POPIS FUNKCÍ KNIHOVNY
PDriver.dll

Obsah:

Úvod

Tato knihovna obstarává veškerou práci s panely, může obsluhovat až 255 panelů. Při inicializaci je vytvořeno samostatné vlákno, které na pozadí provádí čtení, ukládání, formátování a podobně.

V programu, který tuto knihovnu použije, je nutné zajistit aby nebyly volány dvě funkce současně (pokud jsou volány z více vláken). V opačném případě může dojít ke zhroucení programu.

Funkce, které běží v rámci knihovny ve vlákně, se spustí voláním příslušné funkce, která pouze spustí operaci ve vlákně a poté je ukončena. Průběh provádění je možné monitorovat funkcí „***pdControl_fnc***". Pokud je spuštěna některá funkce na pozadí a je volána jiná funkce běhu na pozadí, je vrácen chybový kód **ERR_THREAD_BUSY** (vlákno je obsazeno). V průběhu provádění na pozadí je možné volat jakoukoliv jinou funkci komunikace s panelem. Funkce na pozadí jsou v tabulkách zvýrazněny žlutě. Na konci je souhrn všech funkcí na pozadí.

Veškeré chyby jsou zaznamenány do souboru **PDriver.log** který je uložen ve stejném direktoráři jako tato knihovna.

Parametr int cmd_id ve všech funkcích udává kód funkce a id panelu
Spodních 8 bitů udává číslo panelu, maska je 0x000000FF
Horních 24 bitů udává kód funkce.

INICIALIZACE A UKONČENÍ

Před použitím funkci je nutné knihovnu nejprve inicializovat, tímto krokem se vytvoří a inicializují všechny základní proměnné a funkce. Dále je vytvořeno vlákno, které umožňuje komunikaci s panelem na pozadí. Při inicializaci je načten soubor „command.src", který obsahuje text všech příkazů zobrazovacího programu.

HANDLE pdCreateClient( );

Před ukončení programu je pak nutné funkce a proměnné uzavřít. Při uzavření jsou zrušeny všechny proměnné, které byli vytvořeny v průběhu práce, také je ukončeno vlákno panelu.

void pdCloseClient(HANDLE client);

Parametr „***HANDLE client***" je pro použití v **COM** objektech, Pro každý program, který COM použije se vytvoří samostatné rozhraní pro komunikaci s panely. Současná verze toto neumožňuje a parametr se zadává nulový „***NULL***"
Funkce neprovádí odebrání panelů, to se musí provést předem funkcí **pdArrData** s parametrem CMD_ARRDATA_RAMOVE_ALL

Dalším krokem po inicializaci je načtení nastavení a jazykové mutace. Všechny tyto operace lze provádět i v průběhu práce s knihovnou, samozřejmě pokud se načítá nastavení, musí se zastavit všechna komunikace s panelem. Funkce totiž nejprve zruší nastavení všech panelů a pak je opět načte ze souboru Pokud by byla zároveň prováděna komunikace s panelem, dojde k zhroucení programu.

int pdFiles(int cmd, char *nfile, int size);

Definice cmd	Kód	Popis
CMD_FILES_SETPANELS_LOAD	0x00000100	Funkce provede načtení souboru s nastavením, pokud je funkce volaná, opětovně neprovádí nic.
CMD_FILES_SETPANELS_RELOAD	0x00000200	Zruší nastavení a načte jej znova ze souboru.
CMD_FILES_SETPANELS_SAVE	0x00000400	Uložení nastavení do souboru.
CMD_FILES_SET_LANGUAGE	0x00000800	Nastavení jazyka.
CMD_FILES_SET_PATHGP2	0x00001000	V současné verzi nevyužito.

char *nfile – pointer na pole char ve kterém je řetězec se jménem souboru včetně direktoráře.
int size – velikost datového pole řetězce.

```
void Init( )
{
      char cstr[255];
            // inicializace PDriver.dll
      pdCreateClient( );
            // načtení nastaveni jméno musí být včetně direktoráře
      strcpy( cstr "C:\PDriver.set" );
      pdFiles( CMD_FILES_SETPANELS_LOAD, cstr, 255 );
            // načtení stringů v češtině (czech_PDriver.ln)
      strcpy( cstr "czech" );
      pdFiles( CMD_FILES_SET_LANGUAGE, cstr, 255 );
}
```

Ze jména jazyka funkce nejprve vytvoří jméno souboru včetně direktoráře a pak tento soubor načte. Jazykový soubor obsahuje texty chybových hlášení. Pokud tato

hlášení nebudete využívat není nutné je načítat. V případě že tuto funkci použijeme bez načteného jazykového souboru vrátí text „Chyba n“, za ‘n‘ bude dosazeno číslo chyby.
Formát jména souboru je: „**\Language\%s_PDriver.ln**“
Je-li PDriver.dll přímo v C:\ bude vypadat takto: „**C:\Language\czech_PDriver.ln**“
Popis formátu jazykového souboru naleznete v dokumentaci „***Jazykové mutace programu Panel II***“

```
void Destroy( )
{
      char cstr[255];
            // uložení nastaveni jméno musí být včetně direktoráře
      strcpy( cstr "C:\PDriver.set" );
      pdFiles( CMD_FILES_SETPANELS_SAVE, cstr, 255 );
            // vyjmutí všech panelů
      pdArrData( CMD_ARRDATA_RAMOVE_ALL, NULL, 0 );
            // ukončení PDriver.dll
      pdCloseClient( NULL );
}
```

Do souboru s nastavením jsou uložena následující data ze záznamu DataPanel. Uložení nastavení je možné provádět při ukončení programu, nebo vždy po změně nastavení.

```
struct szData
{
      char   szName[20];          // jméno panelu
      char   szPort;              // sériový port, na který je panel napojen
      char   szAdres;             // adresa panelu
      char   szSizeBuf;           // Délka komunikačního paketu 32
      char   szLine;              // počet řádků panelu
      char   szColum;             // počet znaků na řádek
      char   szDot;               // počet bodů na výšku
      COLORREF szColorRow[10];    // barvy řádku
      COLORREF szColorBack;       // barva pozadí řádku
      char   szTypePanel;         // typ panelu - index verze firmware
      char   szFree1;             // volné
      bool   szOnCRC;             // zapnutí kontrolního součtu
      char   szPhoneNumber[20];   // telefonní číslo
      char   szIpAdres[16];       // IP adresa
      char   szSegment[5];        // velikost segmentů na řádku
      char   szSpace;
      BYTE   szDefaultLS;
      char   szFree2[7];          // volné
      int    szIpPort;            // IP port
      char   szGSM;               // povoleni GSM modemu 0/1 ne/ano
      char   szFree4;             // volné
}szData;
```

SPRÁVA PANELŮ A NASTAVENÍ

Pro každý panel se vytvoří samostatné datové pole, ve kterém se uchovává nastavení a data související s připojeným panelem. Všechny panely jsou řazeny do seznamu, kde je každý panel reprezentován číslem (index – id)

```
int  pdArrData( int cmd_id, char *name, int iParam );
```

Definice cmd	Kód	Popis
CMD_ARRDATA_PANEL_INSERT	0x03000100	Vložení nového panelu na pozici danou číslem panelu v cmd_id
CMD_ARRDATA_PANEL_ADD	0x03000200	Přidání nového panelu na konec
CMD_ARRDATA_PANEL_REMOVE	0x03000400	Odebrání panelu s indexem v cmd_id name – NULL, iParam - 0
CMD_ARRDATA_PANEL_RAMOVE_ALL	0x03000800	Odebrání všech panelů name – NULL, iParam – 0
CMD_ARRDATA_PANEL_COUNT	0x03001000	Vrací Počet definovaných panelů name – NULL, iParam – 0
CMD_ARRDATA_SEARCH_NAME	0x03002000	Vrací id panelu podle jména, iParam – 0
CMD_ARRDATA_DEFAULT_FONT	0x03004000	Nastaví default fonty pro simulátor name – NULL, iParam – 0
CMD_ARRDATA_GET_PANELNAME	0x03004000	Vrací jméno panelu podle jeho id iParam – velikost pole name

***char** *name* -* pointer na jméno panelu (pole char o velikosti 21 bajtů).
***int iParam* -* bit 0 – 8 číslo COM portu, bit 9 – 16 adresa panelu

Funkce vrací stávající počet panelů po provedení požadované akce. V případě že operaci nelze provést nebo panel neexistuje vrací „-1".

Použití příkazu **CMD_ARRDATA_DEFAULT_FONT** nemá vliv na funkčnost knihovny, pouze nastaví do *„DataPanel.LoadNameFonts"* jména základních fontů v panelu. Tyto jména využívá simulátor v programu „Panel II"

Nastavení je uloženo v datovém poli. Pointer na datové pole panelu je možné načíst funkcí:

LPVOID GetPtrPanel(int id);

```
typedef struct DataPanel
{
public:
   char  PanelName[SIZE_NAME_PANEL];      // popisovací jméno panelu
            // Nastavení komunikace
   TPCOM tpcom;                           // nastaveni sériového portu
   BYTE  EnableGSM;                       // nastaveni GSM modemu  0/1 ne/ano
   char  PhoneNumber[SIZE_PHONE_NUMBER];  // telefonní číslo
   char  IPadres[SIZE_IPADRES];           // IP adresa (port = -1)
   long  IPport;                          // IP port
   char  free;                            // VOLNE
            // Nastavení který si program načítá v případě potřeby z panelu
   char  LoadVer;                         // verze firmware panelu
   char  LoadiFonts;                      // počet fontů
   int   LoadAdrFonts;                    // adresa fontu
   char  LoadNameFonts[16][20];           // jména fontů
            // Nastavení a data EEPROM panelů
   char  LoadClaster;                     // počet clastrů
   char  LoadBlokInCL;                    // sektorů v clastru (jeden blok = 254b)
   WORD  LoadSizeRAM;                     // velikost pracovní paměti
   bool  DataLoad;
   BYTE  DataBOOT[256];                   // načtený BOOT sektor
   BYTE  DataFAT[256];                    // FAT tabulka
   BYTE  DataDIR[SIZE_DIR][256];          // Direktorář
   BYTE  DataFileFlash[32];               // jméno souboru ve FLASH paměti
            // Nastaveni simulátoru
   COLORREF SetColorLED[15];              // nastaveni barvy řádku
   COLORREF SetColorBack;                 // nastaveni barvy pozadí řádku
   BYTE  SetColor[15];                    // barva led 4 = R, 2 = G, 1 = B
   BYTE  SetShade;                        // odstínu na jednu barvu
   char  SetLine;                         // počet řádků panelu
   Int   SetX;                            // délka panelu
```

```
   int   SetY;                          // výška řádku
   int   SetVer;                        // verze programu - pořadové číslo
   char  SetSegment[5];
   char  SetSpace;
   char  SetDefaultLS;
   BYTE  SetRTC;                        // hodiny RTC
   BYTE  SetAnalog[16];                 // analogové vstupy a komparátory
   BYTE  SetParael[16];                 // nastaveni paralelních vstupů
   BYTE  SetFInput[16];                 // nastaveni frekvenčních vstupů
   BYTE  Free[256];                     // volné místo
   LPVOID *pSimulator;                  // pointer na simulátor
   LPVOID *ptrUser[PTR_USER];
}DATA_PANEL;
```

```
void Seting( )
{
            // přidání nového panelu jména nový - sériový port 1 adresa 1
      pdArrData( CMD_ARRDATA_PANEL_ADD, "nový", 0x0101 );
            // počet definovaných panelů
      int count = pdArrData( CMD_ARRDATA_PANEL_COUNT, NULL, 0 );
            // pointer na datovou strukturu panelu
      DATA_PANEL* ptrPanel = (DATA_PANEL*)GetPtrPanel( 0 );
            // nastaveni komunikace TCP/IP
      ptrPanel->tpcom.port = -1;
      strcpy( ptrPanel->IPadres, "192.168.1.1" );
      ptrPanel->IPport = 100;
}
```

MONITOROVÁNÍ FUNKCÍ, PROVÁDĚNÝCH VE VLÁKNĚ

Některé funkce komunikace s panelem jsou prováděny ve vlákně v rámci knihovny. Příslušným příkazem se spustí, funkce která toto provede je poté ukončena. Stav provádění ve vlákně je pak možné kontrolovat následující funkcí.

`int  pdControl_fnc( int control );`

Touto funkci je možné zjistit zda je provádění ve vlákně ukončeno, průběh provádění, případné číslo chyby a podobně. Dále je možné řídit zápis souboru.

Funkce které je možné provádět.

Definice cmd	Kód		Popis
CONTROL_GETSTATE	0		Zjistit stav průběhu
CONTROL_NULL_PROGRESS	7		Vynulování čítače průběhu
CONTROL_GET_ERRCODE	8		Vrací číslo chyby
CONTROL_FILE_CANCEL	1	Odpověď na STATUS_FILE_EXIST	ukončit zápis
CONTROL_FILE_SKIP	2		přeskočit tento soubor
CONTROL_FILE_SKIP_ALL	3		přeskočit všechny existující
CONTROL_FILE_OVERWRITE	4		přepsat tento soubor
CONTROL_FILE_OVERWRITE_ALL	5		přepsat všechny existující soubory
CONTROL_FILE_INI_FLASH	12	Odpověď na STATUS_FILE_INI	uložit do FLASH
CONTROL_FILE_INI_EEPTROM	13		uložit do EEPROM
CONTROL_TERMINATE	255		Přerušení spuštěné funkce

Funkce vrací 32 bitové číslo, ve kterém je zaznamenán stav. Spodních 16 bitů obsahuje počítadlo průběhu zpracování, horních 16 bitů pak obsahuje stav.

Definice cmd	Kód	Popis
STATUS_FILE_NEXT	0x00010000	V bufru je připraveno jméno ukládaného souboru
STATUS_FILE_EXIST	0x00020000	Soubor existuje – ukládání zastaveno a čeká na zprávu jak pokračovat.
STATUS_END	0x00040000	Provádění funkce na pozadí ukončeno
STATUS_RUNING	0x00080000	Provádění funkce na pozadí běží
STATUS_FILE_INI	0x00200000	Ukládány soubor je inicializační soubor – program čeká na volbu kam jej má uložit

Tuto funkci je nejideálnější použít pomocí časovače kde se kontrola bude provádět asi jednou za 100ms. Zobrazení průběhu je možné provést pomocí **ProgressControl**, jeho hodnoty jsou od 0 do 100. Základní použití této funkce je následující.

```
   SetTimer( 1, 100, NULL );              // spuštěni časovače monitoringu
   m_Progress.SetRange( 0, 100 );         // nastavení zobrazení průběhu
                                          // m_Progress je MFC objekt CProgress

// monitoring průběhu
void CDlgDemo::OnTimer( UINT nIDEvent )
{
                                          // načte stav průběhu čteni direktoráře
   int stat = pdControl_fnc( CONTROL_GETSTATE );
   if ((stat & 0xFFFF0000) == STATUS_END) {                // čtení ukončeno?
      int err = pdControl_fnc( CONTROL_GET_ERRCODE );      // vrací číslo chyby
                        // odeslání zprávy funkce přerušena
      if (err == ERR_TERMINATE) { SendMessage( WM_USER_END, 0, 0 ); break; }
                        // odeslání zprávy nastala chyba
      if (err != 0) { SendMessage( WM_USER_ERR, err, 0 ); break; }
      SendMessage( WM_USER_END, err, 0 ); break; }   // odeslání zprávy ukončení
   }
   m_Progress.SetPos( stat & 0xFFFF );    // zobrazeni průběhu čtení

   CDialog::OnTimer(nIDEvent);
}
```

OTEVŘENÍ A ZAVŘENÍ SPOJENÍ S PANELEN

Před komunikací s panelem je nutné nejprve otevřít spojení s panelem. V případě sériového portu se otevření nastavených portů zpravidla provede po inicializaci a načtení nastavení příkazem **CMD_CONNECT_COMSET**, uzavření portů se provede před ukončením „**pdCloseClient**" příkazem **CMD_CONNECT_COMDONE**, pokud je v průběhu změněno nastavení portů je nutné nejprve porty uzavřít a následně zase otevřít.

Při komunikaci TCP/IP se otevře spojení před začátkem komunikace příkazem **CMD_CONNECT_COMDONE** a po skončení se opět uzavře **CMD_CONNECT_DISCONNECT**. Přes TCP/IP adresu může s panelem být otevřeno pouze jedno spojení a pokud necháte spojení otevřené, znemožníte tím spojení od jiných uživatelů. Pokud použijete příkazem **CMD_CONNECT_CONNECT** opakovaně zvýší se pouze počítadlo otevření o jednu, uzavření komunikace musíte provést pak stejným počtem (z důvodu více vláknového programování, aby jedno vlákno nezavřelo spojení dokud je používáno druhým vláknem). Před ukončením programu je nutné uzavřít všechna spojení.

U GSM modemu je situace stejná jako u IP adresy pouze s tím rozdílem, že nemá počítadlo otevření.

```
int  pdConnect( int cmd_id );
```

Definice cmd	Kód	Popis
CMD_CONNECT_COM_SET	0x01000100	Nastavení všech COM portů
CMD_CONNECT_COM_GET	0x01000200	Vrací nastavení COM portů, 32 bitové číslo jeden bajt jeden port 1 – port nastaven 0 – port nenastaven
CMD_CONNECT_COM_DONE	0x01000400	Uzavření všech COM portů
CMD_CONNECT_IP_CONNECT	0x01000800	Otevření spojení TCP/IP
CMD_CONNECT_IP_DISCONNECT	0x01001000	Ukončení spojení TCP/IP
CMD_CONNECT_IP_RECONECT	0x01001800	Ukončí spojení TCP/IP a opět jej otevře
CMD_CONNECT_GSM_CONNECT	**0x01002000**	**Spustí vytáčení čísla GSM modemu**
CMD_CONNECT_GSM_DISCONNECT	**0x01004000**	**Zavěšení GSM modemu**
CMD_CONNECT_GSM_STATE	0x01008000	Vrací stav modemu, podle stavu registrů, nekomunikuje s modemem GSM_STATE_NO_ANSWER – neodpovídá GSM_STATE_NO_CONNECT – zavěšen GSM_STATE_CONNECT – spojení aktivní GSM_STATE_OTHER_NUMBER – spojen na jiné číslo
CMD_CONNECT_GSM_GET_MODEM	**0x01010000**	**Spustí zjištění stavu modem, stav uloží do interních registrů programu, po ukončení je možné stav zjistit funkcí CMD_CONNECT_GSM_STATE**
CMD_CONNECT_GSM_GETCOUNTACTIVE	0x01020000	Vrací počet aktivních modemů

CMD_CONNECT_COMSET – nastavit port *DataPanel.tpcom.port.*
CMD_CONNECT_CONNECT – port *DataPanel.tpcom.port* nastavit na -1 a do *DataPanel.IPadres* musí obsahovat řetězec IP adresy panelu (např. 192.168.1.1) a port *DataPanel.IPport*
CMD_CONNECT_GSM_CONNECT se nastaví *DataPanel.tpcom.port* a *DataPanel.*PhoneNumber

V případě, že je příkaz otevření TCP/IP "**CMD_CONNECT_CONNECT**" použit na panel, který má nastavenou komunikaci pomocí sériového portu, neprovede se nic. Při zavření TCP/IP je to stejné. Tím je před každou komunikací možné použít funkci otevření TCP/IP. Všechny tři příkazy **CMD_CONNECT_IP_....** vrací pokud proběhne správně místo ***ERR_NULL*** stav GSM modemu. ***ERR_CONNECT*** v případě, že je spojení aktivní a ***ERR_NOCONNECT*** pokud není.

```
void Main( )
{
      Init( );          // inicializace
            // nastaveni sériových portů
      pdConnect( CMD_CONNECT_COM_SET );

      // ......   komunikace s panely pres RS232 nebo RS 485

            // panel 1 má nastavenou komunikaci TCP/IP
      pdConnect( CMD_CONNECT_IP_CONNECT | 1 );

      // .....    komunikace s panelm 1 pomocí protokolu TCP/IP

            // zavření spojení TCP/IP
      pdConnect( CMD_CONNECT_IP_DISCONNECT | 1 );
            // uvolnění sériových portů
      pdConnect( CMD_CONNECT_COM_SET );
      Destroy( );       // ukončení
}
```

```
void ConnectGSM( int id )
{
      int stat

      switch (pdConnect( CMD_CONNECT_GSM_STATE | id ))
      {
      case ERR_NOMODEM :       break;     // modem není použit

      case GSM_STATE_CONNECT : break;     // spojení je aktivní

      case GSM_STATE_OTHER_NUMBER :       // je vytočeno jiné číslo
                  // předcházející spojeni ukončit
            pdConnect( CMD_CONNECT_GSM_DISCONNECT | id );

      case GSM_STATE_NO_CONNECT :         // spojeni není aktivní
            pdConnect( CMD_CONNECT_GSM_CONNECT | id )
         // monitorováni průběhu vytáčení čísla, tato funkce ovšem vezme veškerý
         // čas procesoru proto je lepší pro tuto funkci použít časovač
         // jak je uvedeno u funkce pdControl_fnc
            for ( ; ; ) {
                  int stat = pdControl_fnc( CONTROL_GETSTATE );
                  if ((stat & 0xFFFF0000) == STATUS_END) {
                        m_Err = pdControl_fnc( CONTROL_GET_ERRCODE );
                        switch( m_Err )
                        {
                        case ERR_NO_CARRIER :   // číslo neexistuje
                        case ERR_NO_DIALTONE :  // není tón
                        case ERR_NULL :         // vytočeno
                        default :
                        }
                  }
                        // zobrazení průběhu
                  m_Progress.SetPos( stat & 0xFFFF );
            }
            break;
      }
}
```

Pokud je na více panelech použit stejný modem, je nutné, aby ukončení spojení modemu bylo provedeno na stejném panelu u kterého jste spojení navazovaly. Při navázání spojení i ukončení jsou totiž u tohoto panelu nastavovány příkazy stavu modemu. Pokud tak neučiníte ztížíte si tím identifikaci stavu modemu.

ŘÍZENÍ PANELU

```
int  pdPanelControl( int cmd_id, int iParam, BYTE *data, int size };
```

Definice cmd	Kód	Popis
CMD_CONTROL_VERFIRM_ID	0x02000100	Vrací ID verze firmware panelu, při použití v CMD_STRINGS_COMMAND se získá text verze
CMD_CONTROL_VERFIRM_ST	0x02000200	Vrací **záznam verze**, který pošle panel
CMD_CONTROL_GET_BRIGHTNESS	0x02000400	Načte jas panelu nebo stmívání
CMD_CONTROL_SET_BRIGHTNESS	0x02000800	Nastavení jasu a stmívání panelu
CMD_CONTROL_GET_RTC	0x02001000	Vrací reálný čas panelu
CMD_CONTROL_SET_RTC	0x02002000	Nastavení reálného času panelu
CMD_CONTROL_GET_TIMER	0x02004000	Vrací hodnotu časovače
CMD_CONTROL_SET_TIMER	0x02008000	Nastavení časovače

CMD_CONTROL_RESET	0x02010000	Restart Firmware iParam = 0; data = NULL; size = 0
CMD_CONTROL_GET_ANALOG	0x02020000	Vrací hodnotu analogových vstupů
CMD_CONTROL_SET_ANALOG	0x02040000	Nepoužito
CMD_CONTROL_SEND_PSW	0x02080000	Nepoužito
CMD_CONTROL_INFO_PSW	0x02100000	Nepoužito
CMD_CONTROL_GET_RELEVANCE	0x02220000	Zjištění platnosti dat iParam - číslo registru 0 – 3 data = NULL; size = 0;
CMD_CONTROL_SET_RELEVANCE	0x02240000	Nastavení platnosti dat iParam - číslo registru 0 – 3 data = NULL; size = 0;

BYTE *data – pointer na pole BYTE ve kterém se předávají data. Pole musí být definované v programu, který tuto funkci volá.
int size – velikost datového pole řetězce.

Tabulka volby parametru *iParam* u **CMD_CONTROL_GET_LIGHT** a **CMD_CONTROL_SET_LIGHT**

iParam	Kód	Popis
CTR_BRIGHTNESS_CHANGE	0x00020001	Změna intenzity svitu panelu – nastavení není ukládáno po vypnutí a zapnutí panelu, je nastavena původní intenzita. data[0] = nastavovaný jas data[1] = 0
CTR_BRIGHTNESS_SET	0x00020000	Uložení nastavení intenzity. data[0] = nastavovaný jas data[1] = 0
CTR_BRIGHTNESS_SET_DARK	0x00040002	Nastavení a uložení stmívání i jasu data[0] = nastavovaný jas data[1] = horní hranice stmívání data[2] = dolní hranice stmívání data[3] = minimální jas
CTR_BRIGHTNESS_GET	0x00010000	Načtení aktuelního jasu panelu data[0] = nastavený jas data[1] = aktuelní jas data[2] = regulace stmíváním
CTR_BRIGHTNESS_GET_DARK	0x00000002	Načtení aktuelního jasu panelu data[0] = nastavený jas data[1] = horní hranice stmívání data[2] = dolní hranice stmívání data[3] = minimální jas
CTR_BRIGHTNESS_ALL	0x10000000	Nastavení na všech panelech pouze pro funkce nastaveni a zápisu

Zvolený jas je uložen v paměti EEPROM, tato paměť má omezený počet zápisů. Z tohoto důvodu není jas při změně ukládán, až teprve pokud zvolíte správnou intenzitu můžete ji s parametrem **CTR_LIGHT_SET_LIGHT** uložit. Samozřejmě je možné příkaz **CTR_LIGHT_CHANGE** vynechat a jas nastavit rovnou s uložením.

```
int err;
BYTE data[4];

data[0] = brightness;       // požadovaný jas panelu
data[1] = 0;
          // nastavení jasu
err = pdPanelControl( CMD_CONTROL_SET_BRIGHTNESS | 0, CTR_BRIGHTNESS_CHANGE,
                      data, 2 );
          // nastavení jasu na všech panelech
err = pdPanelControl( CMD_CONTROL_SET_BRIGHTNESS, CTR_BRIGHTNESS_ALL |
                      CTR_BRIGHTNESS_SET, data, 2 );
```

```
          // nastavení a uložení jasu
err = pdPanelControl( CMD_CONTROL_SET_BRIGHTNESS | 0, CTR_BRIGHTNESS_SET,
                      data, 2 );

//.............................................
data[0] = brightness; // požadovaný jas panelu
data[1] = HIdark;     // horní hranice stmívání
data[2] = LOdark;     // dolní hranice stmívání
data[3] = min;        // minimální jas
err = pdPanelControl( CMD_CONTROL_SET_BRIGHTNESS | 0, CTR_LIGHT_SET_DARK,
               data, 4 );

//.............................................
err = pdPanelControl( CMD_CONTROL_GET_BRIGHTNESS | 0, CTR_BRIGHTNESS_GET_DARK,
                      data, 4 );
jas    = data[0];     // požadovaný jas panelu
HIdark = data[1];     // horní hranice stmívání
LOdark = data[2];     // dolní hranice stmívání
min    = data[3];     // minimální jas
```

Záznam verze vrácené panelem 18 bajtů

Adresa	Popis
0	Verze desetiny
1	Verze celé číslo
2	0xFF
3	Počet sektoru paměti EEROM
4	Velikost RAM horních 8 bitů
5	Velikost RAM dolních 8 bitů
6	Počet fontů
7	Bloků v jednom sektoru EEPROM (jeden blok 254 bajtů)
8	Počet řádku panelu
9	Počet znaků na řádku (jeden znak 8 bodů)
10	Výška jednoho řádku v bodech
11	Verze modulu download desetiny
12	Verze modulu download celé číslo
13	Maximální velikost druhého programu horních 8 bitů
14	Volné
15	Typ procesoru
16	Adresa volného prostoru ne flash horních 8 bitů
17	Volné

CMD_CONTROL_GET_RELEVANCE a **CMD_CONTROL_SET_RELEVANCE** se používá pro ověření platnosti dat. Panely od verze 9.5 obsahují tyto registry čtyři, starší verze pouze jeden. První registr používá knihovna pro ověření načtení direktoráře z panelu. Po načtení direktoráře do něj uloží náhodnou hodnotu, vždy když je pak třeba pracovat s direktorářem zjistí, zda je v tomto registru v panelu stále stejná hodnota.

```
struct timeb timebuffer;
BYTE relevance[4];

ftime( &timebuffer );
memcpy( data, &timebuffer.time, 5);
                // zápis registru platnosti dat
int err = pdPanelControl( CMD_CONTROL_SET_RELEVANCE | 0, 1, data, 4 );

// ..................................

BYTE data[5];
                // přečtení registru platnosti dat
int err = pdPanelControl( CMD_CONTROL_SET_RELEVANCE | 0, 1, datain, 4 );
```

```
                    // kontrola
  if (memcmp( datain, data, 5 ) != 0) return FALSE;
  return TRUE;
```

formát času pro příkazy **CMD_CONTROL_GET_RTC** a **CMD_CONTROL_SET_RTC**

data	Popis		
0	Vteřiny	dekadicky 0 – 59	(0x00 – 0x59)
1	Minuty	dekadicky 0 – 59	(0x00 – 0x59)
2	Hodiny	dekadicky 0 – 23	(0x00 – 0x23)
3	Den v týdnu	1 – 7	
4	Den	dekadicky 1 – 31	(0x01 – 0x31)
5	Měsíc	dekadicky 1 – 12	(0x01 – 0x12)
6	Rok 2000 - 2099	dekadicky 0 – 99	(0x00 – 0x99)

Při čtení analogových vstupů je načten všech devět hodnot najednou. První hodnota udává teplotu procesoru. Ostatní hodnoty udávají stav jednotlivých vstupů.

```
  _int16    Analog[9];                    // datové pole vstupu
            // načtení hodnot všech vstupů
  int err = pdPanelControl( CMD_CONTROL_GET_ANALOG | 0, 0, (BYTE*)Analog, 18 );

  double t_cpu   = Analog[0] / 100;       // první vstup - teplota procesoru
            // ostatní udávají hodnotu vstupu 1 - 8 na jedno desetinné místo
            // 0xFFFF pokud vstup není požit
  double teplota = Analog[0] / 10;
```

TEXTOVÉ FUNKCE

int pdStrings(int cmd_id, char *cstr, int size, int iParam);

Definice cmd	Kód	Popis
CMD_STRINGS_COMMAND	0x04000100	Vrací text příkazu načteného ze souboru „command.src“, který je načten při spuštění knihovny. iParam – id příkazu (viz. Tabulka)
CMD_STRINGS_VERSION	0x04000400	Vrací text definující verzi firmware panelu (např. "9.60") iParam – id verze.
CMD_STRINGS_ERR	0x04000400	Vrací text chyby. iParam – číslo chyby, pokud je „-1“ je vezme se poslední chyba na příslušném panelu
CMD_STRINGS_WR_FILENAME	0x04000800	Jméno ukládaného souboru.
CMD_STRINGS_WR_FILENAME_SIZE	0x04001000	Jméno ukládaného souboru + velikost.
CMD_STRINGS_GET_PROGRAM	0x04004000	Text programu načteného z panelu Viz. **CMD_FILE_READ_FILE**
CMD_STRINGS_DIR_FILENAME	0x04008000	Jméno souboru v direktoráři Viz. **CMD_FILE_READ_DIRECTORY**
CMD_STRINGS_GSM_GETMODEM	0x01100000	Identifikace modemu Parametr iParam GSM_GET_TYPE – typ modemu GSM_GET_OPERATOR – operátor GSM_GET_POWER – sila signálu
CMD_STRINGS_GSM_INFO	0x04002000	Nepoužito
CMD_STRINGS_FLASH_FILENAME	0x04010000	Jméno ini souboru uloženého ve flash. Musí byt načten direktorář

char *cstr – pointer pole char ve kterém je řetězec. Pole musí byt definované v programu který tuto funkci volá.
int size – velikost datového pole řetězce.

Příkaz **CMD_STRINGS_WR_FILENAME** je možné zjistit jméno souboru, který je právě ukládán do panelu *„panel // jméno“*. V případě, že soubor již existuje je možné příkazem **CMD_STRINGS_WR_FILENAME_SIZE** zjistit jméno souboru a velikost původního souboru a nového souboru v bajtech. Formát je následující *„panel // jméno;původní velikost;nová velikost“*

Direktorář je tabulka, která obsahuje 200 řádků. Příkaz **CMD_STRINGS_DIR_FILENAME** je možné přečíst jména z jednotlivých řádků této tabulky (direktorář je nutné nejprve načíst funkcí **pdPanelFile**). Parametr „iParam“ udává číslo řádky. Pokud je pozice prázdná vrací nulový řetězec.

Zobrazovací program je text, do kterého jsou vloženy příkazy jak se má text zobrazit. Všechny příkazy jsou ohraničeny složenými závorkami. Text jednotlivých příkazů se při spuštění programu načte ze souboru „Language\command.src“. Příkazem **CMD_STRINGS_COMMAND** je možné text příslušného příkazu načíst. V souboru je jeden příkaz na jednom řádku.

2="VLEVO"

Číslo na začátku je identifikační číslo příkazu. V uvozovkách je vlastní text příkazu, tento text je možné přeložit do jiného jazyka.

```
char cstr[512];              // velikost nutno dimenzovat na maximální
                             // délku textu v soboru  *_PDriver.ln
pdStrings( CMD_STRINGS_COMMAND | 0, char cstr, 512, err );
```

Kódy příkazů pro funkci **CMD_STRINGS_COMMAND**

Definice	iParam	Popis
CSTR_UNKNOWN	0	
CSTR_NAME	1	Jméno souboru
CSTR_LEFT	2	Posun vlevo
CSTR_UP_L	3	Posun nahoru zarovnání vlevo
CSTR_DOWN_L	4	Posun dolů zarovnání vlevo
CSTR_UP_C	5	Posun nahoru zarovnání uprostřed
CSTR_DOWN_C	6	Posun dolů zarovnání uprostřed
CSTR_UP_R	7	Posun nahoru zarovnání vpravo
CSTR_DOWN_R	8	Posun dolů zarovnání vpravo
CSTR_DOWN	9	Posun dolů zarovnání vlevo
CSTR_UP	10	Posun nahoru zarovnání vlevo
CSTR_STATIC_L	11	Staticky zarovnání vlevo
CSTR_STATIC_C	12	Staticky zarovnání uprostřed
CSTR_STATIC_R	13	Staticky zarovnání vpravo
CSTR_STATIC	14	Staticky zarovnání vlevo
CSTR_PAUSE	15	Pauza
CSTR_DEL_ALL	16	Smazání celého panelu
CSTR_SECOND	17	Reálný čas – vteřiny
CSTR_MINTES	18	Reálný čas – minuty
CSTR_HOUR	19	Reálný čas – hodina
CSTR_WEEK	20	Reálný čas – den v týdnu
CSTR_DAY	21	Reálný čas – den
CSTR_MOON	22	Reálný čas – měsíc
CSTR_YEAR	23	Reálný čas – rok
CSTR_YEAR_2	24	Reálný čas – rok dvoumístně
CSTR_FONT	25	Volba fontu
CSTR_BRIGHTNESS	26	Změna intenzity svitu
CSTR_PRG_THRAED	27	Spuštění druhého programu
CSTR_EXE	28	Spuštění siného programu
CSTR_CALL	29	Provedení programu
CSTR_RADKA	30	Změna řádky a segmentu
CSTR_TEXT_L	31	Zobrazení textu do buferu vlevo

CSTR_TEXT_C	32	Zobrazení textu do buferu uprostřed
CSTR_TEXT_R	33	Zobrazení textu do buferu vpravo
CSTR_VSE_NAHORU	34	Posun nahoru vše
CSTR_VSE_DOLU	35	Posun dolů vše
CSTR_VSE_STATIC	36	Staticky vše
CSTR_TIMER_VIEW	37	Zobrazení časovače
CSTR_EFECT_DROP_ALL	38	Efekt sněžení vše
CSTR_EFECT_DROP	39	Efekt sněžení
CSTR_OBR	40	Zobrazení obrázku
CSTR_ANIM	41	Zobrazení animace
CSTR_TEPLOTA	42	Zobrazení teploty
CSTR_ANALOG	43	Zobrazení analogové hodnoty
CSTR_FREQ	44	Zobrazení frekvence
CSTR_C_PROGRAM	45	Řídicí program – jméno programu
CSTR_C_TIMESTART	46	Řídicí program – čas spuštění
CSTR_C_TIMESTOP	47	Řídicí program – čas vypnutí
CSTR_C_EXTERNI	48	Řídicí program – paralelní vstupy
CSTR_DEL_LINE	49	Smazání řádky
CSTR_SMAZ	50	Smazání
CSTR_COLOR	51	Změna barvy
CSTR_RED	52	Červená barva
CSTR_GREN	53	Zelená barva
CSTR_BLUE	54	Modrá barva
CSTR_VAR	55	Zobrazení proměnné
CSTR_SENSOR	56	Zobrazení senzoru
CSTR_PARAM_MONDAY	1000	Řídicí program – pondělí
CSTR_PARAM_TUESDAY	1001	Řídicí program – úterý
CSTR_PARAM_WEDNESDAY	1002	Řídicí program – středa
CSTR_PARAM_THURSDAY	1003	Řídicí program – čtvrtek
CSTR_PARAM_FRIDAY	1004	Řídicí program – pátek
CSTR_PARAM_SATURDAY	1005	Řídicí program – sobota
CSTR_PARAM_SUNDAY	1006	Řídicí program – neděle
CSTR_PARAM_NO	1007	Ne
CSTR_PARAM_YES	1008	Ano
CSTR_PARAM_ACTIVE	1009	Řídicí program – vstupy aktivní
CSTR_PARAM_WAITNEXT	1010	Řídicí program – vstupy do změny
CSTR_PARAM_GOSUB	1011	Řídicí program – vstupy pouze jednou

ZOBRAZENÍ STATICKÉ ŘÁDKY

Při zobrazení statické řádky by neměl na tuto řádku zobrazovat žádny program, jelikož text takto zobrazený by byl přepsán textem zobrazeným programem. Panely od verze 9.0 neumožňují načtení statické řádky z panelu.

int pdStaticLine(int cmd_id, int *iParam, char *cstr, int size);

Definice cmd	Kód	Popis
CMD_STATIC_GETLINE	0x0A000100	Načtení zobrazeného textu od verze 9.0 firmware nepodporučuje
CMD_STATIC_SETLINE	0x0A000200	Zobrazení textu
CMD_STATIC_GETTIME	0x0A000400	Nepoužito
CMD_STATIC_SETTIME	0x0A000800	Nepoužito

int *iParam – pointer číslo definované v programu, který tuto funkci volá.
Formát je 0x44332211.
11 – číslo řádku 1 – 15.
22 – zarovnání textu 1 vlevo, 2 střed, 3 vpravo.

33 – redukce mezer 1 ano, 0 ne.

char *cstr – pointer na pole char ve kterém je zobrazovaný text. Pole musí být definované v programu, který tuto funkci volá. Maximální délka je 61 znaků plus nula na konci.

int size – velikost datového pole řetězce.

```
void Main( )
{
      Init( );                              // inicializace
      pdConnect( CMD_CONNECT_COMSET );      // nastaveni sériových portů

      char cstr[60];
      strcpy( cstr, "Text" );
            // zobrazení s redukcí mezer na řádce 1 uprostřed
      pdStaticLine(CMD_STATIC_SETLINE | 0, 0x010201, cstr, 60  );

      pdConnect( CMD_CONNECT_COMSET );      // uvolnění sériových portů
      Destroy( );                           // ukončeni
}
```

SOUBOROVÝ SYSTÉM

Každý panel obsahuje paměť, která udrží data i po odpojení napětí. Do této paměti je možné uložit program (text + řídící příkazy). Paměť je formátována podobným způsobem jako disketa (FAT8), takže je možné do panelu nahrát více programů, které můžete libovolně mazat nebo doplňovat. Pokud je do panelu nahráván inicializační souboru (.I), je nabídnuta možnost uložit jej do FLASH paměti.

int pdPanelFileSystem(int cmd_id, BYTE *bdata, int size, int iParam);

Definice cmd	Kód	Popis
CMD_FILESYSTEM_FORMAT	0x05000100	Formátování paměti pro ukládání dat (EEPROM) bdata – záznam BOOT sektoru
CMD_FILESYSTEM_SCAN	0x05000200	NEPOŽITO
CMD_FILESYSTEM_START_LS	0x05000400	Spuštění načtení sektoru iParam – číslo sektoru
CMD_FILESYSTEM_GET_LS	0x05000800	Vrací data sektoru načteného funkcí CMD_FILESYSTEM_START_LS bdata – záznam sektoru

BYTE *bdata – pointer na datové pole **BYTE** o velikosti 256 bajtů, musí být definované v programu, který tuto funkci volá. Pokud není použit musí být **NULL**

int size – Velikost **datového pole bdata**, v případě že datové pole není použito (NULL) zadá se nula

Příkaz formátování zjistí velikost EEPROM, vynuluje BOOT sektor FAT a inicializuje Direktorář. Do datového pole bdata se vloží default záznam BOOT sektoru, standardně je nulový. Po naformátování je do tohoto sektoru uložena velikost EEPROM

```
  BYTE data[256];                   // data BOOT sektoru
  memset( Data, 0, 256 );           // vynulování BOOT sektoru
                                    // spuštění formátování
  pdPanelFileSystem( CMD_FILESYSTEM_FORMAT | 0, data, 256, 0 );

            // standartni monitoring průběhu
```

Příkazem **CMD_FILESYSTEM_START_LS** je možné načíst jakýkoliv sektor EEPROM. Sektor je načten do interního buferu knihovny. Načtená data sektoru je možné pak získat příkazem **CMD_FILESYSTEM_GET_LS**

```
int  iSektor;          // číslo sektoru
BYTE data[256];        // data sektoru
                       // spuštění čtení sektoru
pdPanelFileSystem( CMD_FILESYSTEM_START_LS | 0, NULL, 0, iSektor );

          // standardní monitoring průběhu

                       // načtení dat sektoru
 pdPanelFileSystem( CMD_FILESYSTEM_GET_LS | 0, data, 256, 0 );
```

int pdPanelFile(int cmd_id, char *cstr, int size, int iParam);

Definice cmd	Kód	Popis
CMD_FILE_READ_DIRECTORY	0x07001000	Spuštění načtení direktoráře. Direktorář je načten do bufferu
CMD_FILE_READ_FILE	0x07002000	Spuštění čtení souboru cstr – jméno souboru iParam – definuje velikost buferu pro načtený program
CMD_FILE_WRITE_FILE	0x07008000	Zápis souboru do panelu cstr – text ukládaného souboru
CMD_FILE_DELETE_FILE	0x07010000	Smazání souboru z panelu cstr – jméno souboru iParam – 100 / počet mazaných souborů
CMD_FILE_WRITE_FILE_RAM	0x07020000	Zápis souboru přímo do pracovní paměti panelu cstr – text zobrazovacího programu
CMD_FILE_DELET_FILE_FLASH	0x07040000	Smazání souboru s FLASH paměti

char *cstr – pointer na pole char pro čtený/zapisovaný program nebo jméno souboru. Pole musí být definované v programu, který tuto funkci volá. Maximální délka je 61 znaků plus nula na konci.
int size – velikost datového pole cstr.

V následujícím příkladu je ukázka, jak je možné provést načtení direktoráře:

```
// stlačení tlačítka načtení direktoráře
void CDlgDemo::OnStartReadDir( )
{
                                        // spuštěni čteni direktoráře
   if (pdPanelFile( CMD_FILE_READ_DIRECTORY | 0, NULL, 0, 0 ) != 0) return;
   SetTimer( 1, 100, NULL );            // spuštěni časovače monitoringu
}

// přerušení čteni
void CDlgDemo::OnCance( )
{
   pdControl_fnc( CONTROL_TERMINATE );  // ukončeni čteni
   KillTimer( 1 );                      // zastaveni časovače
}
```

```
// monitoring průběhu
void CDlgDemo::OnTimer( UINT nIDEvent )
{
                                          // načte stav průběhu čteni direktoráře
   int stat = pdControl_fnc( CONTROL_GETSTATE );
   if ((stat & 0xFFFF0000) == STATUS_END) {               // čtení ukončeno?
      int err = pdControl_fnc( CONTROL_GET_ERRCODE );     // vrací číslo chyby
                                          // odeslání zprávy čtení ukončeno
      if (err != ERR_TERMINATE) { SendMessage( WM_USER_END, err, 0 ); break; }
   }
   m_Progress.SetPos( stat & 0xFFFF );    // zobrazeni průběhu čtení

   CDialog::OnTimer(nIDEvent);
}
```

Po načtení direktoráře se zobrazí všechny soubory obsažené v panelu. Pokud před čtením souboru z panelu se neprovede načtení direktoráře, ukončí se funkce čtení souboru chybovým hlášením **ERR_NAME_NOT_EXIST** (jméno souboru neexistuje). Funkce hledá umístění souboru v direktoráři, který je načtený z panelu. V případě, že je prázdný (není načten) soubor nenajde, program pak nezná jeho umístění v paměti panelu a tudíž jej nemůže načíst. Monitoring průběhu čtení je stejný jako v případě čtení direktoráře.

```
// stlačeni tlačítka načtení souboru
void CDlgDemo::OnStartReadDir( )
{
      char name[11];
      strcpy( name, "demo" );       // jméno čteného souboru je demo
                                    // spuštění čtení souboru
      if (pdPanelFile( CMD_FILE_READ_FILE | 0, name, 11, 65536 ) != 0) return;
      SetTimer( 1, 100, NULL );    // spuštění časovače monitoringu
}

            // soubor je se načte do interního buffru knihovny
            // získat text načteného souboru je možné následujícím příkazem
   char cstr[65536];
   pdStrings( CMD_STRINGS_GETPROGRAM, cstr, 65536, 0 );
```

Funkce automaticky po načtení souboru přeloží všechny kódy zobrazovacích příkazů na jejich textovou podobu.

V případě zápisu do panelu direktorář nemusí být načten předem, funkce zápisu si jej v případě potřeby načte sama. Monitoring je rozdílný v tom, že se musí přidat volby pro případ, že soubor existuje nebo je zapisován inicializační soubor. Příznak **STATUS_FILE_NEXT** je nastaven při zahájení zápisu souboru a po přečtení zapisovaného jména funkcí **CMD_STRINGS_WR_FILENAME** je tento příznak vynulován.

Stejně bude vypadat i funkce **CMD_FILE_WRITE_FILE_RAM** (uložení souboru přímo do RAM panelu, s tím rozdílem, že funkce nehlásí, že soubor již existuje.

```
// stlačení tlačítka načtení souboru
void CDlgDemo::OnStartReadDir( )
{
      char cstr[65536];

            // zkopírování textu programu

      if (pdPanelFile( CMD_FILE_WRITE_FILE | 0, cstr, 65536, 0 ) != 0) return;
      SetTimer( 1, 100, NULL );                        // spuštění časovače
}

// časovač - monitoring přeběhu
void CDlgDemo::OnTimer(UINT nIDEvent)
{
```

```
            // stav zápisu
      int i = pdControl_fnc( CONTROL_GETSTATE );      // zjištění stavu
      m_Progress.SetPos( status & 0x0000FFFF );       // ukazatel průběhu zápisu
      if ((status & STATUS_END) != 0) {               // ukončení
            err = pdControl_fnc( CONTROL_GET_ERRCODE );
            SendMessage( WM_USER_END, err, 0 );
      }

            // je nahráván další soubor
      if ((status & STATUS_FILE_NEXT) != 0) {
            pdStrings( CMD_STRINGS_WR_FILENAME, cstr, 40, 0 );
            m_StaticName.SetWindowText( cstr );

      }
            // soubor v panelu existuje
      if ((status & STATUS_FILE_EXIST) != 0) {
            SendMessage( WM_USER_FILEEXIST, 0, 0 );   // volba přepsání souboru
            KillTimer( 1 );
      }

            // zápis ini souboru - výběr flash nebo eeprom
      if ((status & STATUS_FILE_INI) != 0) {
            SendMessage( WM_USER_FILEINI, 0, 0 );     // dialog výběru
            KillTimer( 1 );
      }

      CDialog::OnTimer(nIDEvent);
}
```

Při mazáni souboru není nulováno interní počítadlo průběhu a je nutné je nulovat předem. Ve funkci smazání je nutno udat celkový počet souborů, který se budou mazat, program podle toho spočítá krok průběhu mazání.

```
// spuštěni mazáni souboru
void CDlgPanelExplorer::OnDelete()
{
      char cstr[11];
      m_Counter = m_Count = GetCountDeleteFile( );

            // do proměnné cstr vložit jméno mazaného programu

      pdControl_fnc( CONTROL_NULL_PROGRESS );   // nulovaní pocítadla průběhu
            // smazáni souboru
      pdPanelFile( CMD_FILE_DELETE_FILE | 0, cstr, 11, m_Count );
}

// časovač - monitoring přeběhu
void CDlgDemo::OnTimer(UINT nIDEvent)
{
      char cstr[11];

      if (((stat = pdControl_fnc( CONTROL_GETSTATE )) & 0xFFFF0000) == STATUS_END) {
            err = pdControl_fnc( CONTROL_GET_ERRCODE );     // načtení chyby
                  // Chyba nebo konec mazání
            if ((m_Counter == 0) | (err != 0)) {
                  SendMessage( WM_USER_END, err, 0 );
            }
                  // další soubor
            else {
                        // cstr - další jméno mazaného souboru
                  pdPanelFile( CMD_FILE_DELETE_FILE | 0, cstr, 11, m_Count );
                  m_Counter--;                  // počitadlo snížit o jednu
            }
```

```
        }
        m_Progress.SetPos( stat & 0xFFFF );       // zobrazení průběhu

        CDialog::OnTimer(nIDEvent);
}
```

Smazání souboru z flash paměti se provede následujícím příkazem. Monitoring je stejný jako u čtení direktoráře.

```
pdPanelFile( CMD_FILE_DELET_FILE_FLASH | 0, NULL, 0, 0 );
```

SPOUŠTĚNÍ PROGRAMŮ

Funkce slouží pro spouštění programů v panelu. Dále je možné pomocí této funkce nastavit program tak, aby se spustil po zapnutí panelu.

```
int pdPanelFileControl( int cmd_id, char *cstr, int size,
                        int iParam );
```

Definice cmd	Kód	Popis
CMD_FILECONTROL_SET_INIP	0x06000200	Nastavení spouštěcího programu iParam = 0 – hlavní program iParam = 1 – druhý program
CMD_FILECONTROL_GET_INIP	0x06000400	Načtení jména spouštěcího programu iParam = 0 – hlavní program iParam = 1 – druhý program
CMD_FILECONTROL_CONTROLP	0x06000800	Spouštění programů

char *cstr – pointer na pole char se jménem programu
int size – velikost datového pole cstr.

Tabulka kódu parametru iParam ve funkci CMD_FILECONTROL_CONTROLP

iParam	Kód	Popis
DIRECT_F_RUN	0x00000	Spuštění hlavního programu
DIRECT_F_STOP	0x10000	Zastavení hlavního programu
DIRECT_F_GETRUN	0x20000	Vrací jméno hlavního spuštěného programu
DIRECT_F_OFFVIEW	0x30000	Vypnutí zobrazení pouze firmware do verze 8.00
DIRECT_F_ONVIEW	0x40000	Zapnutí zobrazení pouze firmware do verze 8.00
DIRECT_F_INTERUPT	0x50000	Není implementováno
DIRECT_F_RUNRAM	0x60000	Spuštění programu uloženého přímo v RAM paměti
DIRECT_F_RUN_THREAD	0x70000	Spuštění druhého programu
DIRECT_F_STOP_THREAD	0x80000	Zastavení druhého programu
DIRECT_F_GETRUN_INI	0x90000	Vrací jméno spuštěného inicializačního programu

PŘEKLADAČ

Před uložením programu do panelu je dobré zdrojový text zkontrolovat na chyby. Funkce ukládání sice každý program před uložením přeloží a v případě chyby ji nahlásí. To je ovšem zdlouhavé, pokud je totiž textu obsaženo více programů a v posledním je chyba, jsou nejprve uloženy předchozí programy a pak je teprve nahlášena chyba. Pokud chybu opravíte, tak při ukládání se musí přeskočit předchozí programy, které jsou už v panelu uloženy.

```
int  pdTranslator( int cmd_id, char *data, int *size, int *nFirst,
                   int *nEnd );
```

Definice cmd	Kód	Popis
CMD_TRANSLATOR_ALL	0x08000100	Provede testovací překlad programu. Pro překlad použije verzi nastavenou v DataPanel.SetVer
CMD_TRANSLATOR_ALLVER	0x08000200	Provede testovací překlad programu. Pro překlad použije verzi kterou načte z panelu
CMD_TRANSLATOR_TEXT_TO_PRG	0x08000400	Přeloží program, přeložený program vrací v poli data

char *data - pointer na datové pole **BYTE**, musí být definované v programu, který tuto funkci volá.
int *size - Velikost **datového pole data**
int *nFirs - Znak od kterého má byt text překládán
int *nEnd - Konec Textu

Ve funkci **CMD_TRANSLATOR_TEXT_TO_PRG** se přeložený program uloží do datového pole **data** velikost přeloženého programu je v proměnné **size** horních 8 bitů (24-32) obsahuje atribut programu (zobrazovací , kontrolní, nebo inicializační

```
#define FILE_TYPE_PRG          0          // zobrazovací program
#define FILE_TYPE_CTRL         1          // kontrolní program
#define FILE_TYPE_INIT         4          // inicializační program
```

Pokud překladač narazí na chybu, je funkce ukončena a vrátí číslo chyby. Do proměnné **nFirst** je vložen index prvního znaku, kde se chyba vyskytla. Proměnná **nEnd** obsahuje index posledního znaku, kde se v textu chyba vyskytla.

FONTY

Panely s firmware od verze 9,00 umožňují přehrávání fontů v panelu. Fonty jsou uloženy ve FLASH paměti, kde je volných asi 25000 bajtů (podle toho jak velký je firmware). Jeden font o velikosti 8x8 bodů obsadí 2304 bajtů, takže jich je možné do panelu nahrát maximálně 10. V případě fontu 16x8 to je jen 5. Přesný popis formátu fontu naleznete v dokumentaci o fontech.

```
int  pdFont( int cmd_id, BYTE *data, int size, int *iParam1,
             int *iParam2 );
```

Definice cmd	Kód	Popis
CMD_FONT_READ_HEADER	0x09000100	Načtení tabulky fontů iParam2 - velikost znaku 0xXXYY X - šířka, Y - Výška znaku v bodech
CMD_FONT_WRITE_HEADER	0x09000200	Uložení tabulky fontů iParam1 a iParam2 - adresní rozsah paměti fontů
CMD_FONT_READ	0x09000400	Načtení fontu
CMD_FONT_WRITE	0x09000800	Uložení fontu
CMD_FONT_GET_HEADER	0x09001000	Vrací načtenou tabulku
CMD_FONT_GET	0x09002000	Vrací načtená data fontu

char *data - pointer na datové pole **BYTE**, musí být definované v programu, který tuto funkci volá.
int *size - Velikost **datového pole data**
int *iParam1 - Pointer na proměnou typu int
int *iParam2 - Pointer na proměnou typu int

Paměť fontů je rozdělena na dvě části. První je tabulka fontů o velikosti 512 bajtů. Tato tabulka obsahuje 32 položek pro identifikace jednotlivých fontů. Jedna položka má velikost 32 bajtů a jeho struktura je následující:

```
typedef struct Font
{
      BYTE    Size;                 // velikost popisovače - 32
      char    Name[20];             // jméno fontu je-li koratší zakončené nulou
      _int16  Verzion;              // verze fontu
      _int16  ptrFont;              // 16 bitová adresa dat fontu ve flash
      _int16  ptrMask;              // 16 bitová adresa masky fontu
      BYTE    Height;               // výška znaku v bodech
      BYTE    WidthB;               // sirka znaku v bajtech
      BYTE    Width;                // sirka znaku v bodech
      BYTE    free[2];              // 2 bajty volné
}Font;
```

Proměnné typu _int16 jsou 16 bitové a aby obsahovaly správné číslo, musí se prohodit spodních a horních osm bitů (překladač použitý pro firmware panelu používá opačné řazení bajtu u proměnných). U verze je horních osm bitů celé číslo a spodních osm bitů desetiny (verze 15.5 – 0x0F05). Poslední položka tabulky je rezervovaná pro uložení inicializačního souboru. Adresa umístění je ve stejné proměnné **ptrFont**, velikost je v **ptrMask**. Ostatní proměnné kromě jména nejsou použity.
Pro spuštění načtení tabulky slouží příkaz **CMD_FONT_READ_HEADER**. Proměnna **i** není v současné verzi používána, proměnné **size** funkce vloží rozměr znaku, který je možné na panelu zobrazit. Datové pole není použito. Monitorování běhu funkce se provede standardním způsobem.

```
int i, size, err;

err = pdFont( CMD_FONT_READ_HEADER | 0, NULL, 0, &i, &size );
```

Tabulku po načtení lze získat příkazem CMD_FONT_WRITE_HEADER

```
  int AdrFirst;               // začátek adresného prostoru fontu
  int AdrLast;                // konec adresného prostoru fontu
  int err;
  Font tFonts[26];
            // naplnění tabulky fontu
  err =pdFont( CMD_FONT_GET_HEADER | 0, (BYTE*)&tFonts, 512, &AdrFirst, &AdrLast );
            // prohození bajtu u _int16
  for (int i=0; i<32; i++) {
      tFonts[i].Verzion = (tFonts[i].Verzion >> 8) | (tFonts[i].Verzion << 8);
      tFonts[i].ptrFont = (tFonts[i].ptrFont >> 8) | (tFonts[i].ptrFont << 8);
      tFonts[i].ptrMask = (tFonts[i].ptrMask >> 8) | (tFonts[i].ptrMask << 8);
  }
      // může se provést kontola zda jsou adresy správné a ukazují do prostoru
      // fontů
```

Pokud chceme načíst vlastní fontí získá se z tabulky jeho adresa, výpočtem z rozměru velikost a příkazem **CMD_FONT_READ** se spustí čtení.

```
  int size = tFonts[0].Height * tFonts[0].WidthB * 256;     // velikost fontu
  int aFont = tFonts[0].ptrFont;                            // adresa dat fontu
  int aMask = tFonts[1].ptrMask;                            // adresa masky fontu
```

```
int err = pdFont( CMD_FONT_READ | 0, NULL, size, &aFont, &aMask );
```

Nakonec už stačí přečíst načtená data z bufru příkazem **CMD_FONT_GET**. Font i s maskou je nahrán do stejného bufru za sebou, takže je možné načíst postupně jak je to v následujícím příkladu nebo najednou.

```
BYTE bFont[2048];     // data fontu - velikost pro font 8x8
BYTE bMask[256];      // maska fontu

int fOfset = 0;       // data fontu jsou v bufru od začátku
int mOfset = 2048;    // data masky jsou za daty fontu
int err;

err = pdFont( CMD_FONT_GET | id, bFont, 2048, &fOfset, NULL );
err = pdFont( CMD_FONT_GET | id, bMask, 256, &mOfset, NULL );
```

Při ukládání fontu je situace složitější, paměť je totiž rozdělena na bloky po 512 bajtech. Nejprve je tedy nutné zjistit adresy a velikost fontů, vytvořit si mapu obsazených bloků paměti a následně pak ve volných místech nalézt souvislý prázdný prostor pro data fontu (POZOR data se nesmí rozdělit). Maska fontu může být uložena kamkoliv a též se nesmí rozdělit. Pokud ukládáme více fontů, je možné sloučit například dvě masky do jednoho bloku. Pokud ovšem budeme do bloku, který je z poloviny zaplněn, ukládat data, budou data ve druhé polovině smazány. Při hledání volného místa je nutné počítat s možností, že zde může být uložen i inicializační soubor.

```
int fAdr = SearchFree( 2048 );    // adresa volného prostrou pro font 8x8
int mAdr = SearchFree( 256 );     // adresa volného prostrou pro masku
int iSizeAll = 2048 + 256 + 512; // celková velikost font + maska + tabulka
int err;

BYTE bFont[2048];
BYTE bMask[256];

pdControl_fnc( CONTROL_NULL_PROGRESS );      // nulováni počitadla průběhu

// zápis fontu
err = pdFont( CMD_FONT_WRITE | 0, bFont, 2048, &fAdr, &iSizeAll );

          // čekání na ukončeni zápisu - monitoring průběhu

// zápis masky
err = pdFont( CMD_FONT_WRITE | 0, bMask, 256, &mAdr, &iSizeAll );

          // čekání na ukončeni zápisu - monitoring průběhu
          //

// zápis upraven tabulky o nový font
err = pdFont( CMD_FONT_WRITE_HEADER | id, tFonts, 512, 0, 0 );

          // čekáni na ukončeni zápisu - monitoring průběhu
```

Tabulku fontů je nutné uložit vždy při jakékoliv změně, pokud je třeba vymazat některý font stačí vymazat jej z tabulky a celou ji přepsat v panelu. Pokud v tabulce za mazaným fontem jsou jiné fonty, musí se posunou o jednu pozici dolů. Fonty musí být v tabulce řazeny od začátku za sebou bez prázdných míst.

PROMĚNNÉ

V inicializačním souboru lze nastavit proměnné texty které lze zobrazovat v programu. Pomocí tohoto příkazu lze tyto proměnné měnit.
(viz. popis inicializačního souboru)

```
int  pdVariable( int cmd_id, char *cstr, int size, int *iParam1, int
                 *iParam2 );
```

Definice cmd	Kód	Popis
CMD_VARIABLE_GETTYPE	0x0B000200	Načtení typů přoměnných v panelu
CMD_VARIABLE_GET	0x0B000400	Načtení hodnoty proměnné
CMD_VARIABLE_SET	0x0B000800	Zápis hodnoty do proměnné

Před používáním proměnných je nutné nejprve proměnné načíst, což můžeme udělat podle následujícího příkladu do dynamického pole CPtrArray. Pokud ovšem známe veškeré atributy proměnné, můžeme tento krok vynechat. Současná verze firmware umí pouze textové proměnné.

```
typedef struct VARIABLE
{
      char name[21];
      int iSize;
      int iIndex;
      int iType;
      int iColor;
}VARIABLE;

void GetVariable( )
{
      int iParam1;
      int iParam2;
      char name[21];
      VARIABLE *pVar;         // pointer na proměnnou
      CPtrArray  arrVar;      // pole proměnných
            // do iCount se musí vložit jednička aby se „for" provedl jednou
            // po prvním načteni je do teto proměnné vložen skutečny počet
      int iCount = 1

      for (int var=0; var<iCount; var++) {
            param1 = var;                                 // číslo proměnné
                                                          // načtení proměnné z panelu
            pPanel->err = pdVariable( CMD_VARIABLE_GETTYPE | 0, name, 21, &param1,
                                      &param2 );
            iCount = (iParam2 >> 8) & 0xFF;               // počet proměnných
            if (iCount == 0) break;                       // panel neobsahuje proměnné

            pVar = new VARIABLE;
            pVar->iIndex = iParam1 & 0xFF;                // číslo proměnné
            pVar->iType  = (iParam1 >> 8) & 0xFF;         // typ proměnné
            pVar->iColor = (iParam1 >> 16) & 0xFF;        // barva
            pVar->iSize  = iParam2 & 0xFF;                // velikost proměnné
            strcpy( pVar->name, name );                   // jméno proměnné

            arrVar.Add( (void*)pVar );                    // přidání proměnné do pole
      }
}
```

Načtení proměnné se provede podle následujícího příkladu. Velikost textového pole musíte dát dostatečně velké, jelikož proměnná může mít maximálně 255 znaků, tak definice v příkladu je dostačující pro všechny případy.

```
void GetVariable( VARIABLE pVar )
{
      int param1 = pVar->type;      // typ proměnné
      int param2 = pVar->index;     // index proměnné
      int size   = pVar->size;      // velikost proměnné
      char text[256];               // text proměnné
                                    // načteni textu proměnné
      int err = pdVariable( CMD_VARIABLE_GET | 0, text, size, &param1, &param2 );
}
```

Zápis proměnné je obdobný jako čtení

```
void SetVariable( VARIABLE pVar, char cstr )
{
      int param1 = pVar->type;           // typ proměnné
      int param2 = pVar->index;          // index proměnné
      int size   = pVar->size;           // velikost proměnné
                                         // zápis proměnné
      int err = pdVariable( CMD_VARIABLE_SET | 0, cstr, 256, &param1, &param2 );
}
```

PŘÍMÁ KOMUNIKACE

Pomocí této knihovny lze také komunikovat přímo odesláním daného paketu.

```
int  pdDirectSendPanel( int cmd_id, BYTE *data, int iParam1,
                        int iParam2 );
```

Definice cmd	Kód	Popis
CMD_PANELMSG_SEND	0x0F000100	Přímá komunikace s panelem

char *data - pointer na datové pole **BYTE**, musí být definované v programu, který tuto funkci volá. Obsahuje odesílaná nebo přijatá data a mělo by být veliké minimálně 32 bajtů
int iParam1 - Kód komunikačního příkazu
int iParam1 - Spodních 16 bitů adresa, horních 16 bitů velikost

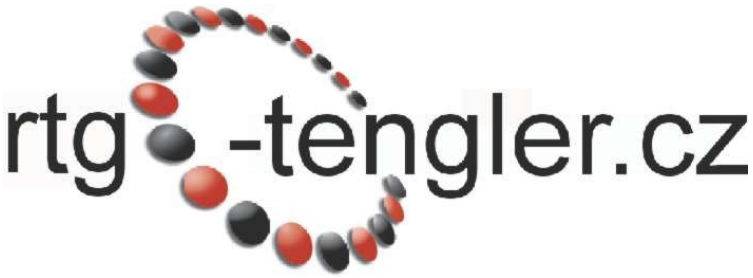
Podrobnější informace o komunikaci s panelem naleznete v dokumentaci „**Popis komunikace**“. Do pole data se vkládá obsah datového pole komunikace. Parametr 1 obsahuje kód příkazu (viz. Tabulka). Parametr 2 pak obsahuje adresu dat a velikost datového pole odesílaného paketu.

Definice cmd	Kód	Popis
MSG_WRITE_FL_PRG	0x22	Zápis do FLASH paměti
MSG_EREASE_BLOCK	0x23	Smazaní bloku FLASH
MSG_WRITE_EEPROM	0x40	Zápis do EEPROM
	0x61	
MSG_WRITE_DIRECTP	0x62	Zápis programu do RAM
MSG_SET_RTC	0x80	Nastavení hodin
MSG_WRITE_VAR	0x68	Zápis proměnné
	0x81	
MSG_SET_OVERTIME	0xE2	Zápis registru platnosti dat
MSG_READ_FL_PRG	0x32	Čtení z FLASH paměti
MSG_READ_EEPROM	0x50	Čtení z paměti EEPROM
	0x71	
MSG_READ_VAR	0x78	Čtení proměnné
MSG_GET_RTC	0x90	Nastavení reálného času
	0x91	
MSG_GET_LIGHT	0xD0	Vrací aktuelní jas panelu
MSG_GET_NAME_FONT	0xE1	Vrací jméno fontu v panelu
MSG_GET_OVERTIME	0xF2	Přečtení registru platnosti dat
MSG_SET_CONTROLP	0xA0	Spouštění programů v panelu
MSG_GET_CONTROLP	0xB0	Vrací číslo spuštěného programu
MSG_SET_LIGHT	0xC0	Nastavení jasu panelu
MSG_GET_VERSION	0xE0	Přečte verzi firmware panelu
MSG_GET_IDETNT	0xE3	Vrací identifikační pole modulu
	0xE5	
MSG_GET_ANALOG	0xE7	Vrací hodnotu analog vstupů
MSG_SET_EEPROM	0xF4	Nastavení EEPROM – při formátování
	0xF5	
MSG_MOD_WR_FLASH	0xF6	Přepnutí módu zápisu do FLASH paměti
MSG_SET_ANALOG	0xF7	Nastavení analogových vstupů

Tabulka chybových kódů

Definice	Kód	Popis
ERR_NULL	0	
COM_ERR_NOINIT	1	COM port není inicializován
COM_ERR_NOCOM	3	COM port neexistuje
COM_ERR_NOSEND	16	Data nelze odeslat na COM port
COM_ERR_NOREAD	33	Data ze nelze přijmou
COM_ERR_NORE	34	Panel na zvoleném portu neodpovídá
COM_ERR_DATALONG	35	Přijatý paket je delší jak 64 bajtů
COM_ERR_NOSTART	36	Přijatý paket neobsahuje start bajt 0xC0
COM_ERR_ADRSRC	37	Přijatá paket má jinou adresu příjemce
COM_ERR_ADRDST	38	Přijatý paket má jinou adresu odesílatele
COM_ERR_NOSTOP	39	Přijatý paket neobsahuje stop bajt 0xC1
COM_ERR_CRC	40	Přijatý paket má chybný kontrolní součet
COM_ERR_NOANSWER	41	Přijatá paket není odpověď od panelu
ERR_NO_CARRIER	48	GSM modem - zavěšení
ERR_NO_DIALTONE	49	GSM modem - Obsazeno
ERR_BUSY	50	GSM modem – modem je zaneprázdněn
ERR_CONNECT	51	GSM modem – Spojení otevřeno
ERR_NOMODEM	52	GSM modem – není připojen modem
ERR_NO_ATH	53	GSM modem – Zavěšení modemu
ERR_NOCONNECT	54	GSM modem – Spojení nelze navázat
ERR_NO_SET	55	GSM modem – modem nelze nastavit
ERR_IP_NOCONNECT	64	IP adresa neexistuje
ERR_IP_NOTRANSMIT	65	Data na IP adresu nelze odeslat
ERR_IP_NORECEIVE	66	Vypršel čas přijmu z IP adresy
ERR_STATE_HEADER_LOADET	96	Tabulka fontů je již načtena
ERR_RELEVANCE	97	Neodpovídá registr platnosti dat
ERR_ID_NOEXIST	120	Index (cmd_id) panelu je chybný – panel neexistuje
ERR_CRC_BOOT	129	Chybný CRC BOOT sektoru - paměť není zformátována
ERR_NAME_NOT_EXIST	130	Jméno programu v panelu neexistuje
ERR_FAT_0	131	Chyba ve FAT tabulce
ERR_FAT_1	132	Chyba ve FAT tabulce
ERR_CRC	133	Chybný CRC načteného sektoru
ERR_NO_MEMORY	136	Paměť panelu je plná
ERR_DIR_FULL	137	Direktorář je plný
ERR_NAME_EXIST	138	Jméno programu v panelu již existuje
ERR_FORMAT_NOT_POSISIBLE	139	Paměť panelu nelze zformátovat
ERR_FONT_NOT_EXIST	140	Font v panelu neexistuje
ERR_NOFONT_DOWNLOAD	141	Panel nepodporuje editaci fontů
ERR_FONTMEMORY_FULL	142	Paměť pro fonty je plná
ERR_UNKNOWN	144	Neidentifikovaná chyba
ERR_PARAMETR_NULL	145	Parametr funkce je nulový
ERR_NO_PASWORD	146	Panel nepodporuje ochranu heslem
ERR_NOMEMORY	147	Nezdařilo se přidělení paměti pro tuto funkci
ERR_FILE_NOT_EXIST	148	Soubor s grafikou pro panel neexistuje
ERR_PARAMERT	149	Chybný parametr funkce
ERR_FNC_COMMAND	150	Chybný command funkce
ERR_FNC_SIZEDATA	151	Chybná velikost datového pole ve funkci
ERR_FNC_PARAMETR	152	Některý z parametrů funkce je chybný
ERR_THREAD_BUSY	153	Již je spuštěná jiná funkce na pozadí
ERR_NO_MODULE	154	Program neobsahuje modul proměnných
ERR_COMMAND_NAME	160	Příkaz je jméno programu
ERR_NO_COMMAND_2	161	
ERR_NO_COMMAND_3	162	
ERR_LONG_NAME	163	Jméno programu je delší jak 10 znaků
ERR_NO_END	164	Program není ukončen
ERR_NO_END_COMMAND	165	Příkaz není uzavřen

ERR_UNKNOWN_TYPE	166	Neznámy typ programu
ERR_NO_PROGRAM	167	Text neobsahuje žádný program
ERR_COMAND_NOT_SUPPORT	168	Panel tento příkaz nepodporuje
ERR_MESSAGE_NO_SUPPORT	169	Panel nepodporuje systém zpráv
ERR_NO_CLOSE_COMMAND	170	Program není ukončen
ERR_PARAMETR_COLOR	171	
ERR_FORMAT_NAME	172	Chybný formát jména programu
ERR_FORMAT	173	Chybný formát příkazu
ERR_SPEED_OUT_OF_EXTENT	176	Parametr příkazu posun může být 1 - 9
ERR_STATIC_OUT_OF_EXCENT	177	Parametr příkazu staticky může být 0 - 255
ERR_PAUSE_OUT_OF_EXCENT	178	Parametr příkazu pauza může být 0 - 255
ERR_FORMAT_FONT_1	179	Chybný formát příkazu font
ERR_FORMAT_FONT_2	180	Chybný formát příkazu font
ERR_LIGHT_OUT_OF_EXCENT	181	Parametr příkazu jas může být 0 - 100
ERR_EXECUTE_FORMAT	182	Chybný formát příkazu spusť
ERR_EXECUTE_OUT_OF_EXCENT	183	Parametr příkazu spusť může být 1 - 100
ERR_EXECUTE_NAME_LONG	184	Jméno v příkazu spusť muže mít max. 10 znaků
ERR_GOTO_NAME_LONG	185	Jméno v příkazu proveď muže mít max. 10 znaků
ERR_LINE_OUT_OF_EXCENT	186	Parametr příkazu řádky může být 1 - 20
ERR_LINE_PARAM	187	Chybný parametr čísla řádku
ERR_PARAM_POZICE_X	188	Grafický příkaz má chybný parametr pozice X
ERR_PARAM_POZICE_Y	189	Grafický příkaz má chybný parametr pozice Y
ERR_PARAM_PICTURE	190	Grafický příkaz má chybný číslo obrázku
ERR	192	Chyba
ERR_FORMAT_TIME	193	Chybný formát času
ERR_FORMAT_COMMAND	194	Chybný formát příkazu
ERR_FORMAT_WEEK	195	Chybný formát dnů v týdnu
ERR_PARAEL_OUT_OF_EXCENT	196	Kód paralelního vstupu může být 0 - 255
ERR_UNKNOWN_TYP_DIRECT	197	Neznámý typ řízení vstupu
ERR_COUNT_CIKL	198	Chybný typ počtu provedení programu
ERR_ORDER_COMMAND	199	Chybné pořadí program
ERR_VAR_LONGN	208	Jméno proměnné může být dlouhé max. 20 znaků
ERR_VAR_FORMAT	209	Chybný formát proměnné
ERR_VAR_FORMAT_COLOR	210	Chybný formát barvy proměnné
ERR_FINI_PRG	211	Chybný formát záznamu spouštěného programu
ERR_TIMER_TYPE	212	Neznámý typ časovače
ERR_TIMER_FORMAT	213	Chybný formát zápisu časovače
ERR_FINI_STATIC_FORMAT	214	Chybný formát zápisu statické řádky
ERR_FINI_STATIC_NOVAR	215	Proměnná není definovaná
ERR_FINI_STATIC_LONG	216	Proměnné jsou delší jak statická řádka
ERR_FINI_STRING	217	
ERR_IF_VARTYPE	218	
ERR_IF_CONDITION	219	
ERR_IF_FORMAT	220	
ERR_INI_NOWRITE_FL	224	Soubor nelze zapsat do FLASH paměti
ERR_TERMINATE	255	Provádění funkce bylo přerušeno



Výrobce: **RTG Mělník** tel/fax. 315624739 mob. 603261914
www.rtg-tengler.cz email: rtg@rtg-tengler.cz
www.svetelnepanely.cz

Software: **Vacek Luboš** tel. 606485842 email: vacek@svetelnepanely.cz