Mupus EGSE Visualization Tool Documentation V1.0 (29.11.2006)

Table of Contents

1 System Requirements

The speed of the software depends greatly of the size of the data / structure files in combination with the system's memory and processor speed. The minimal system requirements are therefore as follows:

- X86 compatible CPU with at least 500MHz
- 128MB Memory
- Windows 98 or Linux with WINE emulator
- 2MB hard drive space

Once the structure file grows and the software needs to operate in the so-called 'live' mode, the minimal requirements are not sufficient anymore because evaluation times could exceed an update cycle. To operate smoothly on live conditions with huge data and structure files, the optimal system requirements are:

- X86 compatible CPU with at least a Pentium4 equivalent 3GHz
- 1GB Memory
- Windows XP (WINE untested under heavy load and live mode)
- 2MB hard drive space

2 Feature overview

The purpose of the MUPUS EGSE software mainly consists of quick real-time data evaluation from a user-chosen data file into human readable values of interest. The user tells the software over a so-called structure file the encapsulated data arrangement of the different frame types from the flight hardware. Those data files consist of binary values that need to be structured and transformed with specific formulas that are also defined within those structure files. The structure files follow easy formatting rules and are created mostly from modified copies or directly within an arbitrary text editor.

The user has the choice to combine a particular structure file with a data file, and furthermore has the chance to track that data file periodically for changes (live-mode). Once the software finished loading both files, it is possible to open a browser and follow a structured tree-view to the desired values. The user can choose to open more browsers with different filter / sort settings for his/her convenience. With a popup menu on a certain variable, the user has the choice to:

- Set an alarm
- Plot with the option for a timed x-axis
- Filter in or out
- Sort
- Show in the original hexadecimal data

Furthermore, the software consists of a colored log window that outputs possible formatting and handling errors as well as informative texts and messages embedded in the data file. The log window can be saved to a text or rtf file.

3 Working with projects

The main window after starting the program lets the user choose to create a new project or open existing ones. Opening or creating multiple projects is possible. Within the project-setup-window, the user has an overview about the current plots and alarms, as well as general settings like structure and data files.

The project can be in two states, either offline or online. These terms refer to the connectivity to the specified structure and data files. The status is displayed on the status-bar at the bottom and can be triggered with the upper left button. A project always starts offline and the user has to provide at least a structure and a data file, plus if the user wants to start with the 'live-mode', that keeps track of changes within the data-file(s). Once the user has activated the project, it starts to load it. After that point, it is impossible to change the main settings. The user has now the chance to create browser windows by clicking on the second button. The third and fourth buttons enable / disable (not delete) all alarms.

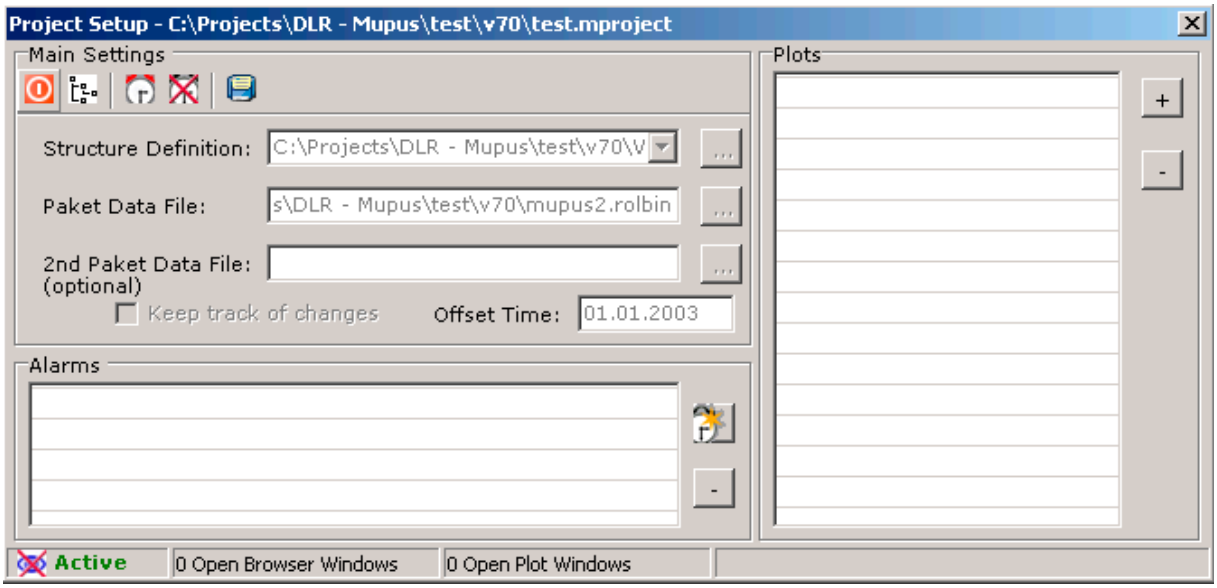

Figure 1: The project setup window provides an overview of the current state

4 Functional details

This section describes how the software works in detail, with special hard-coded functionality and flexible schemes and possible shortcut-only functionalities.

4.1 The structure file

The structure file consists of tabulator separated ASCII entries and usually ends with '.structure'. The user can choose any text editor to manipulate those files. Everything after a semicolon is considered as a comment within a line.

4.1.1 Sections

Every structure contains several sections (or containers). They describe successively following data and have to start at the beginning of a line and have the following format:

```
[Science Frame Mode:Name:Length]
```

The optional science frame mode number marks the encapsulation of that container to the 'ScienceFrame.Mode' variable. This way the software encapsulates the following definitions into the sub-frame SUB_SC within the ScienceFrame if the mode specified in the section matches the mode variable in the ScienceFrame. The length field marks the actual length in bytes of the underlying container. A special case exists for the BaseFrame, where the software tries to determine the length according to the synchronization variable. This value must be the correct block length, even if the defined variables do not reach the end.

The only exception to that is the templates section (4.1.3). If this section is required, it has to be the first section that appears within a structure file and must have a length of zero. It can only appear once and is not a data container. It follows a special format described later.

4.1.2 Variables

Every section contains of different variable types, all of them must be tabulator separated and have the following format:

```
Offset Length Type Name Function Description
```

The description is optional and can consist of an informative text with up to 1024 characters. The different parts are described in detail in the next sections.

4.1.2.1 Offset and Length

The beginning entries mark the offset within that container and their length. They have a special formatting to be bit-precise. This allows the bitwise definition for flags as well as up to 32bit integers. *Important: these definitions apply after the byte order of the input data file was changed!* A comma separates two positive integer numbers to distinguish the byte and bit part. A few examples:

```
0,1   0,5   NUMBER     my_number  my_number  A five bit number
6     0,1   BOOLEAN    my_flag    NA         A flag (one bit)
-1    -1    NUMBER     my_virtual my_number+1     A virtual var
```

The integer value after the comma is optional and ranges from zero to 31. A special case arises if both, the offset and length, are set to a negative value. In this case, the transform function cannot contain the variable name itself because it does not have an underlying variable. It is therefore called a virtual variable and can be used to build up special functions of interest from different existing variables.

The offset plus length should not exceed the length specified in the section. However, this is possible but poses the risk to read beyond the end of the file.

4.1.2.2 Variable types

The type is specified in uppercase letters and can consist of the following:

- NUMBER: The variable to read is setup as a floating-point number and is therefore transformable by any function. The initial value before transformation is always a natural number.

- HEX: This has the same properties as NUMBER, but is presented in the browser view as a hexadecimal number.
- BOOLEAN: The variable usually consists of a length of "0,1" and is translated to true and false in the browser window. No transformation can be applied to this type. However, the transform function field must be filled with an 'NA' for 'Not Applicable'.
- TEXT: Represents an ASCII string with a fixed length. If the software reads this variable, it causes to print it out on the log-window. No transformation can be applied, so the function field requires an 'NA'.
- SUBFRAME: This type encapsulates another frame type into the existing section. The name can consist of a section name (4.1.1) to direct encapsulate a frame or special names designed to dynamically link different sub-frames at this place. These names are preset in behavior. No transformation can be applied, so the function field requires an 'NA'. Please check the 'Predefined behavior' section for further information on automatisms.

4.1.2.3 Name and Transform Function

The name of a variable should be chosen to be unique among all variable names and should not contain spaces or an ampersand (&). These names can be used within the same section in the transform function field. It is not possible to use variables from different sections.

The transform function must contain at least a variable name. The function distinguishes between raw values and already transformed values. It does this by looking at the end of a variable name used in this function. If it is an ampersand, it uses the already transformed value of this variable, otherwise its raw integer value.

The user can combine several variables in arithmetic equations in that field. The parser consists of the following operators and functions:

+	:	Addition
-	:	Substraction
*	:	Multiplication
/	:	Division
^	:	Exponential (only positive numbers for the base)
(	:	Open parenthesis
)	:	Close parenthesis
MOD	:	MOD operator. Integers only. (If used with floating point numbers, it will round the values to integers)

ABS	(Range: -1e4932..1e4932)
ATAN	(Range: -1e4932..1e4932)
COS	(Range: -1e18..1e18)
EXP	(Range: -11356..11356)
LN	(Range: 0..1e4932)
ROUND	(Range: -1e9..1e9)
SIN	(Range: -1e18..1e18)
SQRT	(Range: 0..1e4932)
SQR	(Range: -1e2446..1e2446)
TRUNC	(Range: -1e9..1e9)

The following example shows the simplest behavior:

```
0     2     NUMBER     var_a      var_a
```

In this case, no transformation is applied to 'var_a', and 'var_a' is the same as 'var_a&'. The following examples show transformations that are more complex:

```
0     2     NUMBER     var_a      0.6*var_a
2     2     NUMBER     var_b      var_a&+var_b
-1    -1    NUMBER     virtual_a  var_a&*var_b&
```

This case transforms var_a to 60% of its raw value and var_b& is calculated by that transformed value plus its own value. 'virtual_a' consist now of both transformed values multiplied with each other. Functions cannot consist of their own transformed value, as well as virtual variables cannot contain their variable name at all because they do not have a raw counterpart.

4.1.3 Templates

The templates section appears in the structure file always at the beginning and is optional. The length of this section must be zero and the name is Templates.

The structure within this section is different from the rest of the structure file. The user can enter default transform functions that apply to more than one variable to save work. There are two tab-separated fields per line: the first is the name of the template and the second is the according transform function. The special identifier VAR plays the role of the variable name where it is later used. An example of a template section:

```
[Templates:0]
NONE            VAR
5V_CURRENT      VAR*0.3662/2
12V_CURRENT     VAR*0.3662/4
```

Now the user can use them within the rest of the sections in the transform function column. To use a template, it must be written in relation brackets. The software resolves the name from the templates section and replaces it with the according transform function, additionally replacing VAR with the current variable name. So the line

```
0     2     NUMBER     var_a      <NONE>     My untransformed var
```

would be the same as writing

```
0     2     NUMBER     var_a      var_a      My untransformed var
```

Care must be taken if specific variable names are used within a template entry. Although it is possible, it will not always lead to the right result since it cannot be used outside a certain section where this variable name is specified. On the other hand, combining templates with variables is possible, for example:

```
0     2     NUMBER     var_a      <5V_CURRENT>*var_b
```

4.1.4 Predefined behavior and sub-frames

To let the software determine some basic dynamic links for sub-frames, certain section and variable names must exist in the structure file in order to operate correctly.

Type	Name	Description
Section	BaseFrame	Root frame, all read operations start here
Boolean	HK_SCIENCE_SWITCH	Determines if a house-keeping frame or a science frame goes into the sub-frame SUB_HKSC
Subframe	SUB_HKSC	Depending on HK_SCIENCE_SWITCH, either the HKFrame or the ScienceFrame are encapsulated here
Section	HKFrame	See above
Section	ScienceFrame	See above
Number	MODE	Only in ScienceFrame; determines which Sub-science frame goes into SUB_SC, this is decided by the optional first number in the sections!
Subframe	SUB_SC	See above
Subframe	SUB_ADC	This resolves to 'ADC-Subframe-Type-' plus the string representation of the number SUBTYPE_ADC
Number	SUBTYPE_ADC	See above
Subframe	SUB_HEAT	This resolves to 'ADC-Subframe-Type-' plus the string representation of the number SUBTYPE_HEAT
Number	SUBTYPE_HEAT	See above

Except for the BaseFrame, all other built-in automatisms are optional. This leaves the software open to any other binary data format.

4.2 Browser

The browser acts as an overview to all data of a current project. It shows the transformed value according to an applied transform function. It offers the ability to apply filters, set alarms, define sort conditions, export data, show hex views and to create plots. All these functions are available by right-clicking (enter the popup menu) over a certain element. Browser windows can be created from the project window. Every browser window is its own class, so the user has the choice to create many browser windows with different properties.

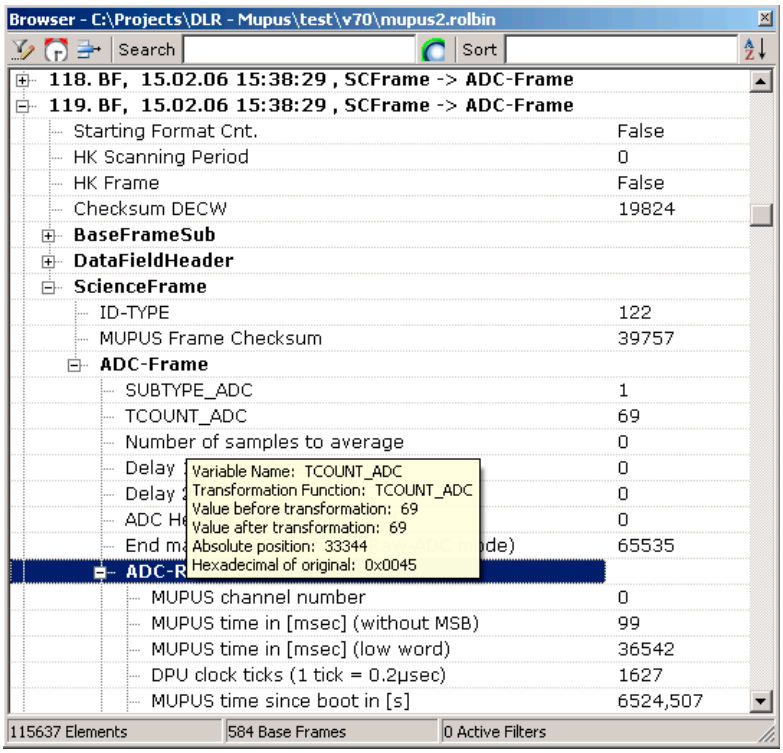

Figure 2: An example of a browser window; BF is a shortcut for BaseFrame, the popup window appears as a mouse hint on items the user hovers over. The buttons on the upper left are Filters, Alarms and export

4.2.1 Filters

Filters can be defined to either explicitly exclude or include certain BaseFrames in the current browser window. A filter is defined by the existence of the entered variable name. If the variable exists, the BaseFrame that contains the variable is either explicitly included or excluded. The user can control the filter settings of a browser window by clicking the upper left button. To create a filter, it is not necessary to enter the variable name in the filter window; instead, the user can use the popup menu to add a certain variable to the filter.

4.2.2 Export

The third button lets the user export the current browser view. To process all data later correctly, it is exported as a matrix, containing all possible variations. This way, all encapsulated sub-frames have access to the information of their parent frames on the same row. The x-axis contains all variable names used in the current browser.

The matrix is stored using the ASCII standard with semicolon-separated entries. To keep the matrix small and effective, the user should employ filters to keep unnecessary columns out.

4.3 Plots

To create a plot of a specified variable the user has two choices. The first one is to hit the ‘plus’ button on the project window where all plots are listed. The disadvantage is that the user needs to know the variable- and frame name. It is much more convenient to use the browser’s popup menu to create a plot of the currently selected variable. Sorting applies according to the browser setting. The user has the choice to zoom in a certain area by either click-and-drag within the plot or by entering the desired areas in the according edit boxes. There is also the possibility to export the currently shown data series as an ASCII file with the third button.

4.3.1 Timed Plots

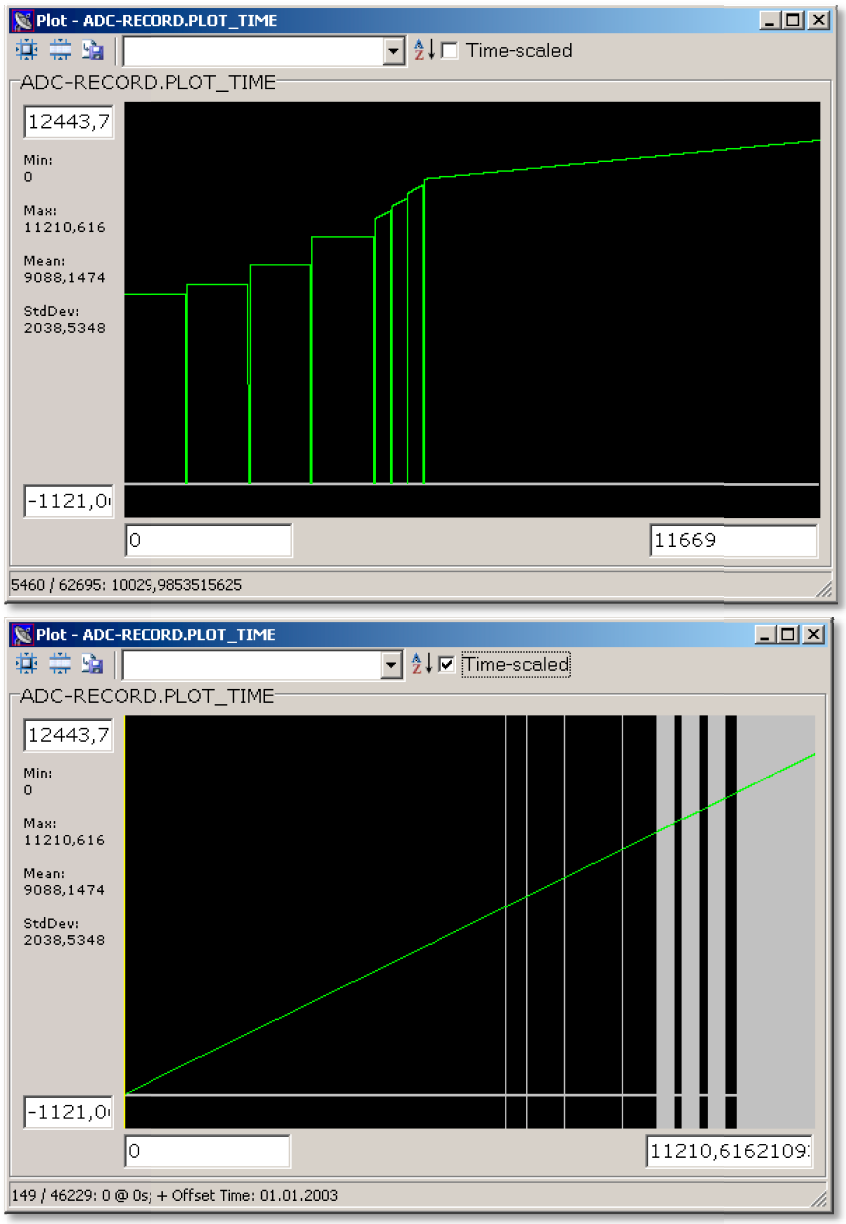

Figure 3: The upper two pictures illustrate the use of a timed plot. The first image plots the variable 'as it comes in' and the second uses itself as a time axis, which results in a straight line.

The 'Time-scaled' checkbox enables this feature. In order to use it with certain variables, the user must provide a variable called PLOT_TIME within the same frame of the desired variable. In order to function well, this variable should be virtualized and contain a function to calculate the according seconds. The plot recognizes this variable and uses it as an X-scaling with linear interpolation. The gray lines indicate an actual value at this position.

4.4 Alarms

Alarms can be used to warn the user when a certain condition on a variable applies. This makes only sense in the live mode. They can be set either from the project window or via the popup menu of the browser.

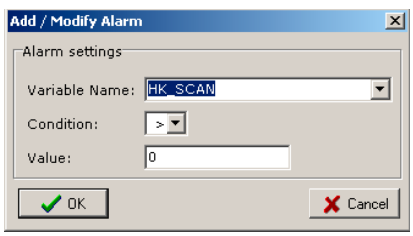

Figure 4: Example of an alarm view