

---

*$\mu$ OS* monitor  
User's manual

---

*Viresco  $\mu$ CS / Micro Control Systems*

*March 13, 2018*

---

**Contents**

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Principle of the operation . . . . .	3
1.2	Communication medium . . . . .	3
1.3	Accessing variables . . . . .	4
1.4	Available types of data . . . . .	4
1.5	Remote procedure execution . . . . .	4
1.6	Portability issues . . . . .	5
<b>2</b>	<b>Installation</b>	<b>6</b>
2.1	Installation of the $\mu OS$ monitor . . . . .	6
2.2	Preparing the destination program . . . . .	6
2.3	Concurrent access . . . . .	7
2.4	Examples . . . . .	10
<b>3</b>	<b>Configuration</b>	<b>11</b>
3.1	Configuration file corectness . . . . .	11
3.2	Available gauges . . . . .	11
3.2.1	7 segment . . . . .	11
3.2.2	Bitfield . . . . .	12
3.2.3	Clock . . . . .	12
3.2.4	Frame . . . . .	13
3.2.5	Complex plot . . . . .	13
3.2.6	Led . . . . .	14
3.2.7	Level meter . . . . .	15
3.2.8	Radar . . . . .	15
3.2.9	Terminal . . . . .	16
3.3	Editing functions . . . . .	17
<b>4</b>	<b>Command line interface</b>	<b>17</b>
4.1	Available commands . . . . .	17
4.2	Examples . . . . .	18
4.3	Values returned from the command line . . . . .	19
<b>A</b>	<b>Properties of different versions of <math>\mu OS</math> monitor</b>	<b>20</b>
<b>B</b>	<b>Revision history</b>	<b>21</b>
<b>C</b>	<b>Revision history</b>	<b>22</b>

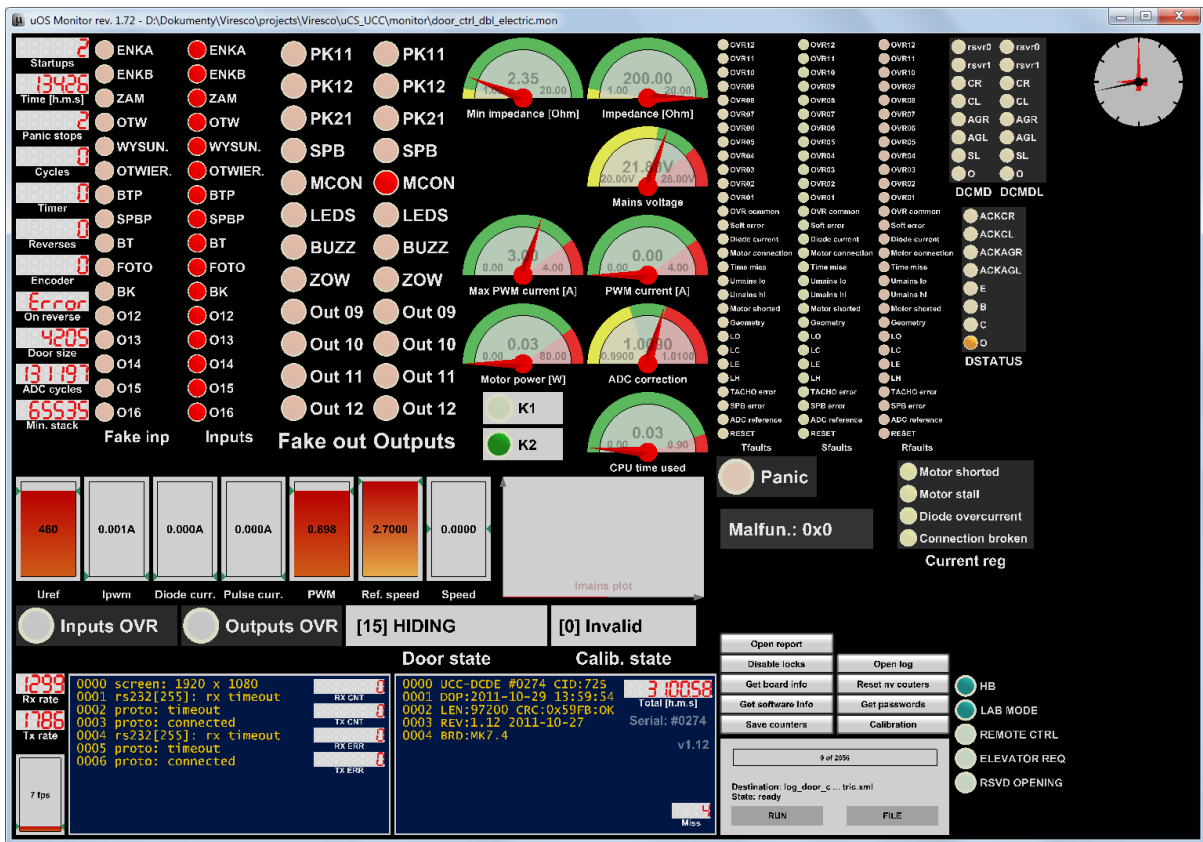
## 1 Introduction

$\mu$ OS monitor is a software used for monitoring and controlling of execution of other programw in real time. It allows building of control applications, process visualization and debugging.

### 1.1 Principle of the operation

Monitor operation is based on the data transmission between itself and the destination programs. The monitor has access to all global variables (the ones that have a fixed and known address) of the program. Each variable can be read and written. Besides the variables access, using of special functions can be utilized <sup>1</sup>.

Figure 1:  $\mu$ OS monitor running



### 1.2 Communication medium

The interaction with the destination program requires that a communication channel between the monitor and the destination program exists.

At the moment the following destination program communication methods exist:

- through the serial port - for the remote destination programs
- through pipes in Windows - for the local programs (the ones that runs on the same machine)

<sup>1</sup> based on the platform

### 1.3 Accessing variables

For the monitor to be able to access all the variables in the destination program, it has to have access to the so-called symbol map. A symbol map is a text file generated by the destination program's linker. Each compiler of any programming language is able to produce such a map.

Map example:

Listing 1: An example of the symbol map generated by the GCC (a fragment)

```

1 [131](sec 2)(fl 0x00)(ty 0)(scl 3)(nx 0) 0x00005e82 get_password
2 [132](sec 4)(fl 0x00)(ty 0)(scl 3)(nx 0) 0x00000803 inputs_old.2616
3 [133](sec 4)(fl 0x00)(ty 0)(scl 3)(nx 0) 0x00000801 outputs_old.2617
4 [134](sec 4)(fl 0x00)(ty 0)(scl 3)(nx 0) 0x000007fd voltages_old.2618
5 [135](sec 1)(fl 0x00)(ty 0)(scl 3)(nx 0) 0x000004bd init_contrast.2332
6 [136](sec -2)(fl 0x00)(ty 0)(scl 103)(nx 2) 0x00000090 param_autogen.c

```

The map file makes up the monitor's input data. The file is analyzed and a list of symbol-address pairs is generated which is used to localize the variables inside the destination program. Such procedure allows to use the original variables names in the  $\mu OS$  monitor configuration instead of artificially prepared names or numerical identifiers.

### 1.4 Available types of data

Types of data accessible in  $\mu OS$  monitor are gathered in table 1

Table 1: Data types

Name	Type	Size [bits]	Localization	Direction
data8u	natural	8	RAM	in/out
data8s	integral	8	RAM	in/out
data16u	natural	16	RAM	in/out
data16s	integral	16	RAM	in/out
data32u	natural	32	RAM	in/out
data32s	integral	32	RAM	in/out
datasingle	floating point <sup>1</sup>	32	RAM	in/out
datadouble	floating point <sup>1</sup>	64	RAM	in/out
datasz	character	N×8	RAM	in/out
terminal	special	N×8	RAM	in
eeprom	special	N×8	EEPROM <sup>2</sup>	in
callback <sup>3</sup>	special	N×8	n.d.	in

### 1.5 Remote procedure execution

Besides accessing variables, also the remote procedure execution (RPC) is possible.

! → Be carefull when using RPC, the software does not verify the procedure's address (even its alignment) !

Remote procedure execution can found to be helpfull for instance during execution of functions that cannot be run by setting the global variables or this operation is too complicated. It can also be used to execute functions "on demand".

A remotely executed procedure has to have the following prototype:

<sup>1</sup> compliant with IEEE-754 standard

<sup>2</sup> available ( $\mu OS$  monitor 1.55) for the Atmel AVR family

<sup>3</sup> execution of an internal monitor's procedure, available only for certain gauges

Listing 2: An example of remotely executed procedure

```

1  /*****
2  * RPC_FUNC_TYPE
3  *
4  * @param[in]      rx - the monitor receive buffer.
5  * @param[out]    tx - the monitor transmit buffer.
6  *
7  * @return         Status of the operation (propagated back to the host).
8  * @retval        TRUE when successful.
9  * @retval        FALSE on error.
10 *
11 * @brief         RPC function prototype.
12 *****/
13 typedef BOOLT (*RPC_FUNC_TYPE)(UOS_PROTO_BUF_TYPE *rx, UOS_PROTO_BUF_TYPE *tx);
14
15 BOOLT rpc_example(UOS_PROTO_BUF_TYPE *rx, UOS_PROTO_BUF_TYPE *tx)
16 {
17     /* rx and tx can be used for any data passed to / returned from the
18        procedure, in this example we are ignoring them. */
19     (void)rx;
20     (void)tx;
21
22     /* Any operations. */
23
24     /* In normal circumstances the procedure returns TRUE, FALSE is reserved
25        for serious errors. */
26     return (TRUE);
27 }

```

As said above, the software does not check the correctness of the procedure's address, not even its alignment. In unfavourable circumstances this can lead to fatal errors arising from the fact that an instruction at incorrect address is executed. The problem will in most cases be a consequence of losing synchronization between the symbol map (which holds the RPC procedure address) and the destination program itself.

To be able to protect ourselves from such an eventuality, one can use a special memory section (or sections) devoted especially for the RPC procedures. After placing these section at a priori defined addresses, we can be sure that the RPC execution will always produce the expected result.

We should remember that the RPC procedure is executed in the same thread the  $\mu OS$  monitor processing procedure does. Regarding this fact, if the time or resources needed to execute the given PRC are not sufficient, using an extra thread / process that is started at the time the RPC is executed should be considered.

## 1.6 Portability issues

When porting the software to other platforms, a special care should be devoted to the following  $\mu OS$  monitor features:

1. The size of a byte is not always equal to 8 bits, on some platforms a char variable can be for instance 16 bits wide. To assure compatibility of  $\mu OS$  monitor when a different byte size is used, the user should define the `BYTE8_SIZE` macro in the right way.

## 2 Installation

### 2.1 Installation of the $\mu$ OS monitor

$\mu$ OS monitor requires a PC computer running Windows 95 (Windows XP, Vista or Windows 7 highly recommended)<sup>4</sup>, an OpenGL compatible graphics card, keyboard and mouse. To use the serial port communication, the serial port itself is required (or any other device that emulates it).

The program is completely standalone, it does not use .NET or any other dependencies. The installer itself is all what is needed. Moreover, after installation the binaries can be copied on a removable drive and executed from it. This solution is already used in couple of distribution packages used by us to perform field firmware updates.

The program is equipped with a standard installer. After starting the installer, the user can choose the destination folder and optionally associate the monitor configuration file extension (\*.mon) with the  $\mu$ OS monitor.

### 2.2 Preparing the destination program

As was told in the introduction, the destination program has to be adapted to cooperate with the monitor. For that to be possible, the destination program has to have ability to call a C procedure or be able to link to a DLL (on Windows).

Servicing  $\mu$ OS monitor is conducted in two stages: a one-time communication module initialization is performed and then a cyclic communication handling subroutine execution is started. Assuming the serial port to be the communication interface and the medium access functions are defined similarly to the ones in listing 8, the initialization will look like this:

Listing 3: Destination program side initialization

```

1  #include <uart.h>
2  #include <uOS_mon.h>
3
4  int main(void)
5  {
6      /* Starts the serial interface. */
7      uart_init();
8
9      /* Configures the communication protocol. */
10     uOS_proto_init
11     (
12         uOS_comm_receive_byte, /* Byte reception procedure. */
13         uOS_comm_transmit_byte, /* Byte sending procedure. */
14         uart_flush_rx_buffer, /* RX buffer flush procedure. */
15         uart_flush_tx_buffer /* TX buffer flush procedure. */
16     );
17
18     for (;;)
19     {
20         /* Normal background activities. */
21         ...
22
23         /* uOS_monitor messages processing. */
24         uOS_mon_cmd_parse();
25     }
26 }
```

<sup>4</sup> the command line interface is not fully functional below Windows XP, also the gauge editing functions may be not accessible

### 2.3 Concurrent access

The software on the destination program side does not make any assumptions as for the system multi-tasking. Since the global data access is being conducted and other processes (threads, interrupts) can also perform access to those variables, there might be a need of excluding others ability of access the variables for the time of `uOS_mon_cmd_parse()` execution. This can be achieved in at least two ways:

Listing 4: Using the critical section

```

1  int main(void)
2  {
3      CRITICAL_SECTION cs;
4
5      for (;;)
6      {
7          /* Normal background activities. */
8          ...
9
10         /* Processing uOS_monitor messages. */
11         enter_critical_section(&cs);
12         uOS_mon_cmd_parse();
13         leave_critical_section(&cs);
14     }
15 }
```

The above procedure may not be acceptable if the time spent in the monitor servicing routine is too long (and also varying based on character reception timeout, see 8).

An alternative way of solving the concurrent access problem is to use dedicated variables devoted especially to interact with the monitor:

Listing 5: Using of dedicated variables

```

1  int global_x;
2  int global_y;
3  int mon_global_x;
4  int mon_global_y;
5
6  void mon_update(void)
7  {
8      /* Copying the read variables. */
9      mon_global_x = global_x;
10
11     /* Copying the written variables. */
12     global_y = mon_global_y;
13 }
14
15 int main(void)
16 {
17     CRITICAL_SECTION cs;
18     InitializeCriticalSection(&cs);
19
20     for (;;)
21     {
22         /* Normal background activities. */
23         ...
24
25         /* Critical section for accessing the variables. */
26         EnterCriticalSection(&cs);
27         mon_update();
28         LeaveCriticalSection(&cs);
29
30         /* Processing monitor messages. */
31         uOS_mon_cmd_parse();
32     }
33 }
```

In the above example the monitor is accessing the dedicated variables `mon_global_x` and `mon_global_y` instead of accessing directly the concurrently accessed variables `global_x` and `global_y`.

If the target environment allows to start additional processes, obviously such a process should be used to service the monitor. A simple example for Windows:

Listing 6: Using a dedicated process

```

1
2 BOOLT done = FALSE;
3 void monitor_thread(void *arg)
4 {
5     (void) arg;
6     CRITICAL_SECTION cs;
7     InitializeCriticalSection(&cs);
8
9     while (!done)
10    {
11        EnterCriticalSection(&cs);
12        /* Procesing uOS_monitor commands. */
13        uOS_mon_cmd_parse();
14        LeaveCriticalSection(&cs);
15
16        Sleep(10);
17    }
18
19    _endthread();
20 }
21
22 int main(void)
23 {
24     _beginthread(monitor_thread, 0, NULL);
25
26     Sleep(10000);
27     done = TRUE;
28     Sleep(1000);
29
30     return (0);
31 }

```

Of course in the case of using a dedicated process, the concurrent access has even bigger significance. That is why the moment of starting the monitor servicing should be chosen carefully or the dedicated monitor variables should be used as in the example above.

Listing 7: Definicja parsera danych

```

1 typedef BOOLT (*UCS_BINARY_PARSER_TYPE)
2     (
3     char *outfile,
4     char *data,
5     UINT32T len
6     );

```



Listing 8: Sample implementation of the communication wrappers

```

1  #include <uOS_proto.h>
2  #include <uOS_mon.h>
3
4  /*****
5  * uOS_comm_transmit_byte
6  *
7  * @param[in]      dst - not used.
8  * @param[in]      byte - byte to be sent.
9  * @param[out]     -
10 * @param[in, out] -
11 *
12 * @return          UOS_STATUS_OK, always.
13 *
14 * @brief           Transmit byte wrapper.
15 *****/
16 UOS_STATUS uOS_comm_transmit_byte(UINT8T dst, UINT8T byte)
17 {
18     (void)dst;
19     uart_transmit_byte(byte);
20
21     return (UOS_STATUS_OK);
22 }
23
24 /*****
25 * uOS_comm_receive_byte
26 *
27 * @param[in]      src - not used.
28 * @param[in]      byte - where to place the result.
29 * @param[out]     -
30 * @param[in, out] -
31 *
32 * @return          UOS_STATUS_OK when a byte was received.
33 * @return          UOS_STATUS_ETIMEOUT when a timeout occurs.
34 *
35 * @brief           Receive byte wrapper.
36 *****/
37 UOS_STATUS uOS_comm_receive_byte(UINT8T src, UINT8T *byte)
38 {
39     unsigned int timeout=100;
40     (void)src;
41
42     do
43     {
44         if (uart_data_in_rx_buffer())
45         {
46             *byte=uart_receive_byte();
47             return (UOS_STATUS_OK);
48         }
49
50         delay_ms(1);
51     }
52     while (timeout-- != 0);
53
54     return (UOS_STATUS_ETIMEOUT);
55 }

```

## 2.4 Examples

Listing 9 contains the code used to create the demo pipe program bundled with the installer.

Listing 9: Pipe demo implementation

```

1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <conio.h>
4  #include "defs.h"
5  #include "uOS_hm.h"
6  #include "uOS_proto.h"
7  #include "uOS_mon.h"
8  #include "named_pipe.h"
9
10  UINT32T mon_cnt;
11  UINT8T mon_lamp, old_lamp;
12  UINT8T MY_MODULE_ADDR = 254;
13
14  static UOS_STATUS uOS_rx_byte(UINT8T src, UINT8T *byte)
15  {
16      return (uOS_pipe_receive_byte(0, byte));
17  }
18
19  static UOS_STATUS uOS_tx_byte(UINT8T dst, UINT8T byte)
20  {
21      return (uOS_pipe_transmit_byte(0, byte));
22  }
23
24  int main(int argc, char *argv[])
25  {
26      uOS_hm_install_logger((UOS_HMLOG_METHOD)printf);
27      uOS_proto_init(uOS_rx_byte, uOS_tx_byte, uOS_pipe_flush_rx,
28                    uOS_pipe_flush_tx);
29      uOS_hm_log("Start...\n");
30
31      if (argc < 2)
32      {
33          pipe_conf[0].name = "PIPE1";
34      }
35      else
36      {
37          pipe_conf[0].name = argv[1];
38      }
39
40      pipe_conf[0].timeout = 500;
41      pipe_conf[0].server = TRUE;
42
43      if (uOS_pipe_open(0) != UOS_STATUS_OK)
44      {
45          uOS_hm_log("Cannot open pipe\n");
46          return (-1);
47      }
48
49      while(!kbhit())
50      {
51          mon_cnt++;
52          if (uOS_mon_cmd_parse() == UOS_STATUS_OK)
53          {
54              if (old_lamp != mon_lamp)
55              {
56                  printf("lamp: %d\n", (int)mon_lamp);
57                  old_lamp = mon_lamp;
58              }
59          }
60      }

```

```

61     uOS_pipe_close (0);
62
63
64     return (0);
65 }

```

### 3 Configuration

All configuration data resides in a XML configuration file. The default file extension is (\*.mon). The installer program can automatically associate this type of files with  $\mu OS$  monitor if the user accepts this.

#### 3.1 Configuration file corectness

The program is equipped with a two-stage configuration file verification mechanism:

1. By means of an embedded XML Schema;
2. By additional correctness checks performed after loading the configuration file and performing the XML Schema validation.

#### 3.2 Available gauges

All gauges available in  $\mu OS$  monitor are defined by thier XSD model. Below you can find a briess description of the gauges along with their XSD specification.

##### 3.2.1 7 segment



Figure 2: 7 segment

Inspired by a 7-segment display. It supports a special format %t which displays time in HH.MM.SS form.

Listing 10: 7 segment gauge model

```

1 <xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
   gauge_7_segment_type">
2 <xsd:all>
3 <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
4 <xsd:element name="name" type="xsd:string"/>
5 <xsd:element name="target" type="target_type"/>
6 <xsd:element name="title" type="xsd:string" minOccurs="0"/>
7 <xsd:element name="x" type="xsd:double"/>
8 <xsd:element name="y" type="xsd:double"/>
9 <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
10 <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
11 <xsd:element name="format" type="xsd:string" minOccurs="0"/>
12 <xsd:element name="digits" type="xsd:positiveInteger"/>
13 <xsd:element name="height" type="xsd:double" minOccurs="0"/>
14 <xsd:element name="width" type="xsd:double" minOccurs="0"/>
15 <xsd:element name="argument" type="xsd:string" minOccurs="0"/>
16 <xsd:element name="actual" type="xsd:double" minOccurs="0"/>
17 <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
18 <xsd:element name="fgcolor" type="color_type" minOccurs="0"/>
19 </xsd:all>
20 </xsd:complexType>

```

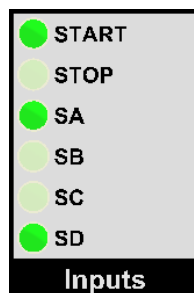


Figure 3: Bitfield

### 3.2.2 Bitfield

A gauge that displays an integer number decomposing it into particular bits. It is possible to set / clear particular bits in output mode (through mouse click).

Listing 11: bitfield gauge model

```

1 <xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
   gauge_bitfield_type">
2 <xsd:all>
3 <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
4 <xsd:element name="name" type="xsd:string"/>
5 <xsd:element name="target" type="target_type"/>
6 <xsd:element name="title" type="xsd:string"/>
7 <xsd:element name="visource" type="xsd:string" minOccurs="0"/>
8 <xsd:element name="labels" type="labels_type"/>
9 <xsd:element name="bits">
10 <xsd:simpleType>
11 <xsd:restriction base="xsd:positiveInteger">
12 <xsd:minInclusive value="1"/></xsd:minInclusive>
13 <xsd:maxInclusive value="32"/></xsd:maxInclusive>
14 </xsd:restriction>
15 </xsd:simpleType>
16 </xsd:element>
17 <xsd:element name="x" type="xsd:double"/>
18 <xsd:element name="y" type="xsd:double"/>
19 <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
20 <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
21 <xsd:element name="width" type="xsd:double" minOccurs="0"/>
22 <xsd:element name="height" type="xsd:double" minOccurs="0"/>
23 <xsd:element name="actual" type="xsd:unsignedInt" minOccurs="0"/>
24 <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
25 <!-- /color is left for backward compatibility, it should be replaced by /fgcolor
   -->
26 <xsd:element name="color" type="xsd:string" minOccurs="0"/>
27 <xsd:element name="txcolor" type="color_type" minOccurs="0"/>
28 <xsd:element name="fgcolor" type="color_type" minOccurs="0"/>
29 <xsd:element name="bgcolor" type="color_type" minOccurs="0"/>
30 <xsd:element name="argument" type="xsd:string" minOccurs="0"/>
31 <xsd:element name="type" minOccurs="0"/>
32 <xsd:simpleType>
33 <xsd:restriction base="xsd:string">
34 <xsd:enumeration value="left aligned"/></xsd:enumeration>
35 <xsd:enumeration value="right aligned"/></xsd:enumeration>
36 <xsd:enumeration value="square left aligned"/></xsd:enumeration>
37 <xsd:enumeration value="square right aligned"/></xsd:enumeration>
38 </xsd:restriction>
39 </xsd:simpleType>
40 </xsd:element>
41 </xsd:all>
42 </xsd:complexType>

```

### 3.2.3 Clock

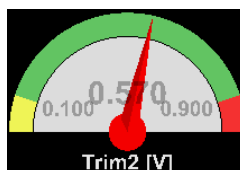


Figure 4: Clock

Classic 180° clock. Allows for setting upper and lower alarm level.

Listing 12: clock gauge model

```

1 <xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
  gauge_clock_type">
2 <xsd:all>
3 <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
4 <xsd:element name="name" type="xsd:string"/>
5 <xsd:element name="target" type="target_type"/>
6 <xsd:element name="title" type="xsd:string" minOccurs="0"/>
7 <xsd:element name="x" type="xsd:double"/>
8 <xsd:element name="y" type="xsd:double"/>
9 <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
10 <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
11 <xsd:element name="format" type="xsd:string"/>
12 <xsd:element name="min" type="xsd:double" minOccurs="0"/>
13 <xsd:element name="max" type="xsd:double" minOccurs="0"/>
14 <xsd:element name="hi" type="xsd:double" minOccurs="0"/>
15 <xsd:element name="lo" type="xsd:double" minOccurs="0"/>
16 <xsd:element name="type">
17 <xsd:simpleType>
18 <xsd:restriction base="xsd:string">
19 <xsd:enumeration value="180"/></xsd:enumeration>
20 </xsd:restriction>
21 </xsd:simpleType>
22 </xsd:element>
23 <xsd:element name="width" type="xsd:double" minOccurs="0"/>
24 <xsd:element name="height" type="xsd:double" minOccurs="0" fixed="0"/>
25 <xsd:element name="actual" type="xsd:double" minOccurs="0"/>
26 <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
27 <xsd:element name="fgcolor" type="color_type" minOccurs="0"/>
28 </xsd:all>
29 </xsd:complexType>

```

### 3.2.4 Frame

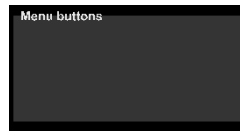


Figure 5: Clock

Frame used for elements grouping.

Listing 13: frame gauge model

```

1 <xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
  gauge_frame_type">
2 <xsd:all>
3 <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
4 <xsd:element name="name" type="xsd:string"/>
5 <xsd:element name="title" type="xsd:string" minOccurs="0"/>
6 <xsd:element name="actual" type="xsd:string" minOccurs="0"/>
7 <xsd:element name="target" type="target_type"/>
8 <xsd:element name="x" type="xsd:double"/>
9 <xsd:element name="y" type="xsd:double"/>
10 <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
11 <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
12 <xsd:element name="width" type="xsd:double" minOccurs="0"/>
13 <xsd:element name="height" type="xsd:double" minOccurs="0"/>
14 <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
15 <xsd:element name="format" type="xsd:string"/>
16 <xsd:element name="txcolor" type="color_type" minOccurs="0"/>
17 <xsd:element name="fgcolor" type="color_type" minOccurs="0"/>
18 <xsd:element name="bgcolor" type="color_type" minOccurs="0"/>
19 <xsd:element name="type" minOccurs="0">
20 <xsd:simpleType>
21 <xsd:restriction base="xsd:string">
22 <xsd:enumeration value="left aligned"/></xsd:enumeration>
23 <xsd:enumeration value="right aligned"/></xsd:enumeration>
24 <xsd:enumeration value="center aligned"/></xsd:enumeration>
25 </xsd:restriction>
26 </xsd:simpleType>
27 </xsd:element>
28 </xsd:all>
29 </xsd:complexType>

```

### 3.2.5 Complex plot

Vector plot, shows two numbers in form of vector (like e.g. complex numbers).

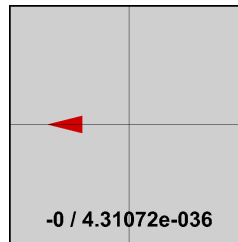


Figure 6: Complex plot

Listing 14: plot cplx gauge model

```

1 <xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
  gauge_plot_cplx_type">
2   <xsd:all>
3     <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
4     <xsd:element name="name" type="xsd:string"/>
5     <xsd:element name="title" type="xsd:string" minOccurs="0"/>
6     <xsd:element name="target" type="target_type"/>
7     <xsd:element name="x" type="xsd:double"/>
8     <xsd:element name="y" type="xsd:double"/>
9     <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
10    <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
11    <xsd:element name="max" type="xsd:double" minOccurs="0"/>
12    <xsd:element name="width" type="xsd:double" minOccurs="0"/>
13    <xsd:element name="height" type="xsd:double" minOccurs="0" fixed="0"/>
14    <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
15    <xsd:element name="format" type="xsd:string"/>
16    <xsd:element name="txcolor" type="color_type" minOccurs="0"/>
17    <xsd:element name="fgcolor" type="color_type" minOccurs="0"/>
18    <xsd:element name="bgcolor" type="color_type" minOccurs="0"/>
19  </xsd:all>
20 </xsd:complexType>

```

### 3.2.6 Led

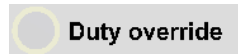


Figure 7: Led

A LED lamp. Its behavior is similar to the *Bitfield*. Possible use as output (through mouse click).

Listing 15: led gauge model

```

1 <xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="gauge_led_type"
  >
2   <xsd:all>
3     <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
4     <xsd:element name="name" type="xsd:string"/>
5     <xsd:element name="target" type="target_type"/>
6     <xsd:element name="title" type="xsd:string"/>
7     <xsd:element name="x" type="xsd:double"/>
8     <xsd:element name="y" type="xsd:double"/>
9     <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
10    <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
11    <xsd:element name="width" type="xsd:double" minOccurs="0"/>
12    <xsd:element name="height" type="xsd:double" minOccurs="0"/>
13    <xsd:element name="actual" type="xsd:double" minOccurs="0"/>
14    <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
15    <!-- /color is left for backward compatibility, it should be replaced by fgcolor
  -->
16    <xsd:element name="color" type="xsd:string" minOccurs="0"/>
17    <xsd:element name="txcolor" type="color_type" minOccurs="0"/>
18    <xsd:element name="fgcolor" type="color_type" minOccurs="0"/>
19    <xsd:element name="bgcolor" type="color_type" minOccurs="0"/>
20    <xsd:element name="type" minOccurs="0">
21      <xsd:simpleType>
22        <xsd:restriction base="xsd:string">
23          <xsd:enumeration value="left aligned"></xsd:enumeration>
24          <xsd:enumeration value="right aligned"></xsd:enumeration>
25          <xsd:enumeration value="square left aligned"></xsd:enumeration>
26          <xsd:enumeration value="square right aligned"></xsd:enumeration>
27        </xsd:restriction>
28      </xsd:simpleType>
29    </xsd:element>
30  </xsd:all>
31 </xsd:complexType>

```

3.2.7 Level meter

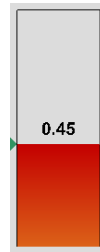


Figure 8: Level meter

This type of gauge visualisates level. It is possible to set origin point (a zero level) different that the minimal level. One can set the output value for this gauge using mouse click or mouse wheel.

Listing 16: level meter gauge model

```

1 <xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
  gauge_level_meter_type">
2 <xsd:all>
3 <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
4 <xsd:element name="name" type="xsd:string"/>
5 <xsd:element name="target" type="target_type"/>
6 <xsd:element name="title" type="xsd:string" minOccurs="0"/>
7 <xsd:element name="visource" type="xsd:string" minOccurs="0"/>
8 <xsd:element name="x" type="xsd:double"/>
9 <xsd:element name="y" type="xsd:double"/>
10 <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
11 <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
12 <xsd:element name="format" type="xsd:string"/>
13 <xsd:element name="inc" type="inc_type" minOccurs="0"/>
14 <xsd:element name="width" type="xsd:double" minOccurs="0"/>
15 <xsd:element name="height" type="xsd:double" minOccurs="0"/>
16 <xsd:element name="min" type="xsd:double" minOccurs="0"/>
17 <xsd:element name="max" type="xsd:double" minOccurs="0"/>
18 <xsd:element name="origin" type="xsd:double" minOccurs="0"/>
19 <xsd:element name="actual" type="xsd:double" minOccurs="0"/>
20 <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
21 <xsd:element name="argument" type="xsd:string" minOccurs="0"/>
22 <xsd:element name="type" minOccurs="0"/>
23 <xsd:simpleType>
24 <xsd:restriction base="xsd:string">
25 <xsd:enumeration value="vertical"/></xsd:enumeration>
26 <xsd:enumeration value="horizontal"/></xsd:enumeration>
27 <xsd:enumeration value="horizontal left aligned"/></xsd:enumeration>
28 </xsd:restriction>
29 </xsd:simpleType>
30 </xsd:element>
31 </xsd:all>
32 </xsd:complexType>
  
```

3.2.8 Radar

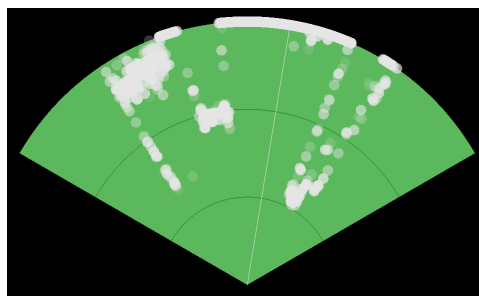


Figure 9: Radar

As the name suggests, this is a visualization of a distance meter mounted on a rotating head. Displaying the history of measurements is achieved by fading of the older points: the newest point has full color, the oldest is almost invisible. The number of points to display is configurable.

Listing 17: radar gauge model

```

1 <xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
  gauge_radar_type">
2 <xsd:all>
3 <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
4 <xsd:element name="name" type="xsd:string"/>
5 <xsd:element name="target" type="target_type"/>
6 <xsd:element name="title" type="xsd:string"/>
7 <xsd:element name="x" type="xsd:double"/>
8 <xsd:element name="y" type="xsd:double"/>
9 <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
10 <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
11 <xsd:element name="width" type="xsd:double" minOccurs="0"/>
12 <xsd:element name="height" type="xsd:double" minOccurs="0"/>
13 <xsd:element name="max" type="xsd:double"/>
14 <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
15 <xsd:element name="points" type="xsd:unsignedInt" minOccurs="0"/>
16 <xsd:element name="type" minOccurs="0">
17 <xsd:simpleType>
18 <xsd:restriction base="xsd:string">
19 <xsd:enumeration value="120"></xsd:enumeration>
20 </xsd:restriction>
21 </xsd:simpleType>
22 </xsd:element>
23 </xsd:all>
24 </xsd:complexType>

```

### 3.2.9 Terminal

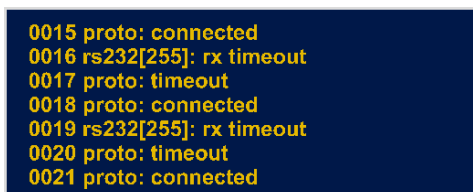


Figure 10: Terminal

Allows simulating of a console. For the destination program to be able to use it, it has to utilize a special data structure and special terminal writing routines (FIFO queueing). Terminal can also be used to interface the internal diagnostics mechanism of the  $\mu OS$  monitor itself. It displays the program's messages (as shown on the terminal picture).

Listing 18: terminal gauge model

```

1 <xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
  gauge_terminal_type">
2 <xsd:all>
3 <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
4 <xsd:element name="name" type="xsd:string"/>
5 <xsd:element name="target" type="target_type"/>
6 <xsd:element name="title" type="xsd:string" minOccurs="0"/>
7 <xsd:element name="x" type="xsd:double"/>
8 <xsd:element name="y" type="xsd:double"/>
9 <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
10 <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
11 <xsd:element name="height" type="xsd:double" minOccurs="0"/>
12 <xsd:element name="width" type="xsd:double" minOccurs="0"/>
13 <xsd:element name="log_file" type="xsd:anyURI" minOccurs="0"/>
14 <xsd:element name="txcolor" type="color_type" minOccurs="0"/>
15 <xsd:element name="prefix" minOccurs="0">
16 <xsd:simpleType>
17 <xsd:restriction base="xsd:string">
18 <xsd:enumeration value="index"></xsd:enumeration>
19 <xsd:enumeration value="time"></xsd:enumeration>
20 </xsd:restriction>
21 </xsd:simpleType>
22 </xsd:element>
23 </xsd:all>
24 </xsd:complexType>

```

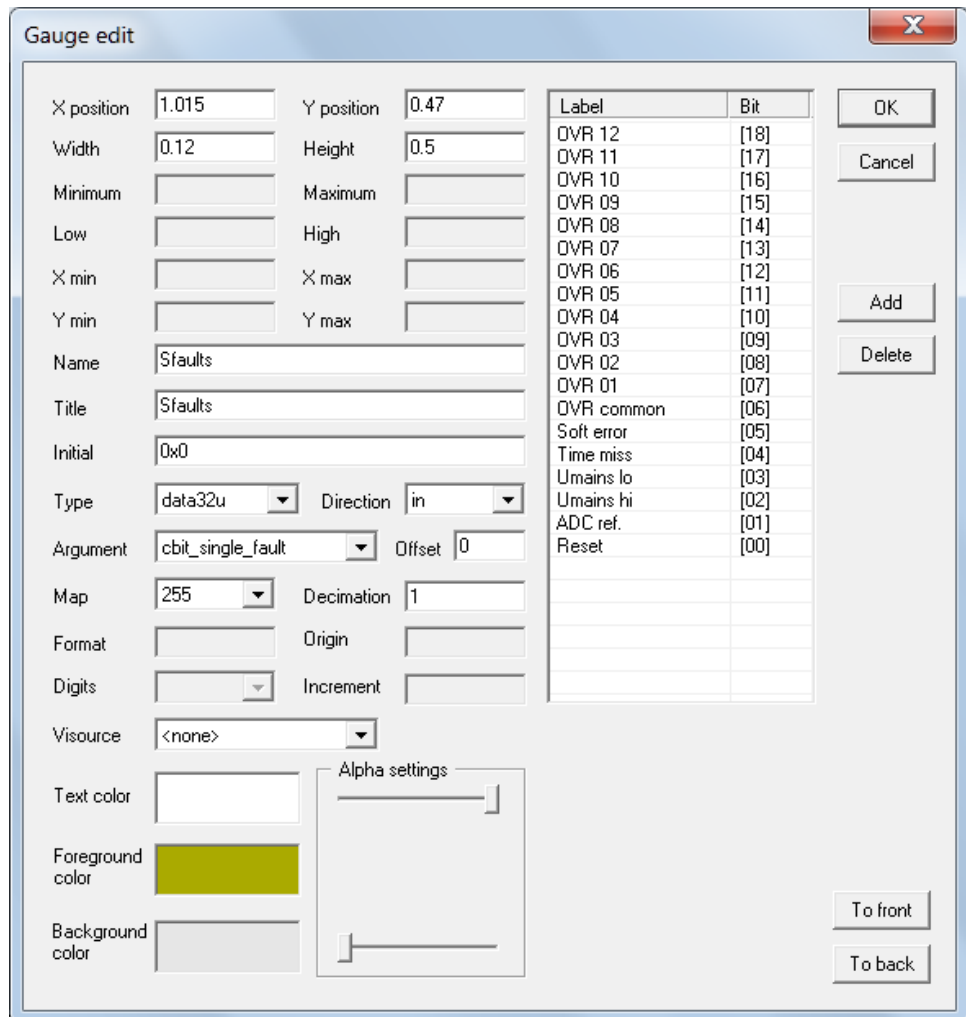


### 3.3 Editing functions

Program is equipped with some editing functions allowing changing of parameters of the particular gauges directly from the program window.

In the current  $\mu OS$  monitor 1.55, it is also possible to create new gauges and copy existing ones.

Figure 11: Editing properties of an enumerator gauge



## 4 Command line interface

The program has a command line interface allowing automation of tasks such as testing or long term acquisition. The command line interface gives the user possibility to access the object defined in the configuration file. The results of the operations are sent in textual form on the standard output (stdout). Description of the command line interface is shown after typing:

```
1 uOS_monitor.exe --help
```

### 4.1 Available commands

At this moment ( $\mu OS$  monitor 1.55) the following commands are available:

- `-h, --help` - prints help;
- `-f, --fullscreen` - runs the program in full screen mode;
- `-r, --redirect` - allows redirecting of the output (for instance using the DOS `>` operator); in normal circumstances the program allocates its own console which makes redirecting impossible;
- `-n, --name` - selects the name of the object that is to be accessed (reading / writing);
- `--geti` - reads an integral number;
- `--getf` - reads a floating point number;
- `--gets` - reads a character array (string);
- `--seti` - writes an integral number;
- `-i, --idof` - reads the destination device identifier;
- `--rpc` - executes remote procedure named `{name}`;
- `--rpc-arg=<string>` - remote procedure argument (optional);
- `--map-dump` - print symbols maps contents and exit.

! → It should be noted that only one `geti` / `getf` / `gets` / `seti` command type can be used at a time. However, the same command can be used multiple times in a single invocation (up to 100 times). For instance, the following invocation:

```
1 uOS_monitor.exe --geti --name="clock" --name="counter" config
   .mon
```

hub

```
1 uOS_monitor.exe --geti -n "clock" -n "counter" config.mon
```

will perform the “clock” and “counter” objects reading and will display them in the order of their occurrence in the command:

```
1 4428.000000
2 3.000000
```

## 4.2 Examples

Let us suppose we have a configuration file the contains a control allowing access to the device operating time:

```
1 <gauge_7_segment>
2 <name>clock</name>
3 <x>0.685000</x>
4 <y>0.930000</y>
5 <width>0.015</width>
6 <digits>6</digits>
7 <decimation>1</decimation>
8 <format>%t</format>
9 <title>Time [h.m.s]</title>
10 <target
11   dir="in"
12   type="data32u"
13   arg="mon_one_second_ticks"
14   map="255" />
15 </gauge_7_segment>
```

Reading its value is possible using the following command:

```
1 uOS_monitor.exe --geti --name="clock" config.mon
```

or

```
1 uOS_monitor.exe --geti -n "clock" config.mon
```

on the other hand, setting the object's value can be accomplished using:

```
1 uOS_monitor.exe --seti=100 --name="clock" config.mon
```

or

```
1 uOS_monitor.exe --seti=100 -n "clock" config.mon
```

where:

- `geti` is the access command;
- `name` defines the object name (same as the one defined by the `<name>` element);
- `tester.mon` is the name of the configuration file.

After execution the demanded value is printed on the console (through the standard output stream).

### 4.3 Values returned from the command line

Successful execution of the command line invocation returns the value of 0. Any error encountered results in a negative value.

## A Properties of different versions of $\mu OS$ monitor

Property	Demo version	Standard version	Professional version
Polish and English documentation	✓	✓	✓
Target software binaries for Windows, AVR, ARM, PIC	✓	✓	✓
Command line interface	✓	✓	✓
The $\mu OS$ monitor DLL library to be used in different programs	✓	✓	✓
Unlimited gauges	✗	✓	✓
Target software source files	✗	✓	✓
$\mu OS$ monitor wrapper for Octave (*.oct)	✗	✗	✓
$\mu OS$ monitor wrapper for Python (*.py)	✗	✗	✓
$\mu OS$ monitor wrapper for C# (*.cs)	✗	✗	✓

## B Revision history

Version	Date	Change descirpion
<b>25</b> <sup>(r3619)</sup>	2018-03-01	Adding support for MATES mod.
<b>24</b> <sup>(r3579)</sup>	2018-02-17	Updated copyright.
<b>23</b> <sup>(r2487)</sup>	2016-08-24	Added limiting of the drawing frequency [resolves #21].
<b>22</b> <sup>(r2245)</sup>	2016-03-12	Added releasing of CPU when there are no gauges on the list.
<b>21</b> <sup>(r2228)</sup>	2016-03-03	Merged mt_support branch.
<b>20</b> <sup>(r2181)</sup>	2016-02-18	Corrected version setting [resolves #19].
<b>19</b> <sup>(r2115)</sup>	2016-02-05	Corected coordinates display [resolves #18].
<b>18</b> <sup>(r1941)</sup>	2016-01-05	Completed implementation of -logger-csv option.
<b>17</b> <sup>(r1938)</sup>	2016-01-04	Added support for -logger-csv option.
<b>16</b> <sup>(r1907)</sup>	2015-12-23	Corrected initial element of the height value for cplx gauge [resolves #13].
<b>15</b> <sup>(r1906)</sup>	2015-12-23	Added missing Critical Section initialization when copying gauges [resolves #6].
<b>14</b> <sup>(r1782)</sup>	2015-09-08	Changed handling of UOS.MON_WM.NEED_REPAINT to improve drawing performance.
<b>13</b> <sup>(r1780)</sup>	2015-09-08	Corrected mouse dragging problem [resolves #10].
<b>12</b> <sup>(r1306)</sup>	2014-10-09	Added handling of NaN for level meter and clock gauge.
<b>11</b> <sup>(r1285)</sup>	2014-10-05	Completed porting to SVN.
<b>10</b> <sup>(r1021)</sup>	2014-03-11	Monitor file name is displayed before program name in the window title.
<b>9</b> <sup>(r1005)</sup>	2014-02-25	Added code to improve responsiveness when e.g. the USB-serial dongle is disconnected from USB.
<b>8</b> <sup>(r1002)</sup>	2014-02-24	Added some more synchronization, made sure that communication will stop before editor is enabled [addresses #6]; added deleting of gauges's critical sections.
<b>7</b> <sup>(r999)</sup>	2014-02-22	Corrected the message pumping when and increased the default resolution.
<b>6</b> <sup>(r998)</sup>	2014-02-22	The communication is not processed when the editor is enabled.
<b>5</b> <sup>(r997)</sup>	2014-02-22	Reduced CPU consumption by calling WaitMessage().
<b>4</b> <sup>(r996)</sup>	2014-02-22	Moved gauge_list to env [see #2].
<b>3</b> <sup>(r990)</sup>	2014-02-21	The communication is done in separate thread [resolves #1 and #2].
<b>2</b> <sup>(r972)</sup>	2014-02-19	Moved uart_conf to env [see #2].
<b>1</b> <sup>(r963)</sup>	2014-02-15	Initial revision.

## C Revision history

Version	Date	Change description
<b>1.105</b>	<i>2014-01-14</i>	Allow embedded map to be empty.
<b>1.104</b>	<i>2013-11-18</i>	Documentation.
<b>1.103</b>	<i>2013-11-17</i>	Refactored to support memory management.
<b>1.102</b>	<i>2013-11-10</i>	Following changes: - removed redundant warning when not used channel is closed; - reworked the build procedure to use parallel build.
<b>1.101</b>	<i>2013-11-06</i>	Following changes: - one more correction to gauge selection mechanism; - added version resource for the DLL.
<b>1.100</b>	<i>2013-10-25</i>	Corrected the case when DLL is attached second time.
<b>1.99</b>	<i>2013-09-12</i>	Rounding to editor grid is done globally just after mouse event (prevents from slipping when dragging).
<b>1.98</b>	<i>2013-09-11</i>	Added gauge refreshing after its editing (decimation counter reset).
<b>1.97</b>	<i>2013-09-11</i>	Fixed behavior when block selection or multi-selection is made.
<b>1.96</b>	<i>2013-07-03</i>	Corrected the case when DLL error occurred before call to <code>dll_monitor_init()</code> .
<b>1.95</b>	<i>2013-06-20</i>	Added support for OEM United States encoding (437).
<b>1.94</b>	<i>2013-05-21</i>	Corrected program crash when <code>dll_monitor_finit()</code> is called multiple times.
<b>1.93</b>	<i>2013-04-27</i>	Improved the message shown when the configuration file does not exist.
<b>1.92</b>	<i>2013-04-03</i>	Error is now returned for DLL build and serial connection error.
<b>1.91</b>	<i>2013-02-02</i>	Improved debug output when using DLL without initializing.
<b>1.90</b>	<i>2012-12-15</i>	Added <code>gauge_frame</code> .
<b>1.89</b>	<i>2012-12-07</i>	Following changes: - optimized gauge validation; - editor grid is less visible; - corrected tab order in gauge edit dialog; - added moving gauges using keyboard.
<b>1.88</b>	<i>2012-11-04</i>	Corrected to properly build DLL after recent changes.
<b>1.87</b>	<i>2012-11-04</i>	Following changes: - added <code>-version</code> command line option; - mouse cursor changes accordingly to the gauge that has focus; - corrected mouse value drag action for horizontal level meter.
<b>1.86</b>	<i>2012-10-20</i>	Added support for "initialized" target direction for LED gauge.
<b>1.85</b>	<i>2012-10-20</i>	Added support for embedded symbol maps.
<b>1.84</b>	<i>2012-09-29</i>	Added missing query deletion when deleting a gauge.
<b>1.83</b>	<i>2012-09-09</i>	Fixed the bug when scene was drawn once with <code>env.quadratic</code> equal to <code>NULL</code> as a result of <code>WM.SIZE</code> message (found the bug when testing on Windows XP).
<b>1.82</b>	<i>2012-07-02</i>	Following changes: - added Standard build configuration; - added <code>cmd_pull_term()</code> ; - fixed double freeing bug in <code>load_map()</code> ; - documentation of the python wrapper.
<b>1.81</b>	<i>2012-06-29</i>	Added support for the "override" command line option.
<b>1.80</b>	<i>2012-05-26</i>	Added API documentation in CHM format.
<b>1.79</b>	<i>2012-05-21</i>	Added displaying of the program release type.
<b>1.78</b>	<i>2012-05-12</i>	Corrected window resizing when exiting editor mode.
<b>1.77</b>	<i>2012-04-02</i>	Corrected the way the configuration file name and the window title are handled.
<b>1.76</b>	<i>2012-03-29</i>	Added <code>-maximized</code> command line option.
<b>1.75</b>	<i>2012-03-16</i>	Added force update on mouse event.

Version	Date	Change description
<b>1.74</b>	<i>2012-03-14</i>	Corrected repetitive paste and double click behavior.
<b>1.73</b>	<i>2012-03-02</i>	Initial support for multi - selection in editor.
<b>1.72</b>	<i>2012-02-07</i>	Added support for gauges subtypes.
<b>1.71</b>	<i>2012-01-19</i>	Initial support for the system callbacks.
<b>1.70</b>	<i>2012-01-13</i>	Added closing of the iconv encoders.
<b>1.69</b>	<i>2012-01-11</i>	Multiple changes: - added support for the gauge subtype; - horizontal subtype for the level meter; - color properties for button and 7 segment display; - string data type for 7 segment display; - the iterators are executed on each execution of the main loop.
<b>1.68</b>	<i>2012-01-06</i>	Initial support for Polish diacritics.
<b>1.67</b>	<i>2011-12-19</i>	Added handling of the USE_CLOSEHANDLE_ERRATA.
<b>1.66</b>	<i>2011-12-13</i>	Following changes: - the console is freed when it was actually allocated; - the gauge data is freed also for the command line mode.
<b>1.65</b>	<i>2011-11-30</i>	Added password dialog box.
<b>1.64</b>	<i>2011-10-24</i>	Added support for the revof command.
<b>1.63</b>	<i>2011-10-12</i>	Many minor editor corrections.
<b>1.62</b>	<i>2011-09-28</i>	Initial support for block transfers (for Bluetooth).
<b>1.61</b>	<i>2011-08-09</i>	Added -load option.
<b>1.60</b>	<i>2011-07-09</i>	Corrected closing communication channels.
<b>1.59</b>	<i>2011-06-15</i>	MY_MODULE_ADDR not declared here.
<b>1.58</b>	<i>2011-05-27</i>	First version with pipe communication support.
<b>1.57</b>	<i>2011-05-13</i>	Command line documentation, support for gauge_radar.
<b>1.56</b>	<i>2011-04-15</i>	Preparations for sockets implementation.
<b>1.55</b>	<i>2011-04-13</i>	Output is printed in case of RPC failure.
<b>1.54</b>	<i>2011-03-21</i>	Corrections related with the channel configuration dialog.
<b>1.53</b>	<i>2011-03-19</i>	Cleaned up.
<b>1.52</b>	<i>2011-03-19</i>	First version with channel edit dialog.
<b>1.51</b>	<i>2011-03-08</i>	Following changes: - corrected saving of gauge_enumerator; - removed old protocol support (the one without UOS_PROTO_MT_SUPPORT); - corrected saving of the //@offset parameter; - corrected flushing of the buffers in query.
<b>1.50</b>	<i>2011-02-11</i>	Added map dump command line option.
<b>1.49</b>	<i>2011-02-07</i>	Initial support for adding gauges in runtime.
<b>1.48</b>	<i>2011-02-01</i>	Added support for -rpc-arg option.
<b>1.47</b>	<i>2011-01-08</i>	Added support for returnig values from RPC.
<b>1.46</b>	<i>2010-12-29</i>	Following changes: - added support for demo configuration; - added building of monitor DLL; - initial implementation of GAUGE_JOYSTICK.
<b>1.45</b>	<i>2010-12-05</i>	Moved uOS_mon_query_exec() before the iterators so the decimation gets / sets the value at the first iteration.
<b>1.44</b>	<i>2010-11-06</i>	First release with RPC support.
<b>1.43</b>	<i>2010-11-06</i>	Preparations for RPC implementation.
<b>1.42</b>	<i>2010-11-03</i>	Added support for bgcolor and fgcolor in GAUGE_GENERIC.
<b>1.41</b>	<i>2010-10-23</i>	Reverted the seti option modification.
<b>1.40</b>	<i>2010-10-22</i>	Added support for the visource property.
<b>1.39</b>	<i>2010-10-19</i>	More support for editing, corrected the seti command.
<b>1.38</b>	<i>2010-10-05</i>	Added support for Z-order.

Version	Date	Change description
<b>1.37</b>	<i>2010-10-02</i>	Following changes: - corrected click events for bitfield and LED; - added support for dialog boxes in fullscreen mode; - changed defines to enums in gauges.h; - added F2 function (maximize / restore window); - code cleanup;
<b>1.36</b>	<i>2010-09-10</i>	Initial support for gauge editing dialog.
<b>1.35</b>	<i>2010-09-06</i>	Added support for CMD_GET_ID / idof command.
<b>1.34</b>	<i>2010-09-04</i>	Added foreground and background color properties to: - GAUGE_LEVEL_METER; - GAUGE_BITFIELD.
<b>1.33</b>	<i>2010-08-22</i>	Added support for configuration file validation.
<b>1.32</b>	<i>2010-08-20</i>	Added gets command line option.
<b>1.31</b>	<i>2010-07-16</i>	Added argtable bug test.
<b>1.30</b>	<i>2010-06-22</i>	Command line interface improved.
<b>1.29</b>	<i>2010-06-05</i>	Command line improvements.
<b>1.28</b>	<i>2010-06-03</i>	Support for the command line interface.
<b>1.27</b>	<i>2010-05-31</i>	First version with argtable2.
<b>1.26</b>	<i>2010-05-27</i>	Minor corrections.
<b>1.25</b>	<i>2010-05-17</i>	Following modifications: - added support for saving terminals to files; - terminal logic rewritten to use line buffers.
<b>1.24</b>	<i>2010-04-08</i>	Corrected uOS_mon_set_term() / uOS_mon_get_term() pair.
<b>1.23</b>	<i>2010-03-29</i>	Monitor terminal re-implemented.
<b>1.22</b>	<i>2010-03-18</i>	Added %t format for gauge_7_segment.
<b>1.21</b>	<i>2010-01-28</i>	Corrected bug in uOS_mon_get_query_data().
<b>1.20</b>	<i>2010-01-23</i>	Support for push query (not tested).
<b>1.19</b>	<i>2010-01-18</i>	Push support in query mode (not tested).
<b>1.18</b>	<i>2010-01-05</i>	Optimizations.
<b>1.17</b>	<i>2009-11-21</i>	Editor - related improvements.
<b>1.16</b>	<i>2009-11-21</i>	First attempt to introduce threads.
<b>1.15</b>	<i>2009-11-17</i>	First release with query support.
<b>1.14</b>	<i>2009-08-20</i>	Corrected closing of serial port, improved terminal output.
<b>1.13</b>	<i>2009-08-15</i>	Changed the messages printed on the terminal.
<b>1.12</b>	<i>2009-08-14</i>	Added support for multiple targets.
<b>1.11</b>	<i>2009-06-18</i>	Following changes: - added retransmission on failure; - improvements for displaying the local terminal.
<b>1.10</b>	<i>2009-04-17</i>	Added prefix field in @ref GAUGE_TERMINAL.
<b>1.9</b>	<i>2009-03-10</i>	Renamed log_parser.h to binary_parsers.h, changed window title.
<b>1.8</b>	<i>2008-12-10</i>	Support for editor.
<b>1.7</b>	<i>2008-06-16</i>	Functions naming changes.
<b>1.6</b>	<i>2008-05-07</i>	Changed command line parsing, fullscreen improvements.
<b>1.5</b>	<i>2008-04-03</i>	Minor corrections.
<b>1.4</b>	<i>2008-03-21</i>	Added msg_loop.c and iterators.c.
<b>1.3</b>	<i>2008-03-12</i>	Added partial mouse support.
<b>1.2</b>	<i>2008-01-29</i>	Keyboard functionality move partially to keyboard.c.