POPIS FUNKCÍ KNIHOVNY
Serial.dll

Obsah:

1. UVOD

Knihovna „**Serial.dll**“ obstarává komunikaci s panelem a to přes sériový port RS232 nebo RS485 tak po síti pomocí IP adresy. Obsahuje také funkce pro řízení GSM modemu. Současná verze programu „**Panel 2“** již tuto knihovnu nepoužívá. Funkce této knihovny jsou zahrnuty v knihovně „**PDriver.dl“l**.

2. Funkce nastavení COM portu (RS232 – RS485)

2.1 bool SetCOM(char *port*, int *baud*, BYTE *StopBits*);

Touto funkcí je možno provést nastavení sériových portů (COM1 – COM6), každý z portů lze nastavit samostatně. Je-li třeba změnit natavení portu, musíte nejprve port uvolnit funkcí *DoneCOM* a pak opětovně nastavit. Nastavení portu 8 bitů, bez parity.

Návratová hodnota:

Typ *bool* udává výsledek nastavení portu, **false** pokud port nelze nastavit (port používá jiná aplikace nebo na tomto počítači není nainstalován) a **true** v případě, že port byl nastaven.

port

číslo nastavovaného portu (1 – COM1 až 6 COM6).

baud

Bytová rychlost sériového portu (***9600***, 19200, 38400, **57600**, 115200). Panely s verzí nižší než 8.00 mají komunikační rychlost 9600, od verze 8.00 je použita rychlost 57600.

StopBits

Nastavení počtu stop bitů, nastavuje se „0“ (jeden stop bit) pouze u starších panelů s verzí ve tvaru data (např. 11.10.99) je třeba nastavit „2“ (dva stop bity).

2.2 void DoneCOM(char *port*);

Uvolnění portu pro použití jinou aplikací případně pro nové natavení s jinými parametri.

port

číslo nastavovaného portu (1 – COM1 až 6 COM6)

2. 3 void SetTimeOut(char *port*, DWORD *ReadTotalTimeoutMultiplier*);

Nastavení času čekání na odpověď od panelu, po vypršení času komunikační funkce vrací chybu (0x22 – panel neodpovídá). Při nastavení portu funkcí SetCOM je tento čas nastaven na 10ms.

port

číslo nastavovaného portu (1 – COM1 až 6 COM6)

ReadTotalTimeoutMultiplier

Hodnota času v milisekundách.

Příklad komunikace pomocí sériového portu:

```
void COM_port( )
{
        // nastaveni portu  COM1, 8, bezparity, jeden stop bits
    SetCOM( 1,  57600, 1 );
    SetTimeOut( 1,  100 );

        // příkazy řízení portu
        // ...

        // uvolnění portu COM1
    DoneCOM( 1 );
}
```

3. Funkce nastavení TCP/IP

Pro komunikaci IP adresou se používají stejné funkce jako při komunikaci přes COM port, s tím rozdílem, že **port** ve struktuře **TPCOM** musí být **-1** (0xFF).

3.1 bool IPConnect(LPCTSTR *strServerIp*, long *IPort*, int *timeout*);

Otevření spojení se síťovou adresou a portem. Otevřít spojení lze pouze s jednou adresou a portem, pokud je spojení již otevřeno (i s jinou adresou) funkce je ukončena.

Návratová hodnota:

Stav otevření spojení. Pokud nelze spojení s IP adresou a portem navázat, vrací funkce **false**. Je-li spojení otevřeno, vrací funkce **true**. Pokud je již nějaké spojení aktivní vrací true.

strServerIp

Pointer na ředězec udávající síťovou IP adresu panelu (format „128.56.22.8“).

IPort

Číslo portu.

timeout

Čas čekání na odpověď od panelu v **milisekundách**. Nepřijde-li do tohoto času od panelu odpověď, je komunikační funkce ukončena s návratovým kédem chyby.

3.2 void IPDisConnected( );

Ukončení spojení se síťovou adresou. Funkce nejprve zjistí, jestli je spojení aktivní, a pokud je aktivní, ukončí jej. Pokud není, funkce se ukončí bez změny.

3.3 bool IPIsConnected( );

Funkce zjišťuje, zda je spojení aktivní.

Návratová hodnota:
Je-li spojení aktivní vrací **true**, v opačném případě vrací **false**.
Příklad komunikace přes IP adresu:

```
void IP_adresa( )
{
        // zavření případného spojení
    if ( IPIsConnected( ) = = true ) IPDisconnected(  );

        // datové pole s IP adresou  120.56.22.8
    char IPadresa[16];                          // 16 bajtů dlouhé datové pole
    strcpy( &IPadresa[0], „128.56.22.8“ );    // řetězec zakončený nulou
        // otevření IP adresy s portem 100. Timeout nastaven na 100 ms
    if ( IPConnect( &Ipadresa[0],  100, 100 ) = = false) return;

        // příkazy řízení portu stejné jako u komunikace RS232
        // ...

        // uvolnění portu COM1
    DoneCOM( 1 );
}
```

4. Komunikační funkce

4.1 BYTE TransmitData(BYTE *command*, TPCOM **tpcom*, TPDATA **tpdata*);

Funkce odesílá data do panelu. Komunikace muže probíhat přes COM port nebo IP adresu.

Návratová hodnota:
Kód chyby pokud nastane při komunikaci s panelem, proběhne-li komunikace v pořádku, panel vrací **0**.

Command
Typ příkazu zápisu dat do panelu.

Tabulka příkazů

Definice	Kód	Popis
MSG_WRITE_EEPROM	64	Zápis do interní paměti panelu EEPROM.
MSG_SET_SLINE	97	Zobrazení statického textu na řádku.
MSG_WRITE_DIRECTP	98	Přímý zápis programu do pracovní paměti panelu.
MSG_SET_RTC	128	Nastavení reálného času panelu.
MSG_SET_OVERTIME	129	Nastavení registru „Overtime“ platnost dat
MSG_SET_CONTROLP	160	Spuštění / zastavení programů
MSG_SET_LIGHT	192	Nastavení intenzity svitu panelu
MSG_SET_EEPROM	244	Nastavení velikosti EEPROM

tpcom
Pointer na datovou TPCOM udávající přes který port má komunikace probíhat.

tpdata
Pointer na strukturu TPDATA obsahující odesílaná data.

4.2 BYTE ReceiveData(BYTE *command,* TPCOM **tpcom*, TPDATA **tpdata*);

Funkce vyžádá data z panelu. Komunikace muže probíhat přes COM port nebo IP adresu.

Návratová hodnota:
Kód chyby pokud nastane při komunikaci s panelem, proběhne-li komunikace v pořádku, panel vrací **0**.

Command
Typ příkazu čtení dat z panelu viz. následující tabulka

Definice	Kód	Popis
MSG_READ_EEPROM	80	Čtení z interní paměti panelu EEPROM.
MSG_GET_SLINE	113	Načtení textu statické řádky.
MSG_GET_RTC	144	Načtení reálného času panelu.
MSG_GET_OVERTIME	242	Nastavení registru „Overtime“ platnost dat
MSG_GET_CONTROLP	176	Vrací číslo spuštěného programu
MSG_GET_LIGHT	208	Nastavení intenzity svitu panelu
MSG_GET_NAME_FONT	225	Přečte jméno fontu z panelu.
MSG_GET_VERZION	224	Načte verzi firmware panelu.

tpcom
Pointer na datovou TPCOM udávající přes který port má komunikace probíhat.

tpdata
Pointer na strukturu TPDATA obsahující přijímaná data.

4.3 BYTE ResetPanelu(char port, BYTE panel);

Provede restart panelu. Panel na tento příkaz neodpovídá.

Návratová hodnota:
Kód chyby pokud nastane při komunikaci s panelem, proběhne-li komunikace pořádku panel vrací **0**.

port
Číslo portu přes který má komunikace probíhat (1-6). Má li komunikace být IP adresou musí mýt hodnotu -1 (0xFF).

panel
Adresa panelu, u většiny panelů má hodnotu 1. Pouze pokud jsou panely připojené na RS485 může být jiná (je uvedena na štítku panelu). Panel řízený IP adresou má vždy adresu 1. Pokud uvedete jinou adresu panel nebude na příkaz reagovat.

4. 4 Struktura určující cíl komunikace

```
typedef struct TPCOM
{
	char port;
	BYTE panel;
	bool OnCheck;
}TPCOM;
```

port

Číslo portu přes který má komunikace probíhat (1-6). Má-li komunikace být IP adresou musí mít hodnotu -1 (0xFF).

panel

Adresa panelu, u většiny panelů má hodnotu 1. Pouze pokud jsou panely připojené na RS485 může být jiná (je uvedena na štítku panelu). Panel řízený IP adresou má vždy adresu 1. Pokud uvedete jinou adresu panel nebude na příkaz reagovat.

OnCheck

Zapnutí provádění kontrolního součtu přenosu. Vypnuto **false**, zapnuto **true.** Kontrolní součet je používán u panelů od verze panelu 7.64, u panelů s nižší verzí musí být kontrolní součet vypnut (false).

4.5 Struktura obsahující data a adresu dat v panelu

```
typedef struct TPDATA
{
	int	adr;
	int	size;
	BYTE *data;
}TPDATA;
```

adr

adresa dat v panelu (například adresa v paměti EEPROM

size

Velikost přijímaných nebo odesílaných dat

data

pointer na datové pole obsahující odesílaná nebo přijímaná data

Tabulka kódů chyb při komunikaci.

Kód	Popis
0x01	Port není nastaven.
0x10	Chyba při odesílání dat.
0x21	Chyba při příjmu dat.
0x22	Panel neodpovídá (vypršel Timeout).
0x41	Chyba při odesílání dat do sítě TCT/IP
0x42	Chyba při příjmu dat TCP/IP (panel neodpověděl v daném čase).

Přijatá data	
0x23	Délku větší než 64 bitů.
0x24	Nenalezen start bajt 0xC0.
0x25	Jiná adresa příjemce (musí být 0x80)
0x26	Jiná adresa odesílatele (panelu) než v odesílaných datech
0x27	Nenalezen stop bajt 0xC1.
0x28	Chybný kontrolní součet.
0x29	Data nejsou odpověď.

5. Příklady funkcí řízení panelu

5.1 Nastavení COM portu

```
TPCOM tpcom;

void SetCom( )
{
        SetCOM( 1,  57600, 1 );

        TPCOM tpcom;            // struktura cílu komunikace
        tpcom.port = 1
        tpcom.adresa = 1;
        tpcom.OnCheck = false
}
```

5.2 Nastavení intenzity svitu panel

```
// parametr jas udává intenzitu svitu v procentech 0 - 100%
BYTE SetJas( char jas  )
{
        BYTE data[2];
        data[0] = jas;          // intenzita svitu panelu
        data[1] = 1;            // panel bude odpovídat (0 = panel odpovídat nebude)

        TPDATA tpdata
        tpdata.adr = 0;
        tpdata.size = 2;
        tpdata.data = &data[0];
        return TransmitData( MSG_SET_LIGHT, &tpcom, &tpdata );

}
```

5.3 Zjištění aktuální intenzity svitu panelu

Načtenou hodnotu svitu uloží funkce na adresu danou pointrem, jas v parametru funkce

```
BYTE GetJas( char *jas  )
{
        TPDATA tpdata
        tpdata.adr = 0;
        tpdata.size = 1;
        tpdata.data = jas;
        return  ReceiveData( MSG_GET_LIGHT, &tpcom, &tpdata );
}
```

5.4 Nastavení reálného času

```
BYTE SetJas( )
{
                // datové pole času a datumu   20.9.2003    22:26:00
        BYTE data[8];
        data[0] = 0x00;                 // vteřiny
        data[1] = 0x29;                 // minuty
        data[2] = 0x22;                 // hodiny
        data[3] = 0x01;                 // týden
        data[4] = 0x20;                 // den
        data[5] = 0x09;                 // měsíc
        data[6] = 0x03;                 // rok
        data[7] = 0x00;

        TPDATA tpdata
        tpdata.adr = 0;
        tpdata.size = 8;
        tpdata.data = &data[0];
        return  = TransmitData( MSG_SET_TRC, &tpcom, &tpdata );
}
```

5.5 Načtení reálného času panelu

```
void GetJas( )
{
                // datové pole času formát stejný jako u zápisu
        BYTE data[8];                   // načtený jas

        TPDATA tpdata
        tpdata.adr = 0;
        tpdata.size = 8;
        tpdata.data = &data[0];
        return = ReceiveData( MSG_GET_RTC, &tpcom, &tpdata );
}
```

5.6 Zobrazení statické řádky

Aby se text zobrazil, nesmí být zapnutý dynamický program.

```
BYTE SetJas(  )
{
        // datové pole zobrazovaného textu
    BYTE data[32];
    data[0] = 1;            // zarovnání textu vlevo
    data[1] = 1;            // zapnutí redukce mezer
    strcpy( &data[2], “Pokusný TEXT“ );

    TPDATA tpdata
    tpdata.adr = 1;             // číslo řádku
    tpdata.size = 32;           // počet bajtů
    tpdata.data = &data[0];
    return = TransmitData( MSG_SET_SLINE, &tpcom, &tpdata );
}
```

5.7 Přečtení statické řádky

```
BYTE GetJas( )
{
        // datové pole statického textu stejný formát jako při zápisu
    BYTE data[64];          // velikost při čtení musí být vžty 64 bajtů

    TPDATA tpdata
    tpdata.adr = 0;             // řádek 1
    tpdata.size = 32;           // velikost dat
    tpdata.data = &data[0];
    BYTE err = ReceiveData( MSG_GET_SLINE, &tpcom, &tpdata );
}
```

První bajt – zarovnání textu 1 = vlevo, 2 = uprost5ed, 3 = vpravo.
Druhý bajt – redukce mezer 0 = redukce vypnuta, 1 redukce zapnuta.
Od třetího bajtu zobrazovaný text

5.8 Formátování paměti EEPROM

Aby bylo možno zapisovat programy do paměti panelu je nutné nejprve paměť EEPROM zformátovat. Popis formátu je v dokumentaci „*Protokol zobrazovacích panelů*“ kapitola „*Formát paměti panelu*“. Formátováním je nejprve nutné zjistit velikost paměti, pak zapsat BOOT záznam, vymazat FAT tabulku a seznam jmen DIR.

```
// FUNKCE formátování paměti
// Zformátuje paměť do velikosti 64k bajtů. Paměť panelu může být osazena dvěma obvody

    // kopie dat z panelu - pole jsou použité v ostatních funkcích
BYTE boot[144];             // BOOT zaznam
BYTE fat[256];              // FAT tabulka
BYTE dir[256];              // jména programů
```

```
BYTE WriteSector( int sector, BYTE *data );
BYTE SaveBOOT( int segment );
BYTE LoadBOOT( int segment );

// FORMATOVÁNÍ PAMĚTI PANELU
void FormatEEPROM(  )
{
        memset( &boot[0], 0, 144 );             // nulování BOOT zaznamu
        boot[8] = 128;
        boot[9] = 1;
        boot[10] = 2;

        if (SaveBOOT( 0  ) != 0) return;        // zápis boot do segmentu 0
        if (LoadBOOT( 0 ) != 0) return;         // paměť nelze zformátovat

                // nastavení nulové velikosti EEPROM
        BYTE data[5];
        data[0] = 0;
        data[1] = 0;
        data[2] = 1;
        data[3] = 2;
        data[4] = 0;
        tpdata.adr = 0;
        tpdata.size = 5;
        tpdata.data &data[0];
        if (TransmitData( MSG_SET_EEPROM, &tpcom, & tpdata ) != 0) return;

                // zjištění velikosti jedné paměti
        BYTE err;
        for (int i=2; i<256; i*=2) {
                if (( err == LoadBOOT( i )) == 0) break;
                if (err != 0x81) return;        // chyba při komunikaci
        }

                // nastavení velikosti jedné paměti EEPROM
        data[4] = i;
        if (TransmitData( MSG_SET_EEPROM, &tpcom, & tpdata ) != 0) return;

                // zjištění zda panel obsahuje i druhou pamět
        if (SaveBOOT( i ) != 0) return;         // zápis boot do segmentu i
        if (LoadBOOT( i ) == 0) i *= 2;         // test druhé paměti

                // nastavení celkové velikosti paměti
        data[0] = i;
        if (TransmitData( MSG_SET_EEPROM, &tpcom, & tpdata ) != 0) return;

                // zápis boot záznamu
        boot[8] = data[0];
        boot[9] = 1;
```

```
        boot[10] = 1;
        boot[11] = 2;
        boot[13] = data-1;
        if (SaveBOOT( i ) != 0) return;             // zápis boot do segmentu 0

                // nulování FAT tabulky druhý segment paměti
        memset( &fat[0], 0, 256 );
        fat[0] = 1;     // boot sektor = obsazeno
        fat[1] = 1;     // fat tabulka = obsazeno
        fat[2] = 1;     // jména programů obsazeno
        if (WriteSector( 1, &fat[0] );

                // nulování seznamu jmen panelu
        memset( &dir[0], 0, 256 );
        if (WriteSector( 2, &dir[0] );
}

// ------------------------------------------------------------------------------------------------------------------------
--------
// FUNKCE zápisu BOOT sektoru
// segment = horních 8 bitů adresy
BYTE SaveBOOT(  int segment )
{
        BYTE err;
                // výpočet kontrolního součtu BOOT sektoru
        int crc = 0;
        for (int i=0; i<142; i++) crc += boot[i];
        crc = crc ^ 1;
        boot[142] = (BYTE)(crc>>8);
        boot[143] = (BYTE)crc;
                // zápis boot záznamu
        TPDATA tpdata;
        tpdata.size = 32;
        int adr = 0;
        while (adr  == 134) {
                tpdata.adr = (segment<<8) | adr;
                tpdata.data = &boot[adr];
                err = TransmitData(  MSG_WRITE_EEPROM, &tpcom, &tpdata );
                if (err != 0) return err;
                adr += 32;
        }
        return 0;
}

// FUNKCE čtení BOOT sektoru
// segment = horních 8 bitů adresy
BYTE LoadBOOT( int segment )
{
```

```
        BYTE err;
                // načtení boot záznamu
        TPDATA tpdata
        tpdata.size = 32;
        int adr = 0;
        while (adr == 134) {
                tpdata.adr = (segment<<8) | adr;
                tpdata.data = &boot[adr]
                err = ReceiveData( MSG_READ_EEPROM, &tpcom, &tpdata );
                if (err != 0) return err;
                adr += 32;
        }
                // ověření kontrolního součtu
        int crc = 0;
        for (int i=0; i<142; i++) crc += boot[i];
        adr = (int)boot[142] << 8 | boot[143];
        if (adr != crc) return 0x81;                    // chybný kontrolní součet
        return 0;
}
```

```
// FUNKCE zápis segmentu (256 bajtů) do paměti panelu
// segment = horních 8 bitů adresy
BYTE WriteSector( int sector, BYTE *data )
{
        BYTE err;
        TPDATA tpdata;
        tpdata.size = 32;

        for (int adr=0; adr<255; adr+=32) {
                tpdata.adr = (sector<<8) | adr;
                tpdata.data = &data[adr];
                if ((err = TransmitData( MSG_WRITE_EEPROM, &tpcom, &tpdata )) != 0) return
err;
        }
}
```

5.9 Načtení „FAT“ a seznamu jmen

```
BYTE OverTime[5];          // registr OverTime

bool TestOverTime(  );
BYTE LoadSector( int sector,  BYTE *data  );
BYTE WriteOverTime(  );

BYTE LoadDIR( )
{
        BYTE err;
                // testuje platnost dat načtených z panelu
                // pokud je registr v OverTime a registr v panelu schodný, funkce je ukončena
```

```
        if (TestOverTime( &tpcom ) == true) return false;

        if ((err = LoadSector( &tpcom, 1,  &fat[0]  )) != 0) return err;  // načtení „fat“
        if ((err = LoadSector( &tpcom, 1, &dir[0]  )) != 0) return err;   // načtení jmen
                // nastavení registru OverTime
        return WriteOverTime( &tpcom );
}
```

```
// TEST registru OverTime
bool TestOverTime(  )
{
        BYTE err;
        TPDATA tpdata;
        BYTE data[5];
                // Načtení registru OverTime
        tpdata.adr = 0;
        tpdata.size = 5;
        tpdata.data = &data[0];
        if ((err = ReceiveData( MSG_GET_OVERTIME, &tpcom, &tpdata );
                // porovnání načteného registru a registu v paměti počítače
        for (int i=0; i<5; i++) if (data[i]  != OverTime[i]) return false;
        return true;
{
```

```
// Zápis registru OverTime
#include <sys/timeb.h>
BYTE WriteOverTime(  )
{
        struct timeb timebuffer;
        memcpy( OverTime[0], &timebuffer.time, 5 );

        TPDATA tpdata;
        tpdata.adr = 0;
        tpdata.size = 0;
        tpdata.data = &OverTime[0];
        return TransmitData( MSG_SET_OVERTIME, &tpcom, &tpdata );
}
```

```
// funkce Načtení sektoru o délce 256 bajtů
BYTE LoadSector( int sector,  BYTE *data  )
{
        BYTE err;
        TPDATA tpdata;
        tpdata.size = 32;

        for (int adr=0; adr<255; adr+=32) {
```

```
                tpdata.adr = (sector<<8) | adr;
                tpdata.data = &data[adr];
                if ((err = ReceiveData( MSG_WRITE_EEPROM, &tpcom, &tpdata )) != 0) return
err;
        }
}
```

5.10 Smazání programu z paměti EEPROM

Smazání programu je provedeno zapsáním „0x00“ na pozici prvního znaku mazaného jména a uvolnění všech segmentů programu ve „fat“.

```
// funkce Načtení sekotru o délce 256 bajtů
BYTE DeletePrg( char *name  )
{
        BYTE err;

        TPDATA tpdata;
        LoadDIR( );                     // načtení jmén a fat

        for (int i=0; i<16; i++) if (strncpy ( &dir[i<<4], name, 10 ) == 0) break;
        if (i == 16) return 0;          // jméno neexistuje
                // smazání jména programu
        dir[(i<<4)+16] = dir[(i<<4)+16] - dir[i<<4];                    // kontrolní součet
        dir[i<<4] = 0;
        tpdata.adr = 0x0200 + (i << 4);     // sektor 2 + adresa jména (číslo jména * 16)
        tpdata.size = 16;
        tpdata.data = &dir[i<<4];
        if ((err = TransmitData( MSG_WRITE_EEPROM, &tpcom, &tpdata )) != 0) return err;
                // smazání programu z „fat“
        int s_del = dir[(i<<4)+14];  // číslo prvního sektoru programu
        int s_next;
        tpdata.size = 1;
        for (; ; ) {
                s_next = fat[s_del];
                fat[s_del] = 0;
                tpdata.adr = 0x0100 + s_del;
                tpdata.data = &fat[s_del];
                if ((err = TransmitData( MSG_WRITE_EEPROM, &tpcom &tpdata)) != 0) return
err;
                if (s_next == 1) break;
                s_del = s_next;
        }
        return 0;
}
```

5.11 Zápis programu do paměti EEPROM

Program je při zápisu do paměti rozdělen do segmentů o velikosti 254 bajtů, každý segment je doplněn o dva bajty kontrolního součtu a uložen do volných segmentů paměti panelu. Pokud program v paměti již existuje je nejprve smazán. Jednotlivé bajty ve „fat“ odpovídají sektorům a do příslušného bajtu segmentu ve „fat“ je nahráno číslo nasledujícího segmentu, pokud je to poslední segment je nastaven na 1.

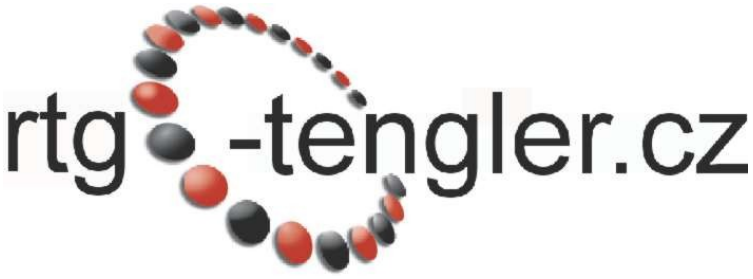
```
// funkce Načtení sekotru o délce 256 bajtů
BYTE SavePrg( char *name, BYTE data, int size  )
{
	BYTE err;
	TPDATA tpdata;

	if ((err = DeltePrg( name )) != 0) return err;		// pokud v programu existuje smazání

	int free = 0;		// číslo volného segmentu v paměti
	int item = 0;		// pozice v datech
	int seg = 0;		// segment
		// jméno programu
	for (int in=0; in<16; in++) if (dir[in<<4]  == 0) break;
	if (in == 16) return 1;				// není volné místo v seznamu
	memcpy( &dir[in<<4] , name, 10 );		// jméno programu
	dir[(in<<4) + 10] = 0;				// atribut zobrazovacího programu
	dir[(in<<4) + 12] = size  >> 8;			// velikost programu
	dir[(in<<4) + 13] = size;
	for (; free<256; free++) if (fat[free] == 0) break;
	dir[(in<<4) + 14] = free;			// první segment programu
	for (int i=err=0; i<15; i++) err += dir[(in<<4)+err];
	dir[(in<<4) + 15] = err;			// kontrolní součet jména
	tpdata.adr = 0x0200 + (in << 4);
	tpdata.size = 16;
	tpdata.data = dir[in<<4];
	if ((err = ReceiveData( MSG_WRITE_EEPROM, tpcom, &tpdata )) != 0) return err;
		// zápis programu
	int size_write = 254;
	tpdata.size = 1;
	for  (; ; ) {
		seg = free
		if (size < 254) size_write = size;			// menší jak segment
		if ((err = WriteSectorCRC( seg, data[item], size - item )) return err;  // zápis segmentu
		item += 254;
		size -= 254
		if (size =< 0) break;
		for (; free<256; free++) if (fat[free] == 0) break;
			// zápis do fat
		fat[seg] = free;
		tpdata.adr = 0x0100 + seg;
```

```
                tpdata.data = &fat[seg];
                if ((err = TransmitData( MSG_WRITE_EEPROM, &tpcom, &tpdata )) != 0) return
err;
        }
                // poslední segment programu ve fat označen 1
        fat[seg] = 1;
        tpdata.adr = 0x0100 + seg;
        tpdata.data = &fat[seg];
        return TransmitData( MSG_WRITE_EEPROM, &tpcom, &tpdata );
}
```

```
// FUNKCE zápis segmentu (256 bajtů) do paměti panelu
// segment = horních 8 bitů adresy
BYTE WriteSectorCRC( int sector, BYTE *data, int size )
{
        BYTE wr_data[256];
        memset( &wr_data[0], 0, 256 );
        memcpy( &wr_data[0], data, size );
                // kontrolní součet segmentu
        int crc = 0;
        for (int i=0; i<254; i++) crc+= wr_data[i]
        wr_data[254] = crc>>8;
        wr_data[255] = crc;
                // zápis sektoru
        TPDATA tpdata;
        tpdata.size = 32;
        for (int adr=0; adr<255; adr+=32) {
                tpdata.adr = (sector<<8) | adr;
                tpdata.data = &data[adr];
                if ((err = TransmitData( MSG_WRITE_EEPROM, &tpcom, &tpdata )) != 0) return
err;
        }
}
```

Typy proměnných

Typ	Bytů	Rozsah
char	1	–128 to 127
BYTE (unsigned char)	1	0 to 255
bool	1	0 - 1
int	4	–2,147,483,648 to 2,147,483,647
DWORD (unsigned int)	4	0 to 4,294,967,295
LPCTSTR	4	Ukazatel na datové pole charakterů



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