

Project properties	
<b>Parameters</b>	<b>version:</b> s6.4, <b>unit:</b> millimeter, <b>stackSize:</b> 5000
<b>Program Files</b>	clean_message.pgx cmd_ack.pgx cmd_reboot.pgx cmd_stop.pgx comm_init.pgx crcl6.pgx dec2hex.pgx emergency_stop.pgx get_config.pgx get_state.pgx mov_pos_t_loop.pgx mov_pos_t_r_loo.pgx move_cur.pgx move_grip.pgx move_pos.pgx move_pos_loop.pgx move_pos_rel.pgx move_pos_rel_lo.pgx move_pos_t_rel.pgx move_pos_time.pgx move_vel.pgx read_buffer.pgx reference.pgx reference_hand.pgx send_message.pgx ser_target_curr.pgx set_target_jerk.pgx set_target_acc.pgx set_target_time.pgx set_target_vel.pgx start.pgx stop.pgx translate.pgx kill_cteni.pgx
<b>Data Files</b>	kom1.dtx
<b>Libraries</b>	io:io

Tool		
Public	Name	Father
	flange	

Frame		
Public	Name	Father
	world	

MDesc				
Public	Name		Public	Name
■	mNomSpeed			

Bool				
Public	Name		Public	Name
■	overrun		■	ready
■	busy			

Num				

<i>Public</i>	<i>Name</i>		<i>Public</i>	<i>Name</i>
	crc			message [20]
	tbl [256]			

<i>String</i>				
<i>Public</i>	<i>Name</i>		<i>Public</i>	<i>Name</i>
	sbirka [50]			

<i>Sio</i>				
<i>Public</i>	<i>Name</i>		<i>Public</i>	<i>Name</i>
	comport			

### *clean\_message ()*

Clear message array

num **i**

```
begin
  //Program sloužící pro přednaplnění pole message hodnotami >=256.
  //Na této skutečnosti je pak založeno rozlišování délky zprávy určené k ode
  slání.
  //Program je volán na začátku každé výstupní funkce.

  //clean_message()

  for i=0 to size(message)-1 step 1
    message[i]=256
  endFor
end
```

**cmd\_ack (num module\_address)**

Acknowledgement of a pending error message

```
num i
num pom1
num pom2
```

```
begin
//Program zastupující výstupní funkci "acknowledgement of a pending error m
essage".
//Tato funkce umožní odblokování modulu hlavice po vzniku závažnější chyby.
//Po správném naplnění výstupní zprávy se zavolá program, který zajistí jej
í odeslání.

//cmd_ack(module_address)
//module_address-adresa modulu, který má příkaz zpracovat

//vycistim si zpravu
call clean_message()
//prvni byte master to slave = 0x05
message[0]=5
//druhy byte adresa modulu predana v parametru funkce default v modulu je 0
x0C(12)
message[1]=module_address
//treti byte delka dat+prikaz
message[2]=1
//ctvrty byte prikaz error acknowledgement 0x8B(139)
message[3]=139
//crc16
call crc16(message,0,crc)
pom1=roundDown(crc/256)
pom2=crc-((roundDown(crc/256))*256)
message[4]=pom2
message[5]=pom1
call send_message(message)
end
```

### **cmd\_reboot (num module\_address)**

The module is restarted

num pom1

num pom2

```
begin
  //Program zastupující výstupní funkci "reboot".
  //Tato funkce umožní restart modulu.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí její odeslání.

  //cmd_reboot(module_address)
  //module_address-adresa modulu, který má příkaz zpracovat

  //vycistim si zpravu
  call clean_message()
  //prvni byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0
  x0C(12)
  message[1]=module_address
  //treti byte delka dat+prikaz
  message[2]=1
  //ctvrty byte prikaz reboot 0xE0(224)
  message[3]=224
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[4]=pom2
  message[5]=pom1
  call send_message(message)
end
```

### **cmd\_stop (num module\_address)**

The module is slowed down and held in the current position

num pom1

num pom2

```
begin
  //Program zastupující výstupní funkci "stop".
  //Tato funkce umožní zastavení modulu a setrvání v této pozici.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí její odeslání.

  //cmd_stop(module_address)
  //module_address-adresa modulu, který má příkaz zpracovat

  //vycistim si zpravu
  call clean_message()
  //první byte master to slave = 0x05
  message[0]=5
  //druhý byte adresa modulu předána v parametru funkce default v modulu je 0x0C(12)
  message[1]=module_address
  //třetí byte délka dat+příkaz
  message[2]=1
  //čtvrtý byte příkaz stop 0x91(145)
  message[3]=145
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[4]=pom2
  message[5]=pom1

  call send_message(message)
end
```

## **comm\_init (num module\_address)**

Initialize communication

```
begin
  //Program sloužící pro unicializaci práce s hlavicí. Program by měl být zavolán za začátku
  //uživatelského programu
  //Proměnná serial input/output je linknuta se sériovým portem 1 a poté jsou nastaveny
  //parametry sériové komunikace. Také je vyprázdněn vstupní buffer sériové linky. Dále jsou nastaveny příznaky
  //busy a ready na výchozí hodnoty po inicializaci. Je spuštěna úloha čtení ze sériové linky.
  //Dále se navazuje spojení dokud není navázáno tzn. příznak ready musí být roven true.

  //comm_init(module_address)
  //module_address-adresa modulu, který má být inicializován

  sioLink(comport,io:portSerial1)
  sioCtrl(comport,"mode","RS232")
  sioCtrl(comport,"baudRate",9600)
  sioCtrl(comport,"bits",8)
  sioCtrl(comport,"stopBits",1)
  sioCtrl(comport,"parity","none")
  sioCtrl(comport,"flowControl","none")
  sioCtrl(comport,"timeout",0)
  clearBuffer(comport)
  busy=true
  ready=false
  taskCreateSync "cteni",0.05,overrun,read_buffer()
  putln("Connection is being established")

  while(ready!=true)
    call cmd_ack(12)
    delay(1)
  endwhile

  call clean_message()
  putln("Connection is established")
end
```

## **crc16 (num &msg, num length, num &crc)**

Vypocet CRC16

```
num i
num pomlength
num temp
num temp1
num temp2
num temp3
string vysl[4]

begin
  //Program určí hodnotu CRC16-IBM(0x8005) vypočtenou ze zprávy &msg a uloží
  ji do proměnné &crc.
  //Pokud je zadaná délka length, počítá se s délkou zprávy zadané v tomto pa
  rameru. Pokud není zadaná(length=0)
  //délka zprávy se určí v cyklu for a uloží do proměnné pomlength se kterou
  dále pracuje program.
  //Určení délky zprávy je založeno na tom, že vstupní pole zprávy je přednap
  lněno hodnotou >=256.
  //Potom tedy údaj >=256 není považován za část zprávy. O přednaplnění pole
  se stará program clean_message,
  //který je volán bezparametrově v každém programu sloužícím pro výstupní op
  eraci.
  //Po určení délky je určen CRC v cyklu for, přičemž se pracuje s hodnotou v
  e zprávě po bytech a zároveň
  //s tabulkou hodnot CRC se kterou tento program pracuje. Tabulka je nadefin
  ována jako globální proměnná.
  //Index do tabulky se určuje pomocí daného výpočtu index = (crc AND 255)XOR
  (msg[i] AND 255)
  //Výpočet CRC16 pak tedy je (crc AND 65280)>>8) XOR tbl[(crc AND 255)XOR(ms
  g[i] AND 255)]

  // crc16(&message,length,&crc)
  // msg - zprava pro konverzi
  // length - delka zpravy (zadaná nebo vypočtená(length=0))
  // crc - proměnná pro výsledek CRC16
  //Určení délky zprávy.
  //Pro každou hodnotu zpávy <256 se délka zprávy zvětší o 1 nebo pokud je dé
  lka zadáná tak se pracuje se zadanou.
  pomlength=0
  if length==0
    for i=0 to size(msg)-1 step 1
      if msg[i]<256
        pomlength=pomlength+1
      else
        endIf
    endFor
  else
    pomlength=length
  endIf

  //Vlastní určení CRC16

  crc=0
  for i=0 to (pomlength-1) step 1
    temp=bAnd(crc,65280)/256
    temp1=bAnd(crc,255)
    temp2=bAnd(msg[i],255)
    temp3=bXor(temp1,temp2)
    crc=bXor(temp,tbl[temp3])
  endFor
end
```

## **dec2hex (num *cislo*, string &*prevod*)**

```
num i
num zbytek
string pom[4]

begin
  //Program slouží pro převod ze tvaru dekadického čísla na číslo se základem
  16
  //Vstupem je převáděné číslo v parametru cislo a výsledný převod je uložen
  do proměnné
  //string& prevod.
  //Výpočet je založen na postupném dělení základem 16.

  //dec2hex(cislo, &prevod)
  //cislo-převáděné číslo
  //prevod-výsledný řetězec

  prevod=""
  if cislo==0
    prevod="0"
  endIf
  while cislo>0
    zbytek=cislo-roundDown(cislo/16)*16
    cislo=roundDown(cislo/16)
    if zbytek>9
      zbytek=zbytek+55
    else
      zbytek=zbytek+48
    endIf
    prevod=prevod+chr(zbytek)
  endWhile

  // obrácení řetězce
  pom=""
  for i=len(prevod) to 0 step -1
    pom=pom+mid(prevod,1,i)
  endFor
  prevod=pom
end
```

### **emergency\_stop (num module\_address)**

The module is stopped as quickly as possible

num pom1

num pom2

```
begin
  //Program zastupující výstupní funkci "emergency stop".
  //Tato funkce umožní okamžité zastavení pohybu modulu, nejrychleji jak je možná a poté
  //se nastaví chyba emergency stop. Před další prací je nutné použít cmd_ack()
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí její odeslání.

  //emergency_stop(module_address)
  //module_address-adresa modulu, který má příkaz zpracovat

  //vycistim si zpravu
  call clean_message()
  //první byte master to slave = 0x05
  message[0]=5
  //druhý byte adresa modulu předána v parametru funkce default v modulu je 0x0C(12)
  message[1]=module_address
  //třetí byte délka dat+příkaz
  message[2]=1
  //čtvrtý byte příkaz emergency stop 0x90(144)
  message[3]=144
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[4]=pom2
  message[5]=pom1
  call send_message(message)
end
```

### **get\_config (num module\_address)**

A range of configuration data can be read from the module

num pom1

num pom2

```
begin
  //Program zastupující výstupní funkci "get config".
  //Tato funkce zajistí odeslání informačních dat z modulu jako typ modulu, v
  erze firmware atd.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí jej
  í odeslání.

  //get_config(module_address)
  //module_address-adresa modulu, který má příkaz zpracovat

  //vycistim si zprávu
  call clean_message()
  //prvni byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0
  x0C(12)
  message[1]=module_address
  //treti byte delka dat+prikaz
  message[2]=1
  //ctvrty byte prikaz get_config 0x80(128)
  message[3]=128
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[4]=pom2
  message[5]=pom1
  call send_message(message)
end
```

**get\_state (num module\_address, num time, num mode)**

Returns the module status

```
num i
num konverze[4]
num pom1
num pom2

begin
  //Program zastupující výstupní funkci "get state".
  //Tato funkce zajistí, že modul odešle svůj status.
  //Status obsahuje: pokud je funkce volána bez parametru tj. time=0 a mode=0
  //2 bytovou informaci, kde každý bit má svůj význam
  //Bit 1-module is refenced, Bit 2-module is moving atd. viz motioncontrol.pd
  f->get state
  //Přitom je tato informace zaslána pouze jednou. Pokud je specifikován para
  metr time
  //informace se odesílá v intervalu daném tímto parametrem.
  //Pokud je specifikován parametr mode, tak se navíc odesílají informace o p
  ozici, rychlosti a proudu.
  //Podle specifikace parametru mode=1 jen pozice mode=2 jen rychlost mode=4
  jen proud.
  //Jejich součty získáme kombinace např mode=7 vše
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí jej
  í odeslání.

  //get_state(module_address,time,mode)
  //module_address-adresa modulu, který má příkaz zpracovat
  //time-čas cyklu odesílání
  //mode-mód přidavných informací viz. popis

  //vycistim si zpravu
  call clean_message()
  //prvni byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0
  x0C(12)
  message[1]=module_address
  //podle zadaneho paramtru bez paramtru=1, s parametrem time=5, s parametrem
  time+mode=6

  if mode==0 and time==0
    message[2]=1
    //ctvrty byte prikaz get_state 0x95(149)
    message[3]=149
    //crc16
    call crc16(message,0,crc)
    pom1=roundDown(crc/256)
    pom2=crc-((roundDown(crc/256))*256)
    message[4]=pom2
    message[5]=pom1

  elseif mode==0 and time!=0
    message[2]=5
    //ctvrty byte prikaz get_state 0x95(149)
    message[3]=149
    //data
    toBinary(time,size(time),"4.01",konverze)
    for i=0 to 3 step 1
      message[4+i]=konverze[i]
    endFor
    //crc16
    call crc16(message,0,crc)
    pom1=roundDown(crc/256)
    pom2=crc-((roundDown(crc/256))*256)
    message[8]=pom2
    message[9]=pom1
  elseif mode!=0 and time!=0
    message[2]=6
    //ctvrty byte prikaz get_state 0x95(149)
```

```
message[3]=149
//data
toBinary(time,size(time),"4.01",konverze)
for i=0 to 3 step 1
  message[4+i]=konverze[i]
endFor
message[8]=mode
//crc16
call crc16(message,0,crc)
pom1=roundDown(crc/256)
pom2=crc-((roundDown(crc/256))*256)
message[9]=pom2
message[10]=pom1
endIf

call send_message(message)

end
```

**kill\_cteni ()**

```
Kill task ,,cteni``
```

```
begin  
  taskKill("cteni")  
end
```

**mov\_pos\_t\_loop (num module\_address, num position\_mm)**

The module is moving in loop between the start position and selected position  
(move\_pos\_time)

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program zastupující výstupní funkci "move position time loop".
  //Tato funkce umožní provádění cyklického pohybu mezi výchozí a zadanou pozici.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí její odeslání.

  //move_position_time_loop(module_address,position_mm)
  //module_address-adresa modulu, který má příkaz zpracovat
  //position_mm-pozice stisku

  //vycistim si zprávu
  call clean_message()
  //prvni byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0x0C(12)
  message[1]=module_address
  //treti byte delka dat+prikaz, zde bude vzdy 5(prikaz 1+4 single precision floating point, little endian = 5)
  message[2]=5
  //ctvrty byte prikaz move_pos_time_loop 0xBB(187)
  message[3]=187
  //data
  toBinary(position_mm,size(position_mm),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1
  busy=true
  call send_message(message)

end
```

**mov\_pos\_t\_r\_loo (num module\_address, num position\_mm)**

Cyclic move\_pos\_t\_rel

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program zastupující výstupní funkci "move position time relatively loop".
  //Tato funkce umožní provádění cyklyckého pohybu mezi výchozí a relativně z
  //adanou pozicí .
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí jej
  //í odeslání.

  //move_position_time_relatively_loop(module_address,position_mm)
  //module_address-adresa modulu, který má příkaz zpracovat
  //position_mm-pozice stisku

  //vycistim si zprávu
  call clean_message()
  //prvni byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0
  //x0C(12)
  message[1]=module_address
  //treti byte delka dat+prikaz, zde bude vzdy 5(prikaz 1+4 single precision
  //floating point, little endian = 5)
  message[2]=5
  //ctvrty byte prikaz move_pos_time_relatively_loop 0xBD(189)
  message[3]=189
  //data
  toBinary(position_mm,size(position_mm),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1

  busy=true
  call send_message(message)
end
```

**move\_cur (num module\_address, num current\_A)**

A current movement is completed

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program zastupující výstupní funkci "move with current".
  //Tato funkce umožní stisk s požadovaným proudem(+je otevírání, -stisk).Po
  dosažení proudu je pohyb ukončen.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí jej
  í odeslání.

  //move_current(module_address,current_A)
  //module_address-adresa modulu, který má příkaz zpracovat
  //position_mm-proud stisku

  //vycistim si zpravu
  call clean_message()
  //prvni byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0
  x0C(12)
  message[1]=module_address
  //treti byte delka dat+prikaz, zde bude vzdy 5(prikaz 1+4 single precision
  floating point, little endian = 5)
  message[2]=5
  //ctvrty byte prikaz move_current 0xB3(179)
  message[3]=179
  //data
  toBinary(current_A,size(current_A),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1

  busy=true
  call send_message(message)
end
```

**move\_grip (num module\_address, num grip\_A)**

Execute grip with current

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program zastupující výstupní funkci "move grip".
  //Tato funkce umožní stisk s požadovaným proudem(+je otevírání, -stisk).Sti
sk je prováděn dokud není zastaven.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí jej
í odeslání.

  //move_grip(module_address,grip_A)
  //module_address-adresa modulu, který má příkaz zpracovat
  //position_mm-proud stisku

  //vycistim si zpravu
  call clean_message()
  //prvni byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0
x0C(12)
  message[1]=module_address
  //treti byte delka dat+prikaz, zde bude vzdy 5(prikaz 1+4 single precision
floating point, little endian = 5)
  message[2]=5
  //ctvrty byte prikaz move_grip 0xB7(183)
  message[3]=183
  //data
  toBinary(grip_A,size(grip_A),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1

  busy=true
  call send_message(message)
end
```

**move\_pos (num module\_address, num position\_mm)**

Move to a defined position

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program zastupující výstupní funkci "move position".
  //Tato funkce umožní pohyb na danou pozici.Pohyb je dokončen po dosažení po
  zice.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí jej
  í odeslání.

  //move_positon(module_address,position_mm)
  //module_address-adresa modulu, který má příkaz zpracovat
  //position_mm-pozice stisku

  //vycistim si zpravu
  call clean_message()
  //prvni byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0
  x0C(12)
  message[1]=module_address
  //treti byte delka dat+prikaz, zde bude vzdy 5(prikaz 1+4 single precision
  floating point, little endian = 5)
  message[2]=5
  //ctvrty byte prikaz move_pos 0xB0(176)
  message[3]=176
  //data
  toBinary(position_mm,size(position_mm),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1
  busy=true
  call send_message(message)
end
```

### **move\_pos\_loop (num module\_address, num position\_mm)**

The module is moving in loop between the start position and selected position (move\_pos)

```
num i
num konverze[4]
num pom1
num pom2

begin
  //Program zastupující výstupní funkci "move position loop".
  //Tato funkce umožní provádění cyklického pohybu mezi výchozí a zadanou pozicí.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí její odeslání.

  //move_position_loop(module_address,position_mm)
  //module_address-adresa modulu, který má příkaz zpracovat
  //position_mm-pozice stisku

  //vycistim si zprávu
  call clean_message()
  //první byte master to slave = 0x05
  message[0]=5
  //druhý byte adresa modulu předána v parametru funkce default v modulu je 0x0C(12)
  message[1]=module_address
  //třetí byte délka dat+příkaz, zde bude vždy 5(příkaz 1+4 single precision floating point, little endian = 5)
  message[2]=5
  //čtvrtý byte příkaz move_pos_loop 0xBA(186)
  message[3]=186
  //data
  toBinary(position_mm,size(position_mm),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1

  busy=true
  call send_message(message)
end
```

**move\_pos\_rel (num module\_address, num position\_mm)**

The module is moved relativ from start position

```
num i
num konverze[4]
num pom1
num pom2

begin
  //Program zastupující výstupní funkci "move position relatively".
  //Tato funkce umožní pohyb na danou pozici.Pohyb je dokončen po dožení pozice.Pozice je dána relativně
  //od výchozí pozice
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí její odeslání.

  //move_positon_relatively(module_address,position_mm)
  //module_address-adresa modulu, který má příkaz zpracovat
  //position_mm-pozice stisku

  //vycistim si zpravu
  call clean_message()
  //prvni byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0x0C(12)
  message[1]=module_address
  //treti byte delka dat+prikaz, zde bude vzdy 5(prikaz 1+4 single precision floating point, little endian = 5)
  message[2]=5
  //ctvrty byte prikaz move_pos_relatively 0xB8(184)
  message[3]=184
  //data
  toBinary(position_mm,size(position_mm),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1

  busy=true
  call send_message(message)
end
```

**move\_pos\_rel\_lo (num module\_address, num position\_mm)**

Cyclic move\_pos\_rel

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program zastupující výstupní funkci "move position relatively loop".
  //Tato funkce umožní provádění cyklického pohybu mezi výchozí a zadanou pozicí. Pozice je dána relativně.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí její odeslání.

  //move_position_relatively_loop(module_address, position_mm)
  //module_address-adresa modulu, který má příkaz zpracovat
  //position_mm-pozice stisku

  //vycistim si zpravu
  call clean_message()
  //prvni byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0x0C(12)
  message[1]=module_address
  //treti byte delka dat+prikaz, zde bude vzdy 5(prikaz 1+4 single precision floating point, little endian = 5)
  message[2]=5
  //ctvrty byte prikaz move_pos_relatively_loop 0xBC(188)
  message[3]=188
  //data
  toBinary(position_mm, size(position_mm), "4.01", konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message, 0, crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1

  busy=true
  call send_message(message)
end
```

**move\_pos\_t\_rel (num module\_address, num position\_mm)**

The module is moved relatively from start position. If the time parameter value is entered, the velocity and acceleration are adjusted in such a way that the position is reached within the specified time

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program zastupující výstupní funkci "move position time relatively".
  //Tato funkce umožní pohyb v na pozici v zadaném čase.Pozice je dána relativně.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí její odeslání.

  //move_position_time_relatively(module_address,position_mm)
  //module_address-adresa modulu, který má příkaz zpracovat
  //position_mm-pozice stisku

  //vycistim si zpravu
  call clean_message()
  //první byte master to slave = 0x05
  message[0]=5
  //druhý byte adresa modulu předána v parametru funkce default v modulu je 0x0C(12)
  message[1]=module_address
  //třetí byte délka dat+příkaz, zde bude vždy 5(příkaz 1+4 single precision floating point, little endian = 5)
  message[2]=5
  //čtvrtý byte příkaz move_pos_time_relatively 0xB9(185)
  message[3]=185
  //data
  toBinary(position_mm,size(position_mm),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1

  busy=true
  call send_message(message)
end
```

### **move\_pos\_time (num module\_address, num position\_mm)**

The module moves to fixed position. If the time parameter value is entered, the velocity and acceleration are adjusted in such a way that the position is reached within the specified time

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program zastupující výstupní funkci "move position time".
  //Tato funkce umožní pohyb na zadanou pozici v daném čase.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí její odeslání.

  //move_position_time(module_address,position_mm)
  //module_address-adresa modulu, který má příkaz zpracovat
  //position_mm-pozice stisku

  //vycistim si zprávu
  call clean_message()
  //první byte master to slave = 0x05
  message[0]=5
  //druhý byte adresa modulu předána v parametru funkce default v modulu je 0x0C(12)
  message[1]=module_address
  //třetí byte délka dat+příkaz, zde bude vždy 5(příkaz 1+4 single precision floating point, little endian = 5)
  message[2]=5
  //čtvrtý byte příkaz move_pos_time 0xB1(177)
  message[3]=177
  //data
  toBinary(position_mm,size(position_mm),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1
  busy=true
  call send_message(message)
end
```

**move\_vel (num module\_address, num velocity)**

A velocity movement is completed

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program zastupující výstupní funkci "move velocity".
  //Tato funkce umožní pohyb zadanou rychlostí.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí její odeslání.

  //move_velocity(module_address,velocity)
  //module_address-adresa modulu, který má příkaz zpracovat
  //velocity-zadaná rychlost pohybu

  //vycistim si zpravu
  call clean_message()
  //první byte master to slave = 0x05
  message[0]=5
  //druhý byte adresa modulu předána v parametru funkce default v modulu je 0x0C(12)
  message[1]=module_address
  //třetí byte délka dat+příkaz, zde bude vždy 5(příkaz 1+4 single precision floating point, little endian = 5)
  message[2]=5
  //čtvrtý byte příkaz move_velocity 0xB5(181)
  message[3]=181
  //data
  toBinary(velocity,size(velocity),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1

  busy=true
  call send_message(message)
end
```

## read\_buffer ()

Read data from buffer

```
num i
num input[100]
num j
num pole_data[20]
num pom1
num pom2
string prevod[4]

begin
  //Program sloužící pro cyklické čtení příchozích zpráv.
  //Pokud ho chceme používat musí se spustit jako sychronní úloha na začátku
  programu např. takto
  //taskCreateSync "cteni",0.05,overrun,read_buffer()
  //Program pak běží pod vlákem cteni a spouští se každých 50ms
  //Před koncem programu voláme taskKill("cteni")
  //Z bufferu se čte tak dlouho, dokud není vše přečteno, poté je vyprázdněn.
  //Zároveň dochází k výpisu přijatých zpráv do konzole uživatele
  //Po zpracování každé jednotlivé zprávy se volá funkce translate(&data) kte
  rá umožní větvení pro každou
  //přijatou zprávu a tedy jejich individuální zpracování
  //Vnitřní cyklus proběhne tolikrát kolik máme přijatých zpráv. Celý program
  je v nekonečném cyklu while(true)
  //kvůli cyklickému spouštění sychronní úlohy

  while(true)
pom1=0
pom2=0
pom1=sioGet(comport,input)
  if pom1!=0
//cls()
//put("Nacteno:")
//put(pom1)
//putln(" ")
//dokud jsou neprečtena data budu cist
while(pom1>pom2)
  call dec2hex(input[pom2],prevod)
  put(prevod)
  pom2=pom2+1
  put(" ")
  call dec2hex(input[pom2],prevod)
  put(prevod)
  pom2=pom2+1
  put(" ")
  j=0

  for i=input[pom2]+2 to 0 step -1
    //pole sloužící pro funkci translate bude obsahovat delku dat+command+dat
a+crc
    pole_data[j]=input[pom2]
    call dec2hex(input[pom2],prevod)
    put(prevod)
    put(" ")
    pom2=pom2+1
    j=j+1
  endFor
  putln(" ")
  call translate(pole_data)

endWhile
  clearBuffer(comport)
  endIf
endWhile
end
```

### **reference (num module\_address)**

A reference movement is completed

num pom1

num pom2

```
begin
  //Program zastupující výstupní funkci "reference run".
  //Tato funkce umožní referenční pohyb.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí její odeslání.

  //reference(module_address)
  //module_address-adresa modulu, který má příkaz zpracovat

  //vycistim si zprávu
  call clean_message()
  //první byte master to slave = 0x05
  message[0]=5
  //druhý byte adresa modulu předána v parametru funkce default v modulu je 0x0C(12)
  message[1]=module_address
  //třetí byte délka dat+příkaz
  message[2]=1
  //čtvrtý byte příkaz reference 0x92(146)
  message[3]=146
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[4]=pom2
  message[5]=pom1
  //zkouška naplnění pole
  //for i=0 to size(message)-1 step 1
  //if message[i]<256
  //putln(message[i])
  // else
  //endif
  // endFor
  busy=true
  call send_message(message)
end
```

### **reference\_hand (num module\_address)**

A manual referencing activity is executed

num pom1

num pom2

```
begin
  //Program zastupující výstupní funkci "reference hand".
  //Tato funkce umožní manuální referenční pohyb.
  //Po správném naplnění výstupní zprávy se zavolá program, který zajistí její odeslání.

  //reference_hand(module_address)
  //module_address-adresa modulu, který má příkaz zpracovat

  //vycistim si zprávu
  call clean_message()
  //první byte master to slave = 0x05
  message[0]=5
  //druhý byte adresa modulu předána v parametru funkce default v modulu je 0x0C(12)
  message[1]=module_address
  //třetí byte délka dat+příkaz
  message[2]=1
  //čtvrtý byte příkaz reference 0x97(151)
  message[3]=151
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[4]=pom2
  message[5]=pom1
  //zkouška naplnění pole
  //for i=0 to size(message)-1 step 1
  //if message[i]<256
  //putln(message[i])
  // else
  //endif
  // endFor

  busy=true
  call send_message(message)
end
```

### **send\_message (num &out\_mess)**

Sends message over serial port.

```
num i
num length
num pole10[10]
num pole11[11]
num pole6[6]

begin
  //Program zajistí odeslání výstupní zprávy
  //Na základě délky zprávy se program větví, aby mohla být zpráva správně od-
  //eslána.
  //Po zjištění délky stejným způsobem jako při výpočtu CRC16 je zpráva odesl-
  //ána.

  //send_message(output_message)

  //zjistim delku zpravy
  length=0

  for i=0 to size(out_mess)-1 step 1
    if out_mess[i]<256
      length=length+1
    else
      endIf
  endFor
  //putln(length)
  //odeslu podle delky zpravy

  switch length
    case 6
      for i=0 to length-1 step 1
        pole6[i]=out_mess[i]
      endFor
      sioSet(comport,pole6)
    break
    case 10
      for i=0 to length-1 step 1
        pole10[i]=out_mess[i]
      endFor
      sioSet(comport,pole10)
    break
    case 11
      for i=0 to length-1 step 1
        pole11[i]=out_mess[i]
      endFor
      sioSet(comport,pole11)
    break
    default
      putln("Nekonzistentni zprava")
    break
  endSwitch
  delay(0.1)
end
```

**ser\_target\_curr (num module\_address, num current)**

The current parametr is now set

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program sloužící pro nastavení požadovaného proudu, při vykonávání pohybo-
  vých funkcí.
  //Po správném sestavení zprávy je tato odeslána do modulu.

  //set_target_current(module_address,current)
  //module_address-adresa modulu, který má příkaz zpracovat
  //current-zadaný proud

  //vycistim si zprávu
  call clean_message()
  //první byte master to slave = 0x05
  message[0]=5
  //druhý byte adresa modulu předána v parametru funkce default v modulu je 0
  x0C(12)
  message[1]=module_address
  //třetí byte délka dat+příkaz, zde bude vždy 5(příkaz 1+4 single precision
  floating point, little endian = 5)
  message[2]=5
  //čtvrtý byte příkaz set_target_current 0xA3(163)
  message[3]=163
  //data
  toBinary(current,size(current),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1

  call send_message(message)
end
```

**set\_target\_jerk (num module\_address, num jerk)**

The jerk parametr is now set

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program sloužící pro nastavení požadovaného trhnutí, při vykonávání pohyb
  ových funkcí.
  //Po správném sestavení zprávy je tato odeslána do modulu.

  //set_target_jerk(module_address,jerk)
  //module_address-adresa modulu, který má příkaz zpracovat
  //jerk-zádané trhnutí

  //vycistim si zpravu
  call clean_message()
  //prvni byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0
  x0C(12)
  message[1]=module_address
  //treti byte delka dat+prikaz, zde bude vzdy 5(prikaz 1+4 single precision
  floating point, little endian = 5)
  message[2]=5
  //ctvrty byte prikaz set_target_jerk 0xA2(162)
  message[3]=162
  //data
  toBinary(jerk,size(jerk),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1

  call send_message(message)
end
```

**set\_target\_acc (num module\_address, num acceleration)**

The acceleration parametr is now set

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program sloužící pro nastavení požadovaného zrychlení, při vykonávání pohybových funkcí.
  //Po správném sestavení zprávy je tato odeslána do modulu.

  //set_target_acceleration(module_address,acceleration)
  //module_address-adresa modulu, který má příkaz zpracovat
  //acceleration-zadané zrychlení

  //vycistim si zprávu
  call clean_message()
  //první byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0x0C(12)
  message[1]=module_address
  //treti byte delka dat+prikaz, zde bude vzdy 5(prikaz 1+4 single precision floating point, little endian = 5)
  message[2]=5
  //ctvrty byte prikaz set_target_acceleration 0xA1(161)
  message[3]=161
  //data
  toBinary(acceleration,size(acceleration),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1

  call send_message(message)
end
```

**set\_target\_time (num module\_address, num time)**

The time parametr is now set

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program sloužící pro nastavení požadovaného času, při vykonávání pohybových
  //funkcí.
  //Po správném sestavení zprávy je tato odeslána do modulu.

  //set_target_time(module_address,time)
  //module_address-adresa modulu, který má příkaz zpracovat
  //time-čas, který má být dosažen

  //vycistim si zpravu
  call clean_message()
  //prvni byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0
  //x0C(12)
  message[1]=module_address
  //treti byte delka dat+prikaz, zde bude vzdy 5(prikaz 1+4 single precision
  //floating point, little endian = 5)
  message[2]=5
  //ctvrty byte prikaz set_target_time 0xA4(164)
  message[3]=164
  //data
  toBinary(time,size(time),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1

  call send_message(message)
end
```

**set\_target\_vel (num module\_address, num velocity)**

The velocity parametr is now set

```
num i
num konverze[4]
num pom1
num pom2
```

```
begin
  //Program sloužící pro nastavení požadované rychlosti, při vykonávání pohyb
  ových funkcí.
  //Po správném sestavení zprávy je tato odeslána do modulu.

  //set_target_velocity(module_address,velocity)
  //module_address-adresa modulu, který má příkaz zpracovat
  //velocity-požadovaná rychlost

  //vycistim si zpravu
  call clean_message()
  //prvni byte master to slave = 0x05
  message[0]=5
  //druhy byte adresa modulu predana v parametru funkce default v modulu je 0
  x0C(12)
  message[1]=module_address
  //treti byte delka dat+prikaz, zde bude vzdy 5(prikaz 1+4 single precision
  floating point, little endian = 5)
  message[2]=5
  //ctvrty byte prikaz set_target_velocity 0xA0(160)
  message[3]=160
  //data
  toBinary(velocity,size(velocity),"4.01",konverze)
  for i=0 to 3 step 1
    message[4+i]=konverze[i]
  endFor
  //crc16
  call crc16(message,0,crc)
  pom1=roundDown(crc/256)
  pom2=crc-((roundDown(crc/256))*256)
  message[8]=pom2
  message[9]=pom1

  call send_message(message)
end
```

**start ()**

```
bool bOk
num cislo
num i
num konverze[4]
string prevod[8]
string vysl[2]

begin
// cls()
// //taskCreateSync "cteni",0.05,overrun,read_buffer()
// putln("Zacinam")
// call comm_init(12)
// //call clean_message()
// //cls()
// //call cmd_ack(12)
// //delay(1)
// if(ready==true)

// if(busy!=true)
// call move_pos(12,14)
// endIf
// //delay(5)
// //call move_grip(12,-1)
// if(busy!=true)
// call move_pos(12,7)
// else
// putln("Busy")
// endIf

// delay(5)
// if(busy!=true)
// call move_pos(12,0.1)
// endIf
// delay(5)
// //call read_buffer()
// //delay(1)
// call emergency_stop(12)
// // putln("Koncim")
// taskKill("cteni")
// endIf
// //call read_buffer()
// //io:portSerial1 = "ssshjshjsh"
// //cislo=10
// //putln("Prevod cisla na IEEE 754:")
// //putln(toString("8.16",cislo))
// // putln(size(cislo))
// //putln(size(konverze))
// // // single precision floating point, little endian
// //toBinary(cislo,size(cislo),"4.01",konverze)
// //s=toString("",konverze[0])+" "+toString("",konverze[1])+" "+toString("",
konverze[2])+" "+toString("",konverze[3])
// //putln("zacatek")
// //putln(konverze[0])
// //putln(konverze[1])
// //putln(konverze[2])
// //putln(konverze[3])
// //putln("konec")
// //prevod=""
// //call dec2hex(konverze[0],vysl)
// //prevod=prevod+vysl+" "
// //call dec2hex(konverze[1],vysl)
// //prevod=prevod+vysl+" "
// //call dec2hex(konverze[2],vysl)
// //prevod=prevod+vysl+" "
// //call dec2hex(konverze[3],vysl)
// //prevod=prevod+vysl
// //putln("Hexa:")
```

```

// //putln(prevod)
// //putln("a zpet na cislo")
// //fromBinary(konverze,4,"4.01",cislo)
// //putln(toString("8.16",cislo))
// // prevod = toString("8.16",cislo)
// // toNum(prevod,cislo,bOk)
// // putln(toString("8.16",cislo))
// //cislo=55785
// //toBinary(cislo,2,"21",konverze)
// //prevod=""
// //call dec2hex(konverze[0],vysl)
// //prevod=prevod+vysl+" "
// //call dec2hex(konverze[1],vysl)
// //prevod=prevod+vysl+" "
// //putln("Hexa:")
// //putln(prevod)
// //cls()
// // for cislo=0 to size(konverze) step 1
// //   putln(konverze[cislo])
// // endFor
// //   putln(size(konverze))
// // 0x07, 0x01, 0x05, 0xB0, 0xEE, 0xEE, 0x56, 0x40 // 0xE47B
// //message[0]= 7
// //message[1]=1
// //message[2]=5
// //message[3]=176
// //message[4]=238
// //message[5]=238
// //message[6]=86
// //message[7]=64
// //cls()
// //naplneni zpravy hodnotami vetsimi nez 256 aby bylo mozne urcit pocet by
tu
// //naplnim hodnotami
// //message[0]=5
// //message[1]=1
// //message[2]=1
// //message[3]=146
// //putln("Velikost pole je: ")
// //putln(size(message))
// //call crc16(message,0,crc)
// //putln(crc)
// //prevod=""
// //call dec2hex(crc,vysl)
// //prevod=prevod+vysl+" "
// //putln("crc16 je:")
// //putln(prevod)
// //call cmd_ack(12)
end

```

***stop ()***

```
begin
  //popupMsg("Pending movement commands have been canceled")
  resetMotion()
end
```

## **translate (num &data)**

Translate input message

```
num i
num prelozeno
num prevod[4]

begin
  //Program sloužící pro zpracování příchozích zpráv.
  //Umožní větvení podle typu příchozí zprávy
  //Vstupním parametrem je pole dat kdy na začátku je typ příkazu zbytek jsou
  data. Množství dat
  //je dáno typem příkazu

  //translate(&data)
  //data-přijatý příkaz + data

  switch data[1]
  //CMD ERROR
  case 136
    put(sbirka[9])
    if data[2]==217
      putln(sbirka[10])
    endif
    break
  case 138
    put(sbirka[2])
    if data[2]==8 and data[3]==0
      putln(sbirka[3])
    endif
    break
  case 139
    put(sbirka[0])
    if data[2]==79 and data[3]==75
      putln(sbirka[1])
      ready=true
      busy=false
    endif
    break
  case 147
    put(sbirka[4])
    for i=0 to 3 step 1
      prevod[i]=data[i+2]
    endfor
    fromBinary(prevod,size(prevod),"4.01",prelozeno)
    put(prelozeno)
    putln(sbirka[6])
    busy=false
    break
  case 148
    put(sbirka[8])
    for i=0 to 3 step 1
      prevod[i]=data[i+2]
    endfor
    fromBinary(prevod,size(prevod),"4.01",prelozeno)
    put(prelozeno)
    putln(sbirka[6])
    busy=false
    break
  case 176
    put(sbirka[5])
    for i=0 to 3 step 1
      prevod[i]=data[i+2]
    endfor
    fromBinary(prevod,size(prevod),"4.01",prelozeno)
    put(prelozeno)
    putln(sbirka[7])
    break
  default
    putln("Neumim prelozit")
```

```
break
endSwitch
end
```