UNIVERZITA PARDUBICE
ÚSTAV ELEKTROTECHNIKY
A INFORMATIKY

UŽIVATELSKÁ PŘÍRUČKA TŘÍDY CSerialPort

2007

Pavlík Pavel

Obsah

Úvod

Třída `CSerialPort`, jejíž použití je popisované v této příručce, umožňuje komunikaci po sériovém portu, respektive po rozhraní RS-232, v prostředí win32 (Microsoft Windows 98/Me/Nt/2000/Xp). Třída je kvůli kompatibilitě založena pouze na win32 API a standardních funkcích z knihoven jazyka C/C++. Třída má implementována zvláštní vlákna pro čtení i zápis, aby tyto operace nezdržovaly hlavní vlákno aplikace. Ve třídě jsou zahrnuty tyto soubory:

- rs232.h – definice vlastní třídy `CSerialPort`
- rs232.cpp – implementace metod třídy `CSerialPort`
- CSerialBuffer.h – definice třídy `CSerialPort` použité jako buffer pro odesílání dat a úložiště příchozí dat (při konzolové aplikaci)
- CSerialBuffer.cpp – implementace metod třídy `CSerialPort`
- CSerialException.h – definice třídy výjimky používané ve třídě
- CserialDefines.h – obsahuje definy použité ve třídě `CSerialPort` (zprávy a nastavení systémových bufferů)
- debug.h – definice makra TRACE (alternativa MFC TRACE pro API) pro případné ladění
- debug.cpp – implementace TRACE
- main.cpp – obsahuje ukázkovou aplikace

Použití třídy by se dalo shrnout do těchto kroků:

- případné zjištění dostupných portů – `ScanPort()`
- otevření portu, případné nastavení logování – `OpenCom(), StartLogInput(), StartLogOutput()`
- případná změna nastavení – `GetPortSettings(), UpdatePortSettings()`
- posílaní a příjem dat – funkce `Send()`, příjem dat je u GUI aplikace řešen zprávou a u konzolové funkcemi `ReceivedBytes(), GetReceivedBytes()`
- uzavření portu – `CloseCom()`

Nastavení lze samozřejmě měnit i po otevření portu, logování lze ukončit metodami `StopLogInput() a StopLogOutput()`. Je také možné otestovat, jestli je port připojen metodou `IsConnected()`. Více detailů je možné nalézt v dalších částech příručky.

Skenování portů

V případě, že je nutné zjistit, jaké porty COM jsou v PC k dispozici, respektive k jakým portům se lze připojit, je možné využít metodu `ScanPort()`.

```
ULONGLONG ScanPort()const throw(serialException);
```

Tato metoda zkusí otevřít porty `COM1` až `COM8` a výsledek uloží do 64-bitového čísla, které následně vrátí. V prvním bajtu (nejméně významný bajt) čísla je uloženo, jestli je možné se připojit ke `COM1`, ve druhém ke `COM2`, ve třetím ke `COM3` atd. Jestliže je možné se připojit, pak je nastavena hodnota 255 (0xFF), v opačném případě je nastavena hodnota 0 (0x00). Takže např., je-li možné se připojit k portu `COM1` a `COM3` metoda vrátí 0xFF00FF. Metoda nesmí být volána, je-li již port připojen. V tomto případě vrátí výjimku `CSerialException`. Následuje ukázkový kód.

```
CSerialPort SP;
unsigned long long ullPorts;
unsigned char abyPorts[8]={0};

try{
      ullPorts= SP.ScanPort();
}catch (CSerialException& SE)
{
      cout << SE.GetErrorCode() << ": " << SE.GetErrorText() << endl;
      system("Pause");
      return 0;
}
cout << hex << ullPorts << endl;
memcpy(abyPorts,&ullPorts,8);
for(int i=0; i<8; i++)
      if (abyPorts[i]==255)
            cout << "COM" << i+1 << " je mozne se pripojit." << endl;
else cout << "COM" << i+1 << " neni mozne se pripojit." << endl;
```

Výstup:

```
ff00ff
COM1 je mozne se pripojit.
COM2 neni mozne se pripojit.
COM3 je mozne se pripojit.
COM4 neni mozne se pripojit.
COM5 neni mozne se pripojit.
COM6 neni mozne se pripojit.
COM7 neni mozne se pripojit.
COM8 neni mozne se pripojit.
Pokračujte stisknutím libovolné klávesy...
```

Test připojení

K ověření, zda je port připojen, slouží metoda `IsConnected()`.

```
BOOL IsConnected()const;
```

Otevření a zavření portu

O otevření portu se stará metoda `OpenCom()`. V této metodě může nastat výjimka.

```
void OpenCom(const HWND Parent=NULL,
const unsigned int ComNumber=1,
const unsigned int BaudRate=CBR_9600,
const unsigned char ByteSize=8,
const unsigned char Parity=NOPARITY,
const unsigned char StopBits=ONESTOPBIT
)throw (CserialException);
```

Jak je vidět z prototypu metody, může být volána i bez parametrů. V tom případě se metoda pokusí otevřít port `COM1` se standardním nastavením. Následuje popis jednotlivých parametrů.

Parent – handel na rodiče (zadejte `NULL` v případě použití v konzoli)

ComNumber – číslo portu, ke kterému se bude připojovat

BaudRate – rychlost komunikace může mít následující hodnoty

- `CBR_110`
- `CBR_300`
- `CBR_600`
- `CBR_1200`
- `CBR_2400`
- `CBR_4800`
- `CBR_9600`
- `CBR_14400`
- `CBR_19200`
- `CBR_38400`
- `CBR_56000`
- `CBR_57600`
- `CBR_115200`
- `CBR_128000`
- `CBR_256000`

ByteSize – počet datových bitů

Parity – nastavení parity, která může mít následující hodnoty

- `EVENPARITY` - sudá parita (počet 1 bitů + paritní bit = sudé číslo)
- `MARKPARITY` – paritní bit je vždy 1
- `NOPARITY` – bez parity
- `ODDPARITY` – lichá parita (počet 1 bitů + paritní bit = liché číslo)
- `SPACEPARITY` – paritní bit je vždy 0

StopBits – počet stop bitů, může mít následující hodnoty

```
ONESTOPBIT
ONE5STOPBITS
TWOSTOPBITS
```

Pro ukončení komunikace neboli k uzavření portu slouží metoda `CloseCom()`. Může u ní nastat výjimka. Není-li komunikace ukončena před koncem programu, je ukončena destruktorem třídy.

```
void CloseCom()throw (CSerialException);
```

Ukázkový kód otevření portu:

```
CSerialPort SP;

try{
	SP.OpenCom();
}
catch (CSerialException& SerialException)
{
	cout << SerialException.GetErrorCode() << ": "
	     << SerialException.GetErrorText() << endl;
	system("Pause");
	return 0;
}
```

Posílání a příjem dat

Pro odesílání dat je ve třídě implementována přetížená metoda `Send()`. Je možné odesílat klasický ASCII C řetězec ukončený nulou, jeden bajt nebo pole bajtů. Aby byla data odeslána, musí být port připojen.

```
void Send(const char* lpzStr);
void Send(const unsigned char byNum);
void Send(const unsigned char* byNums,
          const unsigned int uiCount);
```

Ukázkový kód

```
CSerialPort SP;

try{
	SP.OpenCom();
}catch (CSerialException& SerialException)
{
	cout << SerialException.GetErrorCode() << ": "
	      << SerialException.GetErrorText() << endl;
	system("Pause");
	return 0;
}
std::string str("Posilam retezec tridou CSerialPort.");
SP.Send(str.c_str());
```

Jestli jsou přijata nějaká nová data, je rodiči poslána zpráva `WM_COMM_RXCHAR`, parametr zprávy `WPARAM` obsahuje ukazatel na přijatá data a parametr `LPARAM` obsahuje jejich délku neboli počet přijatých bajtů.

Ukázkový kód (vyjmuto z procedury okna), zobrazí `MessageBox()` s přijatými daty.

```
case WM_COMM_RXCHAR:
      INT count=static_cast<int>(lParam);
      CHAR *lpzBuffer=new CHAR [count+1];
      FillMemory(lpzBuffer, count+1, 0);
      memcpy(lpzBuffer, reinterpret_cast<char*>(wParam), count);
      MessageBox(hwndDlg,lpzBuffer,"New data",MB_ICONINFORMATION|MB_OK);
      delete [] lpzBuffer;
      break;
```

V případě, že není použita okenní aplikaci, a je tedy po otevření komunikace nastaven rodič na `NULL`, není možné odeslat zprávu s daty. V tomto případě se musí použít jiný přístup. Není-li možné odeslat zprávu s daty jsou data ve třídě ukládána. Pro zjištění počtu bajtů, které už čekají ve třídě na vyzvednutí slouží metoda `ReceivedBytes()`.

```
ULONG ReceivedBytes()const
```

Pro vyzvednutí určitého počtu přijatých bajtů slouží metoda `GetReceivedBytes()`.

```
ULONG GetReceivedBytes(const ULONG ulCount,unsigned char*lpBuf);
```

Následující kód ukazuje jakým způsobem zjistit, jestli jsou ve třídě uložena nějaká nová přijatá data a poté, jak je získat.

```
ULONG ulCount=SP.ReceivedBytes();
if (ulCount>0)
{
      lpzBuf=new char[ulCount+1];
      FillMemory(lpzBuf, ulCount+1, 0);
      SP.GetReceivedBytes(ulCount,
                          reinterpret_cast<unsigned_char*>(lpzBuf));
      cout << "Nova data: " << lpzBuf << endl;
      delete [] lpzBuf;
}else cout << "Zadna nova data." << endl;
```

Získaná data jsou ze třídy smazána. Tento přístup lze použít pouze u konzolové aplikace, v případě, že lze odeslat zprávu s daty, nejsou přijatá data ukládána.

Další možností, jak získávat data nejen při konzolovém využití třídy, je logování.

```
void StartLogInput(const char* lpzFileName, bool bBinaryFile=false)
                   throw (CserialException);

void StopLogInput();
```

Od okamžiku, kdy je zavolána metoda `StartLogInput()`, do okamžiku zavolání `StopLogInput()` nebo ukončení komunikace, jsou veškerá přijatá data logována do zadaného souboru. Data jsou čitelná až po ukončení logování. Jestli bude soubor binární nebo textový je možné ovlivnit parametrem `bBinaryFile`. Metoda může vrátit výjimku třídy `CSerialException`. V následujícím ukázkovém kódu jsou veškerá přijatá data logována do textového souboru „Prijato.txt“.

```
CSerialPort SP;
...//nejaky kod
try{
      SP.StartLogInput ("Prijato.txt");
}catch (CSerialException& SerialException)
{
      cout << SerialException.GetErrorCode() << ": "
            <<SerialException.GetErrorText() << endl;
      system("Pause");
      return 0;
}
...//nejaky kod
```

Ekvivalentem jsou metody pro logování odeslaných dat.

```
void StartLogOutput(const char* lpzFileName, bool bBinaryFile=false)
                    throw (CserialException);

void StopLogOutput();
```

Získání a změna aktuálního nastavení portu

V případě, že je nutné zjistit aktuální nastavení portu, je ve třídě implementována metoda `GetPortSettings()`, která vrací kopii struktury `DCB` s aktuálním nastavením portu. Pro změnu nastavení portu je implementována metoda `UpdatePortSettings()`, která jako parametr přijímá `DCB` strukturu, se kterou se pokusí aktuální nastavení portu změnit. Pro obsáhlost `DCB` struktury je vhodné obě metody zkombinovat. Tedy nejprve nastavení získat, poté upravit a na konec nastavení aktualizovat. Pro použití obou metod musí být port připojen, a může v nich nastat výjimka `CSerialException`. Ovšem při změnách konfigurace portu, je třeba pamatovat na to, že nastavení na obou stranách komunikace musí být shodné. Vzhledem k obsáhlosti `DCB` struktury zde její obsah není uveden. Přehlednou dokumentaci lze nalézt v MSDN Library.

```
void UpdatePortSettings(const DCB& dcb) throw(CSerialException);
DCB GetPortSettings()const throw(CSerialException);
```

Ukázkový kód:

```
CSerialPort SP;
DCB myDcb;
try{
      SP.OpenCom();
      //zmena a uprava nastaveni
      myDcb=SP.GetPortSettings();
      myDcb.BaudRate=RTS_CONTROL_TOGGLE;
      SP.UpdatePortSettings(myDcb);
}
catch (CSerialException& SerialException)
{
      cout << SerialException.GetErrorCode() << ": "
            << SerialException.GetErrorText() << endl;
      system("Pause");
      return 0;
}
```

Zprávy zasílané rodiči a nastavení bufferu

V souboru `CserialDefines.h`, který je také zahrnut ve třídě `CserialPort` se nacházejí dvě skupiny maker. První se týká nastavení bufferů a obsahuje tyto makra:

`MAX_WRITE_BUFFER` – nastavení doporučené velikosti výstupního bufferu portu v bajtech, standardně je nastaveno 1024 bajtů

`MAX_READ_BUFFER` – nastavení doporučené velikosti vstupního bufferu portu v bajtech, standardně je nastaveno na 2048 bajtů

Obě tyto hodnoty by měly být voleny podle použitého zařízení, množství najednou posílaných dat a nastavené rychlosti komunikace. Ovšem tyto hodnoty jsou doporučené, OS Windows je rozhodně příliš striktně nedodržuje, a v případě potřeby velikosti bufferů změní. Při testech bylo zjištěno, že při nastavené extrémně malé velikosti bufferu pro příjem (`MAX_READ_BUFFER=2`), rychlosti `BaudRate=9600` a přijímaném souboru o velikosti zhruba 67 KB data byla přijata bez problémů a za stejnou dobu, jako při nastavení `MAX_READ_BUFFER=2048`. Ve většině případů je tedy možné ponechat standardní nastavení. Hodnoty velikosti bufferů musí být sudá čísla.

`MAX_WRITE_BLOCK` – nastavení, kolik bajtů maximálně se najednou zapíše do výstupního bufferu, respektive se najednou pošle, standardně je nastaveno 256 bajtů

MAX_READ_BLOCK – nastavení, kolik bajtů maximálně se najednou zapíše do vstupního bufferu, respektive se najednou načte, standardně je nastaveno 256 bajtů

Obě tyto hodnoty by měly být logicky menší než doporučené velikosti bufferů a nejlépe několikrát. Nastavení způsobí, např. při standardně nastaveném `MAX_WRITE_BLOCK` a teoretickém nahromadění 1024 bajtů dat k odeslání, že by data byla odeslána po čtyřech částech. Tyto hodnoty tedy zabraňují případnému posílání či přijímání dat po zbytečně velkých částech. Ovšem příliš nízké hodnoty těchto údajů negativně působí na rychlost komunikace. Doporučuji hodnoty vyšší než 32 bajtů, naopak nastavení vyšší než 512 bajtů bude poměrně bez efektu (512 B se v bufferech jen tak nenahromadí). I u těchto údajů je možné s klidným svědomím ponechat standardní nastavení.

Druhá skupina maker obsahuje zprávy, které jsou odesílané rodiči (ovšem není-li `NULL`), jako reakce na událost, která nastala na portu. Tato část je pro vás podstatná jen v případě, že třídu používáte v okenní aplikaci a máte tedy možnost na poslané zprávy reagovat. Např. pošle-li se poslední bajt z výstupního bufferu portu, rodiči přijde zpráva `WM_COMM_TXEMPTY_DETECTED`. V případě, že by jste potřebovaly hodnoty těchto maker z jakéhokoli důvodu změnit, upozorňuji, že to musí být hodnoty vyšší než `WM_USER`. Nejpodstatnější zpráva `WM_COMM_RXCHAR`, která oznamuje příjem dat byla popsána v části příručky o posilání a příjmu dat, a zde již uvedena nebude. Pro všechny ostatní zprávy platí, že jejich `WPARAM` je nastaven na `0` a `LPARAM` obsahuje číslo portu, kde k události došlo (např. pro `COM1` to je 1), až na výjimku u zprávy `WM_COMM_ERR_DETECTED`, kde `WPARAM` obsahuje chybu, která nastala.

WM_COMM_BREAK_DETECTED – oznamuje, že bylo na vstupu detekováno přerušení

WM_COMM_CTS_DETECTED – oznamuje, že `CTS(clear-to-send)` signál se změnil

WM_COMM_DSR_DETECTED – oznamuje, že `DSR(data-set-ready)` signál se změnil

WM_COMM_ERR_DETECTED – oznamuje, že nastala line-status chyba, může to být `CE_FRAME`, `CE_OVERRUN`, nebo `CE_RXPARITY`, pravděpodobně je to (ve stejném pořadí), důsledek špatného nastavení baudrate, ztrátou znaku či znaků, nebo chybou parity

WM_COMM_RING_DETECTED – oznamuje, že byl detekován ring indikátor

WM_COMM_RLSD_DETECTED – oznamuje, že `RLSD(receive-line-signal-detect)` signál se změnil

WM_COMM_RXFLAG_DETECTED – oznamuje, že událost znaku byla přijata, vyvolá se po nastavení atributu struktury `DCB EvtChar`

WM_COMM_TXEMPTY_DETECTED – oznamuje, že byl z výstupního bufferu odeslán poslední znak a nyní je buffer prázdný.

Při běžném využití třídy v nějaké jednodušší aplikaci není nutné na tyto zprávy nějakým způsobem reagovat.

Návratové hodnoty funkcí vláken

Jak bylo zmíněno v úvodu, třída má implementována zvláštní vlákna pro čtení i zápis, aby tyto operace nezdržovaly hlavní vlákno aplikace. Tyto vlákna mohou mít různé návratové hodnoty:

Vlákno pro odesílání:

10 – vlákno skončilo po normálním průběhu

5 – vlákno skončilo po chybě v odesílání dat

Vlákno pro příjem:

20 – vlákno skončilo po normálním průběhu

17 – vlákno skončilo po chybě ve zpracování události na portu, pravděpodobně v důsledku chyby v příjmu dat

15 – vlákno skončilo po vážné chybě při čekání na událost na portu

Ukázka jednoduché konzolové aplikace

Tato jednoduchá aplikace se pokusí otevřít port se standardním nastavením a začne logovat přijatá i odeslaná data. V zakomentované části je ukázána případná změna nastavení již otevřeného portu. Případné výjimky při inicializaci i ukončení aplikace jsou odchyceny. V menu aplikace je na výběr možnost poslání řetězce (s možností zadat i mezery) a možnost zjistit, jestli nebyla přijata nějaká nová data. Po zadání jiné volby než jsou znaky 0 až 9 je aplikace ukončena. Následuje zdrojový kód.

```
#include <iostream>
#include "rs232.h"

using namespace std;

int main(int argc,  char* argv[])
{
	CSerialPort SP;
	DCB myDcb;
	try
	{
		SP.OpenCom();//otevření portu
		//začátek logování
		SP.StartLogInput("prijato.txt");
		SP.StartLogOutput("poslano.txt");
		//případná změna nastaveni
		//např. nastaveni rizeni toku TRS on TX
		/*myDcb=SP.GetPortSettings();
		myDcb.BaudRate=RTS_CONTROL_TOGGLE;
		SP.UpdatePortSettings(myDcb);*/
	}
	catch (CSerialException& SE)
	{
		cout << SE.GetErrorCode() << ": "
			<< SE.GetErrorText() <<endl;
		system("Pause");
		return 0;
	}
	//jednoduchý program
	if (SP.IsConnected())
	{
		bool bPokracuj=true;
		while(bPokracuj)
		{
			cout << "******** MENU ********" << endl;
			cout << "1- posli retezec" << endl;
			cout << "2- zjisti nova data" << endl;
			cout << "jine znaky nez 0..9 konec \n\n"
			cout << "Zadejte volbu : ";
			int volba;
			cin >> volba;
			if (cin.fail())
			{
				cin.clear();
				bPokracuj=false;
			}else
			{
				char *lpzBuf=0;
				ULONG ulCount;
				switch (volba)
				{
					case 1: lpzBuf=new char[100];
					FillMemory(lpzBuf, 100, 0);
					cout<<endl<<"Zadejte text: ";
					cin.get();
					cin.getline(lpzBuf,99);
					SP.Send(lpzBuf);
					cout << "Odeslano... "<<endl;
```

```
                        delete [] lpzBuf;
                        break;

                        case 2:
                        ulCount=SP.SizeOfReceived();
                        if (ulCount>0)
                        {
                         lpzBuf=new char[ulCount+1];
                         FillMemory(lpzBuf,
                                    ulCount+1, 0);
                         SP.GetReceived(ulCount,
                                    reinterpret_cast
                                    <unsigned char*>
                                    (lpzBuf));
                         cout << endl <<"Nova data: "
                              << lpzBuf << endl;
                         delete [] lpzBuf;
                        }else
                          cout << "Zadna nova data."
                               << endl;
                         break;
                  }
                  system("Pause");
                  system("CLS");
            }
      }
      try
      {
            //uzavření portu,
            //automaticky se ukončí i logování
            SP.CloseCom();
      }
      catch (CSerialException& SE)
      {
            cout << SE.GetErrorCode() << ": "
                  << SE.GetErrorText() << endl;
      }
}

system("Pause");
return 0;
}
```