μOS **monitor**
Instrukcja Obsługi

Viresco μCS / Micro Control Systems

13 marca 2018

Spis treści

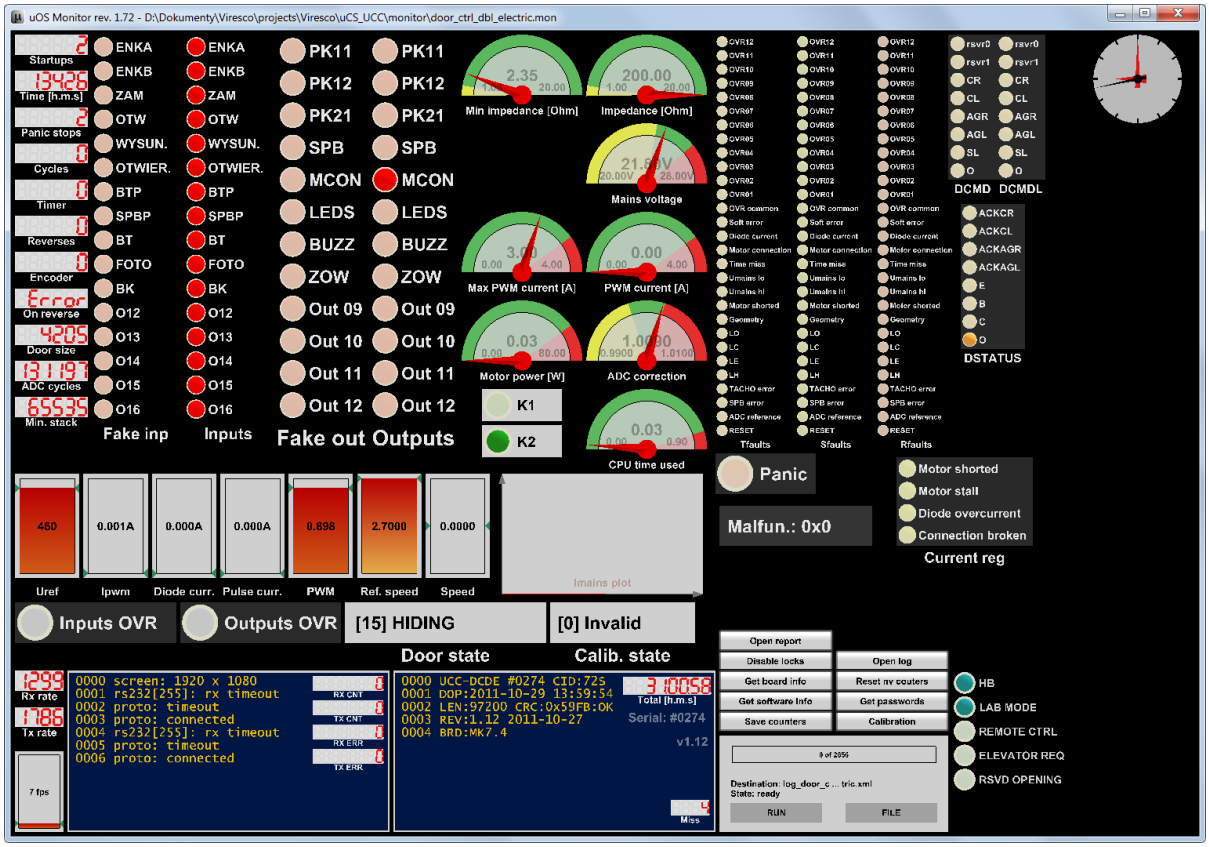
1 Wprowadzenie

µOS monitor jest programem służącym do monitorowania i kontrolowania pracy innych programów w czasie rzeczywistym. Umożliwia budowanie aplikacji sterujących, wizualizację procesów i diagnozowanie problemów.

1.1 Zasada działania

Działanie monitora opiera się na transmisji danych do i z programów docelowych. Monitor ma dostęp do wszystkich zmiennych globalnych (których adres jest znany) w programie. Każda zmienna może zostać odczytana i zapisana. Oprócz dostępu do zmiennych możliwe jest też korzystanie z funkcji specjalnych[1].

Rysunek 1: Uruchomiony *µOS* monitor



1.2 Medium komunikacyjne

Aby interakcja z programem docelowym była możliwa, musi istnieć kanał komunikacyjny pomiędzy monitorem a programem docelowym.

W chwili obecnej dostępne są następujące sposoby komunikacji z programem docelowym:

- przez łącze szeregowe - dla zdalnych programów docelowych
- przez pipe'y w systemie Windows - dla programów lokalnych (uruchomionych na tej samej maszynie)

[1] w zależności od platformy

1.3 Dostęp do zmiennych

By był możliwy dostęp do wszystkich zmiennych w programie docelowym, monitor musi mieć dostęp do tzw. mapy symboli. Mapa symboli to plik tekstowy generowany przez linker programu docelowego. Każdy kompilator dowolnego języka programowania ma możliwość wyprodukowania takiej mapy.

Przykład mapy:

Listing 1: Przykład mapy symboli generowanej przez GCC (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
```

Plik mapy stanowi dane wejściowe dla monitora który analizuje ten plik tworząc listę par symbol-adres która służy do lokalizacji zmiennych w programie docelowym. Procedura ta pozwala używać oryginalnych nazw zmiennych w konfiguracji *µOS* monitor zamiast sztucznie spreparowanych nazw bądź identyfikatorów liczbowych.

1.4 Dostępne typy danych

Typy danych dostępnych w *µOS* monitor zostały zebrane w tabeli 1

Tablica 1: Typy danych

Nazwa	Rodzaj	Rozmiar [bitów]	Lokalizacja	Kierunek
data8u	naturalny	8	RAM	we/wy
data8s	całkowity	8	RAM	we/wy
data16u	naturalny	16	RAM	we/wy
data16s	całkowity	16	RAM	we/wy
data32u	naturalny	32	RAM	we/wy
data32s	całkowity	32	RAM	we/wy
datasingle	zmiennoprzecinkowy[1]	32	RAM	we/wy
datadouble	zmiennoprzecinkowy[1]	64	RAM	we/wy
datasz	znakowy	N×8	RAM	we/wy
terminal	specjalny	N×8	RAM	we
eeprom	specjalny	N×8	EEPROM[2]	we
callback [3]	specjalny	N×8	n.d.	we

1.5 Zdalne wywoływanie procedur

Oprócz dostępu do zmiennych możwliwa jest również zdalne wywoływanie funkcji (RPC).

! → Zachowaj ostrożność przy używaniu RPC, oprogramowanie w żaden sposób nie weryfikuje poprawności adresu procedury (nawet jego wyrównania)!

Zdalne wywoływanie procedur może okazać się pomocne np. przy uruchamianiu funkcji których nie da się uruchomić poprzez przestawienie zmiennych globalnych lub operacja taka jest zbyt skomplikowana. Można również go użyć do uruchamiania wybranych funkcji “na żądanie”.

[1] zgodny ze standardem IEEE-754
[2] dostępne (*µOS* monitor 1.55) dla mikrokontrolerów z rodziny AVR firmy Atmel
[3] wywołanie procedury wewnętrznej monitora, dotyczy tylko niektórych kontrolek

Procedura wywoływana zdalnie musi mieć następujący prototyp:

Listing 2: Przykład deklaracji procedury wywoływanej zdalnie

```
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 i tx nie sa uzywane w biezacej implementacji. */
18    (void)rx;
19    (void)tx;
20
21    /* Dowolne czynnosci. */
22
23    /* W normalnym przypadku procedura powinna zwrocic TRUE, wartosc FALSE
24       jest zarezerwowana dla bledow krytycznych. */
25    return (TRUE);
26 }
```

Jak wspomniano wyżej, oprogramowanie nie sprawdza ani poprawności adresu procedury, ani nawet jej wyrównania. W niekorzystnych przypadkach może to prowadzić do fatalnych w skutkach błędów wynikających z uruchomienia instrukcji pod nieprawidłowym adresem. Problem ten najczęściej będzie spowodowany faktem rozsynchronizowania mapy symboli (która zawiera adresy funkcji RPC) i zawartości samego programu docelowego.

Aby zabezpieczyć się przed taką ewentualnością można posłużyć się specjalną sekcją lub sekcjami pamięci przeznaczonymi wyłącznie dla procedur RPC. Po osadzeniu tych sekcji pod z góry ustalonymi adresami możemy być pewni że wywołanie danej procedury RPC zawsze spowoduje zamierzony skutek.

Należy pamiętać że procedura RPC jest wywoływana w tym samym wątku co procedura przetwarzania komunikatów *µOS* monitor . Jeśli czas lub zasoby potrzebne na wykonanie danej RPC są z tego względu niewystarczające, należy rozważyć użycie dodatkowego wątku / procesu który jest uruchamiany w momencie wywołania RPC.

1.6 Uwagi dotyczące przenośności

Przy przenoszeniu programu na kolejne platformy należy zwrócić szczególną uwagę na następujące cechy *µOS* monitor :

1. Rozmiar bajtu nie zawsze równy jest 8 bitom, na niektórych platformach zmienna typu znakowego jest np. 16 bitowa. Aby zapewnić zgodność *µOS* monitor przy innym rozmiarze bajtu należy upewnić się że makro BYTE8_SIZE jest prawidłowo zdefiniowane.

2.

2 Instalacja

2.1 Instalacja μOS monitor

Do poprawnej pracy μOS monitor wymaga komputera PC z systemem co najmniej Windows 95 (zalecany Windows XP, Windows Vista lub Windows 7) [4]), karty graficznej zgodnej z OpenGL, myszy i klawiatury.

Program jest wyposażony w standardowy instalator. Po uruchomieniu mamy możliwość wyboru katalogu docelowego oraz opcjonalnego skojarzenia plików konfiguracyjnych (*.mon) z μOS monitor .

2.2 Przygotowanie programu docelowego

Jak wspomniano we wprowadzeniu program docelowy musi zostać przystosowany do współpracy z monitorem. Aby było to możliwe, program docelowy musi mieć możliwość wywołania procedury w języku C lub dołączenia biblioteki DLL (dotyczy Windows).

Obsługa μOS monitor jest realizowana dwuetapowo: wykonana jest jednorazowa inicjalizacja modułu odpowiedzialnego za komunikację z monitorem oraz cykliczne uruchamianie podprogramu odsługującego komunikację. Zakładając że medium transmisyjnym jest interfejs szeregowy a funkcje dostępu do tego medium są zdefiniowane podobnie jak na listingu 8, inicjalizacja będzie wyglądać następująco:

Listing 3: Inicjalizacja po stronie programu docelowego

```
1  #include <uart.h>
2  #include <uOS_mon.h>
3
4  int main(void)
5  {
6     /* Uruchamia interfejs szeregowy. */
7     uart_init();
8
9     /* Konfiguruje protokół komunikacyjny. */
10    uOS_proto_init
11       (
12       uOS_comm_receive_byte, /* Funkcja do odbioru znaku. */
13       uOS_comm_transmit_byte,/* Funkcja do nadawania znaku. */
14       uart_flush_rx_buffer,  /* Funkcja oproznienia
15                                 bufora odbiorczego. */
16       uart_flush_tx_buffer   /* Funkcja oproznienia
17                                 bufora nadawczego. */
18       );
19
20    for(;;)
21    {
22       /* Normalne czynnosci w tle. */
23       ...
24
25       /* Przetwarzanie komunikatow uOS_monitor. */
26       uOS_mon_cmd_parse();
27    }
28 }
```

2.3 Wielodostęp

Oprogramowanie po stronie programu docelowego nie czyni żadnych założeń co do wielowątkowości systemu. Ponieważ jest realizowany dostęp do globalnych zmiennych, a inne procesy (wątki, przerwania) mogą również realizować dostęp do tych zmiennych, może zaistnieć potrzeba wykluczenia dostępu przez inne procesy do

[4] interfejs linii poleceń nie jest w pełni funkcjonalny poniżej Windows XP

zmiennych na czas wywołania `uOS_mon_cmd_parse()`. Można to zrealizować na dwa sposoby:

Listing 4: Użycie sekcji krytycznej

```
1  int main(void)
2  {
3     CRITICAL_SECTION cs;
4
5     for(;;)
6     {
7        /* Normalne czynnosci w tle. */
8        ...
9
10       /* Przetwarzanie komunikatow uOS_monitor. */
11       enter_critical_section(&cs);
12          uOS_mon_cmd_parse();
13       leave_critical_section(&cs);
14    }
15 }
```

Powyższy sposób może nie być do zaakceptowania jeśli czas przebywania w procedurze obsługi monitora jest zbyt duży (oraz zależny od limitu czasu odbioru znaku, zobacz 8).

Alternatywnym sposobem rozwiązania problemu wielodostępu jest użycię dedykowanych zmiennych przeznaczonych specjalnie do interakcji z monitorem:

Listing 5: Użycie zmiennych dedykowanych

```
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     /* Przepisanie zmiennych czytanych. */
9     mon_global_x = global_x;
10
11    /* Przepisanie zmiennych zapisywanych. */
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       /* Normalne czynnosci w tle. */
23       ...
24
25       /* Sekcja krytyczna dostepu do zmiennych. */
26       EnterCriticalSection(&cs);
27          mon_update();
28       LeaveCriticalSection(&cs);
29
30       /* Przetwarzanie komunikatow uOS_monitor. */
31       uOS_mon_cmd_parse();
32    }
33 }
```

W powyższym przykładzie monitor realizuje dostęp do dedykowanych zmiennych `mon_global_x` oraz `mon_global_y` zamiast bezpośrednio do zagrożonych wielodostępem zmiennych `global_x` i `global_y`.

Jeśli środowisko docelowe umożliwia uruchomienie dodatkowego procesu to oczywiście można go użyć do obsługi funkcji monitora. Prosty przykład dla Windows:

Listing 6: Wykorzystanie dedykowanego wątku

```
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             /* Przetwarzanie komunikatow uOS_monitor. */
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 }
```

Oczywiście w przypadku użycia osobnego wątku wielodostęp ma jeszcze większe znaczenie dlatego należy zwrócić uwagę na moment uruchomienia obsługi monitora lub użyć zmiennych dedykowanych jak w przykładzie powyżej.

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: Przykładowa implementacja przelotek komunikacyjnych

```
#include <uOS_proto.h>
#include <uOS_mon.h>

/*****************************************************************************//**
* uOS_comm_transmit_byte
*
* @param[in]       dst - not used.
* @param[in]       byte - byte to be sent.
* @param[out]      -
* @param[in,out]   -
*
* @return          UOS_STATUS_OK, always.
*
* @brief           Transmit byte wrapper.
*****************************************************************************/
UOS_STATUS uOS_comm_transmit_byte(UINT8T dst, UINT8T byte)
{
    (void)dst;
    uart_transmit_byte(byte);

    return (UOS_STATUS_OK);
}

/*****************************************************************************//**
* uOS_comm_receive_byte
*
* @param[in]       src - not used.
* @param[in]       byte - where to place the result.
* @param[out]      -
* @param[in,out]   -
*
* @return          UOS_STATUS_OK when a byte was received.
* @return          UOS_STATUS_ETIMEOUT when a timeout occurs.
*
* @brief           Receive byte wrapper.
*****************************************************************************/
UOS_STATUS uOS_comm_receive_byte(UINT8T src, UINT8T *byte)
{
    unsigned int timeout=100;
    (void)src;

    do
    {
        if (uart_data_in_rx_buffer())
        {
            *byte=uart_receive_byte();
            return (UOS_STATUS_OK);
        }

        delay_ms(1);
    }
    while (timeout-- != 0);

    return (UOS_STATUS_ETIMEOUT);
}
```

2.4 Examples

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

Listing 9: Pipe demo implementation

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include "defs.h"
#include "uOS_hm.h"
#include "uOS_proto.h"
#include "uOS_mon.h"
#include "named_pipe.h"

UINT32T mon_cnt;
UINT8T mon_lamp, old_lamp;
UINT8T MY_MODULE_ADDR = 254;

static UOS_STATUS uOS_rx_byte(UINT8T src, UINT8T *byte)
{
   return (uOS_pipe_receive_byte(0, byte));
}

static UOS_STATUS uOS_tx_byte(UINT8T dst, UINT8T byte)
{
   return (uOS_pipe_transmit_byte(0, byte));
}

int main(int argc, char *argv[])
{
   uOS_hm_install_logger((UOS_HM_LOG_METHOD)printf);
   uOS_proto_init(uOS_rx_byte, uOS_tx_byte, uOS_pipe_flush_rx,
       uOS_pipe_flush_tx);

   uOS_hm_log("Start...\n");

   if (argc < 2)
   {
      pipe_conf[0].name = "PIPE1";
   }
   else
   {
      pipe_conf[0].name = argv[1];
   }

   pipe_conf[0].timeout = 500;
   pipe_conf[0].server = TRUE;

   if (uOS_pipe_open(0) != UOS_STATUS_OK)
   {
      uOS_hm_log("Cannot open pipe\n");
      return (-1);
   }

   while(!kbhit())
   {
      mon_cnt++;
      if (uOS_mon_cmd_parse() == UOS_STATUS_OK)
      {
         if (old_lamp != mon_lamp)
         {
            printf("lamp: %d\n", (int)mon_lamp);
            old_lamp = mon_lamp;
         }
      }
   }
```

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

3 Konfiguracja

Wszystkie dane konfiguracyjne programu znajdują się w pliku konfiguracyjnym w formacie XML. Domyślne rozszerzenie tego pliku to (*.mon). Instalator może automatycznie skojarzyć ten typ plików z *µOS* monitor jeśli użytkownik wyrazi na to zgodę.

3.1 Poprawność pliku konfiguracyjnego

Program jest wyposażony w dwustopniowy mechanizm weryfikacji poprawności pliku konfiguracyjnego:

1. Poprzez wbudowany schemat XML - XMLSchema
2. Poprzez dodatkowe sprawdziany poprawności wykonywane już po załadowaniu pliku konfiguracyjnego który jest zgodny ze schematem.

3.2 Dostępne kontrolki

Wszystkie kontrolki obsługiwane przez *µOS* monitor opisuje model XSD. Poniżej zebrano krótki opis kontrolek oraz zawarto ich specyfikacje XSD.

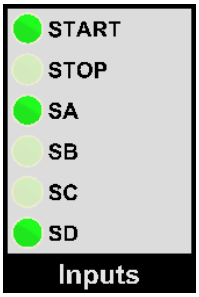
3.2.1 7 segment

Rysunek 2: 7 segment

Wzorowany na wyświetlaczu 7-segmentowym. Obsługuje specjalny format %t który wyświetla czas w formie HH.MM.SS.

Listing 10: Model kontrolki typu 7 segment

```
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>
```



Rysunek 3: Bitfield

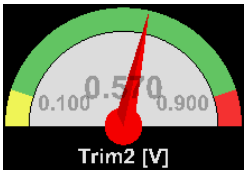
3.2.2 Bitfield

Kontrolka wyświetlająca liczbę całkowitą rozkładając ją na poszczególne bity. Możliwe jest ustawienie / wyzerowanie poszczególnych bitów w trybie wyjściowym (przez kliknięcie myszą).

Listing 11: Model kontrolki typu bitfield

```
<xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
    gauge_bitfield_type">
 <xsd:all>
  <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="target" type="target_type"/>
  <xsd:element name="title" type="xsd:string"/>
  <xsd:element name="visource" type="xsd:string" minOccurs="0"/>
  <xsd:element name="labels" type="labels_type"/>
  <xsd:element name="bits">
   <xsd:simpleType>
    <xsd:restriction base="xsd:positiveInteger">
     <xsd:minInclusive value="1"></xsd:minInclusive>
     <xsd:maxInclusive value="32"></xsd:maxInclusive>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
  <xsd:element name="x" type="xsd:double"/>
  <xsd:element name="y" type="xsd:double"/>
  <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
  <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="width" type="xsd:double" minOccurs="0"/>
  <xsd:element name="height" type="xsd:double" minOccurs="0"/>
  <xsd:element name="actual" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
  <!-- /color is left for backward compatibility, it should be replaced by /fgcolor
      -->
  <xsd:element name="color" type="xsd:string" minOccurs="0"/>
  <xsd:element name="txcolor" type="color_type" minOccurs="0"/>
  <xsd:element name="fgcolor" type="color_type" minOccurs="0"/>
  <xsd:element name="bgcolor" type="color_type" minOccurs="0"/>
  <xsd:element name="argument" type="xsd:string" minOccurs="0"/>
  <xsd:element name="type" minOccurs="0">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:enumeration value="left aligned"></xsd:enumeration>
     <xsd:enumeration value="right aligned"></xsd:enumeration>
     <xsd:enumeration value="square left aligned"></xsd:enumeration>
     <xsd:enumeration value="square right aligned"></xsd:enumeration>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
 </xsd:all>
</xsd:complexType>
```

3.2.3 Clock



Rysunek 4: Clock

Klasyczny zegar 180°. Możliwe jest ustawienie górnego i dolnego poziomu alarmo-

wego.

Listing 12: Model kontrolki typu clock

```
<xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
    gauge_clock_type">
 <xsd:all>
  <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="target" type="target_type"/>
  <xsd:element name="title" type="xsd:string" minOccurs="0"/>
  <xsd:element name="x" type="xsd:double"/>
  <xsd:element name="y" type="xsd:double"/>
  <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
  <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="format" type="xsd:string"/>
  <xsd:element name="min" type="xsd:double" minOccurs="0"/>
  <xsd:element name="max" type="xsd:double" minOccurs="0"/>
  <xsd:element name="hi" type="xsd:double" minOccurs="0"/>
  <xsd:element name="lo" type="xsd:double" minOccurs="0"/>
  <xsd:element name="type">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:enumeration value="180"></xsd:enumeration>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
  <xsd:element name="width" type="xsd:double" minOccurs="0"/>
  <xsd:element name="height" type="xsd:double" minOccurs="0" fixed="0"/>
  <xsd:element name="actual" type="xsd:double" minOccurs="0"/>
  <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="fgcolor" type="color_type" minOccurs="0"/>
 </xsd:all>
</xsd:complexType>
```

3.2.4 Frame

Rysunek 5: Frame

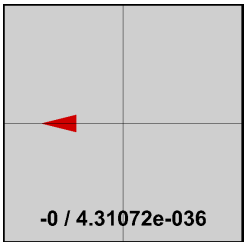
Ramka służąca grupowaniu elementów.

Listing 13: Model kontrolki typu frame

```
<xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
    gauge_frame_type">
 <xsd:all>
  <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="title" type="xsd:string" minOccurs="0"/>
  <xsd:element name="actual" type="xsd:string" minOccurs="0"/>
  <xsd:element name="target" type="target_type"/>
  <xsd:element name="x" type="xsd:double"/>
  <xsd:element name="y" type="xsd:double"/>
  <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
  <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="width" type="xsd:double" minOccurs="0"/>
  <xsd:element name="height" type="xsd:double" minOccurs="0"/>
  <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="format" type="xsd:string"/>
  <xsd:element name="txcolor" type="color_type" minOccurs="0"/>
  <xsd:element name="fgcolor" type="color_type" minOccurs="0"/>
  <xsd:element name="bgcolor" type="color_type" minOccurs="0"/>
  <xsd:element name="type" minOccurs="0">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:enumeration value="left aligned"></xsd:enumeration>
     <xsd:enumeration value="right aligned"></xsd:enumeration>
     <xsd:enumeration value="center aligned"></xsd:enumeration>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
 </xsd:all>
</xsd:complexType>
```

3.2.5 Complex plot

Wykres wzkazowy, pokazuje dwie liczby w postaci wektora (np. liczby zespolone).



Rysunek 6: Complex plot

Listing 14: Model kontrolki typu plot cplx

```
<xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
    gauge_plot_cplx_type">
 <xsd:all>
  <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="title" type="xsd:string" minOccurs="0"/>
  <xsd:element name="target" type="target_type"/>
  <xsd:element name="x" type="xsd:double"/>
  <xsd:element name="y" type="xsd:double"/>
  <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
  <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="max" type="xsd:double" minOccurs="0"/>
  <xsd:element name="width" type="xsd:double" minOccurs="0"/>
  <xsd:element name="height" type="xsd:double" minOccurs="0" fixed="0"/>
  <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="format" type="xsd:string"/>
  <xsd:element name="txcolor" type="color_type" minOccurs="0"/>
  <xsd:element name="fgcolor" type="color_type" minOccurs="0"/>
  <xsd:element name="bgcolor" type="color_type" minOccurs="0"/>
 </xsd:all>
</xsd:complexType>
```

3.2.6 Led

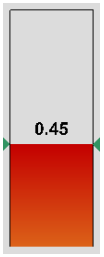
Rysunek 7: Led

Lampka LED. Zachowanie podobne do kontrolki *Bitfield*. Możliwe użycie jako wyjście (poprzez kliknięcie myszką).

Listing 15: Model kontrolki typu led

```
<xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="gauge_led_type"
    >
 <xsd:all>
  <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="target" type="target_type"/>
  <xsd:element name="title" type="xsd:string"/>
  <xsd:element name="x" type="xsd:double"/>
  <xsd:element name="y" type="xsd:double"/>
  <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
  <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="width" type="xsd:double" minOccurs="0"/>
  <xsd:element name="height" type="xsd:double" minOccurs="0"/>
  <xsd:element name="actual" type="xsd:double" minOccurs="0"/>
  <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
  <!-- /color is left for backward compatibility, it should be replaced by fgcolor
      -->
  <xsd:element name="color" type="xsd:string" minOccurs="0"/>
  <xsd:element name="txcolor" type="color_type" minOccurs="0"/>
  <xsd:element name="fgcolor" type="color_type" minOccurs="0"/>
  <xsd:element name="bgcolor" type="color_type" minOccurs="0"/>
  <xsd:element name="type" minOccurs="0">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:enumeration value="left aligned"></xsd:enumeration>
     <xsd:enumeration value="right aligned"></xsd:enumeration>
     <xsd:enumeration value="square left aligned"></xsd:enumeration>
     <xsd:enumeration value="square right aligned"></xsd:enumeration>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
 </xsd:all>
</xsd:complexType>
```

3.2.7 Level meter



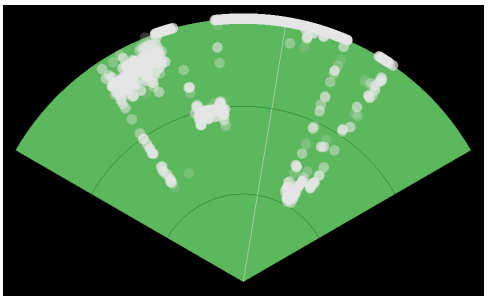
Rysunek 8: Level meter

Ten typ kontrolki obrazuje poziom. Możliwe jest ustawienie punktu odniesienia (poziomu zerowego) innego niż poziom minimalny. Można z jej pomocą ustawiać wartość wyjściową (poprzez kliknięcie lub przewinięcie kółka myszy).

Listing 16: Model kontrolki typu level meter

```
<xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
    gauge_level_meter_type">
 <xsd:all>
  <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="target" type="target_type"/>
  <xsd:element name="title" type="xsd:string" minOccurs="0"/>
  <xsd:element name="visource" type="xsd:string" minOccurs="0"/>
  <xsd:element name="x" type="xsd:double"/>
  <xsd:element name="y" type="xsd:double"/>
  <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
  <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="format" type="xsd:string"/>
  <xsd:element name="inc" type="inc_type" minOccurs="0"/>
  <xsd:element name="width" type="xsd:double" minOccurs="0"/>
  <xsd:element name="height" type="xsd:double" minOccurs="0"/>
  <xsd:element name="min" type="xsd:double" minOccurs="0"/>
  <xsd:element name="max" type="xsd:double" minOccurs="0"/>
  <xsd:element name="origin" type="xsd:double" minOccurs="0"/>
  <xsd:element name="actual" type="xsd:double" minOccurs="0"/>
  <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="argument" type="xsd:string" minOccurs="0"/>
  <xsd:element name="type" minOccurs="0">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:enumeration value="vertical"></xsd:enumeration>
     <xsd:enumeration value="horizontal"></xsd:enumeration>
     <xsd:enumeration value="horizontal left aligned"></xsd:enumeration>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
 </xsd:all>
</xsd:complexType>
```

3.2.8 Radar



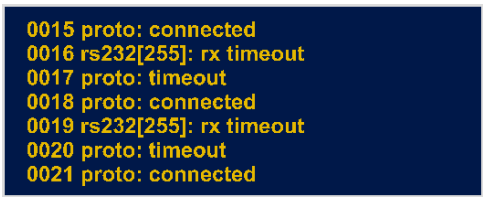
Rysunek 9: Radar

Jak nazwa wskazuje, jest to wizualizacja pracy czujnika odległości umieszczonego na głowicy obortowej. Obrazowanie historii pomiarów jest zrealizowane przez blaknięcie starszych punktów: najnowszy punkt ma pełen kolor, najstarszy jest niemalże niewidoczny. Ilość punktów dowolnie konfigurowalna.

Listing 17: Model kontrolki typu radar

```
<xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
    gauge_radar_type">
 <xsd:all>
  <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="target" type="target_type"/>
  <xsd:element name="title" type="xsd:string"/>
  <xsd:element name="x" type="xsd:double"/>
  <xsd:element name="y" type="xsd:double"/>
  <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
  <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="width" type="xsd:double" minOccurs="0"/>
  <xsd:element name="height" type="xsd:double" minOccurs="0"/>
  <xsd:element name="max" type="xsd:double"/>
  <xsd:element name="decimation" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="points" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="type" minOccurs="0">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:enumeration value="120"></xsd:enumeration>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
 </xsd:all>
</xsd:complexType>
```

3.2.9 Terminal



Rysunek 10: Terminal

Umożliwia symulowanie konsoli. Aby było możliwe korzystanie z niego po stronie programu docelowego, należy skorzystać ze specjalnej struktury danych i ze specjalnych funkcji zapisu do terminala (kolejkowanie FIFO). Terminal można również sprząc z mechanizmem diagnostyki μOS monitor . Wyświetla on wtedy komunikaty programu (jak na rysunku).

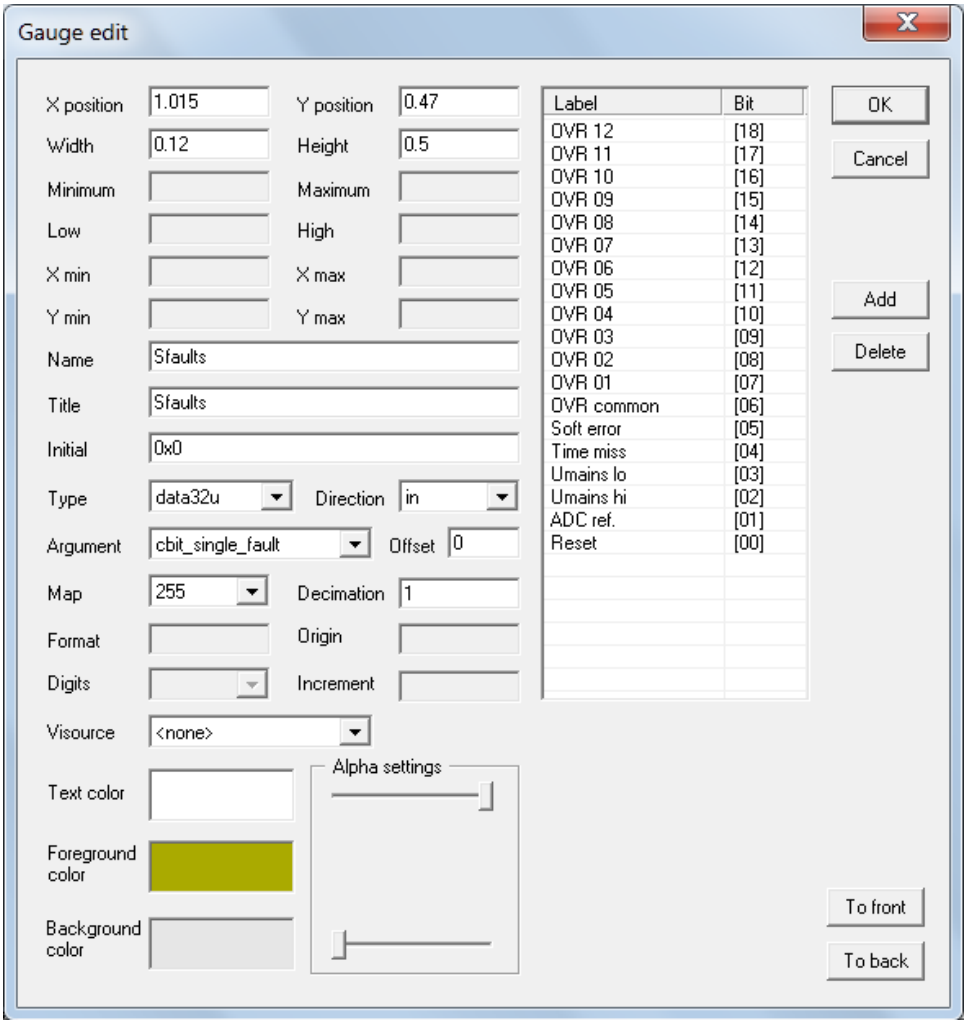
Listing 18: Model kontrolki typu terminal

```
<xsd:complexType xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="
    gauge_terminal_type">
 <xsd:all>
  <xsd:element name="log" type="xsd:boolean" default="false" minOccurs="0"/>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="target" type="target_type"/>
  <xsd:element name="title" type="xsd:string" minOccurs="0"/>
  <xsd:element name="x" type="xsd:double"/>
  <xsd:element name="y" type="xsd:double"/>
  <xsd:element name="z_rot" type="xsd:double" minOccurs="0"/>
  <xsd:element name="z_order" type="xsd:unsignedInt" minOccurs="0"/>
  <xsd:element name="height" type="xsd:double" minOccurs="0"/>
  <xsd:element name="width" type="xsd:double" minOccurs="0"/>
  <xsd:element name="log_file" type="xsd:anyURI" minOccurs="0"/>
  <xsd:element name="txcolor" type="color_type" minOccurs="0"/>
  <xsd:element name="prefix" minOccurs="0">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:enumeration value="index"></xsd:enumeration>
     <xsd:enumeration value="time"></xsd:enumeration>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
 </xsd:all>
</xsd:complexType>
```

3.3 Funkcje edycyjne

Program został wyposażony w ograniczone funkcje edycyjne umożliwiające zmianę parametrów poszczególnych kontrolek bezpośrednio z okna programu.

Rysunek 11: Edycja właściwości kontrolki typu enumerator



4 Interfejs linii poleceń

Program wyposażono w interfejs linii poleceń pozwalający na automatyzację zadań takich jak testowanie lub długotrwała rejestracja. Pozwala on na dostęp do obiektów zdefiniowanych w pliku konfiguracyjnym. Wyniki wszystkich operacji są wysyłane w formie tekstowej na standardowe wyjście (stdout). Opis interfejsu linii poleceń jest dostępny po wpisaniu:

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

4.1 Dostępne komendy

W chwili obecnej (μOS monitor 1.55) dostępne są następujące komendy:

- `-h, --help` - wyświetla pomoc;

- `-f, --fullscreen` - powoduje uruchomienie programu w trybie pełnoekranowym;
- `-r, --redirect` - pozwala przekierować strumień wyjściowy (np. używając operatora > znanego z DOS); normalnie program alokuje nową konsolę co uniemożliwia przekierowanie jego wyjścia;
- `-n, --name` - wybiera nazwę obiektu do którego następuje dostęp (odczyt / zapis);
- `--geti` - odczytuje liczbę całkowitą;
- `--getf` - odczytuje liczbę zmiennoprzecinkową;
- `--gets` - odczytuje ciąg znaków (string);
- `--seti` - zapisuje liczbę całkowitą;
- `--load` - wykonuje transfer bloku pamięci zdefiniowany przez <name>;
- `-i, --idof=<int>` - odczytuje identyfikator urządzenia docelowego na kanale <int>;
- `-v, --revof=<int>` - odczytuje wersję oprogramowania urządzenia docelowego na kanale <int>;
- `--rpc` - wykonuje procedurę zdalną o nazwie <name>;
- `--rpc-arg=<string>` - przekazuje argument procedury zdalnej (opcjonalny);
- `--map-dump` - wyświetla zawartość mapy symboli i kończy działanie.

! → Należy zaznaczyć że tylko jeden typ komendy `geti / getf / gets / seti` może zostać użyty przy pojedynczym wywołaniu programu. Możliwe jest za to użycie tej samej komendy wielokrotnie w tym samym wywołaniu (do 100 razy). Przykładowo następujące wywołanie:

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

lub

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

spowoduje odczytanie obiektów “clock” oraz “counter” i wyświetlenie ich wartości w kolejności wystąpienia w komendzie:

```
4428.000000
3.000000
```

4.2 Przykłady

Załóżmy że plik konfiguracyjny zawiera kontrolkę umożliwiającą dostęp do czasu pracy urządzenia:

```
<gauge_7_segment>
 <name>clock</name>
 <x>0.685000</x>
 <y>0.930000</y>
 <width>0.015</width>
 <digits>6</digits>
 <decimation>1</decimation>
 <format>%t</format>
```

```
 <title>Time [h.m.s]</title>
 <target
   dir="in"
   type="data32u"
   arg="mon_one_second_ticks"
   map="255"/>
</gauge_7_segment>
```

Odczytanie jego wartości jest możliwe po wywołaniu następującego polecenia:

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

lub

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

z kolei ustawienie wartości tego obiektu można uzyskać stosując:

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

lub

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

gdzie:

- `geti` jest komendą dostępu;
- `name` definiuje nazwę obiektu (taką samą jaką definiuje element <name>)
- `tester.mon` jest nazwą pliku konfiguracyjnego

Po uruchomieniu żądana wartość zostaje wydrukowana na konsoli (poprzez strumień standardowego wyjścia).

4.3 Wartości zwracane w trybie linii poleceń

Poprawne wykonanie komendy wydanej z linii poleceń kończy się zwróceniem przez program wartości 0. Jakikolwiek błąd powoduje zwrócenie wartości niezerowej.

5 Programowanie z użyciem μOS monitor

W wersji Professional do programu dołączone są biblioteki umożliwiające używanie oprogramowania z poziomu innych języków: C#, Python oraz GNU Octave.

Wszystkie owijacze wykorzystują `uOS_monitor.dll`.

Należy zaznaczyć że owijacze nie udostępniają wszystkich funkcjonalności dostępnych w okienkowej wersji μOS monitor . Istnieją również różnice pomiędzy implementacjami poszczególnych owijaczy (nie należy się spodziewać że program dla monitora napisany w C# będzie można bez problemu przenieść na przykład do Pythona). Pomimo to podstawowe funkcje mają bardzo podobne działanie. Więcej szczegółów oraz pełna dokumentacja API są dostępne w wersji instalacyjnej w postaci pliku *.chm.

Pliki przykładów dostępne są w podkatalogu `examples` w katalogu instalacji μOS monitor .

Do uruchomienia przykładów potrzebne są:

- `uOS_monitor.dll` (dostępny w μOS monitor Proffessional)

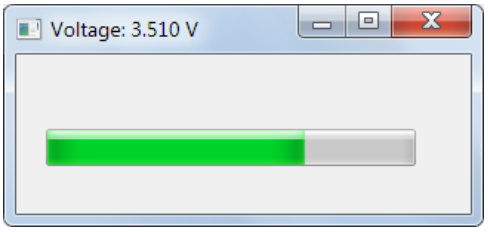
- sterownik Viresco μCS_UCC z oprogramowaniem UCC-IO w wersji 1.19 lub wyższej (instalator można pobrać ze strony http://www.viresco.pl/index.php?option=com_remository&Itemid=59&func=select&id=12)
- plik konfiguracyjny `io.mon` oraz mapa symboli `io.sym` pochodzące z w.w. instalatora

5.1 Python

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jul 02 13:19:02 2012

@author: Lukasz
"""

import sys
sys.path.append(r"d:\Dokumenty\Viresco\projects\Viresco\uOS_monitor\Scripts")
import uOS_monitor
import wx

monitor = []

class GaugeFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Gauge Example', size = (325, 150))
        panel = wx.Panel(self, -1)
        self.count = 0
        self.gauge = wx.Gauge(panel, -1, 500, (20, 50), (250, 25))
        self.gauge.SetBezelFace(3)
        self.gauge.SetShadowWidth(3)

        self.timer = wx.Timer(self)
        self.Bind(wx.EVT_TIMER, self.Update, self.timer)
        self.timer.Start(1000)

    def Update(self, event):
        voltage = monitor.pull_num("Analog 01")
        self.gauge.SetValue(int(voltage * 100.0))
        self.SetTitle("Voltage: {0:0.3f} V".format(voltage))

if __name__ == '__main__':
    monitor = uOS_monitor.uOS_monitor()
    monitor.start(r"d:\Dokumenty\Viresco\projects\Viresco\uCS_UCC\monitor\io.
        mon")
    app = wx.PySimpleApp()
    frame = GaugeFrame()
    frame.Show()
    app.MainLoop()
    monitor.stop()
```

Uruchomienie tego przykładu spowoduje wyświetlenie następującego okienka:

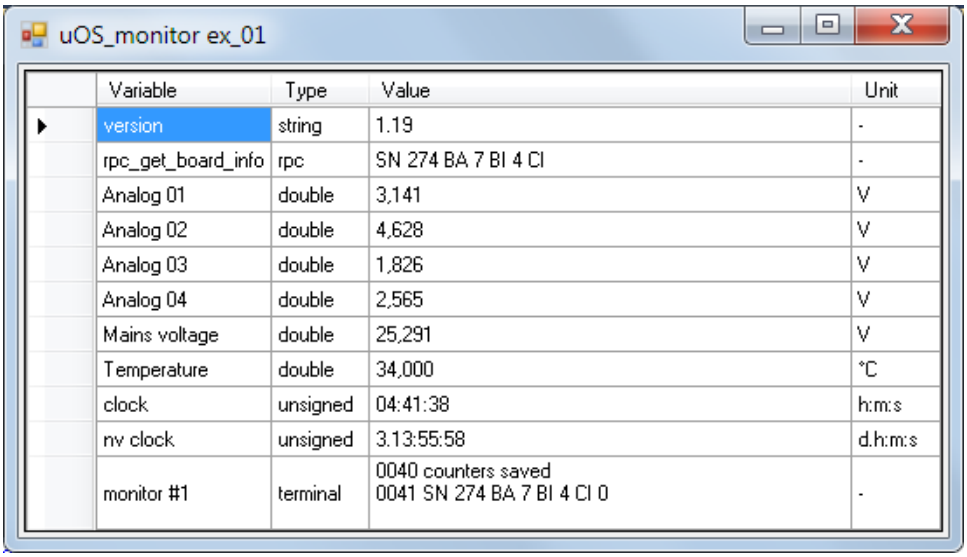
5.2 C#


Voltage: 3.510 V

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Windows.Forms;
9  using Viresco.uOS_monitor;
10
11 namespace ex_01
12 {
13     public partial class Form1 : Form
14     {
15         public Form1()
16         {
17             InitializeComponent();
18         }
19
20         private uOS_monitor monitor;
21
22         private void Form1_Load(object sender, EventArgs e)
23         {
24             monitor = new uOS_monitor("mates_ucc_mk1.mon");
25             dataGridView1.Rows.Add("version", "string", "", "-");
26             dataGridView1.Rows.Add("rpc_get_board_info", "rpc", "", "-");
27             dataGridView1.Rows.Add("Analog 01", "double", "", "V");
28             dataGridView1.Rows.Add("Analog 02", "double", "", "V");
29             dataGridView1.Rows.Add("Analog 03", "double", "", "V");
30             dataGridView1.Rows.Add("Analog 04", "double", "", "V");
31             dataGridView1.Rows.Add("Mains voltage", "double", "", "V");
32             dataGridView1.Rows.Add("Temperature", "double", "", "°C");
33             dataGridView1.Rows.Add("clock", "unsigned", "", "h:m:s");
34             dataGridView1.Rows.Add("nv clock", "unsigned", "", "d.h:m:s");
35             dataGridView1.Rows.Add("monitor #1", "terminal", "", "-");
36             dataGridView1.Rows[dataGridView1.Rows.Count - 1].DefaultCellStyle.
                   WrapMode = DataGridViewTriState.True;
37             dataGridView1.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.
                   AllCellsExceptHeaders;
38
39             timer1.Interval = 250;
40             timer1.Enabled = true;
41         }
42
43         private void timer1_Tick(object sender, EventArgs e)
44         {
45             if (monitor != null)
46             {
47                 foreach (DataGridViewRow r in dataGridView1.Rows)
48                 {
49                     string name = r.Cells["Variable"].Value.ToString();
50
51                     try
52                     {
53                         switch (r.Cells["Type"].Value.ToString())
54                         {
55                             case "double":
56                                 {
```

```
57                                  double value = monitor.CmdPullNumEx(name);
58                                  r.Cells["Value"].Value = value.ToString("0.000");
59                              }
60                              break;
61                          case "unsigned":
62                              {
63                                  // It could be something different,
64                                  // but we are pulling time here.
65                                  double value = monitor.CmdPullNumEx(name);
66                                  TimeSpan s = TimeSpan.FromSeconds(value);
67                                  r.Cells["Value"].Value = s.ToString();
68                              }
69                              break;
70                          case "string":
71                              {
72                                  string value = monitor.CmdPullStrEx(name);
73                                  r.Cells["Value"].Value = value.ToString();
74                              }
75                              break;
76                          case "terminal":
77                              {
78                                  string value = monitor.CmdPullTermEx(name);
79                                  if (value.Length > 0)
80                                  {
81                                      r.Cells["Value"].Value = value.ToString();
82                                  }
83                              }
84                              break;
85                          case "rpc":
86                              {
87                                  // Only once.
88                                  if (r.Cells["Value"].Value.ToString() == string.
                                      Empty)
89                                  {
90                                      string value = monitor.CmdExecuteRpcEx(name,
                                          null);
91                                      r.Cells["Value"].Value = value.ToString();
92                                  }
93                              }
94                              break;
95                      }
96                  }
97                  catch
98                  {
99                  }
100             }
101         }
102     }
103 }
104 }
```

Uruchomienie tego przykładu spowoduje wyświetlenie następującego okienka (zawartość odświeżana co 250 ms - patrz linia 37.):



5.3 GNU Octave

```
1  addpath(".\\..\\Bin");
2
3  file = "d:\\Dokumenty\\Viresco\\projects\\Viresco\\uCS_UCC\\monitor\\io.mon"
4
5  if uOS_monitor("start", file) == 0,
6     [status, out] = uOS_monitor("idof", 255);
7     printf("Connected to: %s\n", out)
8
9     [status, out] = uOS_monitor("revof", 255);
10    printf("Firmware version: %s\n", out)
11
12    [status, out] = uOS_monitor("getf", "Analog 01");
13    printf("Channel #1 voltage: %s V\n", out)
14
15    [status, out] = uOS_monitor("rpc", "rpc_get_board_info");
16    printf("Board information: %s\n", out)
17
18    uOS_monitor("stop");
19 end
```

W tym przykładzie otrzymujemy wynik bezpośrednio w linii poleceń:

```
1  Connected to: uCS_UCC-IO
2  Firmware version: 1.19
3  Channel #1 voltage: 3.136476 V
4  Board information: SN 274 BA 7 BI 4 CI 0
```

A Właściwości poszczególnych wersji μOS monitor

Właściwość	Wersja Demo	Wersja Standard	Wersja Professional
Dokumentacja w języku polskim i angielskim	☑	☑	☑
Pliki binarne programu docelowego dla Windows, AVR, ARM, PIC	☑	☑	☑
Sterowanie z poziomu linii poleceń	☑	☑	☑
Biblioteka μOS monitor DLL do wykorzystania w innych programach	☑	☑	☑
Nielimitiwana ilość kontrolek	☒	☑	☑
Pliki źródłowe dla programu docelowego	☒	☑	☑
Owijacz μOS monitor dla Octave (*.oct)	☒	☒	☑
Owijacz μOS monitor dla Python (*.py)	☒	☒	☑
Owijacz μOS monitor dla C# (*.cs)	☒	☒	☑

B Revision history

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

C Historia zmian

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

Wersja	Data	Opis zmiany
1.74	*2012-03-14*	Corrected repetitive paste and double click behavior.
1.73	*2012-03-02*	Initial support for multi - selection in editor.
1.72	*2012-02-07*	Added support for gauges subtypes.
1.71	*2012-01-19*	Initial support for the system callbacks.
1.70	*2012-01-13*	Added closing of the iconv encoders.
1.69	*2012-01-11*	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.
1.68	*2012-01-06*	Initial support for Polish diacritics.
1.67	*2011-12-19*	Added handling of the USE_CLOSEHANDLE_ERRATA.
1.66	*2011-12-13*	Following changes: - the console is freed when it was actually allocated; - the gauge data is freed also for the command line mode.
1.65	*2011-11-30*	Added password dialog box.
1.64	*2011-10-24*	Added support for the revof command.
1.63	*2011-10-12*	Many minor editor corrections.
1.62	*2011-09-28*	Initial support for block transfers (for Bluetooth).
1.61	*2011-08-09*	Added –load option.
1.60	*2011-07-09*	Corrected closing communication channels.
1.59	*2011-06-15*	MY_MODULE_ADDR not declared here.
1.58	*2011-05-27*	First version with pipe communication support.
1.57	*2011-05-13*	Command line documentation, support for gauge_radar.
1.56	*2011-04-15*	Preparations for sockets implementation.
1.55	*2011-04-13*	Output is printed in case of RPC failure.
1.54	*2011-03-21*	Corrections related with the channel configuration dialog.
1.53	*2011-03-19*	Cleaned up.
1.52	*2011-03-19*	First version with channel edit dialog.
1.51	*2011-03-08*	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.
1.50	*2011-02-11*	Added map dump command line option.
1.49	*2011-02-07*	Initial support for adding gauges in runtime.
1.48	*2011-02-01*	Added support for –rpc-arg option.
1.47	*2011-01-08*	Added support for returnig values from RPC.
1.46	*2010-12-29*	Following changes: - added support for demo configuration; - added building of monitor DLL; - initial implementation of GAUGE_JOYSTICK.
1.45	*2010-12-05*	Moved uOS_mon_query_exec() before the iterators so the decimation gets / sets the value at the first iteration.
1.44	*2010-11-06*	First release with RPC support.
1.43	*2010-11-06*	Preparations for RPC implementation.
1.42	*2010-11-03*	Added support for bgcolor and fgcolor in GAUGE_GENERIC.
1.41	*2010-10-23*	Reverted the seti option modification.
1.40	*2010-10-22*	Added support for the visource property.
1.39	*2010-10-19*	More support for editing, corrected the seti command.
1.38	*2010-10-05*	Added support for Z-order.

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