ComPort library help file

Introduction

About ComPort Library version 2.63

ComPort Library has been in development more than two years. There has been a lot of work put in this stuff. So i simply need some kind of motivation for further develpoment of ComPort Library. I would be very happy if you could take the time and send me a postcard from your country to my home address. I hope that is not too much to ask from you.

ComPort Library help file version
2.63.1

Author information
Name: Dejan Crnila
Year of birth: 1978

Occupation: Student of computer and information science in University of Ljubljana
E-mail: dejancrn@yahoo.com
Home page: http://www2.arnes.si/~sopecrni
Home address: Dolenja vas 111, 3312 Prebold, SLOVENIA

Special thanks to Paul Doland (E-mail: pdoland@flash.net), who provided C++ Builder support for ComPort Library.

Key features

- Platforms: Windows NT 4.0, Windows 2000, Windows 95, Windows 98
- Languages: Delphi 3, 4, 5, 6 and C++ Builder 3, 4, 5
- Components: TComPort, TComDataPacket, TComComboBox, TComRadioGroup, TComLed, TComTerminal
- Asynchronous or synchronous Read/Write operations
- Detailed flow control settings
- Read/Write operation timeouts
- Use of multithreading for monitoring port events
- Build terminal application without a line of code

- Source code included (cca 7000 lines)
- A Delphi context-sensitive help file
- and much more …

What's new in version 2.63

ComPort Libarary version 2.63 is not compatible with some older versions. Some methods and properties have been changed, so if you have been using older versions, you will have to change a code of your program a bit.

New in version 2.63

- Delphi 6 support
- TComStream class
- TComPort.EventThreadPriority property

- EnumComPorts non-admin (WinNT/2000) bug fixed
- Fixed hang up bug on port close
- Other minor fixes
- Optimization

Programming with ComPort library

Using TComPort component

Enumerating ports

Before setting serial port number, it is useful to call EnumComPorts procedure to enumerate serial ports on local machine. Application can then assign a member of TStrings result to Port property.

Example

```
begin
  EnumComPorts(ComboBox1.Items);
  // do some stuff
  if ComboBox1.ItemIndex > -1 then
    ComPort1.Port := ComboBox1.Items[ComboBox1.ItemIndex];
end;
```

Opening and closing port

Before most of the TCustomComPort methods can be succesfully called, serial port has to be opened. There are two ways to open serial port. Application can call Open method or set Connected property to True. To end a session, call Close method or set Connected property to False.

Example

```
begin
  ComPort1.Open; // open serial port
  // do some stuff here
  ComPort1.Close; // end session
end;
```

Asynchronous and synchronous operations

Read and write operations on serial port can be performed in two modes, asynchronous or synchronous. In synchronous mode, method that peforms operation on port does not return until operation is completed (or aborted). In asynchronous mode, method returns immidiately and does not wait for the operation to be completed. After calling asynchronous operation, the result of the method might not yet be defined, so the application should call wait method to make sure the operation is completed. Between asynchronous call of operation and wait method, the application can perform any other tasks that are not dependend on asynchronous operation result.

Each asynchronous operation has to be prepared before it is performed. To prepare asynchronous operation, call InitAsync method. This method inits PAsync type paramater. Each method that deals with asynchronous operations has PAsync parameter. Use the same parameter as in InitAsnyc method, since the TCustomComPort has to know which operation it is refering to. After the operation is completed, call DoneAsync method to free the resources.

Example (asynchronous operation)

```
var
  Operation1: PAsync;

begin
  InitAsync(Operation1);
  try
    ComPort1.WriteStrAsync('Hello', Operation1);
    // do some stuff here
    ComPort1.WaitForAsync(Operation1);
  finally
    DoneAsync(Operation1);
  end;
end;
```

Writing to port

Write operations can be performed very easily when using TCustomComPort. There are four methods that deal with writing data.

Method	Description
Write	Writes non-typed variable to output buffer.
WriteAsync	Writes non-typed variable to output buffer in asynchronous mode.
WriteStr	Writes string type variable to output buffer.
WriteStrAsync	Writes string type variable to output buffer in asnychronous mode.

Application should also properly set write timeouts. See TComTimeouts class for more detailed description.

Example

```
var
  Str: String;

begin
  Str := 'Hello';
  ComPort1.WriteStr(Str); // string type variable
  ComPort1.Write(Str[1], Length(Str)); // no defined type

end;
```

Reading from port

Reading from input buffer can be performed in two ways. Usually application calls one of the read methods inside OnRxChar event, which triggers when charachter(s) arrive in input buffer. If read method is called inside OnRxChar event, read timeouts should be set to no wait, that is, read method checks input buffer and returns immidiately, since the number of bytes in input buffer is

already known. Application can also call read method outside OnRxChar, but it should set read timeouts properly. See TComTimeouts for more details.

If component is linked to other component that needs incoming data, like TComDataPacket or TCustomComTerminal, OnRxChar event is is not called, however, the component calls OnRxBuf event. The application can not read the data from input buffer inside OnRxBuf event, since it has already been read. The data is placed automatically by the component in Buffer parameter of OnRxBuf event. Whether OnRxChar or OnRxBuf event is called, can be checked with TriggersOnRxChar property.

Method	Description
Read	Reads from input buffer to non-typed variable.
ReadAsync	Reads from input buffer to non-typed variable in asynchronous mode.
ReadStr	Reads from input buffer to string type variable.
ReadStrAsync	Reads from input buffer to string type variable in asnychronous mode.

Example (inside OnRxChar)

```
procedure TForm1.ComPort1RxChar(Sender: TObject; Count: Integer);
var
  Str: String;
begin
  ComPort1.ReadStr(Str, Count);
  // do something with Str variable
end;
```

Example (outside OnRxChar)

```
var

  Str: String;

begin
  // set timeouts here or at design time
  ComPort1.ReadStr(Str, NumberOfBytes);
  // do something with Str variable
end;
```

Example (inside OnRxBuf)

```
procedure TForm1.ComPort1RxBuf(Sender: TObject; const Buffer; Count: Integer);
begin
  // application does not have to read data from input buffer
  // data is already in buffer parameter
  HandleData(Buffer, Count); // handle data

end;
```

Aborting asynchronous operation

Asynchronous operations can easily be aborted. Unfortunately, a specific operation cannot be aborted, however, all operations in progress can be aborted simultaneously. If operation is aborted, WaitForAsync method raises EComPort exception with WinCode property set to ERROR_OPERATION_ABORTED.

Example

```
var
  Operation1: PAsync;

begin
  InitAsync(Operation1);
  try
    ComPort1.WriteStrAsync('Hello', Operation1);

    // some stuff
    if { some condition } then
      ComPort1.AbortAllAsync;
    // some stuff
    ComPort1.WaitForAsync(Operation1);
  finally
    DoneAsync(Operation1);
  end;
end;
```

Changing properties at runtime

All TCustomComPort properties except SyncMethod can be changed at runtime while application is connected to serial port. Changes are applied immidiately. If Port property is changed while connected, serial port is closed and reopened. If properties are changed between BeginUpdate and EndUpdate methods, changes are not applied until EndUpdate method is called. If you have to change more than one property at once, wrap them inside BeginUpdate, EndUpdate methods.

Example

```
begin
  ComPort1.Open;
  // do some stuff
  ComPort1.FlowControl.ControlDtr := dtrEnable; // changes are applied
  // do some stuff

  BeginUpdate; // prevent changes from being applied
  ComPort1.Parity.Bits := prOdd;
  ComPort1.FlowControl.XonXoffIn := True;
  EndUpdate; // apply changes
  // do some stuff
  ComPort1.Close;
end;
```

Storing and loading settings

Application can easily store and load serial port settings using StoreSettings and LoadSettings methods. Settings can be stored into configuration file or registry. StoredProps property determines which properties need to be stored.

Example (Registry)

```
begin
  // store settings to registry
  ComPort1.StoreSettings(stRegistry, '\HKEY_LOCAL_MACHINE\Software\ComPortTest');
  // load settings
 ComPort1.LoadSettings(stRegistry, '\HKEY_LOCAL_MACHINE\Software\ComPortTest');

end;
```

Example (Configuration file)

```
begin
  // store settings to configuration file
  ComPort1.StoreSettings(stIniFile, 'c:\ComPortTest.ini');
  // load settings
  ComPort1.LoadSettings(stIniFile, 'c:\ComPortTest.ini');
end;
```

Using buffer functions

Application can clear input and/or ouput buffer using ClearBuffer method. Make sure that there is no asynchronous operation in progress while calling ClearBuffer, because it can cause unexpected problems.

Example

```
begin
  // do some stuff
  if { some condition} then
    ComPort1.ClearBuffer(True, False);  // clear input buffer
  // do some stuff
end;
```

Detecting signals and port state

Change of signals (CTS, DSR, RLSD) can be detected using OnXChange events. Ring indicator can also be detected using OnRing event. Application can check the state of signals anytime with Signals method.

By calling StateFlags method, application can determine whether the transmission is in progress or in wait state due to some reason.

Example

```
procedure TForm1.ComPort1CTSChange(Sender: TObject; OnOff: Boolean);

begin
```

```
  if OnOff then
    PrintMessage('CTS high')
  else
    if fCtlHold in ComPort1.StateFlags then  // if transmission is waiting
      PrintMessage('CTS low, transmission waiting')
    else
      PrintMessage('CTS low');
end;
```

Changing output flow control signals

Output flow control signals such as RTS and DTR can be handled manually, unless they are set to rtsHandshake and dtrHandshake. Use SetRTS and/or SetDTR methods to set signals to low or high state. The same thing can be accomplished using FlowControl property during a session, but it is much faster if application uses SetX methods.

Example

```
begin
  ComPort1.SetRTS(False); // lower RTS signal
  // do some stuff
  ComPort1.SetRTS(True); // set RTS to high

end;
```

Retrieving error information

When error occurs on serial port, OnError event is triggered. Check Errors parameter of event to get information on error that occured on serial port. If application is not using OnError event, it can check last errors on port using LastErrors method. LastErrors returns a set of error flags which indicate error type and clears error buffer. Therefore, if application calls LastErrors again, it returns empty result set, unless a new error occured between two calls.

Application should not call LastErrors method inside OnError event, since it has already been called in inside TCustomComPort to set Errors parameter.

Example

```
procedure TForm1.ComPortError(Sender: TObject; Errors: TComErrors);
begin
  if ceRxParity in Errors then
    ShowMessage('Paity error occured');
end;
```

Waiting for events

When Events property is not empty, a special thread is created for monitoring port events when application calls Open method. This is possible only when you use TComPort component in an application (thread) which creates message loop. Most applications have message loop (GUI, NT services). However, if you want to use TComPort component in a console application, you have to set Events property to empty before calling Open method or your application will crash. To monitor events in console application, you have to use WaitForEvent method.

Example

```
var
  ComPort: TComPort;
  Events: TComEvents;

begin
  ComPort := TComPort.Create(nil);
  try
    ComPort.Events := [];  // do not create monitoring thread
    ComPort.Open;  // open port
    Events := [evCTS, evDSR, evRLSD];  // define events to wait for
    ComPort.WaitForEvents(Events, nil, WaitInfinite);  // wait until at least one event happens
    if evCTS in Events then

      WriteLn('CTS Changed');  // CTS signal changed state
    ComPort.Close;  // close port
  finally
    ComPort.Free;
  end;
end;
```

Reading data in packets

About packets

When application is connected to some sort of serial device like data loggers which constantly send data to PC, data is usually sent in packets. Packet is a string of charachters, usually of constant size, with some start and/or stop charachters. Application can parse incoming data inside OnRxChar event handler, but its much easier to use TComDataPacket component, which does parsing process automatically.

One TCustomComPort can have more than one TComDataPacket components linked, so application can easily recieve packets of different sizes and forms.

When packet is formed OnPacket event is triggered with packet string as parameter. Any data that is discarded during the process of packet forming goes through OnDiscard event.

If TComDataPacket component is linked to TCustomComPort component, OnRxChar event of TCustomComPort is not triggered, so application has to use OnRxBuf event handler if it also needs unparsed incoming data.

Start and stop conditions

Application sets up packet by setting start and stop conditions. This is accomplished by setting StartString, StopString and Size properties of TComDataPacket component.

If StartString property is not empty, packet starts when StartString string has arrived in input buffer. If StartString is empty, packet starts immidiately when any charachter arrives in input buffer.

Stop condition is defined by StopString and Size property. If Size is 0, size stop condition is ignored. If Size is larger than 0, packet is ended when packet size reaches Size charachters. If StopString is not empty, packet stops when StopString string arrives in input buffer. If Size is larger than 0 and StopString is not empty, packet is ended when size of packet reaches Size charachters or

StopString string arrives in input buffer. If Size is 0 and StopString is 0 then packet is not defined and OnPacket has the same effect as OnRxBuf event handler.

Example

Some PBX device sends data to PC in packets, starting with STX (start-of-text) charachter and ending with ETX (end-of-text) charachter. Before opening serial port, application should set up start and stop condition like this:

```
begin
  // set start and stop condition for packet
  ComDataPacket1.StartString := #2;
  ComDataPacket1.StopString := #3;
  // now open the port
  ComPort1.Open;
end;
```

Note

ComDataPacket1 has to be linked to ComPort1 component.

Custom packets

Some packets have specific form and sometimes application can't use standard start and stop conditions. To form a custom packet, application has write OnCustomStart and/or OnCustomStop event handler of TComDataPacket component. If OnCustomStart event handler is assigned, StartString start condition is ignored and it is defined inside event handler. Simillary, if OnCustomStop event handler is assigned, StopString and Size stop conditions are ignored and defined in OnCustomStop event handler.

Example

Data from serial device is sent to PC in the following format: #packet_size#packet_data
where packet_size is size of packet including header and packet_data is raw data.

```
var
  CurrentPacketSize: Integer;

procedure ComDataPacket1CustomStart(Sender: TObject; const Str: string; var Pos: Integer);
var
  P1, P2: Integer;
begin
  Pos := 0; // no start position yet
  P1 := System.Pos('#', Str);

  if P1 = 0 then
    Exit; // no start charachter found
  P2 := System.Pos('#', Copy(Str, P1 + 1, Length(Str) – P1)); // second # charachter
  if P2 = 0 then
    Exit;  // no second start charachter found
  P2 := P2 + P1;
  Pos := P1; // start position of packet defined at first # charachter
  try
```

```
    // determine packet size
   CurrentPacketSize := StrToInt(Copy(Str, P1 + 1, P2 – P1 - 1));

  except
   CurrentPacketSize := P2 – P1 + 1; // packet size is corrupted, cut only #…# part
  end;
end;

procedure ComDataPacket1CustomStop(Sender: TObject; const Str: string; var Pos: Integer);
begin
  if CurrentPacketSize <= Length(Str) then
    Pos := CurrentPacketSize; // set stop position
end;
```

ComPort library reference

TComPort is a serial communication component.

Unit
CPort

Description
Use TComPort component to easily communicate with external devices on RS232 connection, such as modems, bar code readers, PBX and so on. TCustomComPort introduces several properties for detailed setting of serial port, numerous methods to write and read from port and events for monitoring port. Write and read operations can be performed synchronously or asynchronously.

TComTerminal is a VT52/VT100/ANSI terminal control.

Unit
CPortCtl

Description
TComTerminal control recieves data from TCustomComPort component and puts it on the screen. The control supports connection to VT52, VT100 and ANSI terminal servers. Application can also provide custom escape code parsing in OnGetEscapeCodes event.

TComData packet performs read operation in packets.

Unit
CPort

Description
Use TComDataPacket component to easily read data from input buffer in packets. The application can set stop and start condition for packet by setting TComDataPacket properties. One TCustomComPort component can have more than one TComDataPacket component linked.

TComComboBox is a windows combo box for selecting serial settings.

Unit
CPortCtl

Description

Drop TComComboBox control on the form to easily select common serial settings. TComComboBox can be linked to TCustomComPort descendand to to apply the desired settings to serial port.

TComRadioGroup is a group of radio buttons for selecting serial settings.

Unit
CPortCtl

Description
Drop TComRadioGroup control on the form to easily select common serial settings. TComRadioGroup can be linked to TCustomComPort descendand to to apply the desired settings to serial port.

TComLed is a led indicator for serial signals.

Unit
CPortCtl

Description
Use TComLed control to easily show the state of CTS, DSR, RLSD, Ring, Tx and Rx signals. Application can provide custom bitmaps to show the state of signals. TComLed can be linked to TCustomComPort component to automatically update the control when signal(s) change.

Stream class for com port data

Unit
CPort

Description
Use TComStream class when you want send and recieve data as a stream.

EComPort is the exception class for failures on serial port.

Unit
CPort

Description
EComPort exception is raised on error that occurs inside TCustomComPort component. Check EComPort properties for additional information about exception.

Procedures

InitAsync

Inits PAsync variable.

procedure InitAsync(var AsyncPtr: PAsync);

Description
Call InitAsync procedure init PAsync type variable. PAsync type is used to identify operation in all methods that deal with asynchronous operations. After the operation is completed, application is responsible for freeing PAsync type variable with DoneAsync procedure.

DoneAsync

Frees PAsync variable.

procedure DoneAsync(var AsyncPtr: PAsync);

Description
Call DoneAsync procedure to free PAsync type variable after asynchronous operation is completed.

EnumComPorts

Enumerates serial ports on local computer.

procedure EnumComPorts(Ports: TStrings);

Description
Call EnumComPorts to fill Ports paramater with installed com ports on local computer. Ports parameter is a TString descendant created and maintained by the application. Application can assign Ports member to Port property to set port number.

Conversion functions

Use these functions to easily convert one type to other.

procedure BaudRateToStr(BaudRate: TBaudRate): String;
procedure DataBitsToStr(DataBits: TDataBits): String;
procedure StopBitsToStr(StopBits: TStopBits): String;
procedure ParityToStr(Parity: TParityBits): String;
procedure FlowControlToStr(FlowControl: TFlowControl): String;

procedure StrToBaudRate(Str: String): TBaudRate;

procedure StrToDataBits(Str: String): TDataBits;
procedure StrToStopBits(Str: String): TStopBits;
procedure StrToParity(Str: String): TParityBits;
procedure StrToFlowControl(Str: String): TFlowControl;

ComTerminalFont

Default terminal font.

var ComTerminalFont: TFont;

Description
ComTerminalFont is default font fot TCustomComTerminal component and its descendants.

List of error codes

Code	Constant	Meaning
1	CError_OpenFailed	Failed to open serial port.
2	CError_WriteFailed	Error occured while writing to port.
3	CError_ReadFailed	Error occured while reading from port.
4	CError_InvalidAsync	PAsync type parameter is invalid.
5	CError_PurgeFailed	PurgeComm API function failed.
6	CError_AsyncCheck	Unable to check asynchronous operation.
7	CError_SetStateFailed	SetCommState API function failed.
8	CError_TimeoutsFailed	SetCommTimeouts API func. failed.
9	CError_SetupComFailed	SetupComm API function failed.

10	CError_ClearComFailed	ClearCommError API func. failed.
11	CError_ModemStatFailed	GetCommModemStatus failed.
12	CError_EscapeComFailed	EscapeCommFunction failed.
13	CError_TransmitFailed	TransmitChar method failed.
14	CError_SyncMeth	SyncMethod cannot be changed while connected.
15	CError_EnumPortsFailed	EnumPorts function failed
16	CError_StoreFailed	Failed to store settings
17	CError_LoadFailed	Failed to load settings
18	CError_RegFailed	Link (un)registration failed
19	CError_LedStateFailed	Cannot change led state if ComPort is selected
20	CError_ThreadCreated	Cannot wait for event if event thread is created
21	CError_WaitFailed	WaitForEvent method failed