new File(scriptsDir));
94
95            fileTree.expandRow(0);
96            FSController.getInstance().mainPanel.repaint();
97      }
98    }
99
100   private void walkDir(File folder) {
101       File[] listOfFiles = folder.listFiles();
```

```
102         for (int i = 0; i < listOfFiles.length; i++) {
103           File current = listOfFiles[i];
104           if(current.isDirectory()) {
105             walkDir(current);
106           } else if(current.isFile()) {
107               File f = listOfFiles[i];
108
109               if(f.canExecute()) {
110                 // Get help info
111                 String help = "";
112                 try {
113                   String token = FSController.getInstance()
                          .getSocket().token;
114               Process child;
115               child = new ProcessBuilder(f.getAbsolutePath
                      (), "--help").start();
116               BufferedReader out = new BufferedReader(new
                      InputStreamReader(child.getInputStream()))
                      ;
117               child.waitFor();
118
119               String line;
120               while ((line = out.readLine()) != null)
121                 help += line+"\n";
122
123                 } catch (IOException e) {
124               e.printStackTrace();
125             } catch (InterruptedException e) {
126               e.printStackTrace();
127             }
128
129             insertScriptAsNode(f.getAbsolutePath(), help);
130               }
131           }
132         }
133     }
134
135     private void insertScriptAsNode(String path, String
          help) {
136
137       DefaultMutableTreeNode top = treeTop;
138
139       // Make sure the directories are inserted
140       String[] splittedPath = path.split("/");
```

```
141     String scriptName = splittedPath[splittedPath.length
            -1];
142     int skipCount = scriptsDir.split("/").length;
143
144     // Skip until "scripts" folder and stop before the
            actual script
145     for(int i=skipCount;i<splittedPath.length-1;i++) {
146       String dirName = splittedPath[i];
147       boolean isInserted = false;
148
149       // Is directory already in the tree?
150       for(Enumeration<DefaultMutableTreeNode> e = top.
              children(); e.hasMoreElements();) {
151         DefaultMutableTreeNode existing = e.nextElement()
                ;
152         if(existing.toString().equals(dirName)) {
153           isInserted = true;
154           top = existing;
155           break;
156         }
157       }
158
159       // Directory is not in tree
160       if(!isInserted) {
161         DefaultMutableTreeNode newDir = new
                DefaultMutableTreeNode(dirName);
162         top.add(newDir);
163         top = newDir;
164       }
165     }
166
167     // Now insert the script node
168     top.add(new ScriptTreeNode(scriptName, path, help));
169   }
170
171   @Override
172   public void valueChanged(TreeSelectionEvent arg0) {
173     DefaultMutableTreeNode node = (DefaultMutableTreeNode
            ) fileTree.getLastSelectedPathComponent();
174
175     // Nothing is selected
176     if(node == null) return;
177
178     if(node.isLeaf()) {
```

```
179         try {
180           ScriptTreeNode scriptNode = (ScriptTreeNode) node
                ;
181           scriptPanel.removeAll();
182           scriptPanel.add(new LocalScriptPanel(scriptNode.
                name, scriptNode.path, scriptNode.help),
                BorderLayout.NORTH);
183           FSController.getInstance().mainPanel.repaint();
184         } catch(ClassCastException cce) {
185           // Tree is not loaded yet
186         }
187       }
188     }
189 }
```

dtusat/components/MainPanel.java

```
1  package dtusat.components;
2
3  import java.awt.BorderLayout;
4  import java.awt.Component;
5  import java.awt.FlowLayout;
6
7  import javax.swing.BorderFactory;
8  import javax.swing.ImageIcon;
9  import javax.swing.JLabel;
10 import javax.swing.JPanel;
11 import javax.swing.JScrollPane;
12 import javax.swing.JSplitPane;
13 import javax.swing.JTextArea;
14 import javax.swing.JToolBar;
15
16 import dtusat.FSController;
17 import dtusat.Logger;
18
19 public class MainPanel extends JPanel implements Logger {
20
21   public JToolBar toolBar;
22   public JSplitPane splitPane;
23   public JPanel logPanel, buttonPanel;
24   public JTextArea logArea;
25   FSController controller;
26   private JScrollPane logScrollPane;
```

```
27    public JLabel lockStatus;
28    private JLabel connectedStatus;
29
30    public MainPanel() {
31      controller = FSController.getInstance();
32      setLayout(new BorderLayout());
33
34      // Toolbars
35      toolBar = new JToolBar();
36      add(toolBar, BorderLayout.PAGE_END);
37
38      // Connected status
39      connectedStatus = new JLabel();
40      showDisconnectedStatus();
41      toolBar.add(connectedStatus);
42
43      // Lock status
44      lockStatus = new JLabel();
45      showUnLockedStatus();
46      toolBar.add(lockStatus);
47
48      // Splitter
49      splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT)
            ;
50      add(splitPane, BorderLayout.CENTER);
51
52      // Log
53      logArea = new JTextArea();
54      logArea.setBorder(BorderFactory.createEmptyBorder
            (5,5,5,5));
55      splitPane.setRightComponent(new JScrollPane(logArea))
            ;
56
57    }
58
59    public void showConnectedStatus() {
60      connectedStatus.setIcon(new ImageIcon("src/dtusat/
            icons/green.png"));
61      connectedStatus.setText("Connected");
62    }
63
64    public void showDisconnectedStatus() {
65      connectedStatus.setIcon(new ImageIcon("src/dtusat/
            icons/red.png"));
```

```
66      connectedStatus.setText("Disconnected");
67    }
68
69    public void showLockedStatus() {
70      lockStatus.setIcon(new ImageIcon("src/dtusat/icons/
            green.png"));
71      lockStatus.setText("Locked");
72    }
73
74    public void showUnLockedStatus() {
75      lockStatus.setIcon(new ImageIcon("src/dtusat/icons/
            red.png"));
76      lockStatus.setText("Unlocked");
77    }
78
79    public void log(String msg) {
80      logArea.setText(msg+"\n"+logArea.getText());
81      logArea.setSelectionStart(0);
82      logArea.setSelectionEnd(0);
83    }
84
85    public void setTopPanel(Component component) {
86      splitPane.setLeftComponent(component);
87      splitPane.setDividerLocation(0.66);
88      splitPane.updateUI();
89    }
90
91  }
```

dtusat/components/MainTabs.java

```
1   package dtusat.components;
2
3   import javax.swing.BorderFactory;
4   import javax.swing.JTabbedPane;
5
6   public class MainTabs extends JTabbedPane {
7
8     public MainTabs() {
9       setTabPlacement(JTabbedPane.LEFT);
10
11      addTab("Server␣Scripts", new ServerScriptsPanel());
12      addTab("Local␣Scripts", new LocalScriptsPanel());
```

```
13      addTab("Command␣Sequences", new CommandSequencesPanel
            ());
14      addTab("Health␣Status", new HealthPanel());
15    }
16
17  }
```

dtusat/components/ScriptPanel.java

```
1   package dtusat.components;
2
3   import java.awt.BorderLayout;
4   import java.awt.Color;
5   import java.awt.Dimension;
6   import java.awt.FlowLayout;
7   import java.awt.Font;
8   import java.awt.Insets;
9   import java.awt.event.ActionEvent;
10  import java.awt.event.ActionListener;
11
12  import javax.swing.BoxLayout;
13  import javax.swing.JButton;
14  import javax.swing.JLabel;
15  import javax.swing.JPanel;
16  import javax.swing.JTextArea;
17  import javax.swing.JTextField;
18  import javax.swing.border.TitledBorder;
19
20  import dtusat.FSController;
21  import dtusat.FSResponse;
22
23  public class ScriptPanel extends JPanel {
24
25    String name, path, help;
26    JPanel descriptionPanel;
27    JLabel nameLabel;
28
29    JTextArea helpTextArea;
30    JTextField argumentsTextField;
31    JButton executeButton;
32    JLabel argumentsLabel;
33    JTextArea outputArea;
34
```

```
35    public ScriptPanel(String name, String path, String
          help) {
36      this.name = name;
37      this.path = path;
38      this.help = help;
39
40      setLayout(new BorderLayout());
41      setBorder(new TitledBorder(name));
42
43      JPanel boxLayout = new JPanel();
44      boxLayout.setLayout(new BoxLayout(boxLayout,
            BoxLayout.Y_AXIS));
45      add(boxLayout, BorderLayout.NORTH);
46
47      // Description
48      descriptionPanel = new JPanel();
49      descriptionPanel.setLayout(new BoxLayout(
            descriptionPanel, BoxLayout.Y_AXIS));
50      boxLayout.add(descriptionPanel);
51
52      nameLabel = new JLabel(name);
53      helpTextArea = new JTextArea(help);
54      helpTextArea.setEditable(false);
55      helpTextArea.setBackground(this.getBackground());
56      helpTextArea.setMargin(new Insets(5,5,5,5));
57      helpTextArea.setFont(Font.decode("monospaced"));
58      descriptionPanel.add(helpTextArea);
59
60      // Execution Panel
61      JPanel executionPanel = new JPanel(new FlowLayout(
            FlowLayout.LEFT));
62      boxLayout.add(executionPanel, BorderLayout.SOUTH);
63
64      argumentsLabel = new JLabel("Arguments:␣");
65      executionPanel.add(argumentsLabel);
66
67      argumentsTextField = new JTextField(20);
68      executionPanel.add(argumentsTextField);
69
70      executeButton = new JButton("Execute");
71      executeButton.addActionListener(new ActionListener()
            { public void actionPerformed(ActionEvent e) {
            execute();} });
72      executionPanel.add(executeButton);
```

```
73
74      // Outpanel
75      outputArea = new JTextArea();
76      outputArea.setMargin(new Insets(5,5,5,5));
77      boxLayout.add(outputArea);
78    }
79
80    protected String getArguments() {
81      return argumentsTextField.getText();
82    }
83
84    public void execute() {
85    }
86  }
```

dtusat/components/ScriptTreeNode.java

```
1   package dtusat.components;
2
3   import javax.swing.tree.DefaultMutableTreeNode;
4   import javax.swing.tree.MutableTreeNode;
5
6   public class ScriptTreeNode extends
        DefaultMutableTreeNode implements MutableTreeNode {
7
8     String name, path, help;
9
10    public ScriptTreeNode(String name, String path, String
          help) {
11      super(name);
12      this.name = name;
13      this.path = path;
14      this.help = help;
15    }
16
17  }
```

dtusat/components/ServerScriptPanel.java

```
1   package dtusat.components;
2
3   import dtusat.FSCallback;
4   import dtusat.FSController;
```

```
5  import dtusat.FSResponse;
6
7  public class ServerScriptPanel extends ScriptPanel {
8
9    public ServerScriptPanel(String name, String path,
         String help) {
10     super(name, path, help);
11   }
12
13   public void execute() {
14     outputArea.setText("");
15     FSController.getInstance().getSocket().request("
         run_script␣"+path+"␣"+getArguments(), new
         FSCallback() {
16       public void onResponse(FSResponse response) {
17         if(response.isPartial())
18           outputArea.append(response.dataAsString());
19       }
20     });
21   }
22 }
```

dtusat/components/ServerScriptsPanel.java

```
1  package dtusat.components;
2
3  import java.awt.BorderLayout;
4  import java.awt.FlowLayout;
5  import java.awt.GridLayout;
6  import java.awt.event.ActionEvent;
7  import java.awt.event.ActionListener;
8  import java.util.Enumeration;
9
10 import javax.swing.BoxLayout;
11 import javax.swing.ImageIcon;
12 import javax.swing.JButton;
13 import javax.swing.JLabel;
14 import javax.swing.JPanel;
15 import javax.swing.JScrollPane;
16 import javax.swing.JSplitPane;
17 import javax.swing.JTree;
18 import javax.swing.event.TreeSelectionEvent;
19 import javax.swing.event.TreeSelectionListener;
```

```
20 import javax.swing.tree.DefaultMutableTreeNode;
21 import javax.swing.tree.TreeNode;
22 import javax.xml.bind.annotation.XmlElementRef.DEFAULT;
23
24 import org.json.JSONArray;
25 import org.json.JSONException;
26 import org.json.JSONObject;
27
28 import dtusat.FSController;
29 import dtusat.FSResponse;
30 import dtusat.FSCallback;
31 import dtusat.FSSocket;
32
33 public class ServerScriptsPanel extends JPanel implements
       TreeSelectionListener {
34
35   FSSocket socket;
36   JScrollPane scriptView;
37   JPanel refreshPanel, scriptPanel;
38   JButton refreshButton;
39   private DefaultMutableTreeNode treeTop;
40   private JTree fileTree;
41   private JSplitPane splitPane;
42   private JPanel leftPanel;
43   private JPanel leftNorthPanel;
44   private JPanel rightPanel;
45
46   public ServerScriptsPanel() {
47     socket = FSController.getInstance().getSocket();
48
49     setLayout(new BorderLayout());
50
51     // Splitpane
52     splitPane = new JSplitPane();
53     add(splitPane, BorderLayout.CENTER);
54
55     // Left
56     leftPanel = new JPanel(new BorderLayout());
57     splitPane.setLeftComponent(leftPanel);
58
59       // North
60       leftNorthPanel = new JPanel(new FlowLayout(
             FlowLayout.LEFT));
61       leftPanel.add(leftNorthPanel, BorderLayout.NORTH);
```

```
62
63          refreshButton = new JButton("Refresh List", new
                ImageIcon("src/dtusat/icons/refresh.png"));
64          refreshButton.addActionListener(new
                ActionListener() { public void actionPerformed
                (ActionEvent e) {refreshList();}});
65          leftNorthPanel.add(refreshButton);
66
67        // Center
68        treeTop = new DefaultMutableTreeNode("Server
              Scripts");
69        fileTree = new JTree(treeTop);
70        fileTree.addTreeSelectionListener(this);
71        leftPanel.add(new JScrollPane(fileTree),
              BorderLayout.CENTER);
72
73      // Right Panel
74      rightPanel = new JPanel(new BorderLayout());
75      splitPane.setRightComponent(rightPanel);
76
77        // Script Panel
78        scriptPanel = new JPanel();
79        scriptPanel.setLayout(new BorderLayout());
80        scriptView = new JScrollPane(scriptPanel);
81        rightPanel.add(scriptView, BorderLayout.CENTER);
82
83    }
84
85    private void refreshList() {
86      socket.request("list_scripts", new FSCallback() {
87        public void onResponse(FSResponse response) {
88          if(response.isSuccess()) {
89            treeTop.removeAllChildren();
90
91            JSONArray scripts = response.dataAsArray();
92
93            try {
94              for(int i=0;i<scripts.length();i++) {
95                JSONObject script = (JSONObject) scripts.
                      get(i);
96                insertScriptAsNode(script.getString("path")
                      , script.getString("help"));
97              }
98            } catch (JSONException e) {
```

```
99              e.printStackTrace();
100           }
101
102           fileTree.expandRow(0);
103           FSController.getInstance().mainPanel.repaint();
104         }
105       }
106     });
107   }
108
109   private void insertScriptAsNode(String path, String
        help) {
110
111     DefaultMutableTreeNode top = treeTop;
112
113     // Make sure the directories are inserted
114     String[] splittedPath = path.split("/");
115     String scriptName = splittedPath[splittedPath.length
          -1];
116
117     // Stop before the actual script
118     for(int i=0;i<splittedPath.length-1;i++) {
119       String dirName = splittedPath[i];
120       boolean isInserted = false;
121
122       // Is directory already in the tree?
123       for(Enumeration<DefaultMutableTreeNode> e = top.
            children(); e.hasMoreElements();) {
124         DefaultMutableTreeNode existing = e.nextElement()
              ;
125         if(existing.toString().equals(dirName)) {
126           isInserted = true;
127           top = existing;
128           break;
129         }
130       }
131
132       // Directory is not in tree
133       if(!isInserted) {
134         DefaultMutableTreeNode newDir = new
              DefaultMutableTreeNode(dirName);
135         top.add(newDir);
136         top = newDir;
137       }
```

```
138       }
139
140       // Now insert the script node
141       top.add(new ScriptTreeNode(scriptName, path, help));
142     }
143
144     @Override
145     public void valueChanged(TreeSelectionEvent e) {
146       DefaultMutableTreeNode node = (DefaultMutableTreeNode
              ) fileTree.getLastSelectedPathComponent();
147
148       // Nothing is selected
149       if(node == null) return;
150
151       if(node.isLeaf()) {
152         try {
153           ScriptTreeNode scriptNode = (ScriptTreeNode) node
                  ;
154           scriptPanel.removeAll();
155           scriptPanel.add(new ServerScriptPanel(scriptNode.
                  name, scriptNode.path, scriptNode.help),
                  BorderLayout.NORTH);
156           FSController.getInstance().mainPanel.repaint();
157         } catch(ClassCastException cce) {
158           // Tree is not loaded yet
159         }
160       }
161     }
162 }
```

dtusat/components/SubSystemPanel.java

```
1  package dtusat.components;
2
3  import java.awt.BorderLayout;
4  import java.awt.Insets;
5
6  import javax.swing.BoxLayout;
7  import javax.swing.Icon;
8  import javax.swing.ImageIcon;
9  import javax.swing.JLabel;
10 import javax.swing.JPanel;
11 import javax.swing.JTextArea;
```

```
12
13 public class SubSystemPanel extends JPanel {
14
15   public static final int GREEN = 0;
16   public static final int YELLOW = 1;
17   public static final int RED = 2;
18
19   public SubSystemPanel(String name, int status, String
       description) throws UnknownStatusException {
20     setLayout(new BorderLayout());
21     //setBorder(BorderFactory.createLineBorder(Color.
         BLACK, 1));
22
23     JPanel containerPanel = new JPanel();
24
25     //containerPanel.setLayout(new BoxLayout(
         containerPanel, BoxLayout.Y_AXIS));
26     //add(containerPanel, BorderLayout.NORTH);
27     containerPanel.setLayout(new BorderLayout());
28     add(containerPanel, BorderLayout.CENTER);
29
30     JLabel headerLabel = new JLabel(name, iconForStatus(
         status), JLabel.LEFT);
31     JTextArea descriptionTextArea = new JTextArea(
         description);
32     descriptionTextArea.setBackground(this.getBackground
         ());
33     descriptionTextArea.setMargin(new Insets(2,5,5,5));
34     descriptionTextArea.setLineWrap(true);
35
36     containerPanel.add(headerLabel, BorderLayout.NORTH);
37     containerPanel.add(descriptionTextArea, BorderLayout.
         CENTER);
38   }
39
40   private Icon iconForStatus(int status) throws
       UnknownStatusException {
41     if(!(status == GREEN || status == YELLOW || status ==
         RED))
42       throw new UnknownStatusException();
43
44     ImageIcon icon = null;
45     switch(status) {
```

```
46          case GREEN: icon =  new ImageIcon("src/dtusat/icons
                /green.png"); break;
47          case YELLOW: icon = new ImageIcon("src/dtusat/icons
                /yellow.png"); break;
48          case RED: icon =  new ImageIcon("src/dtusat/icons/
                red.png"); break;
49      }
50      return icon;
51    }
52
53  }
```

dtusat/components/UnknownStatusException.java

```
1  package dtusat.components;
2
3  public class UnknownStatusException extends Exception {
4
5  }
```

Bibliography

[1] Eventmachine Documentation. 28-06-2010 <http://eventmachine.rubyforge.org>

[2] JSON Format. 28-06-2010 <http://www.json.org>