Command Editor for DTUsat-2

Alexander Paulsen

Kongens Lyngby 2012
IMM-BSc-2012-25

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

Summary (English)

The goal of this thesis is to develop an editor for easier definition of commands, confirmations, signals and teledata types for the satellite project DTUsat-2. The editor is part of the satellite development process and must support this. An early operational prototype is developed and later extended with more advanced features.

The editor has been developed, documented, tested, tested by a future user and is now used in the project for defining types. The editor features a graphical user interface, a tree for navigation, version numbering, preferences, cross references to other definitions, validation as well as input validation and more advanced version control features.

Resumé (Danish)

Målet for denne afhandling er, at udvikle en editor for nemmere definition af kommandoer, bekræftelser, signaler og teledata for satellit-projektet DTUsat-2. Editoren er en del af og støtter udviklingen af satelliten. En tidlig fungerende prototype er udviklet, og er senere udvidet med flere avancerede funktioner.

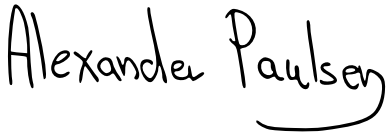
Editoren er blevet udviklet, dokumenteret, testet, testet af en kommende bruger og anvendes nu i projektet til at definere typer. Editoren har en grafisk brugergrænseflade, et træ til navigation, versionsnummerering, præferencer, krydsreferencer til andre definitioner, validering samt validering af input og mere avancerede versionskontrol funktioner.

Preface

This thesis was prepared from the 1st of February to the 1st of July at the department of Informatics and Mathematical Modeling at the Technical University of Denmark in fulfillment of the requirements for acquiring an B.Sc. in Informatics. The thesis was prepared with Hans Henrik Løvengreen as supervisor.

At first a thanks to Hans Henrik Løvengreen for being the supervisor of this project, and for all the research and help he has contributed to the development of the editor, report and project. A thanks to the DTUsat-2 project, without them the editor would not make as much sense. Especially a thank you to Rasmus Ljungmann Pedersen for testing the editor and providing feedback used in this report and the development of the editor. I would also like to thank my family through the developing of the project, a special thank to Thea Paulsen, Christian Paulsen and Benedikte Larsen for correcting and providing feedback on this report and project.

Lyngby, 01-July-2012



Alexander Paulsen, s093259

Contents

Chapter 1

Introduction

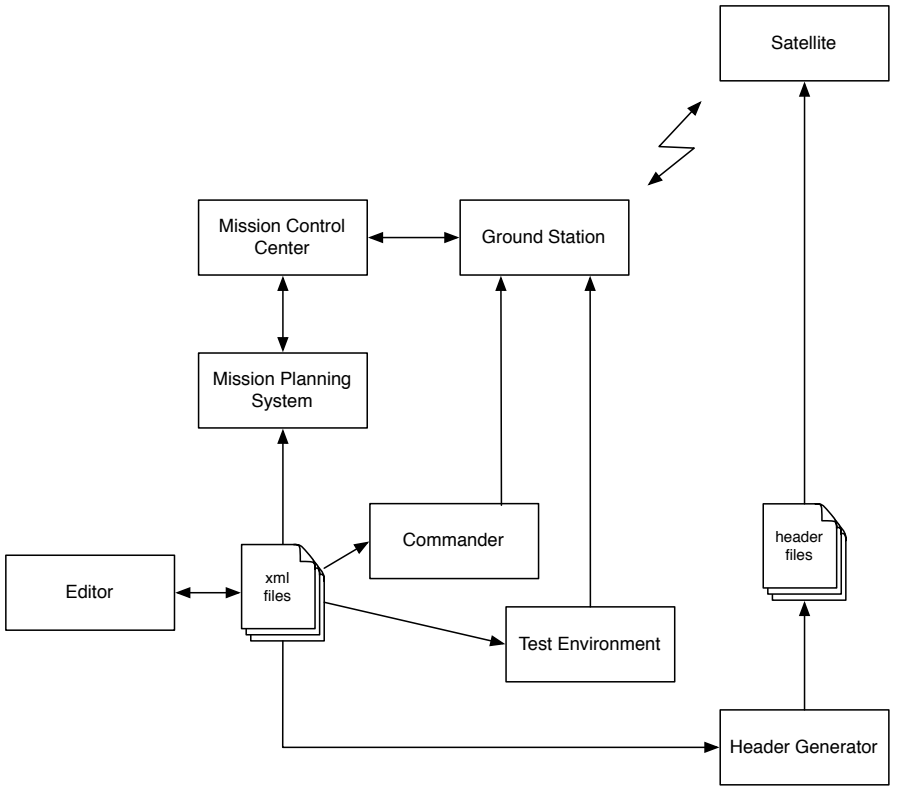
In 2005 the project DTUsat-2 was started on DTU. The goal for the project was to send a satellite out in space. A contest was made and the winning mission was to solve the mystery of bird migration, by tracking cuckoos from space[1][11].

To solve this, both hardware and software were needed and created. The satellite follows the idea of CubeSat – having a small volume and weight[4].

For the satellite to function it has to receive and act on different commands. These commands are defined in a underlying xml-format. To control the satellite's mission, a mission control center and mission planning system are created[12].

As seen in figure 1.1 the Mission Planning System (MPS) transmits commands and controls the satellite via the Mission Control Center (MCC) and the Ground Station (GS). The satellite responds to these by a confirmation and possibly tele data. The satellite and GS, as well as the other components, must be able to run on different hardware and software. To understand each other, they must agree to a common format, this format is defined by the xml-files. For the satellite to understand the commands, c header files are needed, these are created using the tool Header Generator. For test purposes, the MPS and MCC can be replaced by the Commander. Simultaneously with this project, a test environment is developed.

Figure 1.1: Overview of the project. The header files are generated and part of the satellite before launch. When launched the ground station and satellite communicates via a satellite connection.



All these different software components need the command, confirmation, tele data and signal definitions. Because of the increasing amount of definitions, and for making the development of the satellite easier, an editor to make these definitions were needed. A normal xml-editor would not be sufficient due to the amount of tags where some is inline and others not. A special purpose editor would give a much better overview of the defined types, make the definition easier by introducing validation and making cross references from commands to confirmations possible.

1.1 Problem definition

The aim of this thesis is to create an editor that can define commands for DTUsat-2 in an efficient way. The underlying xml-format is used by different components in the system, but an editor for easy definition of new commands with parameters of different types is missing. The editor is part of the satellite development process and must support this.

1.2 Development approach

The development process was iterative. An early operational prototype was wanted, later more features was added for a better user experience. At last more advanced features such as version control was added.

1.3 Report structure

Because of the iterative development of this project, the report is split into two parts. A basic part (chapter 2 to chapter 5) which will describe the more basic part of the program with the graphical user interface, panels and more.

The second part (chapter 6) being about the more advanced part. The advanced part is the editor with a more developer oriented view. For the program to better support a development environment, version history functionality will be implemented.

1.3.1 Report outline

Chapter 1 **Introduction** to the DTUsat-2 project and the editor, problem definition.

The basic part Describing the more basic components of the editor

Chapter 2 **Analysis of requirements** is an analysis on the existing software components and an analysis of the requirements for the editor, leading to a requirements specification.

Chapter 3 **Design** describes how the editor was designed, and why the design became what it is. Describes mainly the design of the implementation, but also the visual design.

Chapter 4 **Implementation** describes the implementation of the editor and the decisions made in the process.

Chapter 5 **Testing** describes the test strategy and how the editor was tested including the results.

The advanced part Describing the more advanced components of the editor, the version control tools. The advanced part is chapter 6.

Section 6.1 **Requirement analysis** is an analysis of the requirements to be met for the version control tools to be a success. Leading on to a requirement specification.

Section 6.2 **Design** of how the version control tools is part of the editor and design of the structure of the implementation.

Section 6.3 **Implementation** of the version control tools and usage of the API.

Section 6.4 **Testing** of the version control tools and its cooperation with the editor.

Chapter 7 **Conclusion** The final conclusion of the thesis discussing the outcome of the editor.

1.3.2 Wording and text style

- The word "type" is used to describe a `CommandType`, `ConfirmationType`, `SignalType` or `TeledataType`. More specific what a type is, is covered in section 2.2.1.
- The word "revision" is used to describe the revision number on the version control server (svn).

- The word “version” is used to describe the version number stated in the xml-file. The version number is covered in 4.2.4. An example could be `0.1.0` (2012/6/26).
- The abbreviations “XML” (Extensible Markup Language) and “SVN” (Apache Subversion) is so common in this report that they are written in lowercase.
- When referring to objects or methods or any thing else implementation specific the `typewriter` font is used.
- When referring to the keywords in the MOSCOW method (The method is described in section 2.3.1.1) the keywords are in **bold** font.

Chapter 2

Requirement analysis

As the whole DTUsat-2 project is under development, all problems are not yet known. The editor is part of this development, new problems in need of a solution is often discovered. Most of the problems are open and thus have no obvious solution.

2.1 Constraints

The editor is a development tool which will be used by a few developers under different operating systems; therefore the tool must be platform-independent. This makes Java an obvious choice of language because of its platform independence, and the widely use in other components of the project.

For reading and writing xml the framework Simple[10] is already being used in the project. Using the same framework to read and create xml as other components in the project removes possible errors when serializing the xml.

2.2 Analysis of existing software components

The whole DTUsat-2 project consists of an increasing number of different components. For an overview look at Figure 1.1 starting on page 2.

It is important for the editor to be a part of this.

2.2.1 Overview of types

Before the MPS was created, each definition of a command, confirmation, tele data or signal was defined in its own java class. This structure was not very flexible. Instead the following, more flexible way to define types was suggested by the authors of the MPS. A type is a definition of a command, confirmation, tele data or signal. A type is defined by a unique name and a unique id, a description and a list of zero or more parameters. Other than that, a Confirmation also holds information on its severity and a comment. A command holds a comment, a common name, a warning, a list of legal subsystems to be send to and a list of legal confirmation to confirm the command. A parameter is defined by a unique name, a description and a name of the sensor it was from. At the moment a parameter can be of six different types of the following manner; integer, byte array, enumeration, double, boolean and string.

In some cases i.e. in `IntParameter` a String is used to hold a number, this is chosen as binary or hexadecimal representation of the number is sometimes wanted instead of the decimal. To use a string, instead of specifying which representation is wanted, is chosen as the solution in the satellite project.

As seen in figure 2.1 a `TypeMap` holds a list of zero or more `Gentypes`. `GenType` is an abstract class; `ConfirmationType`, `SignalType`, `TeledataType` and `CommandType` extends `GenType`. Each previously mentioned type holds different fields. Inherited using getter and setters all holds a `TypeID`, a `name` and a list of zero or more `Parameters`.

A `Parameter`, seen in figure 2.2 on page 10, holds some fields and a `ParameterType` defining what kind of parameter it is. `ParameterType` is an abstract class and inherited by `IntParameter`, `ByteArrayParameter`, `BooleanParameter`, `StringParameter`, `DoubleParameter` and `EnumParameter`. Each parameter type holds different fields to define them; `EnumParameter` holds a list of name and value pairs.

Figure 2.1: Simplified class diagram showing how the different types is designed in the project.

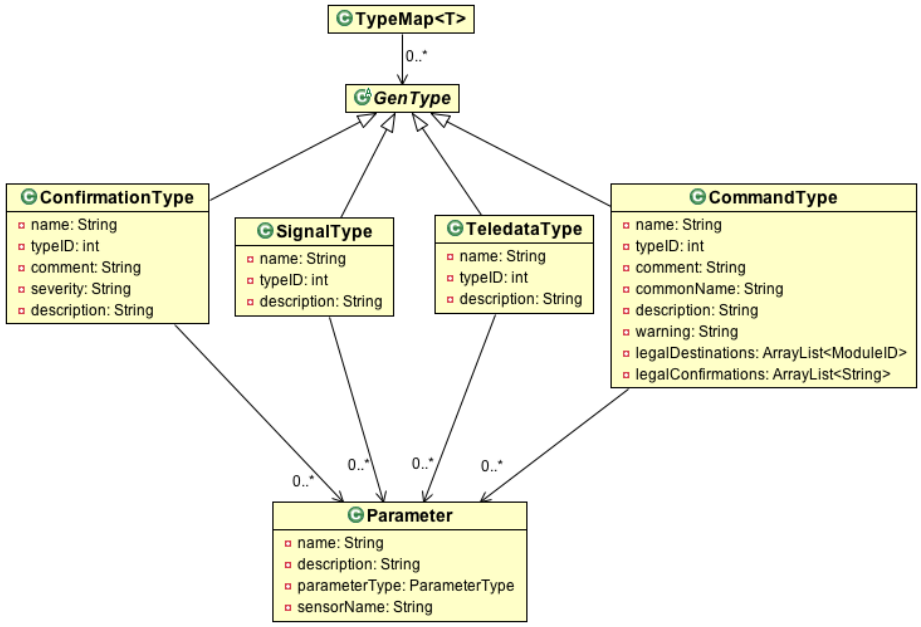
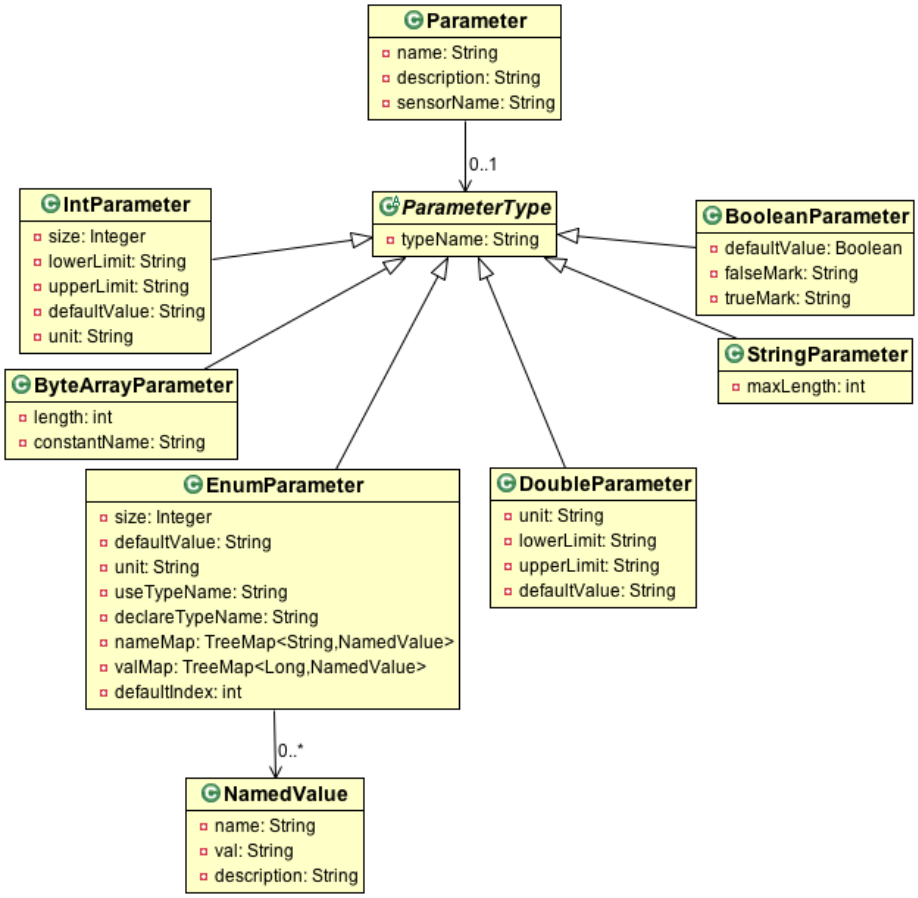


Figure 2.2: Simplified class diagram showing how a parameter and objects related to it is designed in the project.



2.2.2 The already existing editor

When the Types described in 2.2.1 was introduced, a very basic editor was created[12]. This editor was able to show a loaded type map and add types. There was no other functionality and no way to change already defined types.

2.3 Requirements specifications

The requirements specifications are based on the analysis and the already existing editor. Requirements are also discussed with Hans Henrik Løvengreen on what developers on the DTUsat-2 project would need and expect of an editor.

2.3.1 Convention

To create a better overview of the requirements, they are split into functional and non-functional requirements. The functional requirements defines the functionality of the editor. A functionality is described as a set of inputs, the behavior and outputs. Compared to the non-functional requirements being used to judge the operation of the editor[5][9].

> "In general, functional requirements define what a system is supposed to do whereas non-functional requirements define how a system is supposed to be.[9]"

2.3.1.1 The MOSCOW method

To create a better understanding of the importance of each requirement, the MOSCOW method is used. The method uses the keywords **must**, **should**, **could** and **won't**[1]. The keywords are defined as the following[8].

Must A requirement that must be satisfied in the final solution for the solution to be considered a success.

[1] The o's are added to make the word pronounceable.

Should A high-priority item that should be included in the solution if it is possible. This is often a critical requirement, but one which can be satisfied in other ways if strictly necessary.

Could A requirement which is considered desirable, but not necessary. This will be included if time and resources permit.

Won't A requirement that stakeholders have agreed will not be implemented in a given release, but may be considered for the future.

2.3.2 Functional requirements

1. The editor **must** be able to open, save, show and edit the four different type maps.
2. The editor **must** at all times create valid type files.
3. The editor **must** be able to add the right type to each type file.
4. The editor **must** be able to edit all fields of each type.
5. The editor **must** be able to add, remove, rearrange and edit parameters for all types.
6. The editor **must** validate the user input.
7. The editor **should** show a tree structure of the loaded types.
8. The editor **should** implement features for specifying a version number.
9. The editor **could** have different user settings for how to order the tree.
10. The editor **could** be able to create new types based on existing types.
11. The editor **could** suggest legal confirmations from a specified file.
12. The editor **could** validate the xml file to avoid nonsense.
13. The editor **should** be able to set and remember user preferences e.g. preferred size of application and preferred working directory.
14. The editor **won't** implement tabs to display more type files at once. Note that the application can run multiple times simultaneously.

2.3.3 Non-functional requirements

1. The editor **must** be platform independent. Java is chosen as language.
2. The editor **must** feature a Graphical User Interface.
3. The editor **must** be well documented (using JavaDoc) for later development.
4. The editor **must** be tested and working.
5. Some components from the editor **should** be reusable in other parts of the project. Components must be as general as possible, especially the tree.

Chapter 3

Design

This section describes the design of the editor. Not only the visual design but also the design of the software solution.

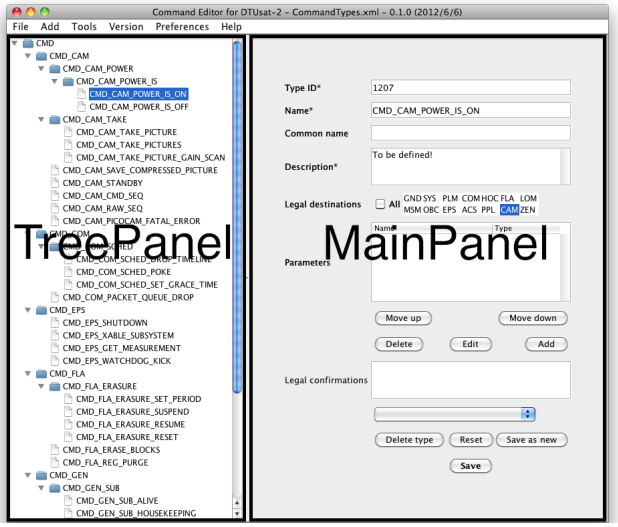
Different ways of showing and editing the definitions of types in the program was possible. A tree structure was chosen to display the types; the user is not expected to know the full names or ids of all types. The types can be sorted into categories (based on name prefix) for a better overview. When selecting a type in the tree, the type definition is copied and loaded into another panel – the `MainPanel`, to be edited. It is very important, that the type is copied into the `MainPanel` to avoid the user changing the type without knowing it.

From the `MainPanel`, if wanted by the user, a parameter can be selected, and then a dialog can be created so the different defined parameters can be edited. The same dialog can be used to create new parameters.

If the user want to, the type can then be saved by pressing the save-button. The original type in the model (located in the `Editor`) is then replaced by the type from the `MainPanel`.

To create a new type, a menu item can be pressed and the `MainPanel` is loaded with an empty type, when that type is saved it can be reached from the tree panel. As seen in figure 3.1 the main window of the editor is divided into two

Figure 3.1: Annotated screenshot of the main window of the editor showing the two panels.



panels, on left the tree (`TreePanel`) and on right the `MainPanel` used to edit the types.

Model-view-controller is a typical software design method. It has the benefits of separating the representation of information from the user's interaction with the information[7].

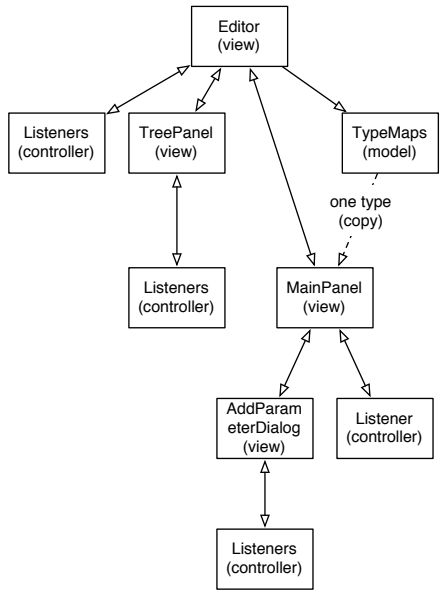
As seen in figure 3.2 everything is not as simple as the optimal model-view-controller idea. `Editor` is in charge of the model and holds two `listeners`. A `button-listener` `EditorListener` and a `window adapter` (`MyWindowAdapter`) listening and waiting for the user to exit.

The `TreePanel` gets the model from `Editor` and holds two `listeners`, `MyTreeSelectionListener` to listen for changes in the selection and `MyTreeModelListener` to listen for changes in the model.

The `MainPanel` has a `listener` for all the buttons (`MainPanelListener`) and can create the `AddParameterDialog` when requested by the user.

The `AddParameterDialog` has two `listeners` the general `AddParameterDialogListener` and a `key-listener` – `MyKeyAdapter`.

Figure 3.2: An overview of the design of the classes of the editor



3.1 Types

The design of the types; `CommandTypes`, `ConfirmationTypes`, `TeledataTypes` and `SignalTypes` are adapted as described in section 2.2.1. The `Editor` holds a `TypeMap` of each of the described types the `TypeMap` holds an `ArrayList` of the given type and `HashMaps` for quick access by id or name.

Chapter 4

Implementation

This section will be about the implementation of the different components and the general programming of the editor. The first part of this section is about the Graphical User Interface and the implementation of it. Later the panels and the `AddParameterDialog` will be described as well as other parts of the editor.

4.1 Graphical User Interface

As described in the non-functional requirements in 2.3.3 the editor must have a GUI. A series of screenshots of the editor is located in appendix B.

4.2 The Editor

The `Editor` object extends the `JPanel` object. It creates the `TreePanel` and the `MainPanel`. It puts both into scrollable panes (`JScrollPane`), which then is added to a split pane (`JSplitPane`) with a horizontal split.

The `Editor` object holds a large amount of menu items, when the object is created, the method `createAndShowGUI` is called which sets up all the menus.

The menu for the editor have seven items, each with subitems. For a complete list refer to appendix C. The seven items is file, add, tools, version, history, preferences and help. The File menu holds items such as new, open, save and save as. Under new, the type map the user wants to create must be chosen (between command, confirmation, teledata and signal.) When opening a file the user must also choose between the four possible type maps; the reason for this is that if the user opens a command type map with errors, it would also be a confirmation, teledata and signal type map with errors. To give the user proper error messages, the user must choose what kind of type map, he wants to open. The reading and writing of xml is covered in the section 4.2.1

Under the add item, the user can choose between adding a comment and/or warning to the type or adding a new type to the type map.

Under tools, the user can validate the type map. The validation is covered in section 4.2.2

Under version, the user can increase the major and/or minor version, the user can also change the version to exactly what he wants. The implementation of version numbering are covered in section 4.2.4

History refers to the version control capabilities of the program and is covered in chapter 6. The user can view both the type and file history.

Under preferences, the default size can be reset (to 800×800 with the divider at 250px from the left) or change the size and placement of the divider to the current shown. The default path (the starting path when selecting files) can both be selected or reset to the home directory. The TreePanel can be changed from sorting alphabetically to sorting by id. The categories for the tree can also be set by a comma separated `String`. Debug mode can be disabled or activated and a confirmations file, used in the MainPanel can be set. The preferences can all be cleared and the user can create a dialog showing the current preferences. The preferences and the implementation, choice of API is covered in section 4.2.6

A help menu with a placeholder for a help file and an about menu about the author of the program and how to request features or report bugs is also present.

4.2.1 Reading and writing xml

As described in section 2.1 the framework Simple is already being used in other components of the project and is therefore the chosen framework. In the beginning of the development of the editor version 1.7.2 of the framework was in use, that version is from the beginning of 2008, later the whole project updated to version 2.6.2 – with no consequences to the editor. The framework works by annotating all instances of data in the xml in a Java class. The classes can then be serialized with the framework and a TypeMap can then be created. An example of an annotated class is `CommandType` shown in listing 4.1 (cut of after the constructor – the full class can be seen on the CD provided in appendix A).

Listing 4.1: CommandType.java annotated to work with the simple xml framework.

```
1  public class CommandType extends GenType implements
       Serializable {
2
3    @Attribute
4    private String name;
5
6    @Attribute
7    private int typeID;
8
9    @Element(required = false)
10   private String comment;
11
12   @Element(required = false)
13   private String commonName;
14
15   @Element
16   private String description;
17
18   @Element(required = false)
19   private String warning;
20
21   @ElementList(required = false)
22   private ArrayList<ModuleID> legalDestinations;
23
24   @Element(required=false)
25   private Boolean packed;
26
27   @ElementList(required = false)
28   private ArrayList<Parameter> parameters;
```

```
29
30    @ElementList(required = false)
31    private ArrayList<String> legalConfirmations;
32
33    public CommandType() {
34
35    }
36    ...
```

An example of a xml-file showing `CommandTypes` is included in Listing 4.2 (cut of to only show two types. – the full file can be seen on the CD provided in appendix A) The types are actual types used in the project, the xml is generated by simple xml framework using the editor.

Listing 4.2: CommandTypes.xml – the xml file used in the project to define `CommandTypes`.

```
1  <DTUsat2CommandTypes>
2      <version>0.1.31 (2012/6/22)</version>
3      <commandType name="CMD_UNDEF" typeID="0">
4          <description>This is a pseudo command to be used in
               situations where no proper command has been
              defined</description>
5          <parameters/>
6      </commandType>
7      <commandType name="CMD_GEN_ALIVE" typeID="1">
8          <description>Request alive response to be sent to
               source module</description>
9          <legalDestinations/>
10     </commandType>
11     <commandType name="CMD_GEN_SEND_HOUSEKEEPING" typeID
           ="2">
12         <description>Request housekeeping data to be sent
               to GND</description>
13         <legalDestinations/>
14         <parameters>
15             <parameter name="hk_set">
16                 <description>Bitvector of desired set of
                       housekeeping data.
17                     Module may decide to send a superset of
                           these.
18                     Default value always results in full
                           housekeeping</description>
19                 <parameterType class="common.types.parameter.
                       IntParameter">
```

```
20                      <size>4</size>
21                      <lowerLimit>0</lowerLimit>
22                      <defaultValue>0xFFFFFFFF</defaultValue>
23                  </parameterType>
24              </parameter>
25          </parameters>
26      </commandType>
27      ...
28  </DTUsat2CommandTypes>
```

The command type `_UNDEF` only defines the required fields. Name and id is defined inline in the commandType tag (it is annotated as an attribute) while description, an element, has its own tag. Note, that parameters is an empty tag. This is an artifact from before when parameters was required. When the tag is empty, it is because it exists but has no info, in Java this means referring to a null pointer, this is the same in the `ElementList` of legal destinations, this means all destinations is legal (per definition) and is an important part of the design of the legal destinations.

The type `CMD_GEN_SEND_HOUSEKEEPING` holds more tags (additional non required tags) but also the `ElementList` parameters. While description is an `Element` and only holds text, an `ElementList` holds elements; `parameter` is in Java its own class with its own annotations, listing 4.3 shows how the `Parameter` object is defined and annotated (also cut of after the constructor – the full class can be seen on the CD provided in appendix A).

Listing 4.3: Parameter.java annotated to work with the simple xml framework.

```
1  public class Parameter implements Serializable, Cloneable
       {
2
3      @Attribute
4      private String name;
5
6      @Element
7      private String description;
8
9      @Element
10     private ParameterType parameterType;
11
12     @Element(required = false)
13     private String sensorName;
14
15     public Parameter() {
```

```
16
17      }
18      ...
```

Like `TypeMap`, `Parameter` has its own `attributes` and `elements` and can have `elementlists` to increase the depth even further.

All the annotations and classes were already created when the development with the editor was started, not much change was needed. The list of parameters was set to be required (it must be annotated if the element is not required, which it was not.) The satellite project wanted a xml-file that were as flat as possible, when no parameters was needed, there was no reason for the xml to need a parameter entry. Making parameters not required in all types (`CommandType`, `ConfirmationType`, `TeledataType` and `SignalType`), of course lead to some null pointers which was then handled.

Until the development of the editor began, the framework has mostly been used to read data in a very specific way. That resulted in garbled getter methods. For instance if `commonName` was not set, the name of the type was just returned instead. This was practical in other uses, but in an editor where the commonName is needed to be shown and set, it is not, the real values in the type is needed. Almost all annotated types, used when reading the xml, was having these "smart" getters. Often the types was also missing important setter classes, other components just read the xml, creating object, bypassing constructors and setters, and later just read the data. Setters and getters for each specific field was therefore created.

4.2.2 Validation of `TypeMaps`

A great and important part of this editor is validation. One of the main reasons for not using a normal xml-editor is because of the very specific conditions for the validation of the xml. Some validation is done, when the user tries to save the type, this is covered in their respective areas, validation in the `MainPanel` is covered in section 4.4.4 and validation in the `addParameterDialog` is covered in section 4.5.1. This section covers the editors validation of the `TypeMap`.

The validation of type maps is located in the `Editor` object the method is called `validateTypeMap`.

At the moment there is only two conditions to be met.

- The id of the type must match with which subsystem it belongs to.
- Command types belonging to a subsystem must at least have that subsystem defined as a legal destination (present in the list of ModuleID's).

For the first validation, the name and the id is needed. The name is defined with `kind_subsystem_name`. An example could be `CMD_CAM_TAKE_PICTURE`, `CMD` indicates it is a command, `CONF DATA SIG` indicates confirmation, teledata and signal respectively. `CAM` indicates that the command comes from the CAM subsystem and `TAKE_PICTURE` is the name of the command. Note, that spaces in the command name is the same separator as between kind and subsystem. To validate the types the type list is looped over (after being sorted by id) and for each type the subsystem and id is compared. There is no clear definition of what subsystems have types in what thousand. If the subsystems match, they must have id's in the same thousand. If they are not in the same subsystem, they must have id's in each their own thousand.

There is of course exceptions. The types with type id < 1000 belongs to the `GEN` subsystem (which does not exist – covered later in this section) these are general commands and some of these have no indication of subsystem. `CMD_UNDEF` is a good example. It belongs to no subsystem, because it is a type which should not be used. But the id indicates it as a general type and should belong to `GEN`, to solve this, if no subsystem is indicated it belongs to `GEN`. This leads to a problem with a fictional type `CMD_UNDEF_CMD` the name being `UNDEF_CMD` and no subsystem indicated, but the name separator and all the other separators are the same and would mistakenly be read as `UNDEF` being the subsystem. This results in types without subsystem implied cannot have spaces (written with _) in the name.

Sometimes subsystems are split. This is the case with `CMD_FLA_ERASURE` which holds commands such as `CMD_FLA_ERASURE_SET_PERIOD` and `CMD_FLA_ERASURE_SUSPEND`. It is not yet defined if these types must be in the same hundreds or how to detect what is split subsystems and what is just names with spaces – without hardcoding the split subsystems names.

The other validation is the validation of legal destinations; the array are checked up against the subsystem indicated by the type. If there is no match the xml is invalid. Again the subsystem `GEN` complicates this. There is no subsystem `GEN` because it is intended for general types. The LegalDestinations field is defined as if the list is null all destinations is legal, this means general commands must have the LegalDestinations field set to null. If the field is set to all (by selecting all), the validation should (and will) fail – this is intended. If later another subsystem is added the general commands should still have legal destinations set to all, this is ensured by setting the legal destinations to null.

Later the validation of other parts of the TypeMap might be wanted, these can easily be added to the method validateTypeMap in Editor. The list the user outputted to the user is an ArrayList of Strings. Errors found in the TypeMap can easily be added and outputted to the user. If the validation of TypeMaps should be more general a interface for validation the TypeMap could be preferred. It is a little unlikely, the editor is the only component actually changing the TypeMaps.

4.2.3 Sorting of TypeMaps

It is preferred that the types in the TypeMap is sorted after id. This functionality is needed when outputting the xml but also in the alternative sort (sort by id) in the TreePanel.

In the Editor object a method called sortTypeMap exists, it sorts the TypeMap using a Comparator the object TypeComparator (created for the purpose) can compare types based on type id and is used with the method sort in the Collections library to sort the TypeMap.

4.2.4 Version numbering

To denote major and minor changes in the xml file a version numbering was needed. The field was already created in the annotations for the xml file, but not used. Three levels of versions was agreed upon. A major version, a minor version and a version that was incremented for each save. Starting on version 0.0.0 each save would result in version 0.0.1 and 0.0.2 and so on, when the user feels minor changes have been made, he can increase the minor version from 0.0.2 to 0.1.0, if he then saves the internal version increases again to 0.1.1 and so on. When the user then feels major changes have been made he can increment to version 1.0.0. Buttons for incrementing major and minor versions are in the menu. If the user regret the incrementing of the version number or the version number is incorrect, the user can set the version number manually in the editor. The version number also holds the date in a year/month/day format. For the date the Calendar object is used. A bug (covered in chapter 5) was that the calendar object counts day starting from 1, but counts months starting from 0. This resulted in the months being off by one.

4.2.5 The title of the application

Along with the type file name the version number is in the title of the application e.g. "Command Editor for DTUsat-2 – CommandTypes.xml – 1.0.0 (2012/6/26)"

4.2.6 User preferences

Different preferences when using the editor can be set; most of the preferences is just for the convenience for the user. For the preferences to work as intended and make sense for the user, they must be saved when changed and loaded when the editor is started up. To save the references the Java Preferences API is used.[2] They are saved in the node /util/editor they can later be recalled from that node by other java applications. The nodes are saved as an xml file in a directory specified by java to save java preferences. The preferences can then be read, set or changed. The following preferences can be set, along with their default values.

Height when starting the program: `800`

Width when starting the program: `800`

Location of the divider when starting the program: `250`

Default folder when choosing files (file to save to, open etc.): `(empty)` meaning the users home directory.

Debug mode : `false`

TreePanel sort by id : `true`. The treepanel sorts by id, if `false` it sorts by name.

Confirmations-file Location of the confirmations file to be used when defining commands. More about this functionality in section 4.4.3

Categories for the tree panel which categories should be created to insert names into: `(empty)`, meaning the tree creates as many categories as possible. More about this in section 4.3

4.3 The tree panel

The `TreePanel` is the basis for the navigation in the editor. The idea is to present the types in the most effective way. Types belong to subsystems and can be closely related to other types based on their name. A tree was wanted to show the types. The structure is almost like a radix tree (space optimized trie) but only text between underscores must be merged. And each leaf must be a type and each type must be a leaf.

In the implementation of the `TreePanel`, the compressed nodes are called categories as they mimic a category the type is inserted into. The panel is a `JPanel` and each node is saved as a `DefaultMutableTreeNode` which can hold an object along with knowledge about its children, parents and more. In the `TreePanel`, that object is just a `String`. Figure 4.1 shows an example of the `TreePanel`; the nodes with blue folder icons are categories and the file icons are types. The icons are standard and can with little effort be changed.

These categories can be altered. A user might want to manually set specific categories, another user might not want to. The user can in the preferences menu set a comma separated list of categories, he want to have in the tree. If that comma separated list is empty (default) the method `collapsableNames` in `TreePanel` are used to determine all the possible names to collapse. The method `addNodes` are then used to collapse these categories into other categories e.g. `CMD_FLA_ERASURE` must then be inserted into `CMD_FLA` which must be inserted into `CMD` (the root node of command type maps). If a type is added, which does not start with `CMD`, it is added as the last child of the root – this should not happen while using the editor normally.

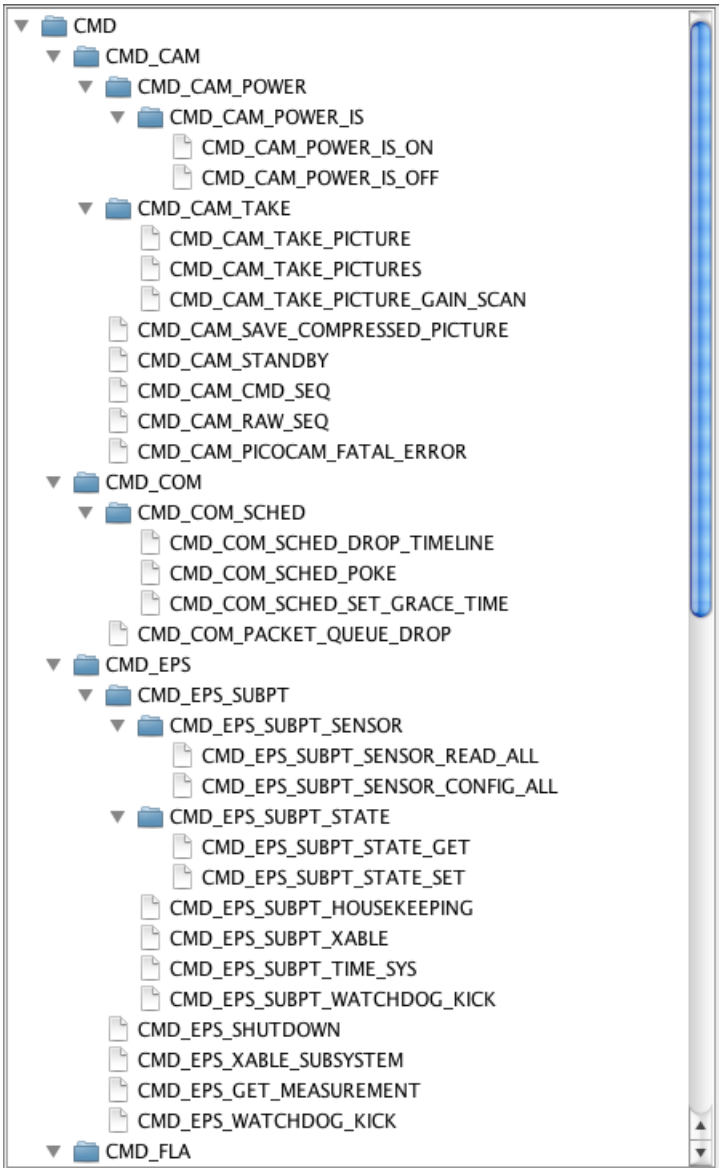
Then the types must be inserted into the tree. The same method as before (`addNodes`) is used. This leads to more than one problem. At first, it must be avoided that `CMD_CAM_TAKE_PICTURES` is inserted into `CMD_CAM_TAKE_PICTURE` this is done by implementing that categories must be split in between of underscores.

Another problem is that leaves can look like it is a category while it is not. E.g. `CMD_CAM_TAKE_PICTURE` is a command type, but `CMD_CAM_TAKE_PICTURE_GAIN_SCAN` is too. The second must not be inserted into the first to comply with what a user would expect.[1] The tree must know that.

To solve that problem there are two solutions, the first is, to before inserting the types into the collapsed category tree, to sort them alphabetic in reverse order.

[1] Note that the problem is handled correctly in figure 4.1

Figure 4.1: An example of the `TreePanel`.
Note that `CMD_CAM_TAKE_PICTURE_GAIN_SCAN` is *not* inserted into `CMD_CAM_TAKE_PICTURE` because `CMD_CAM_TAKE_PICTURE` is *not* a category but a type as well.



Then long names such as `CMD_CAM_TAKE_PICTURE_GAIN_SCAN` will be inserted before shorter names such as `CMD_CAM_TAKE_PICTURE`. The problem with that solution is that the sort of the type is wrong. This could be solved by later sorting the children of each node.

Instead it is chosen to keep the sort of the nodes (now also supporting the nodes being sorted by id because the tree does not interfere with the sort of the types, this means the tree could support all kinds of sorts) and instead remember which nodes should be a category. An extra object for each node to hold, could be created with info about whether or not the node was a category, or instead the selected solution; keep a list of names stating which nodes was a category and then only insert into nodes present in the array.

Other components might request the `TreePanel` to make a selection, i.e. if the `MainPanel` is to delete the selected type, a new type must be selected. The `MainPanel` uses the method `SetSelected()` to get the name of a type to show in the `MainPanel` and the `TreePanel` to select that to prevent the user to be confused. The `TreePanel` uses a depth-first-approach and selects the first leaf in the tree.

In the preferences the sort of the tree can be changed. The categories are sorted by name. Inside the categories sorting by id or name can be changed.

The `TreePanel` rely only on a type map and a specific `listener` (the `listener` must be made for the editor so it can change the type showed in the `MainPanel`, when requested by the user) so that the `TreePanel` in the future can be used in other software components in the project to give a good overview of loaded types.

4.4 The main panel

The `MainPanel` is the panel where the types is defined. The user selects a type in `TreePanel` this is then *copied* to the `MainPanel`. That the type is copied (and all objects inside the object's objects) is very important, if instead the type is passed to the MainPanel, when editing the parameters and saving (the parameter), the parameters are then directly changed in the original type, when the user only agreed to save the parameter, not the whole type. This is because the type object (`GenType`) holds a reference to the parameter. When the type is copied, it is important for the object inside the type is copied too. This is done by letting `Parameter implement Cloneable` and then clone it when the type is copied to the `MainPanel`.

The `MainPanel` holds fields for editing all the different data entries for the given type; id, name, description and parameters. If the type is a `ConfirmationType`, a combo box (a drop-down menu where only one string out of a selection of strings can be chosen). If the type is a `command type`; common name, legal destinations and legal confirmations can be defined. Additional fields warning and comment which is not used as often, can be shown using a menu item in the menu. If the type has no warning or comment defined, these fields are not shown.

The parameters is a list of parameters defined in a table, each parameter can then be edited, deleted, and moved up or down. Additional parameters can also be added. The parameters table is covered in section 4.4.1. Legal destinations must be chosen between the different subsystems and is covered in section 4.4.2. Legal confirmations is a text field for direct input or confirmations loaded from a file can be chosen in a combo box and then inserted into the text field, this is covered in section 4.4.3.

When the user is done defining the type it can be saved. Before saving into the model, the definition is validated, this is covered in section 4.4.4. The user can also define a new type based on an old type. A type can have the id and name changed and the button "save as new" can be pressed and the type is saved as a new type.

To show a new type in the `MainPanel` the method `showNode` is called, it is an overloaded class and can be called with a `String`, a `GenType` and without any parameters. With a `String` it searches in the model for the type with that name. With the type, it shows that type. Without a type it asks the `TreePanel` what type to show. When which type to show is known, the method `loadNode` is called to load the values from the type into the fields. At first all fields are emptied, the type might not use all the fields as the previous type – if showing a `command type` and then a `confirmation type`, nothing must remain.

If the user then saves the type the method `saveNode` is called, which first validates the fields using the method `validateFields` described more closely in section 4.4.4.

If the user tries to exit the program or show another type without saving the type currently shown, the user is asked to save the type or discard it. This is covered in section 4.4.5.

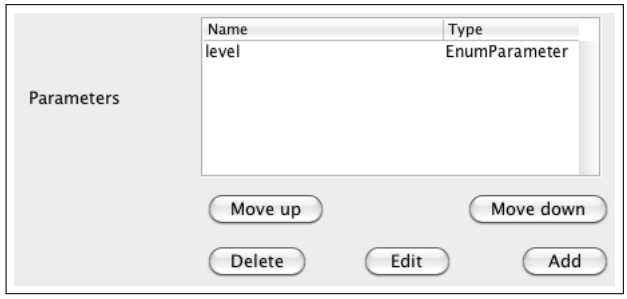
4.4.1 Parameters table

The parameters table follows the idea of a model-view-controller design. With `ParameterTableModel` as the model, a `JTable` as the view and uses the `MainPanelListener` as the controller.

The model holds a list of parameters, the `JTable` shows the name and the type of parameter in the `JTable`. Several buttons are associated with the table. It is important for the user to change the order of parameters; a move up and move down button is created for that. New parameters can be added and already defined parameters can be edited (the parameter are edited in the `AddParameterDialog` described in section 4.5) or deleted. There are buttons for that as well.

When moving or deleting the parameters, to not confuse the user it is wanted to the selection of the same parameter. This selection, must persist. If moving the parameter up, the selection must move up, if moving down the selection must move down. If deleting a parameter, the selection must be on the next parameter in the list (unless it was the last parameter, to keep selection of something, the parameter above must be selected) – this is to make the deletion of many parameters easier. Figure 4.2 shows the parameter table with a loaded parameter.

Figure 4.2: The parameters table in the `MainPanel`. Showing "level" an enumeration parameter.



When the table model is changed (something has been moved, deleted or added) the `ParameterTable` must be repainted. It proved to be important to also call the method `fireTableDataChanged` on the `ParameterTable` to update properly. If the method was *not* called, the scrollbars of the table would not be correct – The user could be able to scroll in a table with too few items (if parameters were deleted), and might not be able to scroll in a table with a large

number of recently added items.

4.4.2 Legal destinations

At the moment there is 14 modules, each having a `ModuleID` defined as an enum in the project, the modules is as the following list:

GND Ground Segment

SYS System services

PLM Platform Manager

COM Communication System

HOC Housekeeping Collector

FLA Flash Control

LOM Log manager

MSM Mass storage manager

OBC Onboard Computer System

EPS Electrical Power System

ACS Attitude Control System

PPL Primary Payload System

CAM Pico Camera System

ZEN Zensor board

These are loaded into a `JList`, when the type is loaded, the type's defined legal destinations is then set. The user can then change this, and when the user saves, the `JList` is converted to an `ArrayList` and saved in the type.

It is defined, that if the legal destinations `ArrayList` is set to null, all destinations are legal, because of that, a checkbox is added to make that choice.

If all modules is selected in the `JList`, that is saved in the type to prevent errors if more modules later is introduced. If the user wants to select all modules (and future modules) he must actively state that using the checkbox. When the

checkbox is checked (or was it when loading the type) the `JList`, where the user can select destinations, is disabled to prevent confusion or the user trying to use both.

The field with no destinations selected can be seen in figure 4.3, with some destinations selected can be seen in figure 4.4 and with the checkbox all selected and the `JList` disabled in figure 4.5.

Figure 4.3: The legal destinations in the `MainPanel`, with no destinations selected.

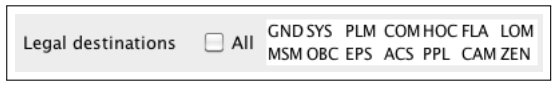


Figure 4.4: The legal destinations in the `MainPanel`, with some destinations selected.

Figure 4.5: The legal destinations in the `MainPanel`, with all destinations selected, and thereby the list disabled – nothing in the `JList` can be selected.

4.4.3 Legal confirmations

In the definition of a type, legal confirmations are a `String`, the field to enter the legal confirmations, the confirmations types that are accepted to be answered from the command type, could just be written in a field like the name. The user must be able to do that, but it could be easier if the user could select the confirmations on a list and would make more sense when the confirmations are already defined.

The input for the legal confirmations is then a text field accompanied with a combo box. The user can freely enter a comma separated list of legal confirmations, or the user can chose a confirmation type (loaded from a confirmation type map which location is defined in the preferences) which then is inserted into the field (and the selection in the combo box is then reset to the initial empty selection).

The power user can now easily just write the legal confirmations, the regular user can choose from the list. At the moment there is no validation on whether the added legal confirmations is unique. This could be validated when the user save by removing all duplicates of confirmations (by using a `TreeSet` which is already used to ensure uniqueness in other parts of the editor).

Figure 4.6: The legal confirmations in the `MainPanel`, the user can write legal confirmations from his own memory.

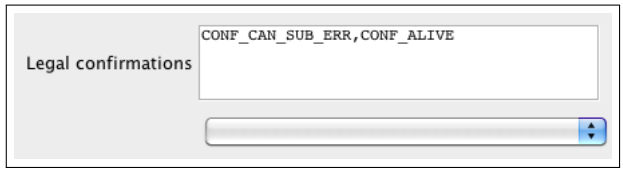
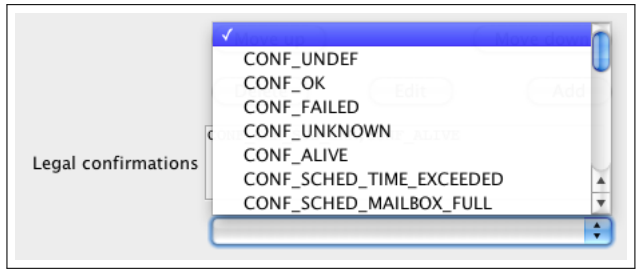


Figure 4.7: The legal confirmations in the `MainPanel`, the user can choose confirmations from a drop-down menu to be added.



4.4.4 Input validation in `MainPanel`

When the user tries to save the type in the `MainPanel` a number of validation checks are made. If the type does not pass the validation the user is prompted with an error message with all the errors from the type. This is greatly preferred over just returning the first error so all can be corrected before trying to save again. The following tests are made to make sure the type is valid.

- Required fields must be set, such as name, id and description.
- id must be a valid base ten number.
- Unique fields must be unique, such as name and id.

4.4.5 Ask to save feature

When the user changes to see another type via the tree or via loading another type map. Unsaved changes might be discarded. To avoid this a "ask to save" feature is implemented.

When the user tries to update the type in the `MainPanel` the user must be asked if he want to save his changes – but only if he had made changes. A method in `MainPanel isChanged` is implemented, if a field is edited, it returns `true` and the user is prompted whether to save or to discard the changes. Each field is compared with its respective data field in the type definition.

When comparing strings, the method `isEdited(String,JTextComponent)` is created to avoid comparing with null pointers, null pointers exists when fields are non required and not filled out. For comparing a parameter list with another parameter list the method `isParametersEdited(ArrayList<Parameter>, ArrayList<Parameter>)` is created, this method also triggers moved parameters.

4.5 The add parameter dialog

As seen in figure 4.8 the dialog have 5 different looks depending on what type of parameter is being defined. The five types is boolean, byte array, enumeration, integer and string. The five types shares three fields, name, description and the optional sensor name. In a combo box the type of parameter can be selected and the dialog changes.

If the parameter is a boolean, a true mark and a false mark can be set. These fields tell what `true` and `false` in the boolean means, an example could be, the often used definition where, true mark being "on" and false mark being "off". The true mark and false mark fields are added a `key listener` updating a combo box with the values. In that combo box, the user can select the default values from the true and false mark. The default being `true` and `false` respectively. This can be seen in figure 4.9

If the parameter is a byte array, only the length must be set using a textfield. If it is a string, only the max length must be set using a textfield. If the parameter is an integer; a unit, lower and upper limit, default value and size can be set. The lower and upper limit must fit into the size, and the default value must fit between the lower and upper limit. The lower and upper limit is optional and

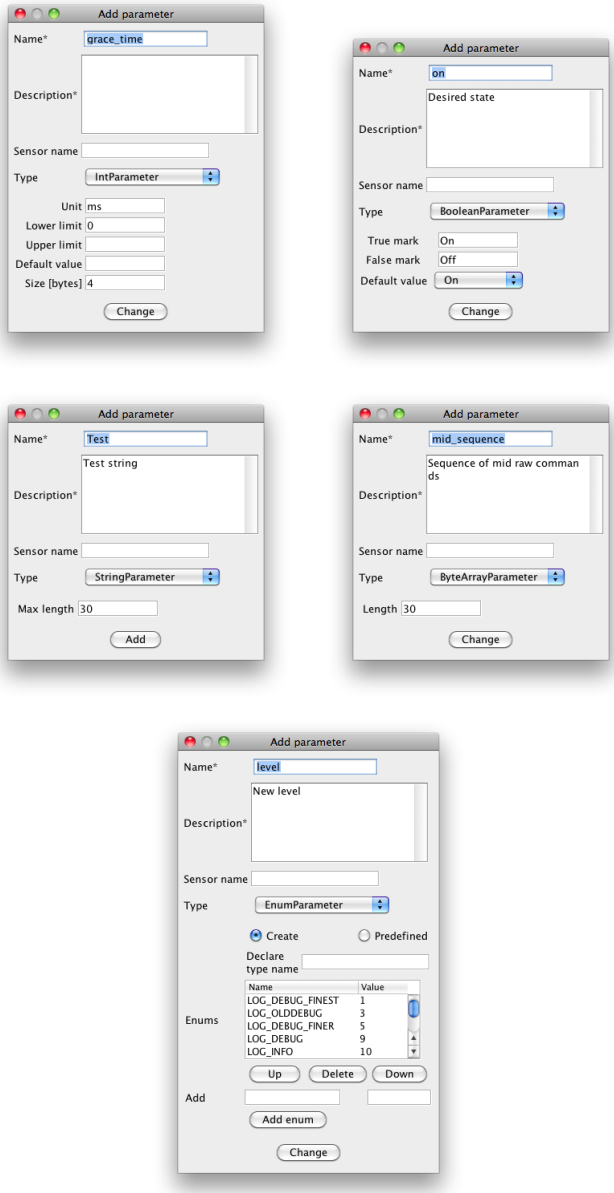
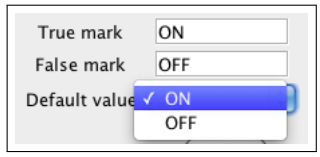


Figure 4.8: The different looks of `AddParameterDialog` depending of what kind of parameter type needs to be defined.

can be restricted by the size given. The size must be a power of two.

Figure 4.9: The selection of default value with changed true mark and false mark.



If the type is an enumeration type, a similar component as when manipulating parameters is used (described in section 4.4.1). The user can move enumerations up and down and delete them. Textfields for a name and a value is in the bottom of the list and when the user presses the "add enum" button the textfields content is moved into the model and shown in the table and the textfields are then cleared. Because of the simplicity of an enumeration – only a name and a value, the enumeration value cannot be changed, the enumeration must instead be deleted and recreated with the correct values, this is chosen to avoid too many buttons (leading to confusion) and because of the few fields an enumeration value is defined by. When defining an enumeration names must be unique, values can be redundant, a check using a `TreeSet` is implemented – the names are when loaded inserted into a `TreeSet` before inserting new names the `TreeSet` is searched to determine if the name already exist.

By request of an user of a prototype, the integer type is the default type when creating parameters. This is based on the user's experience of what parameter type is used most often.

4.5.1 Input validation in `AddParameterPanel`

Most of the fields are optional and therefore there is no validation. The name of the parameter must be unique and is made sure to be that, using a `TreeSet` located in `MainPanel`, the `TreeSet` object is a simple object it is used by inserting a string into it. If the string is not already present it returns success (`true`) and failure (`false`) otherwise. The structure is the same used to make sure the enumeration's names are all unique. Other than that only the description must be set.

The integer parameter has some more specific requirements, the model takes care of that. If an integer parameter is tried created with garbled info an `exception` is `thrown`. That `exception` is then caught and sent directly to the user to act upon via a `dialog`.

4.6 Changes in existing classes

The following changes to the existing classes was made some of them is already mentioned in section 4.2.1.

- `Parameter`s not longer required in `CommandType`, `ConfirmationType`, `TeledataType`, `SignalType`.
- The errors caused by `null pointer exceptions` from the above change.
- Proper `getter`s and `setter`s in a great amount of classes.
- A problem in `TypeMap` when the list of types was empty or not set.
- To properly copy a type to the `MainPanel` `Parameter` was changed to also implement `Cloneable`. Described in section 4.4.

Chapter 5

Testing

Testing of the editor is important. As described in the introduction in chapter 1 a bunch of different software components depend on the generated xml-files. An error in the xml-files can lead to fatal errors in all the components. Minor errors and bugs can be an annoyance to the user and can lead to an unusable editor and must be avoided.

It is important the editor both meets the functional and non-functional requirements specified in chapter 2.3 and the editor is working as expected.

5.1 Strategy

The editor is written in java, a good choice would then be to use JUnit to test the editor. But using JUnit testing on the editor would be ineffective and noncomprehensive because of the amount of GUI components working together. Instead the editor is tested systematically. The editor is also tested by a future user of the editor.

Under the development of the editor, parts of the code are tested as it is done. But systematic testing is then still needed to make sure the components work

together, and is generally a good idea to catch some parts which might not be tested.

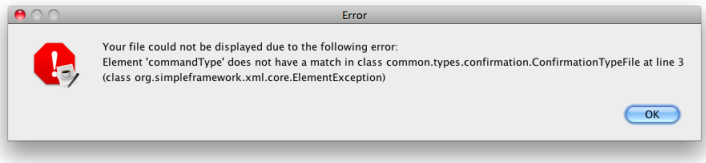
5.2 Systematic test

The following systematic tests is performed unless other is specified, the test gave the expected result.

1. Testing the `Editor`

   1.1 Opening and reading[1] a xml file with `CommandTypes`, `ConfirmationTypes`, `SignalTypes` and `TeledataTypes` with and without errors – When opening a file with errors an error dialog is shown, see figure 5.1

Figure 5.1: Example of an error message displayed when opening a `CommandType` map as a `ConfirmationType` map



   1.2 Saving a `CommandType`, `ConfirmationType`, `SignalType` and `TeledataType` map. The user is warned before saving something invalid and not able to save it. If exceptions are thrown from the simple xml framework they are directed to the user as in figure 5.1.

   1.3 The user can create all the 4 different typemaps. When creating a new empty typemap, a new typemap is created and then the same functionality as when using the menu item "add node" is used to add a new node.
   If the user saves the typemap without saving a node (resulting in an empty typemap) the simple xml framework *wrongly* choses to save the typemap with *no errors to the user*. This results in *invalid* xml, because a typemap is defined to at least hold one type. The reason why the framework can create xml and later not being able to read it, is unknown and beyond the development of this editor.
   This is not considered to be a problem a user will expect at any time

[1] See figure B.2, B.4 and B.5 on page 74

using the editor in a normal way.

When creating a new typemap the savefile is reset, so that when saving, the user is forced to choose a file to save to. (instead of saving into the original file – this was discovered and corrected under the testing)

1.4 Add node only adds the correct node to the typemap e.g. `CommandType` in `CommandType` maps, `SignalType` in `SignalType` maps and so on.

1.5 Validation

1.5.1 Validation of `CommandType`, `ConfirmationType` and `TeledataType` maps find errors when id and module do not match. Note: it will output an error, when modules and ids do not match, but if more modules and ids clash in a special way, it might not tell the user about all of them. When the first errors are corrected, it will find the rest, this means it will *not* say a type map is valid if it is invalid.

1.5.2 Validation of `SignalType` maps, because of the small size of the maps and that they follow another id-number-convention than the other, `SignalType` maps are not supported to be validated. An error message is, met if the user tries to validate a `SignalType` map.

1.5.3 Input validation in `MainPanel`. The required fields must be filled out correctly (id must be a number). One or more of the following errors can be shown: id must be a number, missing name and/or missing description.

1.5.4 Input validation in `AddParameterDialog`; name and description must be filled out, if not an error dialog is displayed.
Errors in the model i.e. when creating an integer with a lower limit higher than the upper limit, the exception is show in an error dialog to the user.
Errors in `ParameterType`s not covered by the model i.e. enums with same names are caught by the `AddParameterDialog` and shown in an error dialog to the user.

1.6 Version numbering

1.6.1 The major and minor version can be increased using the menu.

1.6.2 The lowest internal number is increased by each save.

1.6.3 The version number is saved correctly in the file.

1.6.4 The whole version number (all three numbers) can be set using the menu.
The numbers can then still be increased as usual, the lowest is still auto increased.

If the user inputs an invalid version number the user an error dialog is displayed.

1.6.5 The date is updated and set correctly. (This has not always been the case, read more in section 5.3)

1.6.6 If no version is set when loading a file (the version number was unused before the editor) a default number of `0.0.0` is chosen and set. The user is warned.

1.7 User preferences

1.7.1 The size (and divider placement) is set and remembered correctly. If no size is set the default size is used ($800 \times 800px$ and divider at $250px$ from left)

1.7.2 The default path (both set to the folder of the currently loaded file (set to current) and select) is set and remembered correctly. Only folders can be selected (no files because it is unknown what to do then). Only valid folders can be selected, if later the folder does not exist, it goes to default mode – the users home directory. If no folder is selected the default mode is also used.

1.7.3 Toggle `DEBUG` mode is boolean and when the menu is pressed it switches. The functionality of `DEBUG` mode is not yet agreed upon but some of the warning dialogs are not shown when active.

1.7.4 `TreePanel` preferences is covered in "Testing the `TreePanel`"

1.7.5 "Select confirmations-file" is covered in "Testing the `MainPanel`"

1.7.6 Clear preferences clears the preferences as expected.

1.7.7 Show preferences shows all the loaded preferences, not only some hardcoded values, it displays all the loaded preferences in the `PackageNode` – `user.util.editor` in a Dialog, see figure 5.2. The preferences is saved in a xml-file. Xml for a such xml file is outputted into the terminal.

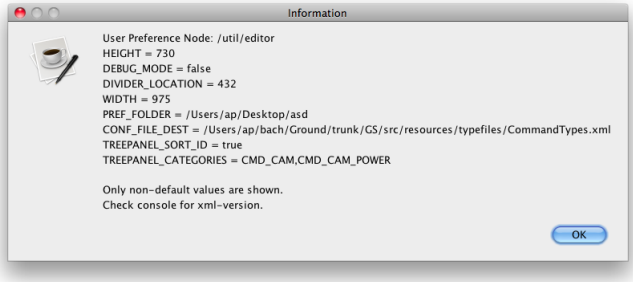
1.8 Testing the `TreePanel`

1.8.1 If two or more types shares a prefix (`CMD_CAM_TAKE_PICTURE` and `CMD_CAM_TAKE_PICTURES` shares `CMD`, `CMD_CAM`,`CMD_CAM_TAKE` but *not* `CMD_CAM_TAKE_PICTURE` or `CMD_CAM_TAKE_PICTUR` etc) a category is created if no categories are defined in the preferences.

1.8.2 Types are only inserted into other categories *not* other types.

1.8.3 Types and categories with same name might exist. If a category exists with the same name as a type, both are present in the tree. If selecting the category in the tree, the type will be shown in the `MainPanel`. It is unknown what a user might expect in this case, but if a node exists with the selected type or category, it is shown in the `MainPanel`.

Figure 5.2: Showing the non-default valued preferences along with which node the preferences is saved to.



1.8.4 If a category is selected and does not also exist as a type nothing in the `MainPanel` is changed. If a category is selected and also exists as a type (very rare, and not yet in any typefile) both the type and category can be selected to show the type.

1.8.5 Categories can be created and saved in preferences. The categories must be unique, no error is shown, but when creating the tree, duplicates are removed as the categories are sorted.
The root category (`CMD` for `CommandTypes`, `CONF` for `ConfirmationTypes`, `SIG` for `SignalTypess` and `DATA` for `TeledataTypes` must exist and is forced as a category.
Defined categories with no types inserted into are not removed, they exist (with a wrong icon – the same as a type (the icons are automatically set, if a category has no children – it has icon as a type)) to be used as a placeholder for not yet defined types.
The preferences for the tree are saved along with the preferences for the editor. When later the tree will be used in other software components in the tree, the preferences can be read from there or can be saved in another node.
When the categories for the tree is set the tree is repainted to show the new types.
The categories can be set to the empty string and causes the tree to create as many categories as possible (when prefixes is shared).
The categories is a comma separated list, if "," is inserted, two empty categories are created. This results in no categories created. This "bug" is kept as a feature if the user wants no categories.

1.8.6 How the tree is sorted can be selected in the preferences. The categories are always sorted alphabetically. The types inside the

categories can be changed to be sorted by id instead of name.

1.8.7 The categories in the tree are not specific to the type map. When defining categories for a `CommandType` map, the categories will also be shown in a `SignalType` map. They are wrongly put into the root node. This could be avoided by having categories defined to each type map or having, what could end being a very long list of categories for all files, the categories not making sense in current type map could then be removed based on its prefix.

1.8.8 If the tree is too big for the panel, a scrollbar is present so the user is able to see all types in the tree.

1.9 Testing the `MainPanel`

1.9.1 The correct fields are updated with the correct data. Unused fields are emptied to not mess with later loaded types (i.e. loading a `SignalType` after a `CommandType` many fields are not used, these should be cleared if a `CommandType` later is loaded – this could be a potentional problem because if non required fields are empty and the text fields then not are updated to reflect that.)

1.9.2 The fields are saved correctly into the type which then is substituted into the typemap. If "save as new" is pressed, the type is inserted as a new type.

1.9.3 The reset button works (almost) as expected. If creating a new type and then pressing reset the id is reset to 0 (all other fields are reset correctly). This is because the new type loaded into the `MainPanel` needs a id. This id is not shown in the field on load, but is still present in the type. This could be fixed by creating an empty constructor for the types.

1.9.4 The delete button works as expected. When a type is deleted the `TreePanel` selects the first type, which is then shown in the main panel. The `TreePanel` updates to reflect the changes in the type map.

1.9.5 The test for uniqueness with name and id is tested to be correct.

1.9.6 When name of types is changed the TreePanel is updated to correctly reflect that.

1.9.7 When the fields comment or warning are empty they are not shown. If they are nonempty they are shown. If the user requests the fields to be shown using the menu they are shown correctly. If the user request the fields to be shown, when it does not make sense in that type (only `CommandType` uses the comment and warning fields) an error message is met. An error message is met too, if no types are loaded (e.g. no type loaded in `MainPanel` or no type map loaded in the `Editor`).

1.9.8 Parameters can be moved up and down, deleted and added. When the parameter is moved up or down, the selection in the table keeps on the selected. When deleting a parameter, the next parameter is selected (so if the user wants to delete 5 parameters, he can select one and just press delete 5 times, instead of selecting, and deleting each parameter)
When adding and removing parameters, where the scrollbars should change (removing the 6th parameter when the view only can show 5 should remove the scrollbars, or adding the 6th parameter in the same case should show the scrollbar) the scrollbars are shown correctly. The scrollbars also show correctly when the types are loaded.

1.9.9 Legal destinations are loaded correctly. If the list is `null` the checkbox is checked and the legal destinations box grayed out (disabled). When selecting or deselecting the checkbox the destinations box is disabled and enabled as expected.

1.9.10 The Legal confirmations file can be loaded in the preferences menu. If no file is loaded, the combo box shows "Select a confirmations-file in preferences", the textbox for legal confirmations can still be used. A file can be selected (starting the selection in the correct folder). If the file is invalid (according to simple xml) the error returned is shown to the user.
When the file is loaded correct, the confirmations can be selected in the combo box. When a item is selected in the combo box, it clears and the confirmation name is inserted into the textbox, the user can still use the textbox manually.
If the file is deleted, with the editor running, nothing happens, because the file is only loaded on startup. If the user wants to reload the file without opening the editor again the same file can be selected in the preferences, when a file is selected the combo box is updated.
If the defined file in the preferences no longer exist, an error is shown and the combobox shows "'..."' It could be argued, that it should show the same text as when no file is loaded, but the user were already warned.

1.9.11 The "Ask to save" feature is tested to work properly. The warning is shown when the user changes the type in the `MainPanel`, a warning is shown whether or not, the user wants to save the type. If the program is closed with unsaved changes in the type, the same warning is shown.
If he choose save and the type fails to be saved, the change of type is dismissed and the user can correct his mistakes and then save.

If it succeed, the node is saved and the new one is shown.
If the user chooses discard, the type selected is shown.
If the user exits the program in a special way (i.e. "cmd + Q" on mac, the equivalent of "alt + F4" on pc, or other forced shutdowns) the program does not know it is getting killed and the message might not be shown.

1.9.12 The `MainPanel` might be to big to fit in the frame (if the editor is only $800px$ high and all fields (with comment and warning) is shown), a scrollbar is then provided.

All requirements stated in section 2.3 are implemented, a few with minor problems as stated above. The editor has been tested on Mac OS X and linux.

5.3 User test

An actual user of the editor has tested an early, but functional version of the editor. The editor did work as expected, but he had the following suggestions to improve the editor. The suggestions are translated from danish and shortened.

1. When saving the file, it should be checked if anything was changed. If not, the version number should not be incremented.
2. When closing the program without saving the file, the user should be asked to save or discard it.
3. It is very annoying that when saving, all categories in the tree are unfolded.
4. On machines using GTK2 the all-checkbox for legal destinations is placed on the subsystem list.
5. When selecting folders in the tree, system errors are printed in the console.
6. Type ID is only a number, the textbox should only accept numbers directly (instead of giving an error when trying to save).
7. The editor would be easier to use, if the buttons with no effect is grayed out (i.e. save and reset)
8. When adding a parameter to a type, the new window opens in (0,0) on the screen, this is bad when using dual monitors.
9. Multi-line fields wraps text on character instead of words.

10. It would be nice, if the parameter list was wider so all values could be shown (instead of just name and type)

Below is my thoughts to his suggestions.

Suggestion **1** and **2** relies almost on the same functionality, and might later be implemented, but not present in the editor as of now. It is important to note the difference in the suggestion and the "Ask to save" feature. The implemented features only asks when a change to an unsaved node in the `MainPanel` is about to be lost.

Suggestion **2** and **3**, is a problem because the user saves the file too often. Suggestion 2 might remove some saves causing suggestion 3. The tree is built to update the whole tree on changes (saving could be changed not to update the tree as it should not change the typemap). A change in name on one type also reloads the whole tree. This is needed to determine if new categories must be created, old must be deleted. The tree could implement functionality to save which categories are unfolded, and when creating the tree only unfold these. This is not implemented as of now.

Suggestion **4**, in agreement with my supervisor it is chosen not to support machines using GTK2 (GTK3 is available as of start of 2011) The reason for the bug is in the layout manager and out of the scope of this editor.

Suggestion **5**, the editor was a prototype, the debug-information is no longer outputted.

Suggestion **6** is a nice-to-have feature. Not implemented as of now. This is the only field accepting only numbers, the users of the editor is expected to know only to input numbers (and if not, they are told that by a dialog when saving).

The functionality of suggestion **7** would need the editor to listen for changes on all textfields. This could be implemented, but is not as of now. The reset field should very rarely be grayed out – only when creating a new type and nothing had been written yet.

Suggestion **8** is implemented. New windows are opened in relation to (on top of) where the editor is on the screen.

The word wrap in suggestion **9** is implemented, the font size is also changed to a smaller to support more text in little space.

The functionality in **10** relies on the user having a wide screen (more than $1280px$) to be shown properly, not all users might have this. (A preference

stating the possession of a “large screen” could be implemented, but is not) It is also unclear how to present the data of the different parameter types properly on one line. This is not implemented, the user still has to press edit on the parameter to show more info than the name and type of parameter.

Most of the suggestions are visual or other nice-to-have features, this shows that the editor is highly usable, but not perfect yet.

Chapter 6

The advanced part

6.1 Analysis of requirements

As stated earlier the aim for the editor is to be a tool to support the development of the satellites definitions of types. Work including software components (code, API etc.), type definitions and more are all shared among the group of developers. The work is shared using a version control server and each developer then retrieves the common work from that and it is expected that each developer is having a working svn directory on their computer. In a large group of developers using a version control server to share the type definition files (type files) it is then a obvious choice for an editor to support that. A developer might want to revert a version of a `TypeFile`, or just parts of it. As of today the revision server holds 2329 revisions. A user might want to know in what revisions the file was changed and compare each version. svn can do a naive comparison of two files using the diff function. If a user wants to know what changes was made to the type `CMD_UNDEF` between version 2329 and 2326, he will write something similar to this in a terminal:

```
svn diff -r 2329:2326 CommandTypes.xml
```

resulting in the following output (cut of to a few lines):

```
1  --- GS/src/resources/typefiles/CommandTypes.xml (revision
       2329)
2  +++ GS/src/resources/typefiles/CommandTypes.xml (revision
       2326)
3  @@ -1,5 +1,5 @@
4   <DTUsat2CommandTypes>
5  -    <version>0.1.21 (2012/5/21)</version>
6  +    <version>0.1.10 (2012/5/20)</version>
7       <commandType name="CMD_UNDEF" typeID="0">
8           <description>This is a pseudo command to be used
                in situations where no proper command has been
                defined</description>
9           <parameters/>
10 @@ -79,7 +79,7 @@
11          <legalConfirmations/>
12      </commandType>
13      <commandType name="CMD_GEN_SUBPT_PING" typeID="30">
14 -        <description>Subsystem passthrough. Sends a value
       to the subsystem, which it should echo back.</
       description>
15 +        <description>Sends a value to the subsystem, which
        it echoes back.</description>
16          <legalDestinations>
17              <moduleID>EPS</moduleID>
18              <moduleID>ACS</moduleID>
19 ...
```

It is easy to se that the version number is changed from `0.1.10` to `0.1.21` but what happened to `CMD_UNDEF` is not, it looks like it was created in revision 2329, but it was not, it was not edited at all. Maybe it was just moved to another line number in the file (the editor sorts the types after id) maybe an empty line or a new type was inserted before. It all leads to the conclusion — a higher level of comparison of type specific changes is needed.

6.1.1 Requirements

The requirements for the advanced part are not very strict and the solutions for them are very open. The requirements for this part use the same MOSCOW defintions as the basic part, these can be seen in section 2.3.1.1. The whole version control part is a **should**; it is not a requirement needed for the editor to be a success (a **must**), it is more a nice-to-have feature. But for the version

control part to be a success, the following needs to be met.

1. The editor **must** show the last revision and if possible version, the type was edited.
2. The editor **must** show what revision and if possible version, the type was created in.
3. The editor **must** show a list of changes the type has experienced.
4. The editor **must** function normally (without version control features) on files not under version control.
5. The editor **should** show information about who to blame for changes based on revision number.
6. The editor **should** show a short history of the `TypeFile`.
7. The editor **should** only fetch the needed revisions and not just fetch all revisions.
8. The editor **could** have the feature of reverting a type to a previous version.

Because the advanced part is the same project – the editor, as the basic part, the non-functional requirements are the same. These can be found in section 2.3.3 on page 13.

6.2 Design

The visual design of the version control features of the programs are pretty straight forward. The user must be able to access three different types of information; the last revision the type was edited and what revision it was created in, the list of changes to the type and an overview of commits of the file. In the main panel the information about when the type last was changed and the introduction of the type is shown above the fields for editing. The change history of the type and the meta data (the history) of the `TypeFile` can be shown in dialogs.

From a model-view-controller point of view, the view is `MainPanel` and the new object `VersionControlDialog`. The controller for `MainPanel` is the usual `MainPanelListener` and for `VersionControlDialog` it is the `VersionControl DialogListener`. The model for the revision history is the object `RevisionTypeMap`.

The RevisionTypeMap provides all the information the view needs. All version control is made through a svn server. An API is used to retrieve the information through the svn server.

6.2.1 Lists of RevisionTypeMap

The RevisionTypeMap holds a list of the revision numbers and two groups of parallel lists.

- revs – A list of the revisions the file was changed in. (according to svn)
- Parallel lists with information about the valid TypeMaps
  - typeMaps – A list of the valid TypeMaps
  - versions – A list of versions matching the valid TypeMaps. The version is the version number stated in the xml-file i.e. "0.1.0 2012/06/22"
  - revisions – A list of revisions matching the valid TypeMaps
  - metaData – A list of meta data (author, date and message from svn) to the revision.
- Parallel lists with information about the invalid TypeMaps
  - invalidRevisions – A list of revisions to the failed TypeMaps (the TypeMaps that could not be loaded by the xml framework)
  - revisionExceptions – A list of strings on why the revision failed from the xml framework.
  - failedMetaData – A list of meta data from the failed revisions.

The RevisionTypeMap also keeps track of how many revisions it has fetched, so it does not need to fetch all revisions to say something about the first.

The RevisionTypeMap is able to only fetch the relevant revisions. If a type is created in the last revision there is no need for fetching and loading all the different revisions of the file.

6.2.2 Methods of RevisionTypeMap

The RevisionTypeMap also holds a number of methods, the most relevant methods are described below.

- The `constructor` takes a client object from the API, a path and a boolean about the cache version (if `false`, all the revisions are loaded. If `true`, none are loaded – they are loaded when needed). When the constructor is called the method `updateLog` is called.
- The method `updateLog` uses the API to connect to the svn server and requests in what revisions the file was changed in according to svn.
- The method `fetch` uses the API to fetch one revision of the file and loads the TypeMap.
- The method `fetchAll` uses the method `fetch` to fetch the rest of the revisions of the file.
- The method `getAllVersions` takes a type as parameter and then fetches all versions. It stops when it reaches a `TypeMap` where the type is not present.
- The method `whenIntroduced` takes a type, fetches all versions it is presented in (using `getAllVersions`) and returns the lowest (first) revision it is presented in.
- The method `whenLastChange` also uses the method `getAllVersions`[1] and compares the type to the older version until it reaches a version, where the type has changed. The method uses the method `unChanged(Gentype, GenType)` to test for changes.
- The method `unChanged(Gentype, GenType)` tests for changes between to types. All fields are tested for changes. The method `unChanged(String, String)` is used for looking for changes between strings, a method is used to avoid problems when comparing with `null pointers`.
- The method `changes` takes a `GenType` and returns a `String` (formatted with HTML) with the changes between the different versions.
- Getter and setter methods for the earlier described lists and easier access to the list of total revisions which is hidden in a callback object[2].

6.2.3 Reuse in `RevisionTypeMap`

The code is designed to reuse as many methods as possible. This is important to give consistent results. Fewer methods implies fewer bugs, few crucial parts

[1] This might easily fetch too many versions. Why this is done anyway is described in section, section 6.2.3

[2] Callback objects are covered in the section 6.3.

of the code is very error prone, these must be kept at a minimum – this is done by reuse.

`whenLastChange` and `whenIntroduced` uses the same method `getAllVersions` to determine the introduction and the last change. It is often not necessary to fetch all versions to find out, when the last change was made (the last change can easily be many revisions later than the introduction.) But the two functions are used together, when `whenLastChange` is called `whenIntroduced` has already been called, and all revisions are already fetched and loaded.

The usage of `getAllVersions` in `whenLastChange` is solely to, if ever, the editor changes to not use the two methods together (or is used in wrong order), `whenLastChange` would still return the correct result. If wanted, the method `getAllVersions` can then be changed to also implement a stop when meeting a type that is changed (when called by `whenLastChange` specified by a `boolean`) using the method `unchanged(GenType, GenType)` to compare the two types.

6.2.4 The `versionControlDialog`

The `versionControlDialog` is used to display info about the changes to the type or the history of the file. The dialog is created with a `boolean` to specify what to show.

If it is to show type history, it gets the currently loaded type from `MainPanel` and uses the method `changes` to retrieve the changes of the type. If it is to show the file history, it fetches the rest of the files (to see which files fails to be serialized, and to get the version numbers of the files). Then it represents the data retrieved from the `RevisionTypeMap` using HTML.

6.3 Implementation

As API the svn API JavaHL is used. JavaHL is well-known (also used in eclipse's (a commonly used IDE) version control) and mostly because of its simplicity. JavaHL is a simple JNI wrapper[3], wrapping around the native svn client. Because of that, or other reasons, the JavaDoc supplied with the API is not very

[3] "The Java Native Interface (JNI) is a programming framework that enables Java code running in a Java Virtual Machine (JVM) to call, and to be called by, native applications (programs specific to a hardware and operating system platform) and libraries written in other languages such as C, C++ and assembly[6]."

comprehensive. It might be expected for the user to know the native functions and thereby understand the API methods. This made the developing process of the advanced part of the editor more difficult.

At first, JavaHL was used to checkout the svn repository, with the following function. (URL is the url to the svn server, client is a object with the interface `svnClientInterface`)

```
client.checkout(URL, "", Revision.HEAD, Revision.HEAD,
    Depth.infinity, false, true);
```

The JavaDoc for the method only documents the names for the variables, not what they are.

```
checkout(String moduleName, String destPath, Revision
    revision, Revision pegRevision, int depth, boolean
    ignoreExternals, boolean allowUnverObstructions)
    Executes a revision checkout.
```

This check out the given url (destination to the svn server) between two revisions to a given depth. The booleans are unclear in the JavaDoc what is, but svn documentation specifies that. None of it is accepted by the svn server and is not used.

At first the checkout was made before fetching anything from the svn server, but the functionality was later abandoned because the editor then might touch with data it should not without asking, and the user of the editor is able to checkout the repository on his own.

To retrieve the log of the file (the revisions the file is edited in) the method `logMessages(String path, Revision pegRevision, RevisionRange[] ranges, boolean stopOnCopy, boolean discoverPath, boolean includeMergedRevisions, String[] revProps, long limit, LogMessageCallback callback)` is used. The method returns nothing, instead a callback object[4] is given as a parameter. The callback object `MyLogMessage Callback` (implementing `LogMessageCallback` from the svn API) is a simple object, when the constructor is called (multiple times), the data is saved in an `ArrayList` and can later be retrieved.

[4]"In object-oriented programming languages without function-valued arguments, such as Java, callbacks can be simulated by passing an instance of an abstract class or interface, of which the receiver will call one or more methods, while the calling end provides a concrete implementation. Such objects are effectively a bundle of callbacks, plus the data they need to manipulate. They are useful in implementing various design patterns such as Visitor, Observer, and Strategy."[3]

The object `MyLogMessageCallback` now holds the revision number of the revisions the file was changed in. It is unknown where in the file a given type is or whether the type exists at all. To make the high level comparison of tracing creation and changes in types the whole file must be loaded each time! To fetch the file in a given revision JavaHL is used. The method `fileContent` is used, given the path and what revision is wanted it returns the file. The file is loaded using simple xml framework as usual, if the serialization using simple successes the `TypeMap`, revision number and meta data is saved. If it fails, the revision number, the exception from the serialization (what went wrong) and meta data (who to blame) is saved.

When fetching a revision the number of fetched revisions is saved in an integer called `iter` in `RevisionTypeMap` so it knows when to stop, and what is already fetched.

6.4 Testing

When the first working prototype of the editor was made, the xml files as definition on types was agreed to be official. The version history of the files reflects that, not many revisions of the files exists and often the definitions are defined, and not changed later. As of now only 15 revisions of CommandTypes.xml exist. To compare the files they must be serialized using the same types, because of changes in the definition of how the files is annotated, the first 5 revisions of CommandTypes.xml cannot be loaded. At the moment only `CommandType` maps are supported.

A screenshot of the history of CommandTypes.xml is shown in figure 6.1. In figure 6.2 the top of the `MainPanel` is shows the information from one type, `CMD_GEN_SUBPT_SENSOR_READ_PIN`, from version control, on when it was introduced and last changed.

In figure 6.3 the changes to the type is shown.

To test the version control's functionality, a test svn server has been set up. It holds a few test types which are changed in different revisions. The table 6.1 and table 6.2 explains the changes to the types. The xml files can be seen in appendix E

The table 6.2 shows the values of the types. The values are shown comma separated id,name,description.

Figure 6.1: The history of CommandTypes.xml shows the revisions along with meta data about the revisions. For failed revisions it shows the exception from the xml framework. Note that the off-by-one-month-bug is showing from revision 2326 to 2332, the version numbers date is off by one month in comparison to the version control server. This bug is now fixed, covered in section 4.2.4

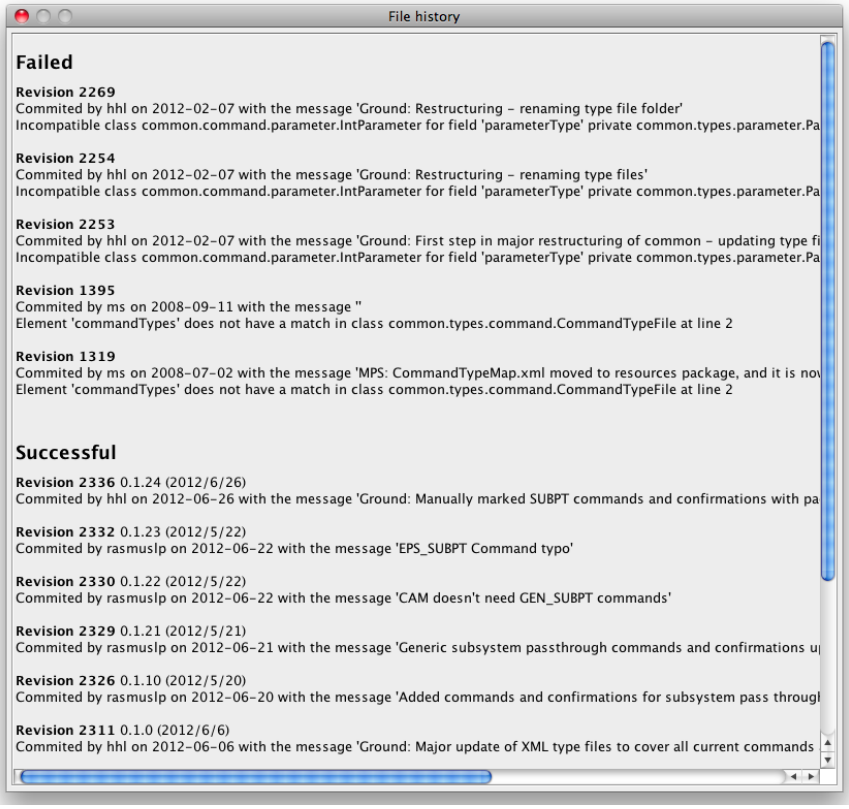


Figure 6.2: Information from version control server is shown if the file is under version control. Author and when (both revision and version, if available) the type was introduced and last edited is shown.

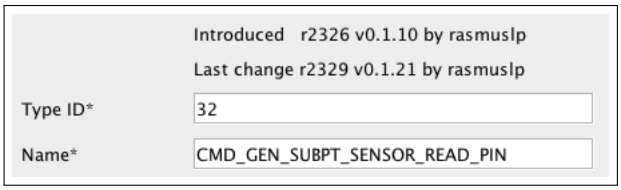


Figure 6.3: Showing history with changes for a type through the different versions.

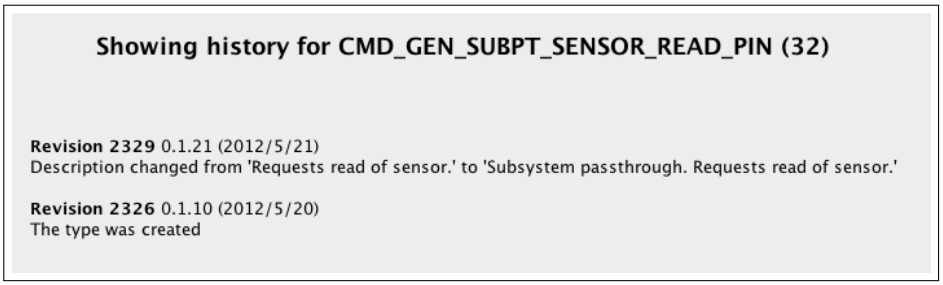


Table 6.1: List of the changes to the types A, B, C and D in the 5 different revisions.

Revision 1 Create A and B

Revision 2 Edit A, create C and D

Revision 3 Edit B and C, delete D

Revision 4 A map with errors

Revision 5 create D

The figure 6.4 shows the values in the `MainPanel` for the 4 types. The revisions fit with the values from table 6.1 and table 6.2. Note that D is introduced in revision 5. It actually was introduced in revision 2, but deleted in revision 3, when a type is reintroduced, the type is known as a new type because it could just randomly share name and id with a former type it has no relations to.

In figure 6.5 the history of each type is shown. As of now, only name and description is looked after for changes. The changes for each type is correct compared to table 6.1 and table 6.2.

The figure 6.6 shows the history of the test file. The failed revision in between working revisions is identified, but all other versions are still loaded. The author date and other meta data are shown correctly.

All requirements stated in section 6.1.1 is implemented, except requirement 8 which is not yet fully implemented. No real issues are found, but some of the points could be extended i.e. following types with changed name/id and only `CommandType` maps are supported could be extended to support them all (even though it is the most used and changed map).

6.4.1 Portability

Because of the use of JavaHL some portability is lost. The use of a JNI wrapper makes it difficult to create a runnable jar file that can run on different operating systems. As described earlier, a JNI wrapper just wraps around native code, a special file for each operating system is then needed, and a jar file for each possible operating system can then be created. Instead a framework such as SVNkit could have been used, but users of the editor is expected to find the correct JavaHL (the user would already have this, if running version control

Table 6.2: The different values of the types A,B,C and D in the 5 different revisions. The format is id,name,description.

	A	B	C	D
r1	0,A,test	1,B,test		
r2	0,A,testing		2,C,test	3,D,test
r3		1,B,testa	2,C,testb	*deleted*
r4	Typemap with errors			
r5				3,D,test

Figure 6.4: Revision introduced and last change of the 4 types A,B,C and D.

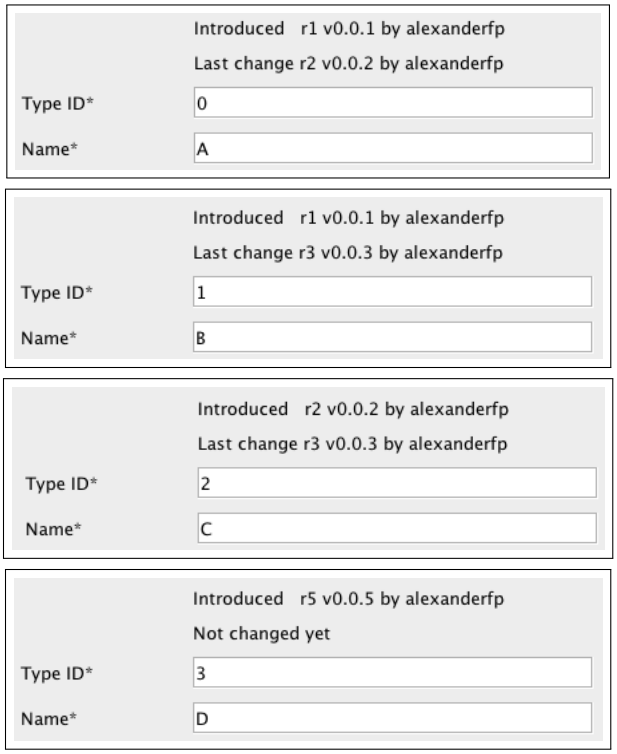


Figure 6.5: Type history for the four types.

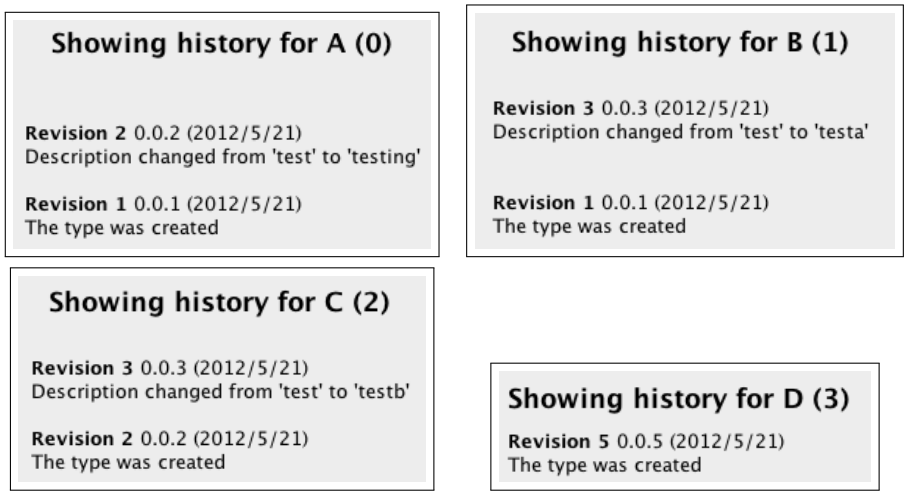


Figure 6.6: Showing history with changes for a type through the different versions.

via eclipse, which is a common IDE) for his operating system and compile the editor.

6.4.2 Resources

Java runs cross platform and are running in a virtual machine, this have some performance problems. Adding to that, when the version control functionality is in use, due to design – because of the high level comparison of types needed, all revisions of the file working on, is loaded into the editor. This can lead to the editor to hog a high amount of memory.

The problem is a very common tradeoff in software development; which is most important, speed or memory usage. When opening the editor, the user must wait at the moment up to two seconds for the editor to connect to the svn server and retrieve the information on the first type loaded into the `MainPanel` (remember that the editor only fetches the relevant versions when checking for last change or introduction).

The memory usage problem could be solved by fetching the versions each time they are needed, this would mean a few seconds wait time each time a new type should be shown.

At the moment, on the machine used for testing, the editor uses 275MB RAM while running with no typemap loaded. If a `CommandType` map with a size of 29KB (0.02832MB) is loaded the memory usage increases with 51MB to 326MB. That is a factor 1800!

When opening the editor with the real `CommandType` map, the editor uses 400MB RAM, but that is only because the older files are smaller; the files would grow as more types are defined.

The reason for that massive increase in ram usage, is the usage of all the objects. A pointer i 64bit java uses 8 bytes. With all the objects pointing to other objects, when serialized the memory usage of a xml-file loaded must be expected to increase greatly.

The solution to solve some of the performance issues would be to:

- When opening a file, the editor will retrieve *when* all types were introduced and when they last were changed. This is saved in a simple data structure able to hold pairs with the key being the name of the type and the value

being the revision, or even better, triplets holding a name, a revision for introduction and a revision for last change.

- When searching for changes, only the two revisions the editor is comparing should be kept in memory at the same time. (When searching for when the type is introduced, only one revision is needed, but it could be done parallel with searching for changes)
- When the revisions are retrieved, the meta data are saved in separate lists, this does not use very much memory and could be kept that way.
- When the user want to see the history of a type, he must be patient. Then all revisions with the type present must be loaded (but only with two type maps in memory at any time)

If implemented, now the user must only wait when opening the file and when the user wants to see the history of a node. A loading screen/dialog could be shown when the user have been waiting for more than 2 seconds, to make sure the user understands the editor is still responsive.

Chapter 7

Conclusion

The editor has been developed and documented with all the important requirements. The editor has been tested, and tested by a user the editor is intended for. As of the 6th of June, the xml-files and the editor is officially used to define commands, confirmations, signals and teledata in the satellite project. Based on that, the editor has fulfilled its purpose.

In the editor a user is able to change all parts of the definitions of all the four types and parameters can be added in a simple way. A cross-reference from commands to confirmations exists and can be used by the user. The editor is also working with a great amount of more advanced features. Some features will later be extended, and features might be added. The `TreePanel` will most likely, with some small changes, be part of other software components in the satellite project.

Because of the importance of the files being valid, some work is put into validation of the type files, these validations can later be extended to contain more tests.

Some parts of the version control functions might in the future, use more than an acceptable amount of ram. Some solutions for that problem is covered in section 6.4.

To extend the editor, a call to the `Generation` tool to generate header files might be added. A way to define and use general types (in a not yet agreed upon format) will most likely also find its way to the editor.

Personally I have developed and/or improved skills in developing software, especially creating a GUI (and what users expect from that), using APIs/frameworks, object-oriented-programming and most importantly to work in a large project, with other students. I have also experienced the joy of creating a program other students will benefit from.

Appendix A

CD – code, documentation & xml files

At the end of this section a CD should be provided with the following files:
(if not, the files can be requested by e-mail to s093259@student.dtu.dk)

- code
  - AddParameterDialog.java
  - AddParameterDialogListener.java
  - DynamicTree.java
  - Editor.java
  - EditorListener.java
  - MainPanel.java
  - MainPanelListener.java
  - MyKeyAdapter.java
  - MyLogMessageCallback.java
  - MyWindowAdapter.java
  - NamedValueTableModel.java
  - ParameterTableModel.java

- RevisionTypeMap.java
- TreePanel.java
- TypeComparator.java
- VersionControlDialog.java
- XMLfilter.java
- common – common code in the project used in the editor, for changes made while developing the editor, refer to section 4.6
  - dtusat2
    - ModuleID.java
  - types/
    - GenType.java
    - TypeMap.java
    - command/CommandType.java
    - command/CommandTypeFile.java
    - confirmation/Confirmation.java
    - confirmation/ConfirmationType.java
    - confirmation/ConfirmationTypeFile.java
    - parameter/BooleanParameter.java
    - parameter/ByteArrayParameter.java
    - parameter/DoubleParameter.java
    - parameter/EnumParameter.java
    - parameter/IntParameter.java
    - parameter/NamedValue.java
    - parameter/Parameter.java
    - parameter/ParameterType.java
    - parameter/StringParameter.java
    - signal/SignalType.java
    - signal/SignalTypeFile.java
    - teledata/TeledataType.java
    - teledata/TeledataTypeFile.java
- Javadoc in html format
- xml/ – xml files
  - CommandTypes.xml
  - ConfirmationTypes.xml
  - SignalTypes.xml
  - TeledataTypes.xml

- test/ – test files used while testing the revision control in section 6.4.
  - CommandTypes1.xml
  - CommandTypes2.xml
  - CommandTypes3.xml
  - CommandTypes4.xml
  - CommandTypes5.xml

Appendix B

Screenshots of the editor

Figure B.1: The editor on startup, with disabled fields.

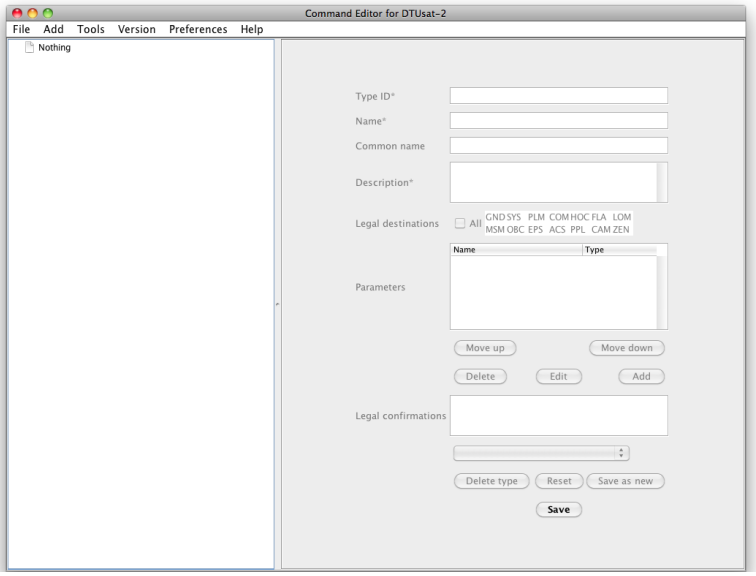


Figure B.2: The editor showing command types.

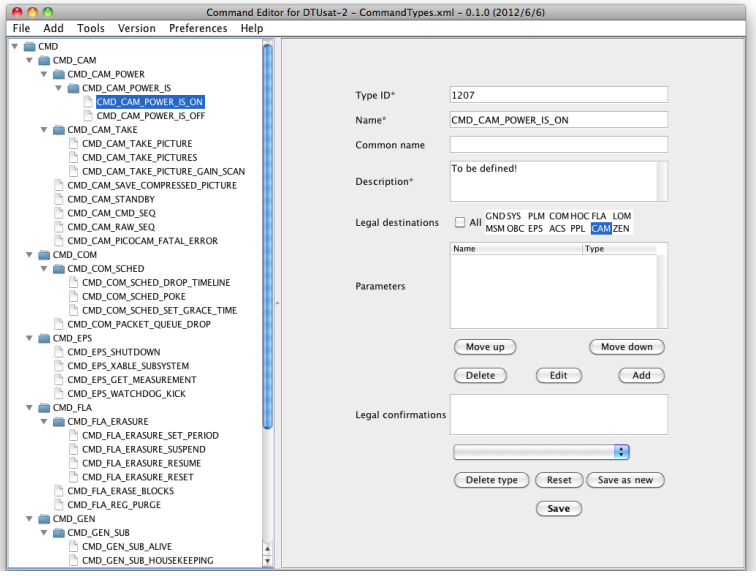


Figure B.3: The editor showing command types with both hidden fields.

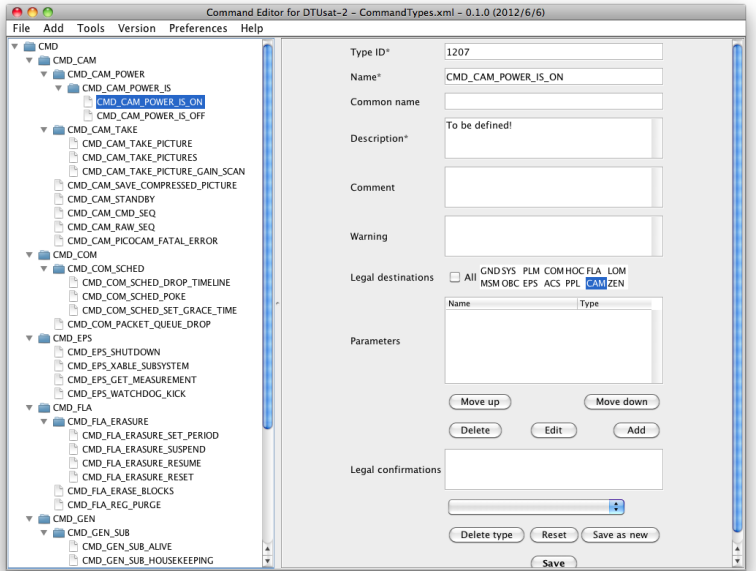


Figure B.4: The editor showing confirmation types.

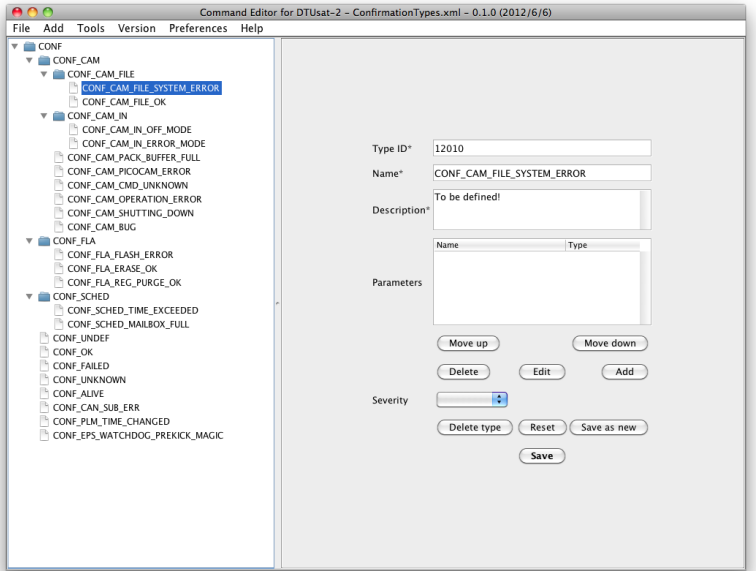


Figure B.5: The editor showing tele data types (the same when showing signal types)

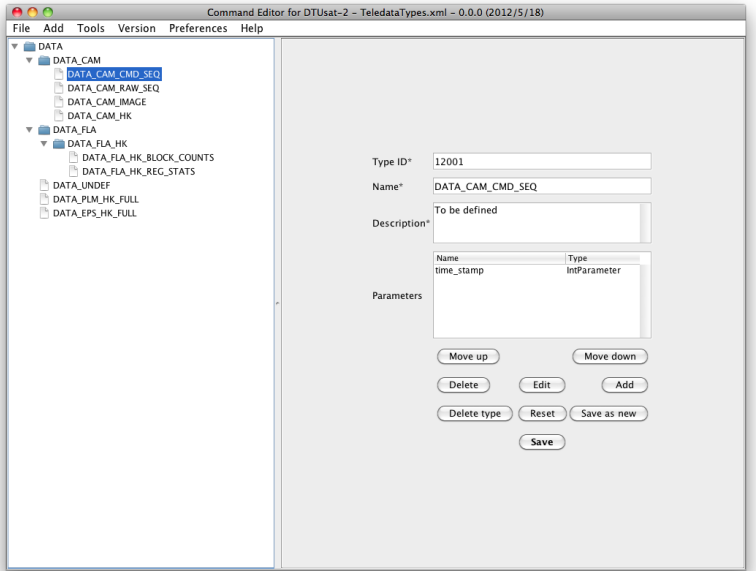
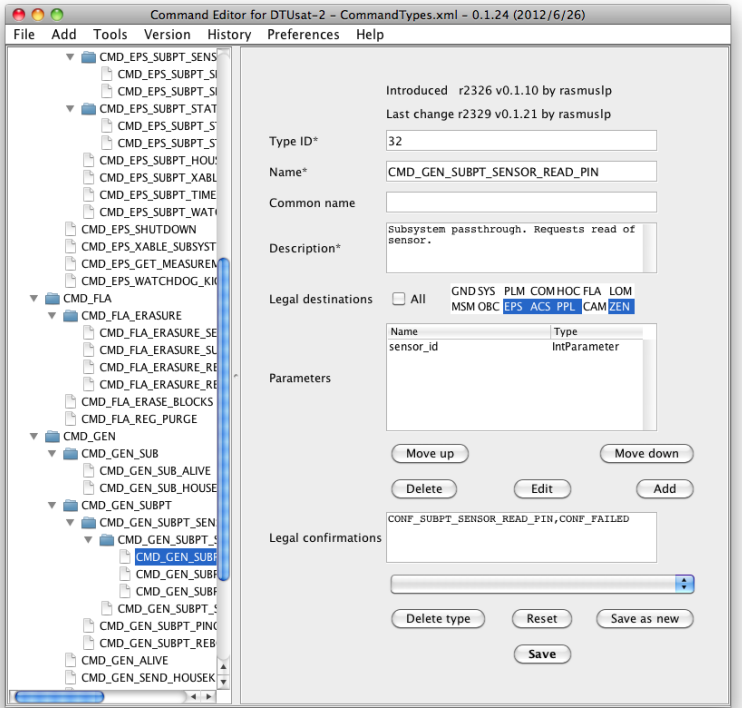


Figure B.6: The editor showing a type under version control. For screenshots of the version control dialogs refer to section 6.4



Appendix C

The menu of the editor

- File
  - New
    - Command Type Map
    - Confirmation Type Map
    - Teledata Type Map
    - Signal Type Map
  - Open
    - Command Type Map
    - Confirmation Type Map
    - Teledata Type Map
    - Signal Type Map
  - Save
  - Save as
- Add
  - Comment
  - Warning
  - New Type
- Tools

- Validate
- Version
  - Major version
  - Minor version
  - Change version
- History
  - View type history
  - View file history
- Preferences
  - Default size
    - Set current
    - Reset
  - Default path
    - Set current
    - Reset
  - TreePanel
    - Toggle TreePanel sort
    - Set categories
  - Toggle DEBUG mode
  - Select confirmations-file
  - Clear preferences
  - Show preferences
- Help
  - Help
  - About

Appendix D

Xml files for the satellite

D.1 CommandTypes.xml

The xml file defining the `CommandType`s.

```
1  <DTUsat2CommandTypes>
2      <version>0.1.24 (2012/6/26)</version>
3      <commandType name="CMD_UNDEF" typeID="0">
4          <description>This is a pseudo command to be used in
               situations where no proper command has been defined</
               description>
5          <parameters/>
6      </commandType>
7      <commandType name="CMD_GEN_ALIVE" typeID="1">
8          <description>Request alive response to be sent to source
               module</description>
9          <legalDestinations/>
10     </commandType>
11     <commandType name="CMD_GEN_SEND_HOUSEKEEPING" typeID="2">
12         <description>Request housekeeping data to be sent to GND</
               description>
13         <legalDestinations/>
14         <parameters>
15             <parameter name="hk_set">
16                 <description>Bitvector of desired set of housekeeping
                       data.
17                     Module may decide to send a superset of these.
```

```
18                  Default value always results in full housekeeping</
                        description>
19              <parameterType class="common.types.parameter.
                    IntParameter">
20                  <size>4</size>
21                  <lowerLimit>0</lowerLimit>
22                  <defaultValue>0xFFFFFFFF</defaultValue>
23              </parameterType>
24          </parameter>
25      </parameters>
26  </commandType>
27  <commandType name="CMD_GEN_SET_LOG_LEVEL" typeID="10">
28      <description>Set log level of destination module</description
            >
29      <legalDestinations/>
30      <parameters>
31          <parameter name="level">
32              <description>New level</description>
33              <parameterType class="common.types.parameter.
                    EnumParameter">
34                  <size>1</size>
35                  <defaultValue>LOG_INFO</defaultValue>
36                  <values>
37                      <namedValue name="LOG_DEBUG_FINEST" val="1"/>
38                      <namedValue name="LOG_OLDDEBUG" val="3"/>
39                      <namedValue name="LOG_DEBUG_FINER" val="5"/>
40                      <namedValue name="LOG_DEBUG" val="9"/>
41                      <namedValue name="LOG_INFO" val="10"/>
42                      <namedValue name="LOG_WARN" val="30"/>
43                      <namedValue name="LOG_ERR" val="40"/>
44                      <namedValue name="LOG_CRIT" val="50"/>
45                  </values>
46              </parameterType>
47          </parameter>
48      </parameters>
49  </commandType>
50  <commandType name="CMD_GEN_SUB_ALIVE" typeID="21">
51      <description>Request alive response from subsystem to be sent
            to source module</description>
52      <legalDestinations>
53          <moduleID>EPS</moduleID>
54          <moduleID>ACS</moduleID>
55          <moduleID>PPL</moduleID>
56          <moduleID>CAM</moduleID>
57          <moduleID>ZEN</moduleID>
58      </legalDestinations>
59  </commandType>
60  <commandType name="CMD_GEN_SUB_HOUSEKEEPING" typeID="22">
61      <description>Request housekeeping from subsystem</description
            >
62      <legalDestinations>
63          <moduleID>EPS</moduleID>
64          <moduleID>ACS</moduleID>
65          <moduleID>PPL</moduleID>
66          <moduleID>CAM</moduleID>
```

```
67              <moduleID>ZEN</moduleID>
68          </legalDestinations>
69          <parameters>
70              <parameter name="hk_set">
71                  <description>Housekeeping set</description>
72                  <parameterType class="common.types.parameter.
                        IntParameter">
73                      <size>4</size>
74                      <lowerLimit>0</lowerLimit>
75                      <defaultValue>4294967295</defaultValue>
76                  </parameterType>
77              </parameter>
78          </parameters>
79          <legalConfirmations/>
80      </commandType>
81      <commandType name="CMD_GEN_SUBPT_PING" typeID="30">
82          <description>Subsystem passthrough. Sends a value to the
                subsystem, which it should echo back.</description>
83          <legalDestinations>
84              <moduleID>EPS</moduleID>
85              <moduleID>ACS</moduleID>
86              <moduleID>PPL</moduleID>
87              <moduleID>ZEN</moduleID>
88          </legalDestinations>
89          <packed>true</packed>
90          <parameters>
91              <parameter name="value">
92                  <description>The value the subsystem must echo.</
                        description>
93                  <parameterType class="common.types.parameter.
                        IntParameter">
94                      <size>4</size>
95                      <lowerLimit>0</lowerLimit>
96                  </parameterType>
97              </parameter>
98          </parameters>
99          <legalConfirmations>
100             <string>CONF_SUBPT_PING</string>
101             <string>CONF_FAILED</string>
102         </legalConfirmations>
103     </commandType>
104     <commandType name="CMD_GEN_SUBPT_REBOOT_DETECT" typeID="31">
105         <description>Subsystem passthrough. Requests indication of
                whether or not the subsystem has rebooted.</description>
106         <legalDestinations>
107             <moduleID>EPS</moduleID>
108             <moduleID>ACS</moduleID>
109             <moduleID>PPL</moduleID>
110             <moduleID>ZEN</moduleID>
111         </legalDestinations>
112         <legalConfirmations>
113             <string>CONF_SUBPT_REBOOT_DETECT</string>
114             <string>CONF_FAILED</string>
115         </legalConfirmations>
116     </commandType>
```

```
117     <commandType name="CMD_GEN_SUBPT_SENSOR_READ_PIN" typeID="32">
118         <description>Subsystem passthrough. Requests read of sensor
                .</description>
119         <legalDestinations>
120             <moduleID>EPS</moduleID>
121             <moduleID>ACS</moduleID>
122             <moduleID>PPL</moduleID>
123             <moduleID>ZEN</moduleID>
124         </legalDestinations>
125         <packed>true</packed>
126         <parameters>
127             <parameter name="sensor_id">
128                 <description>ID of the sensor to read.</description>
129                 <parameterType class="common.types.parameter.
                        IntParameter">
130                     <size>1</size>
131                     <lowerLimit>0</lowerLimit>
132                 </parameterType>
133             </parameter>
134         </parameters>
135         <legalConfirmations>
136             <string>CONF_SUBPT_SENSOR_READ_PIN</string>
137             <string>CONF_FAILED</string>
138         </legalConfirmations>
139     </commandType>
140     <commandType name="CMD_GEN_SUBPT_SENSOR_READ_ADC" typeID="33">
141         <description>Subsystem passthrough. Requests read of sensor
                .</description>
142         <legalDestinations>
143             <moduleID>EPS</moduleID>
144             <moduleID>ACS</moduleID>
145             <moduleID>PPL</moduleID>
146             <moduleID>ZEN</moduleID>
147         </legalDestinations>
148         <packed>true</packed>
149         <parameters>
150             <parameter name="sensor_id">
151                 <description>ID of the sensor to read.</description>
152                 <parameterType class="common.types.parameter.
                        IntParameter">
153                     <size>1</size>
154                     <lowerLimit>0</lowerLimit>
155                 </parameterType>
156             </parameter>
157         </parameters>
158         <legalConfirmations>
159             <string>CONF_SUBPT_SENSOR_READ_ADC</string>
160             <string>CONF_FAILED</string>
161         </legalConfirmations>
162     </commandType>
163     <commandType name="CMD_GEN_SUBPT_SENSOR_CONFIG_TEMPERATURE"
            typeID="34">
164         <description>Subsystem passthrough. Requests configuration of
                temperature sensor.</description>
165         <legalDestinations>
```

```
166             <moduleID>EPS</moduleID>
167             <moduleID>ACS</moduleID>
168             <moduleID>PPL</moduleID>
169             <moduleID>ZEN</moduleID>
170         </legalDestinations>
171         <packed>true</packed>
172         <parameters>
173             <parameter name="sensor_id">
174                 <description>ID of the sensor to configure.</
                        description>
175                 <parameterType class="common.types.parameter.
                        IntParameter">
176                     <size>1</size>
177                     <lowerLimit>0</lowerLimit>
178                 </parameterType>
179             </parameter>
180             <parameter name="alarm">
181                 <description>Alarm threshold value.</description>
182                 <parameterType class="common.types.parameter.
                        IntParameter">
183                     <size>2</size>
184                 </parameterType>
185             </parameter>
186             <parameter name="precision">
187                 <description>Precision of sensor.</description>
188                 <parameterType class="common.types.parameter.
                        IntParameter">
189                     <size>1</size>
190                     <lowerLimit>0</lowerLimit>
191                 </parameterType>
192             </parameter>
193         </parameters>
194         <legalConfirmations>
195             <string>CONF_SUBPT_SENSOR_CONFIG_TEMPERATURE</string>
196             <string>CONF_FAILED</string>
197         </legalConfirmations>
198     </commandType>
199     <commandType name="CMD_GEN_SUBPT_SENSOR_READ_TEMPERATURE" typeID
            ="35">
200         <description>Subsystem passthrough. Requests read of
                temperature sensor.</description>
201         <legalDestinations>
202             <moduleID>EPS</moduleID>
203             <moduleID>ACS</moduleID>
204             <moduleID>PPL</moduleID>
205             <moduleID>ZEN</moduleID>
206         </legalDestinations>
207         <parameters>
208             <parameter name="sensor_id">
209                 <description>ID of the sensor to read.</description>
210                 <parameterType class="common.types.parameter.
                        IntParameter">
211                     <size>1</size>
212                     <lowerLimit>0</lowerLimit>
213                 </parameterType>
```

```
214             </parameter>
215         </parameters>
216         <legalConfirmations>
217             <string>CONF_SUBPT_SENSOR_READ_TEMPERATURE</string>
218             <string>CONF_FAILED</string>
219         </legalConfirmations>
220     </commandType>
221     <commandType name="CMD_CAM_TAKE_PICTURE" typeID="1200">
222         <description>Take a picture</description>
223         <legalDestinations>
224             <moduleID>CAM</moduleID>
225         </legalDestinations>
226         <parameters>
227             <parameter name="capture_type">
228                 <description> </description>
229                 <parameterType class="common.types.parameter.
                        EnumParameter">
230                     <values>
231                         <namedValue name="USE_TRIGGER" val="0xF0"/>
232                         <namedValue name="USE_SHUTTER" val="0x00"/>
233                     </values>
234                 </parameterType>
235             </parameter>
236         </parameters>
237     </commandType>
238     <commandType name="CMD_CAM_TAKE_PICTURES" typeID="1201">
239         <description>Take a series of pictures</description>
240         <legalDestinations>
241             <moduleID>CAM</moduleID>
242         </legalDestinations>
243     </commandType>
244     <commandType name="CMD_CAM_TAKE_PICTURE_GAIN_SCAN" typeID
          ="1202">
245         <comment>Should have a better name.</comment>
246         <description>Set gain scan </description>
247         <legalDestinations>
248             <moduleID>CAM</moduleID>
249         </legalDestinations>
250         <parameters>
251             <parameter name="ADCFloor">
252                 <description>To be defined!</description>
253                 <parameterType class="common.types.parameter.
                        IntParameter">
254                     <size>2</size>
255                     <lowerLimit>0</lowerLimit>
256                 </parameterType>
257             </parameter>
258             <parameter name="ADCCeiling">
259                 <description>To be defined!</description>
260                 <parameterType class="common.types.parameter.
                        IntParameter">
261                     <size>2</size>
262                     <lowerLimit>0</lowerLimit>
263                 </parameterType>
264             </parameter>
```

```
265         </parameters>
266     </commandType>
267     <commandType name="CMD_CAM_SAVE_COMPRESSED_PICTURE" typeID
            ="1203">
268         <description>Save a picture</description>
269         <legalDestinations>
270             <moduleID>CAM</moduleID>
271         </legalDestinations>
272     </commandType>
273     <commandType name="CMD_CAM_STANDBY" typeID="1204">
274         <description>Enter standby mode</description>
275         <legalDestinations>
276             <moduleID>CAM</moduleID>
277         </legalDestinations>
278     </commandType>
279     <commandType name="CMD_CAM_CMD_SEQ" typeID="1205">
280         <description>Send raw command sequence to cam</description>
281         <legalDestinations>
282             <moduleID>CAM</moduleID>
283         </legalDestinations>
284         <parameters>
285             <parameter name="sequence_length">
286                 <description>Length of command byte sequence</
                        description>
287                 <parameterType class="common.types.parameter.
                        IntParameter">
288                     <size>1</size>
289                     <lowerLimit>0</lowerLimit>
290                 </parameterType>
291             </parameter>
292             <parameter name="mid_sequence">
293                 <description>Sequence of mid raw commands</description>
294                 <parameterType class="common.types.parameter.
                        ByteArrayParameter" length="30"/>
295             </parameter>
296         </parameters>
297     </commandType>
298     <commandType name="CMD_CAM_RAW_SEQ" typeID="1206">
299         <description>Send raw command sequence to cam</description>
300         <legalDestinations>
301             <moduleID>CAM</moduleID>
302         </legalDestinations>
303         <parameters>
304             <parameter name="TM_is_returned">
305                 <description>To be defined!</description>
306                 <parameterType class="common.types.parameter.
                        BooleanParameter">
307                     <defaultValue>true</defaultValue>
308                 </parameterType>
309             </parameter>
310             <parameter name="command_length">
311                 <description>Length of command byte sequence</
                        description>
312                 <parameterType class="common.types.parameter.
                        IntParameter">
```

```
313                 <size>1</size>
314                 <lowerLimit>0</lowerLimit>
315             </parameterType>
316         </parameter>
317         <parameter name="raw_command">
318             <description>Raw PicoCam command</description>
319             <parameterType class="common.types.parameter.
                    ByteArrayParameter" length="30"/>
320         </parameter>
321     </parameters>
322 </commandType>
323 <commandType name="CMD_CAM_POWER_IS_ON" typeID="1207">
324     <description>To be defined!</description>
325     <legalDestinations>
326         <moduleID>CAM</moduleID>
327     </legalDestinations>
328 </commandType>
329 <commandType name="CMD_CAM_POWER_IS_OFF" typeID="1208">
330     <description>To be defined!</description>
331     <legalDestinations>
332         <moduleID>CAM</moduleID>
333     </legalDestinations>
334 </commandType>
335 <commandType name="CMD_CAM_PICOCAM_FATAL_ERROR" typeID="1209">
336     <description>To be defined!</description>
337     <legalDestinations>
338         <moduleID>CAM</moduleID>
339     </legalDestinations>
340 </commandType>
341 <commandType name="CMD_PLM_SET_TIME" typeID="2001">
342     <description>Set system time</description>
343     <legalDestinations>
344         <moduleID>PLM</moduleID>
345     </legalDestinations>
346     <parameters>
347         <parameter name="new_time">
348             <description>New time (UTC; ms since epoch)</
                    description>
349             <parameterType class="common.types.parameter.
                    IntParameter">
350                 <size>8</size>
351                 <lowerLimit>0</lowerLimit>
352                 <unit>ms</unit>
353             </parameterType>
354         </parameter>
355     </parameters>
356 </commandType>
357 <commandType name="CMD_PLM_RESET_WATCHDOG_CYCLES" typeID="2002">
358     <description>Reset watchdog cycle</description>
359     <legalDestinations>
360         <moduleID>PLM</moduleID>
361     </legalDestinations>
362 </commandType>
363 <commandType name="CMD_COM_PACKET_QUEUE_DROP" typeID="3002">
364     <description>Clear the downlink packet queue</description>
```

```
365         <warning>Dropping the packet queue may cause loss of
                significant onboard data</warning>
366         <legalDestinations>
367             <moduleID>COM</moduleID>
368         </legalDestinations>
369     </commandType>
370     <commandType name="CMD_COM_SCHED_DROP_TIMELINE" typeID="3501">
371         <description>Clear the scheduler timeline</description>
372         <warning>Dropping the scheduler cancels all planned, uploaded
                commands</warning>
373         <legalDestinations>
374             <moduleID>COM</moduleID>
375         </legalDestinations>
376     </commandType>
377     <commandType name="CMD_COM_SCHED_POKE" typeID="3502">
378         <description>Wake up scheduler</description>
379         <legalDestinations>
380             <moduleID>COM</moduleID>
381         </legalDestinations>
382     </commandType>
383     <commandType name="CMD_COM_SCHED_SET_GRACE_TIME" typeID="3503">
384         <description>Set max latency of scheduled commands</
                description>
385         <legalDestinations>
386             <moduleID>COM</moduleID>
387         </legalDestinations>
388         <parameters>
389             <parameter name="grace_time">
390                 <description> </description>
391                 <parameterType class="common.types.parameter.
                        IntParameter">
392                     <size>4</size>
393                     <lowerLimit>0</lowerLimit>
394                     <unit>ms</unit>
395                 </parameterType>
396             </parameter>
397         </parameters>
398     </commandType>
399     <commandType name="CMD_FLA_ERASE_BLOCKS" typeID="5000">
400         <description>Try to erase dirty flash blocks</description>
401         <legalDestinations>
402             <moduleID>FLA</moduleID>
403         </legalDestinations>
404         <parameters>
405             <parameter name="max_blocks">
406                 <description>Maximum  number of blocks to be erased</
                        description>
407                 <parameterType class="common.types.parameter.
                        IntParameter">
408                     <size>1</size>
409                     <lowerLimit>1</lowerLimit>
410                     <upperLimit>62</upperLimit>
411                     <defaultValue>1</defaultValue>
412                 </parameterType>
413             </parameter>
```

```
414         </parameters>
415     </commandType>
416     <commandType name="CMD_FLA_ERASURE_SET_PERIOD" typeID="5001">
417         <description>Set auto erasure period</description>
418         <legalDestinations>
419             <moduleID>FLA</moduleID>
420         </legalDestinations>
421         <parameters>
422             <parameter name="new_period">
423                 <description>New period (0 disables auto erasure) </
                        description>
424                 <parameterType class="common.types.parameter.
                        IntParameter">
425                     <size>4</size>
426                     <lowerLimit>0</lowerLimit>
427                     <defaultValue>600</defaultValue>
428                     <unit>sec</unit>
429                 </parameterType>
430             </parameter>
431         </parameters>
432     </commandType>
433     <commandType name="CMD_FLA_ERASURE_SUSPEND" typeID="5002">
434         <description>Suspend auto erasure</description>
435         <legalDestinations>
436             <moduleID>FLA</moduleID>
437         </legalDestinations>
438         <parameters/>
439     </commandType>
440     <commandType name="CMD_FLA_ERASURE_RESUME" typeID="5003">
441         <description>Resume auto erasure</description>
442         <legalDestinations>
443             <moduleID>FLA</moduleID>
444         </legalDestinations>
445         <parameters/>
446     </commandType>
447     <commandType name="CMD_FLA_ERASURE_RESET" typeID="5004">
448         <description>Drop any suspension</description>
449         <legalDestinations>
450             <moduleID>FLA</moduleID>
451         </legalDestinations>
452         <parameters/>
453     </commandType>
454     <commandType name="CMD_FLA_REG_PURGE" typeID="5010">
455         <description>Purge (compact) registry</description>
456         <legalDestinations>
457             <moduleID>FLA</moduleID>
458         </legalDestinations>
459         <parameters/>
460     </commandType>
461     <commandType name="CMD_MSM_LIST" typeID="7000">
462         <description>List ???</description>
463         <legalDestinations>
464             <moduleID>MSM</moduleID>
465         </legalDestinations>
466     </commandType>
```

```
467     <commandType name="CMD_MSM_RETR" typeID="7001">
468         <description>To be defined!</description>
469         <legalDestinations>
470             <moduleID>MSM</moduleID>
471         </legalDestinations>
472     </commandType>
473     <commandType name="CMD_OBC_WATCHDOG_EXT_KICK" typeID="8001">
474         <description>Reset external watchdog</description>
475         <legalDestinations>
476             <moduleID>OBC</moduleID>
477         </legalDestinations>
478     </commandType>
479     <commandType name="CMD_EPS_SHUTDOWN" typeID="9000">
480         <description>Shutdown EPS</description>
481         <legalDestinations>
482             <moduleID>EPS</moduleID>
483         </legalDestinations>
484     </commandType>
485     <commandType name="CMD_EPS_XABLE_SUBSYSTEM" typeID="9001">
486         <description>Enable/Disable subsystem</description>
487         <legalDestinations>
488             <moduleID>EPS</moduleID>
489         </legalDestinations>
490         <parameters>
491             <parameter name="system">
492                 <description>Subsystem</description>
493                 <parameterType class="common.types.parameter.
                        EnumParameter">
494                     <values>
495                         <namedValue name="ACS" val="0"/>
496                         <namedValue name="PPL" val="1"/>
497                         <namedValue name="Picocam" val="2"/>
498                     </values>
499                 </parameterType>
500             </parameter>
501             <parameter name="on">
502                 <description>Desired state</description>
503                 <parameterType class="common.types.parameter.
                        BooleanParameter">
504                     <defaultValue>false</defaultValue>
505                     <falseMark>Off</falseMark>
506                     <trueMark>On</trueMark>
507                 </parameterType>
508             </parameter>
509         </parameters>
510         <legalConfirmations/>
511     </commandType>
512     <commandType name="CMD_EPS_GET_MEASUREMENT" typeID="9003">
513         <description>To be defined!</description>
514         <legalDestinations>
515             <moduleID>EPS</moduleID>
516         </legalDestinations>
517     </commandType>
518     <commandType name="CMD_EPS_WATCHDOG_KICK" typeID="9010">
519         <description>Kick whole satellite watchdog</description>
```

```
520         <legalDestinations>
521             <moduleID>EPS</moduleID>
522         </legalDestinations>
523         <parameters>
524             <parameter name="magic_value">
525                 <description>Magic</description>
526                 <parameterType class="common.types.parameter.
                        IntParameter">
527                     <size>4</size>
528                     <lowerLimit>0</lowerLimit>
529                     <upperLimit>0xffffffff</upperLimit>
530                     <defaultValue>0xcc70eec6</defaultValue>
531                 </parameterType>
532             </parameter>
533         </parameters>
534     </commandType>
535     <commandType name="CMD_EPS_SUBPT_HOUSEKEEPING" typeID="9500">
536         <description>Subsystem passthrough. Request housekeeping data
                .</description>
537         <legalDestinations>
538             <moduleID>EPS</moduleID>
539         </legalDestinations>
540         <legalConfirmations>
541             <string>CONF_FAILED</string>
542             <string>CONF_EPS_SUBPT_HOUSEKEEPING</string>
543         </legalConfirmations>
544     </commandType>
545     <commandType name="CMD_EPS_SUBPT_XABLE" typeID="9502">
546         <description>Subsystem passthrough. Request a power state
                change for a subsystem.</description>
547         <legalDestinations>
548             <moduleID>EPS</moduleID>
549         </legalDestinations>
550         <packed>true</packed>
551         <parameters>
552             <parameter name="system">
553                 <description>Subsystem</description>
554                 <parameterType class="common.types.parameter.
                        EnumParameter">
555                     <values>
556                         <namedValue name="ACS" val="0"/>
557                         <namedValue name="PPL" val="1"/>
558                         <namedValue name="Picocam" val="2"/>
559                     </values>
560                 </parameterType>
561             </parameter>
562             <parameter name="on">
563                 <description>Desired power state.</description>
564                 <parameterType class="common.types.parameter.
                        BooleanParameter">
565                     <defaultValue>false</defaultValue>
566                     <falseMark>Off</falseMark>
567                     <trueMark>On</trueMark>
568                 </parameterType>
569                 <sensorName></sensorName>
```

```
570             </parameter>
571         </parameters>
572         <legalConfirmations>
573             <string>CONF_OK</string>
574             <string>CONF_FAILED</string>
575         </legalConfirmations>
576     </commandType>
577     <commandType name="CMD_EPS_SUBPT_TIME_SYS" typeID="9503">
578         <description>Subsystem passthrough. Request power for a
                subsystem for a specific amount of time.</description>
579         <legalDestinations>
580             <moduleID>EPS</moduleID>
581         </legalDestinations>
582         <packed>true</packed>
583         <parameters>
584             <parameter name="system">
585                 <description>Subsystem</description>
586                 <parameterType class="common.types.parameter.
                        EnumParameter">
587                     <values>
588                         <namedValue name="ACS" val="0"/>
589                         <namedValue name="PPL" val="1"/>
590                         <namedValue name="Picocam" val="2"/>
591                     </values>
592                 </parameterType>
593             </parameter>
594             <parameter name="time">
595                 <description>Duration the subsystem should be powered.
                        In seconds.</description>
596                 <parameterType class="common.types.parameter.
                        IntParameter">
597                     <size>1</size>
598                     <lowerLimit>0</lowerLimit>
599                 </parameterType>
600             </parameter>
601         </parameters>
602         <legalConfirmations>
603             <string>CONF_OK</string>
604             <string>CONF_FAILED</string>
605         </legalConfirmations>
606     </commandType>
607     <commandType name="CMD_EPS_SUBPT_WATCHDOG_KICK" typeID="9504">
608         <description>Subsystem passthrough. Kicks the EPS (satelite-
                wide) watchdog.</description>
609         <legalDestinations>
610             <moduleID>EPS</moduleID>
611         </legalDestinations>
612         <parameters>
613             <parameter name="magic_value">
614                 <description>The one true magic value needed for the
                        kick to happen.</description>
615                 <parameterType class="common.types.parameter.
                        IntParameter">
616                     <size>4</size>
617                     <lowerLimit>0</lowerLimit>
```

```
618             </parameterType>
619          </parameter>
620       </parameters>
621       <legalConfirmations>
622          <string>CONF_OK</string>
623          <string>CONF_FAILED</string>
624       </legalConfirmations>
625    </commandType>
626    <commandType name="CMD_EPS_SUBPT_SENSOR_READ_ALL" typeID="9505">
627       <description>Subsystem passthrough. Requests read of all
              sensors.</description>
628       <legalDestinations>
629          <moduleID>EPS</moduleID>
630       </legalDestinations>
631       <legalConfirmations>
632          <string>CONF_FAILED</string>
633          <string>CONF_EPS_SUBPT_SENSOR_READ_ALL</string>
634       </legalConfirmations>
635    </commandType>
636    <commandType name="CMD_EPS_SUBPT_SENSOR_CONFIG_ALL" typeID
          ="9506">
637       <description>Subsystem passthrough. Requests a configuration
              of all (i.e. temperature) sensors.</description>
638       <legalDestinations>
639          <moduleID>EPS</moduleID>
640       </legalDestinations>
641       <packed>true</packed>
642       <parameters>
643          <parameter name="alarm1">
644             <description>Alarm threshold value for temperature
                   sensor 1.</description>
645             <parameterType class="common.types.parameter.
                   IntParameter">
646                <size>2</size>
647             </parameterType>
648          </parameter>
649          <parameter name="precision1">
650             <description>Precision of temperature sensor 1.</
                   description>
651             <parameterType class="common.types.parameter.
                   IntParameter">
652                <size>1</size>
653                <lowerLimit>0</lowerLimit>
654             </parameterType>
655          </parameter>
656          <parameter name="alarm2">
657             <description>Alarm threshold value for temperature
                   sensor 2.</description>
658             <parameterType class="common.types.parameter.
                   IntParameter">
659                <size>2</size>
660             </parameterType>
661          </parameter>
662          <parameter name="precision2">
```

```
663             <description>Precision of temperature sensor 2.</
                    description>
664             <parameterType class="common.types.parameter.
                    IntParameter">
665                 <size>1</size>
666                 <lowerLimit>0</lowerLimit>
667             </parameterType>
668         </parameter>
669     </parameters>
670     <legalConfirmations>
671         <string>CONF_OK</string>
672         <string>CONF_FAILED</string>
673     </legalConfirmations>
674 </commandType>
675 <commandType name="CMD_EPS_SUBPT_STATE_GET" typeID="9507">
676     <description>Subsystem passthrough. Requests the power state
            of all non-boom systems.</description>
677     <legalDestinations>
678         <moduleID>EPS</moduleID>
679     </legalDestinations>
680     <legalConfirmations>
681         <string>CONF_FAILED</string>
682         <string>CONF_EPS_SUBPT_STATE_GET</string>
683     </legalConfirmations>
684 </commandType>
685 <commandType name="CMD_EPS_SUBPT_STATE_SET" typeID="9508">
686     <description>Subsystem passthrough. Requests a power state
            change of all non-boom systems.</description>
687     <legalDestinations>
688         <moduleID>EPS</moduleID>
689     </legalDestinations>
690     <packed>true</packed>
691     <parameters>
692         <parameter name="acs">
693             <description>Desired power state of subsystem.</
                    description>
694             <parameterType class="common.types.parameter.
                    BooleanParameter">
695                 <defaultValue>false</defaultValue>
696                 <falseMark>Off</falseMark>
697                 <trueMark>On</trueMark>
698             </parameterType>
699         </parameter>
700         <parameter name="ppl">
701             <description>Desired power state of subsystem.</
                    description>
702             <parameterType class="common.types.parameter.
                    BooleanParameter">
703                 <defaultValue>false</defaultValue>
704                 <falseMark>Off</falseMark>
705                 <trueMark>On</trueMark>
706             </parameterType>
707         </parameter>
708         <parameter name="picocam">
```

```
709             <description>Desired power state of subsystem.</
                    description>
710             <parameterType class="common.types.parameter.
                    BooleanParameter">
711                 <defaultValue>false</defaultValue>
712                 <falseMark>Off</falseMark>
713                 <trueMark>On</trueMark>
714             </parameterType>
715         </parameter>
716     </parameters>
717     <legalConfirmations>
718         <string>CONF_OK</string>
719         <string>CONF_FAILED</string>
720     </legalConfirmations>
721 </commandType>
722 </DTUsat2CommandTypes>
```

D.2 ConfirmationTypes.xml

The xml file defining the `ConfirmationTypes`.

```
1  <DTUsat2ConfirmationTypes>
2      <version>0.1.16 (2012/6/26)</version>
3      <confirmationType name="CONF_UNDEF" typeID="0">
4          <description>Pseudo confirmation where no proper confirmation
               is known</description>
5          <parameters/>
6      </confirmationType>
7      <confirmationType name="CONF_OK" typeID="1">
8          <description>Generel positive confirmation</description>
9          <parameters/>
10     </confirmationType>
11     <confirmationType name="CONF_FAILED" typeID="2">
12         <description>General negative confirmation</description>
13         <parameters/>
14     </confirmationType>
15     <confirmationType name="CONF_UNKNOWN" typeID="3">
16         <description>Command not known to receiving module</
               description>
17         <parameters/>
18     </confirmationType>
19     <confirmationType name="CONF_ALIVE" typeID="4">
20         <description>Positive response to ALIVE command</description>
21         <parameters/>
22     </confirmationType>
23     <confirmationType name="CONF_CAN_SUB_ERR" typeID="10">
24         <description>A CAN-related error occurred</description>
25         <parameters>
26             <parameter name="command">
27                 <description>Offending command</description>
```

```
28              <parameterType class="common.types.parameter.
                    IntParameter">
29                  <size>1</size>
30                  <lowerLimit>0</lowerLimit>
31              </parameterType>
32          </parameter>
33          <parameter name="error_code">
34              <description>CAN error code (see file?)</description>
35              <parameterType class="common.types.parameter.
                    IntParameter">
36                  <size>1</size>
37                  <lowerLimit>0</lowerLimit>
38              </parameterType>
39          </parameter>
40      </parameters>
41  </confirmationType>
42  <confirmationType name="CONF_SUBPT_PING" typeID="100">
43      <description>Subsystem passthrough. Echo of sent value.</
            description>
44      <packed>true</packed>
45      <parameters>
46          <parameter name="value">
47              <description>The value the subsystem has echoed.</
                    description>
48              <parameterType class="common.types.parameter.
                    IntParameter">
49                  <size>4</size>
50                  <lowerLimit>0</lowerLimit>
51              </parameterType>
52          </parameter>
53      </parameters>
54  </confirmationType>
55  <confirmationType name="CONF_SUBPT_REBOOT_DETECT" typeID="101">
56      <description>Subsystem passthrough. Indicates whether or not
            the subsystem has rebooted.</description>
57      <packed>true</packed>
58      <parameters>
59          <parameter name="reboot_detected">
60              <description>Reboot detected.</description>
61              <parameterType class="common.types.parameter.
                    BooleanParameter">
62                  <defaultValue>false</defaultValue>
63                  <falseMark>False</falseMark>
64                  <trueMark>True</trueMark>
65              </parameterType>
66          </parameter>
67      </parameters>
68  </confirmationType>
69  <confirmationType name="CONF_SUBPT_SENSOR_READ_PIN" typeID
        ="102">
70      <description>Subsystem passthrough. Read sensor value.</
            description>
71      <packed>true</packed>
72      <parameters>
73          <parameter name="sensor_id">
```

```
74              <description>ID of the sensor read.</description>
75              <parameterType class="common.types.parameter.
                    IntParameter">
76                 <size>1</size>
77                 <lowerLimit>0</lowerLimit>
78              </parameterType>
79           </parameter>
80           <parameter name="value">
81              <description>Value of the read sensor.</description>
82              <parameterType class="common.types.parameter.
                    IntParameter">
83                 <size>1</size>
84                 <lowerLimit>0</lowerLimit>
85              </parameterType>
86           </parameter>
87        </parameters>
88     </confirmationType>
89     <confirmationType name="CONF_SUBPT_SENSOR_READ_ADC" typeID
           ="103">
90        <description>Subsystem passthrough. Read sensor value.</
              description>
91        <packed>true</packed>
92        <parameters>
93           <parameter name="sensor_id">
94              <description>ID of the sensor read.</description>
95              <parameterType class="common.types.parameter.
                    IntParameter">
96                 <size>1</size>
97                 <lowerLimit>0</lowerLimit>
98              </parameterType>
99           </parameter>
100          <parameter name="value">
101             <description>Value of the read sensor.</description>
102             <parameterType class="common.types.parameter.
                    IntParameter">
103                <size>2</size>
104                <lowerLimit>0</lowerLimit>
105             </parameterType>
106          </parameter>
107       </parameters>
108    </confirmationType>
109    <confirmationType name="CONF_SUBPT_SENSOR_CONFIG_TEMPERATURE"
          typeID="104">
110       <description>Subsystem passthrough. Sensor configured.</
              description>
111       <packed>true</packed>
112       <parameters>
113          <parameter name="sensor_id">
114             <description>ID of the sensor configured.</description>
115             <parameterType class="common.types.parameter.
                    IntParameter">
116                <size>1</size>
117                <lowerLimit>0</lowerLimit>
118             </parameterType>
119          </parameter>
```

```
120         </parameters>
121     </confirmationType>
122     <confirmationType name="CONF_SUBPT_SENSOR_READ_TEMPERATURE"
            typeID="105">
123         <description>Subsystem passthrough. Read temperature value.</
                description>
124         <packed>true</packed>
125         <parameters>
126             <parameter name="sensor_id">
127                 <description>ID of the sensor read.</description>
128                 <parameterType class="common.types.parameter.
                        IntParameter">
129                     <size>1</size>
130                     <lowerLimit>0</lowerLimit>
131                 </parameterType>
132             </parameter>
133             <parameter name="value">
134                 <description>Value of the temperature sensor read.</
                        description>
135                 <parameterType class="common.types.parameter.
                        IntParameter">
136                     <size>2</size>
137                 </parameterType>
138             </parameter>
139         </parameters>
140     </confirmationType>
141     <confirmationType name="CONF_PLM_TIME_CHANGED" typeID="2001">
142         <description>Time was successfully set</description>
143         <parameters>
144             <parameter name="old_timestamp">
145                 <description>Old timestamp</description>
146                 <parameterType class="common.types.parameter.
                        IntParameter">
147                     <size>8</size>
148                     <lowerLimit>0</lowerLimit>
149                 </parameterType>
150             </parameter>
151         </parameters>
152     </confirmationType>
153     <confirmationType name="CONF_SCHED_TIME_EXCEEDED" typeID="3500">
154         <description>Execution time has been passed - command ignored
                </description>
155     </confirmationType>
156     <confirmationType name="CONF_SCHED_MAILBOX_FULL" typeID="3501">
157         <description>Command could not be delivered to mailbox of
                module</description>
158     </confirmationType>
159     <confirmationType name="CONF_FLA_FLASH_ERROR" typeID="5000">
160         <description>An (unexpected) error occurred during a flash
                operation</description>
161         <parameters>
162             <parameter name="code">
163                 <description>Error code from flash file system </
                        description>
```

```
164             <parameterType class="common.types.parameter.
                    EnumParameter">
165                 <values>
166                     <namedValue name="
                            FLASHFS2_UNEXPECTED_BLOCK_SIZE_OR_COUNT" val
                            ="-1"/>
167                     <namedValue name="FLASHFS2_OUT_OF_BOUNDS" val
                            ="-2"/>
168                     <namedValue name="FLASHFS2_WRITE_ERROR" val
                            ="-3"/>
169                     <namedValue name="FLASHFS2_ERASE_ERROR" val
                            ="-4"/>
170                     <namedValue name="
                            FLASHFS2_METADATA_WRITE_ATTEMPTED" val="-5"/>
171                     <namedValue name="FLASHFS2_INCONSISTENT_DATA" val
                            ="-6"/>
172                     <namedValue name="FLASHFS2_NO_DIRTY_BLOCKS" val
                            ="-7"/>
173                     <namedValue name="FLASHFS2_NO_FREE_BLOCKS" val
                            ="-8"/>
174                     <namedValue name="FLASHFS2_FAILED_TO_GET_LIMITS"
                             val="-9"/>
175                     <namedValue name="FLASHFS2_READ_ERROR" val
                            ="-10"/>
176                     <namedValue name="
                            FLASHFS2_DATA_STORE_ALREADY_EXISTS" val
                            ="-11"/>
177                     <namedValue name="FLASHFS2_INVALID_DATA_STORE_ID"
                              val="-12"/>
178                     <namedValue name="
                            FLASHFS2_UNKNOWN_PREVIOUS_DATA_STORE" val
                            ="-13"/>
179                     <namedValue name="
                            FLASHFS2_IMAGE_NOT_OPENED_FOR_WRITING" val
                            ="-14"/>
180                     <namedValue name="
                            FLASHFS2_IMAGE_OPENED_FOR_WRITING" val
                            ="-15"/>
181                     <namedValue name="FLASHFS2_REGISTRY_CORRUPT" val
                            ="-16"/>
182                     <namedValue name="FLASHFS2_REGISTRY_KEY_NOT_FOUND
                            " val="-17"/>
183                     <namedValue name="
                            FLASHFS2_REGISTRY_INVALID_LENGTH" val="-18"/>
184                     <namedValue name="FLASHFS2_REGISTRY_NO_ROOM" val
                            ="-19"/>
185                     <namedValue name="FLASHFS2_IMAGE_BROKEN" val
                            ="-20"/>
186                     <namedValue name="FLASHFS2_INVALID_IMAGE_REF" val
                            ="-21"/>
187                     <namedValue name="FLASHFS2_IMAGE_NOT_FOUND" val
                            ="-22"/>
188                     <namedValue name="FLASHFS2_ALREADY_SEALED" val
                            ="-23"/>
```

```
189                     <namedValue name="FLASHFS2_NOT_SEALED" val
                            ="-24"/>
190                     <namedValue name="FLASHFS2_IMAGE_ALREADY_OPENED"
                            val="-25"/>
191                     <namedValue name="FLASHFS2_INVALID_MODE" val
                            ="-26"/>
192                 </values>
193             </parameterType>
194         </parameter>
195     </parameters>
196 </confirmationType>
197 <confirmationType name="CONF_FLA_ERASE_OK" typeID="5001">
198     <description>Erase command successful</description>
199     <parameters>
200         <parameter name="blocks_erased">
201             <description>Number of blocks erased</description>
202             <parameterType class="common.types.parameter.
                    IntParameter">
203                 <size>2</size>
204                 <lowerLimit>0</lowerLimit>
205             </parameterType>
206         </parameter>
207     </parameters>
208 </confirmationType>
209 <confirmationType name="CONF_FLA_REG_PURGE_OK" typeID="5002">
210     <description>Registry purge command successful</description>
211     <parameters>
212         <parameter name="entries">
213             <description>Number of registry entries copied</
                    description>
214             <parameterType class="common.types.parameter.
                    IntParameter">
215                 <size>2</size>
216                 <lowerLimit>0</lowerLimit>
217             </parameterType>
218         </parameter>
219     </parameters>
220 </confirmationType>
221 <confirmationType name="CONF_EPS_SUBPT_HOUSEKEEPING" typeID
        ="9500">
222     <severity></severity>
223     <description>Subsystem passthrough. Complete housekeeping
            from subsystem.</description>
224     <packed>true</packed>
225     <parameters>
226         <parameter name="acs">
227             <description>Power state of subsystem.</description>
228             <parameterType class="common.types.parameter.
                    BooleanParameter">
229                 <defaultValue>false</defaultValue>
230                 <falseMark>Off</falseMark>
231                 <trueMark>On</trueMark>
232             </parameterType>
233         </parameter>
234         <parameter name="ppl">
```

```
235             <description>Power state of subsystem.</description>
236             <parameterType class="common.types.parameter.
                    BooleanParameter">
237                 <defaultValue>false</defaultValue>
238                 <falseMark>Off</falseMark>
239                 <trueMark>On</trueMark>
240             </parameterType>
241         </parameter>
242         <parameter name="picocam">
243             <description>Power state of subsystem.</description>
244             <parameterType class="common.types.parameter.
                    BooleanParameter">
245                 <defaultValue>false</defaultValue>
246                 <falseMark>Off</falseMark>
247                 <trueMark>On</trueMark>
248             </parameterType>
249         </parameter>
250         <parameter name="sensor_value_adc1">
251             <description>Read sensor value from ADC 1.</description
                    >
252             <parameterType class="common.types.parameter.
                    IntParameter">
253                 <size>2</size>
254                 <lowerLimit>0</lowerLimit>
255             </parameterType>
256         </parameter>
257         <parameter name="sensor_value_adc2">
258             <description>Read sensor value from ADC 2.</description
                    >
259             <parameterType class="common.types.parameter.
                    IntParameter">
260                 <size>2</size>
261                 <lowerLimit>0</lowerLimit>
262             </parameterType>
263         </parameter>
264         <parameter name="sensor_value_adc3">
265             <description>Read sensor value from ADC 3.</description
                    >
266             <parameterType class="common.types.parameter.
                    IntParameter">
267                 <size>2</size>
268                 <lowerLimit>0</lowerLimit>
269             </parameterType>
270         </parameter>
271         <parameter name="sensor_value_adc4">
272             <description>Read sensor value from ADC 4.</description
                    >
273             <parameterType class="common.types.parameter.
                    IntParameter">
274                 <size>2</size>
275                 <lowerLimit>0</lowerLimit>
276             </parameterType>
277         </parameter>
278         <parameter name="sensor_value_adc5">
```

```
279             <description>Read sensor value from ADC 5.</description
                    >
280             <parameterType class="common.types.parameter.
                    IntParameter">
281               <size>2</size>
282               <lowerLimit>0</lowerLimit>
283             </parameterType>
284           </parameter>
285           <parameter name="sensor_value_adc6">
286             <description>Read sensor value from ADC 6.</description
                    >
287             <parameterType class="common.types.parameter.
                    IntParameter">
288               <size>2</size>
289               <lowerLimit>0</lowerLimit>
290             </parameterType>
291           </parameter>
292           <parameter name="sensor_value_adc7">
293             <description>Read sensor value from ADC 7.</description
                    >
294             <parameterType class="common.types.parameter.
                    IntParameter">
295               <size>2</size>
296               <lowerLimit>0</lowerLimit>
297             </parameterType>
298           </parameter>
299           <parameter name="sensor_value_adc8">
300             <description>Read sensor value from ADC 8.</description
                    >
301             <parameterType class="common.types.parameter.
                    IntParameter">
302               <size>2</size>
303               <lowerLimit>0</lowerLimit>
304             </parameterType>
305           </parameter>
306           <parameter name="sensor_value_adc9">
307             <description>Read sensor value from ADC 9.</description
                    >
308             <parameterType class="common.types.parameter.
                    IntParameter">
309               <size>2</size>
310               <lowerLimit>0</lowerLimit>
311             </parameterType>
312           </parameter>
313           <parameter name="sensor_value_adc10">
314             <description>Read sensor value from ADC 10.</
                    description>
315             <parameterType class="common.types.parameter.
                    IntParameter">
316               <size>2</size>
317               <lowerLimit>0</lowerLimit>
318             </parameterType>
319           </parameter>
320           <parameter name="sensor_value_adc11">
```

```
321             <description>Read sensor value from ADC 11.</
                    description>
322             <parameterType class="common.types.parameter.
                    IntParameter">
323                 <size>2</size>
324                 <lowerLimit>0</lowerLimit>
325             </parameterType>
326         </parameter>
327         <parameter name="sensor_value_adc12">
328             <description>Read sensor value from ADC 12.</
                    description>
329             <parameterType class="common.types.parameter.
                    IntParameter">
330                 <size>2</size>
331                 <lowerLimit>0</lowerLimit>
332             </parameterType>
333         </parameter>
334         <parameter name="sensor_value_pin1">
335             <description>Read sensor value from PIN 1.</description
                    >
336             <parameterType class="common.types.parameter.
                    IntParameter">
337                 <size>1</size>
338                 <lowerLimit>0</lowerLimit>
339             </parameterType>
340         </parameter>
341         <parameter name="sensor_value_temperature1">
342             <description>Read sensor value from temperature sensor
                    1.</description>
343             <parameterType class="common.types.parameter.
                    IntParameter">
344                 <size>2</size>
345             </parameterType>
346         </parameter>
347         <parameter name="sensor_value_temperature2">
348             <description>Read sensor value from temperature sensor
                    2.</description>
349             <parameterType class="common.types.parameter.
                    IntParameter">
350                 <size>2</size>
351             </parameterType>
352         </parameter>
353     </parameters>
354 </confirmationType>
355 <confirmationType name="CONF_EPS_SUBPT_SENSOR_READ_ALL" typeID
        ="9505">
356     <description>Subsystem passthrough. All read sensor values.</
            description>
357     <parameters>
358         <parameter name="sensor_value_adc1">
359             <description>Read sensor value from ADC 1.</description
                    >
360             <parameterType class="common.types.parameter.
                    IntParameter">
361                 <size>2</size>
```

```
362                 <lowerLimit>0</lowerLimit>
363             </parameterType>
364         </parameter>
365         <parameter name="sensor_value_adc2">
366             <description>Read sensor value from ADC 2.</description
                    >
367             <parameterType class="common.types.parameter.
                    IntParameter">
368                 <size>2</size>
369                 <lowerLimit>0</lowerLimit>
370             </parameterType>
371         </parameter>
372         <parameter name="sensor_value_adc3">
373             <description>Read sensor value from ADC 3.</description
                    >
374             <parameterType class="common.types.parameter.
                    IntParameter">
375                 <size>2</size>
376                 <lowerLimit>0</lowerLimit>
377             </parameterType>
378         </parameter>
379         <parameter name="sensor_value_adc4">
380             <description>Read sensor value from ADC 4.</description
                    >
381             <parameterType class="common.types.parameter.
                    IntParameter">
382                 <size>2</size>
383                 <lowerLimit>0</lowerLimit>
384             </parameterType>
385         </parameter>
386         <parameter name="sensor_value_adc5">
387             <description>Read sensor value from ADC 5.</description
                    >
388             <parameterType class="common.types.parameter.
                    IntParameter">
389                 <size>2</size>
390                 <lowerLimit>0</lowerLimit>
391             </parameterType>
392         </parameter>
393         <parameter name="sensor_value_adc6">
394             <description>Read sensor value from ADC 6.</description
                    >
395             <parameterType class="common.types.parameter.
                    IntParameter">
396                 <size>2</size>
397                 <lowerLimit>0</lowerLimit>
398             </parameterType>
399         </parameter>
400         <parameter name="sensor_value_adc7">
401             <description>Read sensor value from ADC 7.</description
                    >
402             <parameterType class="common.types.parameter.
                    IntParameter">
403                 <size>2</size>
404                 <lowerLimit>0</lowerLimit>
```

```
405             </parameterType>
406         </parameter>
407         <parameter name="sensor_value_adc8">
408             <description>Read sensor value from ADC 8.</description
                    >
409             <parameterType class="common.types.parameter.
                    IntParameter">
410                 <size>2</size>
411                 <lowerLimit>0</lowerLimit>
412             </parameterType>
413         </parameter>
414         <parameter name="sensor_value_adc9">
415             <description>Read sensor value from ADC 9.</description
                    >
416             <parameterType class="common.types.parameter.
                    IntParameter">
417                 <size>2</size>
418                 <lowerLimit>0</lowerLimit>
419             </parameterType>
420         </parameter>
421         <parameter name="sensor_value_adc10">
422             <description>Read sensor value from ADC 10.</
                    description>
423             <parameterType class="common.types.parameter.
                    IntParameter">
424                 <size>2</size>
425                 <lowerLimit>0</lowerLimit>
426             </parameterType>
427         </parameter>
428         <parameter name="sensor_value_adc11">
429             <description>Read sensor value from ADC 11.</
                    description>
430             <parameterType class="common.types.parameter.
                    IntParameter">
431                 <size>2</size>
432                 <lowerLimit>0</lowerLimit>
433             </parameterType>
434         </parameter>
435         <parameter name="sensor_value_adc12">
436             <description>Read sensor value from ADC 12.</
                    description>
437             <parameterType class="common.types.parameter.
                    IntParameter">
438                 <size>2</size>
439                 <lowerLimit>0</lowerLimit>
440             </parameterType>
441         </parameter>
442         <parameter name="sensor_value_pin1">
443             <description>Read sensor value from PIN 1.</description
                    >
444             <parameterType class="common.types.parameter.
                    IntParameter">
445                 <size>1</size>
446                 <lowerLimit>0</lowerLimit>
447             </parameterType>
```

```
448             </parameter>
449             <parameter name="sensor_value_temperature1">
450                 <description>Read sensor value from temperature sensor
                        1.</description>
451                 <parameterType class="common.types.parameter.
                        IntParameter">
452                     <size>2</size>
453                 </parameterType>
454             </parameter>
455             <parameter name="sensor_value_temperature2">
456                 <description>Read sensor value from temperature sensor
                        2.</description>
457                 <parameterType class="common.types.parameter.
                        IntParameter">
458                     <size>2</size>
459                 </parameterType>
460             </parameter>
461         </parameters>
462     </confirmationType>
463     <confirmationType name="CONF_EPS_SUBPT_STATE_GET" typeID="9507">
464         <description>Subsystem passthrough. The power state of all
                subsystems.</description>
465         <packed>true</packed>
466         <parameters>
467             <parameter name="acs">
468                 <description>Power state of subsystem.</description>
469                 <parameterType class="common.types.parameter.
                        BooleanParameter">
470                     <defaultValue>false</defaultValue>
471                     <falseMark>Off</falseMark>
472                     <trueMark>On</trueMark>
473                 </parameterType>
474             </parameter>
475             <parameter name="ppl">
476                 <description>Power state of subsystem.</description>
477                 <parameterType class="common.types.parameter.
                        BooleanParameter">
478                     <defaultValue>false</defaultValue>
479                     <falseMark>Off</falseMark>
480                     <trueMark>On</trueMark>
481                 </parameterType>
482             </parameter>
483             <parameter name="picocam">
484                 <description>Power state of subsystem.</description>
485                 <parameterType class="common.types.parameter.
                        BooleanParameter">
486                     <defaultValue>false</defaultValue>
487                     <falseMark>Off</falseMark>
488                     <trueMark>On</trueMark>
489                 </parameterType>
490             </parameter>
491         </parameters>
492     </confirmationType>
493     <confirmationType name="CONF_CAM_PACK_BUFFER_FULL" typeID
            ="12007">
```

```
494         <description>To be defined!</description>
495     </confirmationType>
496     <confirmationType name="CONF_CAM_PICOCAM_ERROR" typeID="12008">
497         <description>To be defined!</description>
498     </confirmationType>
499     <confirmationType name="CONF_CAM_CMD_UNKNOWN" typeID="12009">
500         <description>To be defined!</description>
501     </confirmationType>
502     <confirmationType name="CONF_CAM_FILE_SYSTEM_ERROR" typeID
            ="12010">
503         <description>To be defined!</description>
504     </confirmationType>
505     <confirmationType name="CONF_CAM_OPERATION_ERROR" typeID
            ="12011">
506         <description>To be defined!</description>
507     </confirmationType>
508     <confirmationType name="CONF_CAM_SHUTTING_DOWN" typeID="12012">
509         <description>To be defined!</description>
510     </confirmationType>
511     <confirmationType name="CONF_CAM_IN_OFF_MODE" typeID="12013">
512         <description>To be defined!</description>
513     </confirmationType>
514     <confirmationType name="CONF_CAM_IN_ERROR_MODE" typeID="12014">
515         <description>To be defined!</description>
516     </confirmationType>
517     <confirmationType name="CONF_CAM_BUG" typeID="12015">
518         <description>To be defined!</description>
519     </confirmationType>
520     <confirmationType name="CONF_CAM_FILE_OK" typeID="12016">
521         <description>To be defined!</description>
522         <parameters>
523             <parameter name="file_id">
524                 <description>Image no of saved image</description>
525                 <parameterType class="common.types.parameter.
                        IntParameter">
526                     <size>4</size>
527                     <lowerLimit>0</lowerLimit>
528                 </parameterType>
529             </parameter>
530         </parameters>
531     </confirmationType>
532 </DTUsat2ConfirmationTypes>
```

D.3 SignalTypes.xml

The xml file defining the `SignalTypes`.

```
1  <DTUsat2SignalTypes>
2      <version>0.1.0  (2012/6/6)</version>
3      <signalType name="SIG_OK" typeID="0">
4          <description></description>
```

```
5         <parameters/>
6       </signalType>
7       <signalType name="SIG_PING" typeID="6">
8         <description></description>
9         <parameters/>
10      </signalType>
11      <signalType name="SIG_PONG" typeID="7">
12        <description></description>
13        <parameters/>
14      </signalType>
15      <signalType name="SIG_REQUEST_PACKETS" typeID="8">
16        <description></description>
17        <parameters/>
18      </signalType>
19      <signalType name="SIG_PKT_ACKNOWLEDGEMENT" typeID="10">
20        <description></description>
21        <parameters/>
22      </signalType>
23      <signalType name="SIG_PACKETS_STOP_AND_RESET" typeID="11">
24        <description></description>
25        <parameters/>
26      </signalType>
27      <signalType name="SIG_PACKET_PARSE_ERROR" typeID="20">
28        <description></description>
29        <parameters/>
30      </signalType>
31      <signalType name="SIG_PACKET_SIZE_MISMATCH" typeID="30">
32        <description></description>
33        <parameters/>
34      </signalType>
35      <signalType name="SIG_UNKNOWN_SIGNAL_TYPE" typeID="51">
36        <description></description>
37        <parameters/>
38      </signalType>
39      <signalType name="SIG_UNKNOWN_PACKET_TYPE" typeID="52">
40        <description></description>
41        <parameters/>
42      </signalType>
43      <signalType name="SIG_DEST_MBOX_FULL" typeID="111">
44        <description></description>
45        <parameters/>
46      </signalType>
47      <signalType name="SIG_UNKNOWN_ERROR" typeID="114">
48        <description></description>
49        <parameters/>
50      </signalType>
51      <signalType name="SIG_SCHED_QUEUE_FULL" typeID="150">
52        <description></description>
53        <parameters/>
54      </signalType>
55      <signalType name="SIG_SCHED_PACKET_EXISTS" typeID="151">
56        <description></description>
57        <parameters/>
58      </signalType>
59      <signalType name="SIG_FLASH_ERROR" typeID="160">
```

```
60          <description></description>
61          <parameters/>
62      </signalType>
63      <signalType name="SIG_UPLINK_RESET" typeID="200">
64          <description></description>
65          <parameters/>
66      </signalType>
67      <signalType name="SIG_UPLINK_ERROR" typeID="201">
68          <description></description>
69          <parameters/>
70      </signalType>
71      <signalType name="SIG_UPLINK_TIMEOUT" typeID="202">
72          <description></description>
73          <parameters/>
74      </signalType>
75      <signalType name="SIG_UPLINK_SENT" typeID="203">
76          <description></description>
77          <parameters/>
78      </signalType>
79      <signalType name="SIG_WRONG_MODULE_ID" typeID="112">
80          <description>Command addresses unknown module </description>
81      </signalType>
82  </DTUsat2SignalTypes>
```

D.4 TeledataTypes.xml

The xml file defining the `TeledataTypes`.

```
1   <DTUsat2TeledataTypes>
2       <version>0.1.0  (2012/6/6)</version>
3       <teledataType name="DATA_PLM_HK_FULL" typeID="2100">
4           <description>PLM full housekeeping set</description>
5           <parameters>
6               <parameter name="count">
7                   <description> </description>
8                   <parameterType class="common.types.parameter.
                        IntParameter">
9                       <size>4</size>
10                      <lowerLimit>0</lowerLimit>
11                  </parameterType>
12              </parameter>
13              <parameter name="cycles">
14                  <description> </description>
15                  <parameterType class="common.types.parameter.
                        IntParameter">
16                      <size>4</size>
17                      <lowerLimit>0</lowerLimit>
18                  </parameterType>
19              </parameter>
20              <parameter name="sent">
21                  <description> </description>
```

```
22              <parameterType class="common.types.parameter.
                    IntParameter">
23                  <size>2</size>
24                  <lowerLimit>0</lowerLimit>
25              </parameterType>
26          </parameter>
27          <parameter name="era">
28              <description> </description>
29              <parameterType class="common.types.parameter.
                    IntParameter">
30                  <size>2</size>
31                  <lowerLimit>0</lowerLimit>
32              </parameterType>
33          </parameter>
34      </parameters>
35  </teledataType>
36  <teledataType name="DATA_FLA_HK_BLOCK_COUNTS" typeID="5100">
37      <description>FLA housekeeping: Data store block counts</
            description>
38      <parameters>
39          <parameter name="bad_count">
40              <description> </description>
41              <parameterType class="common.types.parameter.
                    IntParameter">
42                  <size>2</size>
43                  <lowerLimit>0</lowerLimit>
44              </parameterType>
45          </parameter>
46          <parameter name="blank_count">
47              <description> </description>
48              <parameterType class="common.types.parameter.
                    IntParameter">
49                  <size>2</size>
50                  <lowerLimit>0</lowerLimit>
51              </parameterType>
52          </parameter>
53          <parameter name="free_count">
54              <description> </description>
55              <parameterType class="common.types.parameter.
                    IntParameter">
56                  <size>2</size>
57                  <lowerLimit>0</lowerLimit>
58              </parameterType>
59          </parameter>
60          <parameter name="dirty_count">
61              <description> </description>
62              <parameterType class="common.types.parameter.
                    IntParameter">
63                  <size>2</size>
64                  <lowerLimit>0</lowerLimit>
65              </parameterType>
66          </parameter>
67          <parameter name="image_count">
68              <description> </description>
```

```
69              <parameterType class="common.types.parameter.
                    IntParameter">
70                  <size>2</size>
71                  <lowerLimit>0</lowerLimit>
72              </parameterType>
73          </parameter>
74          <parameter name="reg_count">
75              <description> </description>
76              <parameterType class="common.types.parameter.
                    IntParameter">
77                  <size>2</size>
78                  <lowerLimit>0</lowerLimit>
79              </parameterType>
80          </parameter>
81          <parameter name="sched_count">
82              <description> </description>
83              <parameterType class="common.types.parameter.
                    IntParameter">
84                  <size>2</size>
85                  <lowerLimit>0</lowerLimit>
86              </parameterType>
87          </parameter>
88          <parameter name="packet_count">
89              <description> </description>
90              <parameterType class="common.types.parameter.
                    IntParameter">
91                  <size>2</size>
92                  <lowerLimit>0</lowerLimit>
93              </parameterType>
94          </parameter>
95      </parameters>
96  </teledataType>
97  <teledataType name="DATA_FLA_HK_REG_STATS" typeID="5200">
98      <description>FLA housekeeping: Registry statistics</
            description>
99      <parameters>
100         <parameter name="size">
101             <description>Size of registry in bytes (excluding
                    provisional header)</description>
102             <parameterType class="common.types.parameter.
                    IntParameter">
103                 <size>2</size>
104                 <lowerLimit>0</lowerLimit>
105             </parameterType>
106         </parameter>
107         <parameter name="block_no">
108             <description> </description>
109             <parameterType class="common.types.parameter.
                    IntParameter">
110                 <size>2</size>
111                 <lowerLimit>-1</lowerLimit>
112             </parameterType>
113         </parameter>
114     </parameters>
115 </teledataType>
```

```
116     <teledataType name="DATA_UNDEF" typeID="0">
117         <description>Dummy data</description>
118     </teledataType>
119     <teledataType name="DATA_CAM_CMD_SEQ" typeID="12001">
120         <description>To be defined</description>
121         <parameters>
122             <parameter name="time_stamp">
123                 <description>Time stamp of ???</description>
124                 <parameterType class="common.types.parameter.
                        IntParameter">
125                     <size>8</size>
126                     <lowerLimit>0</lowerLimit>
127                 </parameterType>
128             </parameter>
129         </parameters>
130     </teledataType>
131     <teledataType name="DATA_CAM_RAW_SEQ" typeID="12002">
132         <description>To be defined</description>
133         <parameters>
134             <parameter name="time_stamp">
135                 <description>Time stamp of ???</description>
136                 <parameterType class="common.types.parameter.
                        IntParameter">
137                     <size>8</size>
138                     <lowerLimit>0</lowerLimit>
139                 </parameterType>
140             </parameter>
141         </parameters>
142     </teledataType>
143     <teledataType name="DATA_CAM_HK" typeID="12005">
144         <description>Camera housekeeping</description>
145         <parameters>
146             <parameter name="raw_cam_hk">
147                 <description>Raw housekeeping from PicoCam</description
                        >
148                 <parameterType class="common.types.parameter.
                        ByteArrayParameter" length="24"/>
149             </parameter>
150             <parameter name="hk_sent">
151                 <description> </description>
152                 <parameterType class="common.types.parameter.
                        IntParameter">
153                     <size>2</size>
154                     <lowerLimit>0</lowerLimit>
155                 </parameterType>
156             </parameter>
157             <parameter name="alive_sent">
158                 <description> </description>
159                 <parameterType class="common.types.parameter.
                        IntParameter">
160                     <size>2</size>
161                     <lowerLimit>0</lowerLimit>
162                 </parameterType>
163             </parameter>
164             <parameter name="number_of_packets_not_saved">
```

```
165             <description> </description>
166             <parameterType class="common.types.parameter.
                    IntParameter">
167                 <size>2</size>
168                 <lowerLimit>0</lowerLimit>
169             </parameterType>
170         </parameter>
171     </parameters>
172 </teledataType>
173 <teledataType name="DATA_CAM_IMAGE" typeID="12003">
174     <description>To be defined</description>
175     <parameters>
176         <parameter name="time_stamp">
177             <description>Time stamp of ???</description>
178             <parameterType class="common.types.parameter.
                    IntParameter">
179                 <size>8</size>
180                 <lowerLimit>0</lowerLimit>
181             </parameterType>
182         </parameter>
183     </parameters>
184 </teledataType>
185 <teledataType name="DATA_EPS_HK_FULL" typeID="9100">
186     <description>To be defined</description>
187     <parameters>
188         <parameter name="alive_sent">
189             <description> </description>
190             <parameterType class="common.types.parameter.
                    IntParameter">
191                 <size>4</size>
192                 <lowerLimit>0</lowerLimit>
193             </parameterType>
194             <sensorName>alive_sent</sensorName>
195         </parameter>
196     </parameters>
197 </teledataType>
198 </DTUsat2TeledataTypes>
```

Appendix E

Xml files for test purposes

The files defining the `CommandTypes` on the test svn server, from the test in section 6.4.

E.1 CommandTypes.xml:1

```
<DTUsat2CommandTypes>
    <version>0.0.1 (2012/5/21)</version>
    <commandType name="A" typeID="0">
        <description>test</description>
    </commandType>
    <commandType name="B" typeID="1">
        <description>test</description>
    </commandType>
</DTUsat2CommandTypes>
```

E.2 CommandTypes.xml:2

```
<DTUsat2CommandTypes>
    <version>0.0.2 (2012/5/21)</version>
    <commandType name="A" typeID="0">
        <description>testing</description>
```

```
5       </commandType>
6       <commandType name="B" typeID="1">
7           <description>test</description>
8       </commandType>
9       <commandType name="C" typeID="2">
10          <description>test</description>
11      </commandType>
12      <commandType name="D" typeID="3">
13          <description>test</description>
14      </commandType>
15  </DTUsat2CommandTypes>
```

E.3 CommandTypes.xml:3

```
1   <DTUsat2CommandTypes>
2       <version>0.0.3 (2012/5/21)</version>
3       <commandType name="A" typeID="0">
4           <description>testing</description>
5       </commandType>
6       <commandType name="B" typeID="1">
7           <description>testa</description>
8       </commandType>
9       <commandType name="C" typeID="2">
10          <description>testb</description>
11      </commandType>
12  </DTUsat2CommandTypes>
```

E.4 CommandTypes.xml:4

```
1   <DTUsat2CommandTypes>
2       <version>0.0.3 (2012/5/21)</version>
3       <fail><!-- this will make this xml file invalid -->
4   </DTUsat2CommandTypes>
```

E.5 CommandTypes.xml:5

```
1   <DTUsat2CommandTypes>
2       <version>0.0.5 (2012/5/21)</version>
3       <commandType name="A" typeID="0">
4           <description>testing</description>
5       </commandType>
6       <commandType name="B" typeID="1">
7           <description>testa</description>
8       </commandType>
9       <commandType name="C" typeID="2">
10          <description>testb</description>
11      </commandType>
12      <commandType name="D" typeID="3">
```

```
13          <description>test</description>
14      </commandType>
15  </DTUsat2CommandTypes>
```

Bibliography

[1] DTUsat-2 project, internal document. Solving the Mystery of Bird Migration: Tracking Small Birds from Space.

[2] http://docs.oracle.com/javase/1.4.2/docs/api/java/util/prefs/Preferences.html. Java Preferences in oracle's JavaDoc. Last checked 1/7-2012.

[3] http://en.wikipedia.org/wiki/Callback_(computer_programming). Wikipedia – Callback objects. Last checked 1/7-2012.

[4] http://en.wikipedia.org/wiki/CubeSat. Wikipedia – CubeSat. Last checked 1/7-2012.

[5] http://en.wikipedia.org/wiki/Functional_requirements. Wikipedia – Functional requirements. Last checked 1/7-2012.

[6] http://en.wikipedia.org/wiki/Java_Native_Interface. Wikipedia – Java Native Interface. Last checked 1/7-2012.

[7] http://en.wikipedia.org/wiki/Model_view_controller. Wikipedia – Model-view-controller. Last checked 1/7-2012.

[8] http://en.wikipedia.org/wiki/MoSCoW_Method. Wikipedia – MoSCoW method. Last checked 1/7-2012.

[9] http://en.wikipedia.org/wiki/Non Functional_Requirements. Wikipedia – Non-functional requirements. Last checked 1/7-2012.

[10] http://simple.sourceforge.net/. Website for Simple (xml-framework). Last checked 1/7-2012.

[11] http://www.dtusat.dtu.dk/. Website for the DTUsat-2 project. last checked 1/7-2012.

[12] Michael Sørensen and Mathias Holm Andersen. Bachelor thesis on DTU in 2008. Mission Planning System.