3.4 串口通信综合套件:ComPort

名称：ComPort Library version 2.61

适用：Delphi3、4、5

来源： www.Torry.net

版权： Dejan Crnila

联系： E-mail: dejancrn@yahoo.com

Home page: http://www2.arnes.si/——sopecrni

性质： 免费软件

评级： ★★★★

光盘： 该控件位于安装目录“套件\ ComPort”

3.4.1 控件评测

图 3-60 ComPort 控件包所含的控件

ComPort Library 是一组通信组件。它提供在 Windows 95/98 和 NT4.0 /5.0 下的端口通信功能。它包括：TComPort、TComData- Packet、TComComboBox、TComRadioGroup、TComLed、TComTerminal 等。利用它们可以实现对端口的异步或同步读/写操作，能对数据流进行详细的控制设置，还能监控端口事件的多线程应用，甚至能构造无代码行的终端应用程序，总之使你开发 Delphi 应用程序变的更简单。

此控件包是完全免费的，并且包括源代码和示例，的确不可多得。不过开发作者有一个小小的要求，如果你使用这些控件，请给他寄一张明信片以激励他进一步开发。

3.4.2 使用详解

安装方法

1．对 ComPort.zip 解压，得到三个压缩包，它们分别是源文件、示例文件和帮助文件的压缩包。

2．建立新的目录，对压缩包再分别解压。

3．运行 Delphi 5，在面板中打开菜单 File\Open，在对话框中运行安装目录下的 CportLib5.dpk，在弹出的窗口中点击“Install”（若你原先装有该控件包的旧版本，应首先卸载旧版本）。

4．在面板中打开菜单 Tools\Environment Options\Library，将安装目录添加到 Library Path 中去。

注意：2.60 版本的端口库与一些旧版本不兼容。其中一些方法和属性已改变，所以，如果你以前用的是旧版本，必须稍微改动一下你的程序代码。

控件简介

"CportLib"	
	**TComPort** TComPort 是一个串口通信组件。利用 TComPort 组件可以方便地与用 RS232 接口连接的外部设备进行通信，例如调制解调器，条形码读卡机，PBX（专用分组交换机）等等。TCustomComPort 类引入了一些详细设置串口的属性，以及许多为监控串口从串口读/写事件的方法。
	**TComTerminal** TComTenminal 控件从 TCustomComPort 接收数据并显示在屏幕上。此控件支持到 VT52, VT100 和 ANSI 终端服务器的连接。
	**TComDataPacket** TComDataPacket 执行信息包的读操作。利用 TComDataPacket 组件能方便地从信息包的输入缓冲区读数据。应用程序通过设置 TComDataPacket 的属性能为信息包设定起始和停止条件。
	**TComComboBox** TComComboBox 是一个选择串口设置的 windows 组合框。它能被链接到 TCustomComPort 派生类以便在串口上实现所需的串口设置。
	**TComRadioGroup** TComRadioGroup 是一个选择串口设置的单选按钮组。
	**TComLed** TComLed 是一个串口信号发光二极管指示器。利用 TComLed 控件能够方便地显示 CTS, DSR, RLSD, Ring, Tx 和 Rx 信号的状态。

3.4.2.1  TComPort

用途

TComPort 是一个串口通信组件。利用 TComPort 组件可以方便地与用 RS232 接口连接的外部设备进行通信，例如调制解调器，条形码读卡机，PBX（专用分组交换机）等等。TCustomComPort 类引入了一些详细设置串口的属性，以及许多为监控串口从串口读/写事件的方法。读/写操作可以是同步也可以是异步完成。

主要属性

BaudRate：TBaudRate——设置字符传输速率。BaudRate 属性代表的是字符通过 RS232 接口的收发传输速度，接口两侧必须有相同的 BaudRate。如果 BaudRate 的值是 brCustom，

CustomBaudRate 属性用来设置波特率。

Buffer：TComBuff——指定串口缓冲器的大小，其大小仅与串口驱动器有关，而串口驱动器可适用于任意大小。

Connected：Boolean——显示串口连接是否正在起作用。设置为 True 打开串口，设置 False 关闭串口连接。在程序中可检查 Connected 属性来判断串口是否打开。另外需要说明的是，设置 Connected 为 True 与调用 Open 方法等同，而设置 Connected 为 False 与调用 Close 方法等同。

DataBits：TDataBits——设置一字节的数据位数。利用此属性指定可传送和接收字节的位数。值得注意的是 5 比特数据位加 2 比特停止位是无效的组合，6，7 或 8 比特数据位加 1.5 比特停止位也是无效的组合。

DiscardNull：Boolean——设置空字节是否被抛弃，当设为 True，则空字节在接收时就被抛弃。

EventChar：Char——设定事件通知的 ASCII 值。当事件字符接收时，触发 OnRxFlag 和 OnRxChar 事件。

Events：TComEvents——为特定事件指定是否设置事件监控。如果至少有一个事件设置了事件监控，则产生一个专用线程监控串口事件。

FlowControl：TComFlowControl——设置流控制类型。其控制类型可以为硬件、软件或无属性。

Parity：TComParity——设置使能奇偶位检查和奇偶方案。

Port：TPort——设定串口数目。Port 值遵循 COMportnumber 或 portnumber 语法，如果在通话期间改变了 Port 值，则旧端口被关闭，打开一个新的端口。应用程序能够调用 EnumComPorts 过程统计当前计算机上的端口数目。

StopBits：TStopBits——设定每个字节的停止位。StopBits 的值可被设定为：

sbOneStopBit	每字节 1 位停止位
sbOne5StopBits	每字节 1.5 位停止位
sbTwoStopBits	每字节 2 位停止位

注意：5 比特数据位加 2 比特停止位是无效的组合，6，7 或 8 比特数据位加 1.5 比特停止位也是无效的组合。

StoredProps：TStoredProps——定义当调用 StoreSettings(或 LoadSettings)方法时是存储还是调用属性。StoredProps 可被设定为如下值的组合：

spBasic	端口数　波特率　数据位　停止位　流控制
spFlowControl	详细的流控制设置
spBuffer	输入输出缓冲器大小
spTimeouts	Timeout 的值
spParity	详细的奇偶位设置

spOthers　　EventChar 和 DiscardNull 属性

SyncMethod：TSyncMethod——SyncMethod 定义事件将调用哪个线程，为如下值之一：

smThreadSync　　事件在应用程序主线程里被调用

smWindowSync　　事件在 Open 方法里被调用

smNone　　事件在特定监视线程里被调用

注意：SyncMethod 一般设为 smThreadSync。Windows NT 操作环境下一般用 smWindowSync 来定义在哪个线程调用事件。

Timeouts：TComTimeouts——为读/写操作设定时间超时。

TriggersOnRxChar：Boolean——显示数据到达输入缓冲器时调用哪个事件。利用 TriggersOnRxChar 属性检查在运行期间数据到达输入缓冲器时调用哪个事件。如果 TriggersOnRxChar 为 True，调用 OnRxChar 事件；如果 TriggersOnRxChar 为 False，调用 OnRxBuf 事件。

主要方法

procedure AbortAllAsync

中断所有未决的异步操作。调用 AbortAllAsync 方法取消所有未决的异步操作。如果在调用 AbortAllAsync 方法之后调用了 WaitForAsync 方法，将产生 EcomPort 异常错误，并发出 ERROR_OPERATION_ABORTED 消息。

注意：应用程序依然需要负责释放所有调用 DoneAsync 方法的未决操作的异步指针。

procedure ApplyBuffer

将缓冲器大小送往串口驱动器。

procedure ApplyDCB

对串口进行设置。ApplyDCB 方法填写 DCB 记录并将它送往串口驱动器。

procedure ApplyTimeouts

串口时间超时设置。ApplyTimeouts 方法填充 COMMTIMEOUTS 记录并将它送往串口驱动器。

procedure BeginUpdate

防止串口设置被突然改掉。当改变了任意属性且 Connected 属性被设为 True 时，调用 BeginUpdate 方法来防止端口设置被应用。调用 BeginUpdate 后，直到调用了 EndUpdate 方法后，端口设置才能有效。这对于应用程序一次改变多个设置是很有用的。

procedure ClearBuffer(Input, Output: Boolean)

清除输入/输出缓冲器。调用 ClearBuffer 方法清除输入/输出缓冲器，如果 Input 参数设为 True，输入缓冲器被清除；如果 Output 参数设为 True，输出缓冲器被清除。

procedure Close

Close 方法关闭串口连接，如果串口已经关闭，Close 方法什么也不处理，应用程序不能进行读/写操作。

注意：调用 Close 方法与设置 Connected 属性为 False 是等同的。

constructor Create(AOwner: TComponent); override

产生一个 TCustomComPort 类的实例。

destructor Destroy; override

析构 TCustomComPort 对象。不要在应用程序中直接调用 Destroy 方法，而是调用 Free 方法。Free 证实 TCustomComPort 组件没有被释放时才调用 Destroy。这时如果端口没关闭，首先关闭端口。

procedure EndUpdate

应用端口设置。调用 EndUpdate 方法使调用 BeginUpdate 方法后新设置的端口属性有效。

function InputCount: Integer

返回输入缓冲器的字节数。

function IsAsyncCompleted(AsyncPtr: PAsync): Boolean

显示异步操作是否完成。调用 IsAsyncCompleted 得到异步操作状态，如果返回为 True，异步操作已经完成；如果返回为 False，异步操作还在执行。AsyncPtr 参数是被 InitAsync 方法初始化的变量。

function LastErrors: TComErrors

LastErrors 函数返回一组上次调用 LastErrors 后发生的错误。下列的各项错误都可能在串口发生：

ceFrame	帧同步错误
ceRxParity	奇偶校验错误
ceOverrun	字符缓冲器溢出，下一个字符被丢失
ceBreak	中断错误
ceIO	通信过程中的 I/O 错误
ceMode	请求方式不被支持
ceRxOver	输入缓冲器溢出
ceTxFull	输出缓冲器溢出

procedure LoadSettings(StoreType: TStoreType; LoadFrom: String)

从注册表或文件中加载 TCustomComPort 属性设置。LoadSettings 方法从注册表或文件中加载 TCustomComPort 属性设置，这些属性的加载依赖于 StoredProps 属性。StoredProps 属性值为下列之一：

stRegistry	LoadFrom 参数是注册表键值
stIniFile	LoadFrom 参数是文件名

procedure Open

打开串口连接。Open 方法打开串口连接且设置串口为 TCustomComPort 类定义的属性。如果串口已经打开，此方法不处理任何事情。

注意：串口以互斥方式打开，也就是说，一旦串口被某一应用程序打开，它再不能被同一应用程序或其他应用程序打开。

function OutputCount: Integer

返回输出缓冲器的字节数。

function Read(var Buffer; Count: Integer): Integer

从输入缓冲器读数据。调用 Read 函数读 Count 字节数据到 Buffer 变量。此函数直到 Count 字节数据被读完或时间片被用完才返回。返回值为实际读入数据的字节数。

注意：Buffer 变量的大小必须大于 Count 字节。

function ReadAsync(var Buffer; Count: Integer; var AsyncPtr: PAsync): Integer

以异步方式从输入缓冲器读数据。调用 ReadAsync 函数读 Count 字节数据到 Buffer 变量。函数立即返回，并不等待操作执行完，通过调用 WaitForAsync 函数来确定操作执行完。返回值是实际读入数据的字节数。AsyncPtr 参数是被 InitAsync 初始化的 Pasync 记录。操作执行完后应用程序负责调用 DoneAsync 方法释放 AsyncPtr 指针，返回值是实际读取的字节数。

注意：Buffer 变量的大小必须大于 Count 字节。

function ReadStr(var Str: String; Count: Integer): Integer

从输入缓冲器读入数据到 String 变量。调用 ReadStr 函数从输入缓冲器读入数据到 Str 变量，此函数直到 Count 字节数据被读完或时间片被用完才返回。返回值为实际读入数据的字节数。

注意：参数 Count 设为 0 时不要调用此方法。

function ReadStrAsync(var Str: String; Count: Integer; var AsyncPtr: PAsync): Integer

以异步方式从输入缓冲器读入数据到 String 变量。调用 ReadStrAsync 函数读 Count 字节数据到 Str 变量。函数立即返回，并不等待操作执行完，通过调用 WaitForAsync 函数来确定操作执行完。返回值是实际读入数据的字节数。AsyncPtr 参数是被 InitAsync 初始化的 PAsync 结构。操作执行完后应用程序负责调用 DoneAsync 方法释放 AsyncPtr 指针。

注意：如果 WaitForAsync 的返回值不同于参数 Count 的值，程序中必须利用 SetLength 函数手动调整 Str 参数的长度，参数 Count 设为 0 时不要调用 ReadStrAsync 方法。

procedure SetBreak(OnOff: Boolean)

设置中断或非中断传输状态。

利用 SetBreak 函数设置中断或非中断传输状态，OnOff 参数可为下列值之一：

True	暂缓字符传输且置为中断传输状态
False	恢复字符传输且置为非中断传输状态

procedure SetDTR(OnOff: Boolean)

使能 DTR 信号。利用 SetDTR 函数使能 DTR（数据终端准备好）信号。OnOff 参数为下列值之一：

True	设置 DTR 信号
False	清除 DTR 信号

注意：如果 FlowControl.ControlDTR 属性设为 dtrHandshake，调用 SetDTR 函数应用程序将产生一个 EcomPort 异常错误。

procedure SetRTS(OnOff: Boolean)

使能 RTS 信号。利用 SetRTS 函数使能 RTS（请求发送）信号，OnOff 参数为下列值之一：

True	设置 RTS 信号
False	清除 RTS 信号

注意：如果 FlowControl.ControlRTS 属性设为 rtsHandshake，调用 SetRTS 函数应用程序将产生一个 EcomPort 异常错误。

procedure SetXonXoff(OnOff: Boolean)

模拟 Xon/Xoff 字符。利用 SetXonXoff 函数模拟 Xon/Xoff 字符到达，OnOff 参数为下列值之一：

True	模拟 Xon 字符到达
False	模拟 Xoff 字符到达

procedure ShowSetupDialog

显示串口调整对话框。调用 ShowSetupDialog 方法打开串口调整对话框，如果串口已被打开且用户点击 OK 按钮，则不需再打开串口就应用新设置。

function Signals: TComSignals

显示哪个信号是高电平。调用 Signals 函数来决定信号的状态，如果返回值中包含某一特殊信号，则此信号是高电平状态。要想改变信号的状态利用 OnXChange 事件。Signals 函数能判断下列信号的状态：

csCTS	清除发送
csDSR	（数据设备准备好）DCE 就绪
csRing	Ring 指示
csRLSD	接收线信号检测

function StateFlags: TComStateFlags

调用 StateFlages 函数获得串口通信设备的信息。Flags 有如下意义：

fCtlHold	如果此标记被设置，则说明传送要等待 CTS（清除发送）信号被发送出去
fDsrHold	如果此标记被设置，则说明传送要等待 DSR（DCE 就绪）信号被发送出去
fRlsHold	如果此标记被设置，则说明传送要等待 RLSD（接收线信号检测）信号被发送出去

fXoffHold	如果此标记被设置，则说明传送要等待，因为 Xoff 字符被接收
fXoffSent	如果此标记被设置，则说明传送暂停，直到 Xoff 字符被发送给一个系统，且此系统认为下一个字符是 Xon（而不管实际字符是什么）。
fEof	标志 EOF 字符是否被接收。如果此标记被设置，则说明 EOF 字符被接收
fTxim	如果此标记被设置，则有一字符队列，此队列是通过 TransmitChar 函数发送到设备上的。此种字符的发送在通信设备的输出缓冲器中要先于其他字符

procedure StoreSettings(StoreType: TStoreType; StoreTo: String)

StoreSettings 方法将 TCustomComPort 类的属性值存储到注册表或文件中，这些属性的存储取决于 StoredProps 属性，StoreType 参数为下列值之一：

stRegistry	StoreTo 参数为注册表键值
stIniFile	StoreTo 参数为文件名

procedure TransmitChar(Ch: Char)

传送一个字符。利用 TransmitChar 方法传送一个字符（优先于输出缓冲器中任意未传送的字符）。此函数在发送一个中断字符（如 CTRL+C）到主机系统时是非常有用的。字符传送遵从标准的流控制和信号交换机制。

注意：如果设备没有发送字符，TransmitChar 不能被重复调用。一旦 TransmitChar 将一个字符放入输出缓冲器，这个字符必须在再次调用此函数前被发送出去。如果前一字符没有被发送出去，TransmitChar 产生一个错误。

function WaitForAsync(var AsyncPtr: PAsync): Integer

等待异步操作执行完。调用 WaitForAsync 函数确定异步操作已执行完。该函数直到异步操作执行完或放弃执行此操作函数才返回。AsyncPtr 参数是一个被 InitAsync 初始化的变量。

注意：应用程序还要利用 DoneAsync 方法释放 AsyncPtr 变量。

function Write(const Buffer; Count: Integer): Integer

将数据写到输出缓冲器。调用 Write 函数将 Count 字节数据从 Buffer 变量写到输出缓冲区，函数执行直到 Count 字节数据被写完或时间片用完才返回。

function WriteAsync(const Buffer; Count: Integer; var AsyncPtr: PAsync): Integer

以异步方式将数据写到输出缓冲器。调用 WriteAsync 函数将 Count 字节数据从 Buffer 变量写到输出缓冲区。函数立即返回，并不等待操作执行完，通过调用 WaitForAsync 函数来确定操作执行完。返回值是实际写入数据的字节数。AsyncPtr 参数是被 InitAsync 初始化的 Pasync 记录。操作执行完后应用程序负责调用 DoneAsync 方法释放 AsyncPtr 指针。

function WriteStr(const Str: String): Integer

将字符串写到输出缓冲器。调用 WriteStr 函数将 Str 变量写到输出缓冲器，此函数直到

字符串被写完或时间片被用完才返回。返回值为实际写入缓冲区的字节数。

function WriteStrAsync(const Str: String; var AsyncPtr: PAsync): Integer

以异步方式将字符串写到输出缓冲器。调用 WriteStrAsync 函数将 Str 变量写到输出缓冲区。函数立即返回，并不等待操作执行完，通过调用 WaitForAsync 函数来确定操作执行完。返回值是实际写入数据的字节数。AsyncPtr 参数是被 InitAsync 初始化的 Pasync 记录。操作执行完后应用程序负责调用 DoneAsync 方法释放 AsyncPtr 指针。

procedure CreateHandle; virtual

创建串口句柄。Open 方法调用 CreateHandle，CreateHandle 是真正建立连接，调用 CreateHandle 后，Handle 属性有效。

procedure DestroyHandle; virtual

析构串口句柄。DestroyHandle 在 Close 方法内部被调用。串口连接关闭且 Handle 属性值无效。

procedure DoAfterClose; dynamic

调用 OnAfterClose 事件句柄。串口被成功关闭后，DoAfterClose 方法被调用。DoAfterClose 方法调用 OnAfterClose 事件句柄。TCustomComPort 类的子类可以重载这个方法以执行特殊的行为。

注意：不要直接调用 DoAfterClose 方法，因为它会自动被 TCustomComPort 调用。

主要事件

OnAfterClose

串口关闭后触发。利用 Close 方法或 Connected 属性在关闭串口后写 OnAfterClose 事件句柄。如果串口关闭操作失败，OnAfterClose 事件不被触发。

OnAfterOpen

串口打开后触发。利用 Open 方法或 Connected 属性在打开串口后写 OnAfterOpen 事件句柄以执行特定的操作。如果串口打开操作失败，OnAfterOpen 事件不被触发。

OnBeforeClose

串口关闭前发生。利用 Close 方法或 Connected 属性在关闭串口前写 OnBeforeClose 事件句柄。如果在 OnBeforeClose 事件句柄中发生异常错误，串口不被关闭。

OnBeforeOpen

串口打开前触发。利用 Open 方法或 Connected 属性在打开串口前写 OnBeforeOpen 事件句柄。如果在 OnBeforeOpen 事件句柄中发生异常错误，串口不被打开。

OnBreak

在输入时检测到中断后触发。

OnCTSChange

CTS 信号状态改变时触发。当 CTS（清除发送）信号状态改变时写 OnCTSChange 事件句柄以执行特定操作。如果 OnOff 参数为 True，CTS 从低电平转换到高电平；如果为 False，

从高电平转换到低电平。

OnDSRChange

DSR 信号改变时触发。当 DSR（DCE 就绪）信号状态改变时写 OnDSRChange 事件句柄。如果 OnOff 参数为 True，DSR 从低电平转换到高电平；如果为 False，从高电平转换到低电平。

OnError

线路状态发生错误时触发。当线路状态发生错误时写 OnError 事件句柄。线路状态错误类型有 ceFrame, ceOverrun, ceRxParity。

注意：如果应用程序在 OnError 事件内调用 LastErrors 方法，它将返回空结果。

OnRing

检测到有 Ring 指示时发生。当检测到 Ring 指示器时写 OnRing 事件句柄，此事件在开发调制解调器应用程序时非常有用。

OnRLSDChange

RLSD 信号状态改变时触发。当 RLSD 信号状态改变时写 OnRLSDChange 事件句柄。如果 OnOff 参数为 True，RLSD 从低电平转换到高电平；如果为 False，从高电平转换到低电平。

OnRx80Full

输入缓冲区的 80%已被占用时触发。当输入缓冲区的 80%已被占用时写 OnRx80Full 事件句柄。这对应用程序是一个警告，即输入缓冲区即将溢出，除非有某类流控制规定。

OnRxChar

输入缓冲区中有字符到达时触发。当输入缓冲区中有字符到达时写 OnRxChar 事件句柄以获得最近到达的字符的信息，Count 是输入缓冲区等待读入数据的字节数，OnRxChar 事件经常用于从输入缓冲区中读字符。

OnRxBuf

数据从输入缓冲区中被读出后触发。写 OnRxChar 事件句柄以得到已到达输入缓冲器中的数据。但当它已被其他的诸如 TComDataPacket 或 TCustomComTerminal.组件读出且放入 Buffer 参数时，应用程序不能从输入缓冲区中读出数据。因为它已经被读过了。

注意：应用程序能通过 TriggersOnRxChar 参数检查是因为 OnRxBuf 或是 OnRxChar 被调用。

OnRxFlag

事件字符到达时触发。当事件字符到达时写 OnRxFlag 事件句柄，用 EventChar 属性来设置事件字符为 ASCII 码。

注意：一个事件字符也能被放入输入缓冲区。事件和非事件字符的不同之处在于事件字符触发 OnRxChar 和 OnRxFlag 事件，而非事件字符仅仅触发 OnRxChar 事件。

OnTxEmpty

输出缓冲器空时触发。输出缓冲器空时写 OnTxEmpty 事件句柄，即输出缓冲器中所有的字符都已被传送出去。

3.4.2.2 TComTerminal

用途

TComTenminal 控件从 TCustomComPort 接收数据并显示在屏幕上。此控件支持到 VT52, VT100 和 ANSI 终端服务器的连接。应用程序在 OnGetEscapeCodes 事件中能提供自定义的转义码解析。

主要属性

AltColor：TColor——设置终端屏幕的字体颜色。当终端接收到要求改变当前字体颜色的转义码时，字体颜色自动改变。

AppendLF：Boolean——使能添加换行功能。利用 AppendLF 属性指定是否在每次接收到回车字符（ASCII 码为 13）后置换行字符（ASCII 码为 10），如果为 True，则在回车后添加换行符。

ArrowKeys：TArrowKeys——当预置 ArrowKeys 为下列值之一时，定义怎样控制光标键的反应：

akTerminal	光标键用作终端键，按下光标键就给服务器送一个转义码
akWindows	光标键用作 windows 键

AutoSize: Boolean——确定是否根据其内容自动改变控件的大小。

Caret：TTermCaret——设置插入符号类型。设置 Caret 属性指定终端控件插入符号类型。插入符号或终端光标在聚焦于终端控件时是可见的。Caret 可为下列值之一：

tcBlock	块类型的插入符号
tcUnderline	下划线类型的插入符号
tcNone	插入符号不可见

Color：TColor——设定背景颜色。用 Color 属性为终端屏幕设置背景颜色，默认背景颜色为黑色。

Connected：Boolean——显示连接是否起作用。设置 Connected 属性为 True 打开到终端服务器的连接；设为 False 断开连接。应用程序通过检查 Connected 的值判断是否连接。

注意：如果 ComPort 属性值为零，Connected 属性值永远为 False。

ComPort：TCustomComPort——设定 TCustomComPort 控件。设置 ComPort 属性以连接 TCustomComTerminal 控件和 TCustomComPort 控件，ComTerminal 从 ComPort 组件获得所有的数据，几个 TComTerminal 类控件能与一个 ComPort 控件连接。

Columns：Integer——设定栏数。利用 Columns 属性设置终端屏幕的栏数，默认值为 80。

Emulation：TTermEmulation——设定仿真类型。设置 Emulation 属性定义如何解释转义码。大多数终端服务器是与 VT100/ANSI 兼容的，如果 Emulation 值为 teNone，在 OnGetEscapeCodes 事件句柄中提供默认的转义码的处理。Emulation 属性为下列值之一：

teVT100orANSI	与 VT100/ANSI 兼容的转义码
teVT52	与 VT52 兼容的转义码
teNone	不解释转义码

Font：TFont——设置终端的字体。利用 Font 属性设置终端的字体。TCustomComTerminal 只能用其自己固定的几种字体，如果所选择的字体与终端要求不匹配，则终端字体设置为默认值。

Force7Bit：Boolean——设置是否处理 7 位的字符。利用 Force7Bit 属性能够对 7 位的字符的处理，若为 True，所有接受到的字符都按 7 位来处理。

LocalEcho：Boolean——使能局部响应。利用 LocalEcho 属性使能局部响应，若为 True，所有通过终端客户送到服务器的 ASCII 字符显示在屏幕上，若为 False，局部字符在屏幕上没有反应。

注意：如果客户向终端服务器发送数据后，终端服务器没有回送数据，推荐你设置 LocalEcho 为 True。

SendLF：Boolean——使能发送换行符。利用 SendLF 属性指定是否在每次回车字符（ASCII 码为 13）后发送换行字符（ASCII 码为 10），如果为 True，则在回车后发送换行符。

Rows：Integer——指定行数。利用 Rows 属性设定终端屏幕的行数，默认是 24。

WrapLines：Boolean——使能行卷标。利用 WrapLines 属性指定是否启用行卷标，如果为 True，当光标到达当前行的末尾时自动转到下一行的开头；若为 False，行末的字符交互重叠。

主要方法

procedure DoChar(Ch: Char)

当需要将非转义码字符显示在终端屏幕上时调用 DoChar 方法，此方法调用 OnChar 事件句柄。可以重载此方法，使得字符显示在终端屏幕上同时执行某操作。

注意：不要直接调用 DoChar 方法，因为 TCustomComTerminal 类组件会自动调用它。

procedure DoGetEscapeCodes(var EscapeCodes: TEscapeCodes)

当 TCustomComTerminal 需要处理转义码时调用 DoGetEscapeCodes 方法，此方法调用 OnGetEscapeCodes 事件句柄。当 TCustomComTerminal 需要自定义处理转义码过程可重载此方法。

注意：不要直接调用 DoGetEscapeCodes 方法，因为 TCustomComTerminal 类组件会自动调用它。

procedure DoStrRecieved(var Str: string)

当要从输入缓冲区读出字串时调用 DoStrRecieved 方法，此方法调用 OnStrRecieved 事

件句柄。可以根据需要重载此方法。

注意：不要直接调用 DoStrRecieved 方法，因为 TCustomComTerminal 类组件会自动调用它。

procedure DoUnhandledCode(Code: TEscapeCode; Data: string)

当终端不能解释转义码时调用 DoUnhandledCode 方法，此方法调用 OnUnhandledCode 事件句柄。可根据需要重载此方法。

注意：不要直接调用 DoUnhandledCode 方法，因为 TCustomComTerminal 类组件会自动调用它。

procedure LoadFromStream(Stream: TStream)

调用 LoadFromStream 方法将从流中加载屏幕缓冲区。

procedure MoveCaret(AColumn, ARow: Integer)

调用 MoveCaret 方法移动插入符号到一新的位置。AColumn 参数表示新的列位置，ARow 参数表示新的行位置。如果 AColumn 或 Arow 超出屏幕范围，则插入符号放置到屏幕的边缘。

procedure WriteStr(Str: string)

调用 WriteStr 方法将字串显示在屏幕上，此方法并不将数据送到串口。Str 参数不能包含转义码，因为当调用此方法时终端不能解释它。如果应用程序需要向终端传送转义码，应该调用 WriteEscCode 方法。

procedure WriteEscCode(ACode: TEscapeCode; AParams: TParams)

调用 WriteEscCode 方法将转义码传送到终端屏幕，但并不通过串口送到终端服务器。ACode 参数是应用程序需要传送的转义码，AParams 参数指定转义码的次序。

procedure SaveToStream(Stream: TStream)

将屏幕缓冲区存入到一个流中。

注意：SaveToStream 方法不存储字符的属性，例如颜色，亮度等等。

procedure SelectFont

打开终端字体设置对话框，在对话框内允许用户进行固定字体之间的选择。

procedure ShowSetupDialog

显示终端设置对话框。调用 ShowSetupDialog 方法打开终端设置对话框，用户点击 OK 按钮，新的设置就立即被终端采用。

主要事件

OnChar

字符显示在屏幕上时触发。当一个非转义码字符显示在屏幕上时写 OnChar 句柄，这在应用程序需要将输入数据记录到文件或输出到打印机时是非常有用的。

OnGetEscapeCodes

终端需要定义转义码时触发。写 OnGetEscapeCodes 事件句柄以提供自定义的转义码处

理办法。如果 Emulation 值不是 teNone，OnGetEscapeCodes 事件被忽略；如果 Emulation 值是 teNone 且 EscapeCodes 参数为 nil，控件不能处理转义码。

OnStrRecieved

从串口接收到字符串时触发。当从串口输入缓冲区读出字串时写 OnStrRecieved 事件句柄，OnStrRecieved 事件是在处理任何转义码之前被调用。所以要在字符串被处理之前来进行修改，可在 OnStrRecieved 事件中进行。

OnUnhandledCode

转义码没有被处理时触发。当转义码没有缺省处理操作或没有被识别时写 OnUnhandledCode 事件句柄，Code 参数是该转义码的值，Data 参数是把该转义码作为 String 类型。

3.4.2.3 TComDataPacket

用途

TComDataPacket 执行信息包的读操作。利用 TComDataPacket 控件能方便地从信息包的输入缓冲区读数据。应用程序通过设置 TComDataPacket 的属性能为信息包设定起始和停止条件。一个 TCustomComPort 类控件能与一个以上的 TComDataPacket 控件链接。

主要属性

Buffer：String——存储接收到的数据。利用 Buffer 属性从 ComPort 控件存取原始数据。

CaseInsensitive：Boolean——为起始和停止字符串设定大小写检查。若为 True，TComDataPacket 忽略大小写检查；若为 False，则包括大小写检查。

ComPort: TCustomComPort——指定 TCustomComPort 控件。设置 ComPort 属性用以链接 TComDataPacket 控件和 TCustomComPort 控件。控件从 ComPort 控件获得所有的数据。多个 TComDataPacket 控件能与相同的 ComPort 控件链接。

IncludeStrings: Boolean——说明是否包含起始和停止字符串。若为 False，则在 OnPacket 事件内不包含起始和停止字符串。

MaxBufferSize: Integer——指定缓冲区的最大长度。若缓冲区的大小达到 MaxBufferSize 的值，缓冲区将被丢弃并且缓冲区内的当前信息包将被删除。

Size: Integer——指定信息包停止位的长度。如果当前信息包的长度等于或大于 Size 的值，该信息包被结束且 OnPacket 事件被触发，如果 Size 值为 0，不设置停止位。

注意：若 StopString 不为空且在信息包内存在 StopString 字符，信息包立即被结束而不管其 Size 值；若 StopString 为空且 Size 值为 0，OnPacket 事件与 TCustomComPort 控件的 OnRxBuf 事件有相同的功能。

StartString: String——指定信息包起始字符串。若 StartString 不为空，StartString 字符到达输入缓冲区时信息包开始；若 StartString 为空，任何字符到达输入缓冲区时信息包开始。

注意：若 TComDataPacket 指派了 OnCustomStart 事件，StartString 值都被忽略且停止位在 OnCustomStop 事件句柄内定义。

StopString: String——指定信息包停止字符串。若 StopString 不为空，StopString 字符到达输入缓冲区时信息包结束，OnPacket 事件发生。

注意：如果 Size 不为 0 且大于当前信息包的长度，信息包立即结束而忽略停止位。OnPacket 事件与 TCustomComPort 控件的 OnRxBuf 事件有相同的功能。

主要方法

procedure DoDiscard(Str: String)

当数据丢弃时调用 DoDiscard 方法，此方法调用 OnDiscard 事件句柄。其派生类能重载此过程。

注意：不要直接调用 DoDiscard 方法，因为 TComDataPacket 会自动调用它。

procedure DoPacket(Str: String)

信息包形成时调用 DoPacket，此方法调用 OnPacket 事件句柄。其派生类可根据需要重载此方法。

注意：不要直接调用 DoPacket 方法，因为 TComDataPacket 会自动调用它。

procedure HandleBuffer

分离信息包内的数据。从输入缓冲区读出数据将它置入 Buffer 后调用 HandleBuffer 方法。HandleBuffer 利用起止位将接收到的数据分离开后放入信息包，其派生类能重载此函数以执行自定义的包分离操作。

procedure Notification(AComponent: TComponent; Operation: TOperation)

当对象被创建或被析构时对事件通知做出响应。对象被创建或被析构时自动调用 Notification。调用了这个方法后，Notification 检查由 ComPort 属性指定的 TCustomComPort 控件是否要被析构。控件被析构，Notification 设置 ComPort 属性值为 nil。

主要事件

OnCustomStart

控件需要自定义包的起始位时发生。生成自定义包时写 OnCustomStart 事件，若 TComDataPacket 已有指派的 OnCustomStart 事件句柄，起始位在 OnCustomStart 事件句柄内定义。Str 参数是当前缓冲区，Pos 参数是包的起始位。应用程序必须设置 Pos 参数以定义包的起始位。如果 Str 参数中没有与自定义起始位相匹配的数据，应用程序需设置 Pos 参数为 0。

OnCustomStop

控件需要自定义包的停止位时发生。生成自定义包时写 OnCustomStop 事件。若 TComDataPacket 已有指派的 OnCustomStop 事件句柄，停止位在 OnCustomStop 事件句柄内定义。Str 参数是当前缓冲区，Pos 参数是包的停止位。应用程序必须设置 Pos 参数以定义包的停止位。如果 Str 参数中没有与自定义停止位相匹配的数据，应用程序需设置 Pos 参数为 0。

OnDiscard

丢弃数据时发生。数据丢弃时写 OnDiscard 事件，通过 OnDiscard 事件后任何数据不再出现 OnPacket 事件中。

OnPacket

包结束时发生。生成包时写 OnPacket 事件，出现一位停止位时结束包。信息包字符串保存在参数 Str 中。

3.4.2.4 TComComboBox

用途

TComComboBox 是一个选择串口设置的 windows 组合框。它能被链接到 TCustomComPort 派生类以便在串口上实现所需的串口设置。

主要属性

AutoApply: Boolean——说明改变的设置是否自动应用到串口。若 AutoApply 为 True 且 ComPort 不为空，当组合框里的选项变化时，新的选择设置将自动应用到串口。

注意：如果 TCustomComPort 组件改变了在组合框里显示的属性，组合框里这些选项并不自动更新，应用程序必须调用 UpdateSettings 方法。

ComPort: TCustomComPort——指定 TCustomComPort 控件。将 TComComboBox 控件和 TCustomComPort 组件链接起来以更新其改变。

ComProperty: TcomProperty——选择串口设置。设置 ComProperty 属性选择 TCustomComPort 的哪个属性需在组合框内显示。ComProperty 为下列值之一：

cpNone	组合框内没有属性显示
cpPortNum	PortNum 属性在组合框内显示
cpBaudRate	BaudRate 属性在组合框内显示
cpDataBits	DataBits 属性在组合框内显示
cpStopBits	StopBits 属性在组合框内显示
cpParity	ParityBits 属性在组合框内显示
cpFlowControl	FlowControl 属性在组合框内显示

主要方法

procedure Change

如果允许将应用新设置，组合框内的当前选项改变时自动调用 Change 方法。若 AutoApply 为 True 且 ComPort 不为空，链接的 TCustomComPort 组件将发生相应的变化。Change 方法调用 OnChange 事件。

procedure UpdateSettings

更新组合框内的选项。调用 UpdateSettings 方法更新组合框内的选项，显示

TCustomComPort 属性设置的正确状态。若 ComPort 为空，此方法无效。

procedure ApplySettings

将设置应用到 TCustomComPort 控件。调用 ApplySettings 方法将当前组合框内的选项应用到 TCustomComPort 组件。若 ComPort 为空，此方法无效。

注意：若 AutoApply 为 True，选项改变时自动被应用。

procedure Notification(AComponent: TComponent; Operation: TOperation); override

对象被创建或析构时响应事件通知。组件被创建或被析构时自动调用 Notification。调用了这个继承的方法后，Notification 检查被 ComPort 属性指定的 TCustomComPort 控件是否要被析构。若控件被析构，Notification 设置 ComPort 属性值为空。

3.4.2.5 TComRadioGroup

用途

TComRadioGroup 是一个选择串口设置的单选按钮组。使用 TComRadioGroup 控件易于选择通用的串口设置。TComRadioGroup 能被链接到 TCustomComPort 派生类以便在串口上实现所需的串口设置。

主要属性

AutoApply: Boolean——说明改变的设置是否自动应用到串口。

注意：如果 TCustomComPort 组件改变了在单选按钮组里显示的属性，单选框里这些选项并不自动更新，应用程序必须调用 UpdateSettings 方法。

ComPort: TCustomComPort——指定 TCustomComPort 组件。设置 ComPort 属性以链接 TComRadioGroup 控件和 TCustomComPort 组件。

ComProperty: TComProperty——选择串口设置。设置 ComProperty 属性选择 TCustomComPort 的哪个属性需在单选按钮组内显示。ComProperty 为下列值之一：

cpNone	单选按钮组内没有属性显示
cpPortNum	PortNum 属性在单选按钮组内显示
cpBaudRate	BaudRate 属性在单选按钮组内显示
cpDataBits	DataBits 属性在单选按钮组内显示
cpStopBits	StopBits 属性在单选按钮组内显示
cpParity	ParityBits 属性在单选按钮组内显示
cpFlowControl	FlowControl 属性在单选按钮组内显示

主要方法

procedure Click

调用此方法应用新设置。单选按钮组内的当前设置改变时自动调用 Change 方法。若 AutoApply 为 True 且 ComPort 不为空，新设置立即被应用到与其链接的 TCustomComPort

组件中去，然后 Click 调用 OnChange 事件。

procedure UpdateSettings

更新单选按钮组内的选项。若 ComPort 为空，此方法无效。

procedure ApplySettings

将设置应用到 TCustomComPort 控件。调用 ApplySettings 方法将当前单选按钮组内的选项应用到 TCustomComPort 控件。若 ComPort 为空，此方法无效。

注意：若 AutoApply 为 True，选项改变时自动被应用。

procedure Notification(AComponent: TComponent; Operation: TOperation); override

对象被创建或析构时响应事件通知。

3.4.2.6 TComLed

用途

TComLed 是一个串口信号发光二极管指示器。利用 TComLed 控件能够方便地显示 CTS, DSR, RLSD, Ring, Tx 和 Rx 信号的状态。应用程序能提供自定义位图以显示信号的状态。当信号改变时，TComLed 能被链接到 TCustomComPort 组件以自动更新。

主要属性

ComPort: TCustomComPort ——设置 ComPort 属性链接 TComLed 控件和 TCustomComPort 控件以显示 LedSignal 信号的状态。若 ComPort 不为空，当串口信号改变时，TComLed 状态自动改变。

GlyphOn: TLedBitmap——设置信号状态打开时的自定义位图。当 State 值为 lsOn，位图被显示出来。

注意：Kind 属性值必须为 lkCustom。

GlyphOn: TLedBitmap——设置信号状态关闭时的自定义位图。当 State 值为 lsOff，位图被显示出来。

注意：Kind 属性值必须为 lkCustom。

Kind: TLedKind——选择位图显示串口信号状态，有 6 幅预定义位图可供选择。Kind 属性可为下列值之一：

LsRedLight	红灯
LsBlueLight	蓝灯
LsGreenLight	绿灯
LsYellowLight	黄灯
LsPurpleLight	紫灯
LsBulb	灯泡

LedSignal: TComLedSignal ——选择串口信号。设置 LedSignal 属性选择待监控的串口

信号。若 ComPort 不为空，TComLed 状态自动改变以显示所选信号的状态。LedSignal 可为下列值之一：

lsConn	串口连接/断开连接状态
lsCTS	CTS 信号状态
lsDSR	DSR 信号状态
lsRLSD	RLSD 信号状态
lsRing	探测到 Ring 信号
lsRx	数据接收信号状态
lsTx	数据发送信号状态

State: TLedState——设置 State 属性以改变 TComLed 状态。

注意：ComPort 不为空时手工改变 State 属性是无效的。

RingDuration: Integer——设置响铃信号的持续时间。Ring 信号与其他信号有些不同，TCustomComPort 并不报告其状态，它仅仅报告 Ring 信号是否被探测到，基于这一点，当探测到 Ring 信号时置其状态为 lsOn，持续时间过后置其状态为 lsOff。

主要方法

procedure DoChange(AState: TLedState)

调用 OnChange 事件。State 值改变时调用 DoChange 方法，其派生类能重载此方法。不要直接调用 DoChange 方法，因为它能被 TComLed 自动调用。

procedure Notification(AComponent: TComponent; Operation: TOperation)

对象被创建或析构时响应事件通知。

procedure Paint

响应 WM_PAINT 消息时调用 Paint 方法，它将绘制一个 TComLed 控件。其派生类能重载此方法为控件增加自定义绘图功能。

主要事件

OnChange

State 属性改变时触发。State 属性被手工或自动改变时触发 OnChange 事件。Astate 参数是 State 属性的新值。

以上是对控件包中的每个控件的简介，下面将结合实际讲解如何使用控件包中控件实现一些常用的功能。

- **列举端口**

在对串口进行编号之前，调用 EnumComPorts 过程对本机上的串口进行列举。

例：

```
begin
  EnumComPorts(ComboBox1.Items);
  //插入你的代码
```

```
  if ComboBox1.ItemIndex > -1 then
    ComPort1.Port := ComboBox1.Items[ComboBox1.ItemIndex];
end;
```

● **打开或关闭串口**

为了成功调用 TCustomComPort 大多数方法，应先打开串口。打开串口的方法有两种：在程序中调用 Open 方法或设置 Connected 属性为 True。要结束一次会话，可调用 Close 方法或设置 Connected 为 False。

例

```
begin
  ComPort1.Open; // 打开串口
  //插入你的代码
  ComPort1.Close; // 结束会话
end;
```

● **同步和异步操作**

在串口上执行读写操作有两种模式，同步或异步方式。在同步模式状态下，执行操作的方法直到操作完成（或中断）才返回；在异步模式状态下，执行操作的方法立即返回而不等操作完成。

每个异步操作在执行前都必须准备好。为了准备异步操作，应调用 InitAsync 方法。此方法初始化 PAsync 类型的参数。每个与异步操作有关的方法都有 PAsync 参数。因为 TCustomComPort 知道它所进行的操作，可以在 InitAsnyc 方法中使用相同的参数。在操作完成后，调用 DoneAsync 释放资源。

例（异步操作）：

```
var
  Operation1: PAsync;
begin
  InitAsync(Operation1);
  try
    ComPort1.WriteStrAsync('Hello', Operation1);
    //插入你的代码
    ComPort1.WaitForAsync(Operation1);
  finally
    DoneAsync(Operation1);
  end;
end;
```

● **写端口**

使用 TCustomComPort 能非常容易的实现写操作。写数据有四种方法：

Write	向输出缓冲区中写入无类型变量

WriteAsync	以同步方式向输出缓冲区中写入无类型变量
WriteStr	向输出缓冲区中写入字符串型变量
WriteStrAsync	以异步方式向输出缓冲区中写字符串型变量

应用程序应恰当设置写超时，详细的信息请看 TComTimeouts 类

例：

```
var
  Str: String;
begin
  Str := 'Hello';
  ComPort1.WriteStr(Str); // 字符串类型的变量
  ComPort1.Write(Str[1], Length(Str)); // 没有定义类型
end;
```

● **读端口**

从输入缓冲区中读入数据有两种方式。通常应用程序在 OnRxChar 事件中调用两种读方法中的一种，OnRxChar 事件是在字符到达输入缓冲区时触发。如果读方法在 OnRxChar 事件中被调用，读超时应设为不等待，也就是说，读方法检查输入缓冲区并立即返回，因为输入缓冲区的字节数是已知的。程序也能在 OnRxChar 事件外调用读方法，但必须正确的设置超时。

如果控件与其他需要引入数据的控件（如 TComDataPacket 或 TCustomComTerminal）相链接。OnRxChar 事件不能被调用，但是控件可调用 OnRxBuf 事件。因为数据已被读取过，应用程序不能从输入缓冲区中读取数据。但是数据已自动放在 OnRxBuf 事件的 Buffer 参数，通过检查 TriggersOnRxChar 属性可以确定是 OnRxChar 还是 OnRxBuf 事件被调用。

Read	向输出缓冲区中写入无类型变量
ReadAsync	以同步方式向输出缓冲区中写入无类型变量
ReadStr	向输出缓冲区中写入字符串型变量
ReadStrAsync	以异步方式向输出缓冲区中写字符串型变量

例（在 OnRxChar 事件中读数据）：

```
procedure TForm1.ComPort1RxChar(Sender: TObject; Count: Integer);
var
  Str: String;
begin
  ComPort1.ReadStr(Str, Count);
  // 对 Str 变量进行处理
end;
```

Example （在 OnRxChar 外读数据）

```
var
  Str: String;
begin
  // 在设计期或在此设置超时
  ComPort1.ReadStr(Str, NumberOfBytes);
  // 对 Str 变量进行处理
end;
```

例(在 OnRxBuf 中)

```
procedure  TForm1.ComPort1RxBuf(Sender:  TObject;  const  Buffer;  Count:
Integer);
begin
  //应用程序不需要从输入缓冲区中读数据
  // 数据已经在 buffer 参数中
  HandleData(Buffer, Count); // 处理数据
end;
```

- **中断异步操作**

异步操作能容易的被中断。虽然某个特别的操作不能被中断，但是进程中的所有线程都可以同步中断。如果操作被中断，WaitForAsync 方法将引发 EcomPort 异常并将 WinCode 属性设为 ERROR_OPERATION_ABORTED。

例：

```
var
  Operation1: PAsync;
begin
  InitAsync(Operation1);
  try
    ComPort1.WriteStrAsync('Hello', Operation1);
    // 插入你的代码
    if { some condition } then
      ComPort1.AbortAllAsync;
    // 插入你的代码
    ComPort1.WaitForAsync(Operation1);
  finally
    DoneAsync(Operation1);
  end;
end;
```

- **在运行期改变属性**

程序运行并与某个串口相连接时，除了 SyncMethod 外，TCustomComPort 所有的属性都能被更改。改变的属性立即被应用。当正连接着某个端口，Port 属性被改变时，串口将被

关闭并重新打开。如果属性在 BeginUpdate 方法和 EndUpdate 之间改变，此属性直到调用 EndUpdate 方法才应用。如果你一次不止改变一个属性，应在 BeginUpdate 和 EndUpdate 之间进行设置。

例：

```
begin
  ComPort1.Open;
  // 插入你的代码
 ComPort1.FlowControl.ControlDtr := dtrEnable; // 应用改变的设置
  // 插入你的代码
  BeginUpdate; // 防止应用改变的设置
  ComPort1.Parity.Bits := prOdd;
  ComPort1.FlowControl.XonXoffIn := True;
  EndUpdate; // 应用改变的设置
  // 插入你的代码
  ComPort1.Close;
end;
```

- **存储和加载设置**

程序通过 StoreSettings 和 LoadSettings 方法能很容易的加载和存储串口设置。设置可以存储在配置文件或注册表中，这由 StoredProps 属性来决定存储在哪一个当中。

例（存储在注册表）

```
begin
  // 将设置存储在注册表中
  ComPort1.StoreSettings(stRegistry,'\HKEY_LOCAL_MACHINE\Software\ComPortT
est');
  // 载入设置
 ComPort1.LoadSettings(stRegistry,
'\HKEY_LOCAL_MACHINE\Software\ComPortTest');

end;
```

例 (存储在配置文件)

```
begin
  // 将设置存储在配置文件中
  ComPort1.StoreSettings(stIniFile, 'c:\ComPortTest.ini');
  // 载入设置
  ComPort1.LoadSettings(stIniFile, 'c:\ComPortTest.ini');
end;
```

● **使用缓冲区函数**

应用程序能通过 ClearBuffer 方法来清除输入和（或）输出缓冲区。在调用 ClearBuffer 方法时应确保进程中没有异步操作，否则会引起一些想不到的问题。

例：

```
Begin
  //插入你的代码
  if { some condition} then
    ComPort1.ClearBuffer(True, False);  // clear input buffer
  //插入你的代码
end;
```

● **探测信号和端口状态**

在 OnXChange 事件中能检测到信号(CTS, DSR, RLSD)的改变，也可以在 OnRing 事件中检测到 Ring 信号。应用程序可以在任何时候通过调用 Signals 方法来检测信号的状态。

通过调用 StateFlags 方法，应用程序可以确定传输是处于进行中还是处于等待状态。

例：

```
procedure TForm1.ComPort1CTSChange(Sender: TObject; OnOff: Boolean);
begin
  if OnOff then
    PrintMessage('CTS high')
  else
    if fCtlHold in ComPort1.StateFlags then  // 如果传输在等待中
      PrintMessage('CTS low, transmission waiting')
    else
      PrintMessage('CTS low');
end;
```

● **改变输出流控制信号**

输出流控制信号（如 RTS 和 DTR）能被手工处理，除非它们被设置为 rtsHandshake 和 dtrHandshake。采用 SetRTS 或 SetDTR 方法可将信号设置为高电平或低电平状态。虽然在会话中设置 FlowControl 属性也能完成相同的功能，但是前种方法更快。

例：

```
begin
  ComPort1.SetRTS(False); // 设置RTS 为低电平
//插入你的代码
  ComPort1.SetRTS(True); // 设置RTS 为高电平
end;
```

● **获取错误信息**

当串口上发生错误时，触发 OnError 事件。检查事件中的 Errors 参数可获取发生在串口

上的错误信息。如果不使用 OnError 事件，它也可以通过调用 LastErrors 方法来检测端口上的上一次错误。LastErrors 将返回指示错误类型的标志并清空错误缓冲区。

例：

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

- **等待事件**

当事件属性不为空时，应用程序调用 Open 方法将创建一个特别的线程来监视端口事件。这只有在你的程序创建了消息循环时才有可能。大多数程序(GUI, NT 服务)都有消息循环。但是，如果你想在一个控制台程序中使用 TComPort 控件，你必须在程序中调用 Open 方法设置 Events 为空。

例：

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

begin
  ComPort := TComPort.Create(nil);
  try
    ComPort.Events := [];  //不创建监视线程
    ComPort.Open;  // 打开端口
    Events := [evCTS, evDSR, evRLSD];  // 定义等待的事件
    ComPort.WaitForEvents(Events, WaitInfinite);  // 等待直到至少有一个事件发生
    if evCTS in Events then
    WriteLn('CTS Changed');  // CTS 信号状态改变
    ComPort.Close;  // 关闭端口
  finally
    ComPort.Free;
  end;
end;
```

3.4.3 范例剖析

3.4.3.1 范例一

下面是一个 Comport 控件对串口进行操作的应用示例：

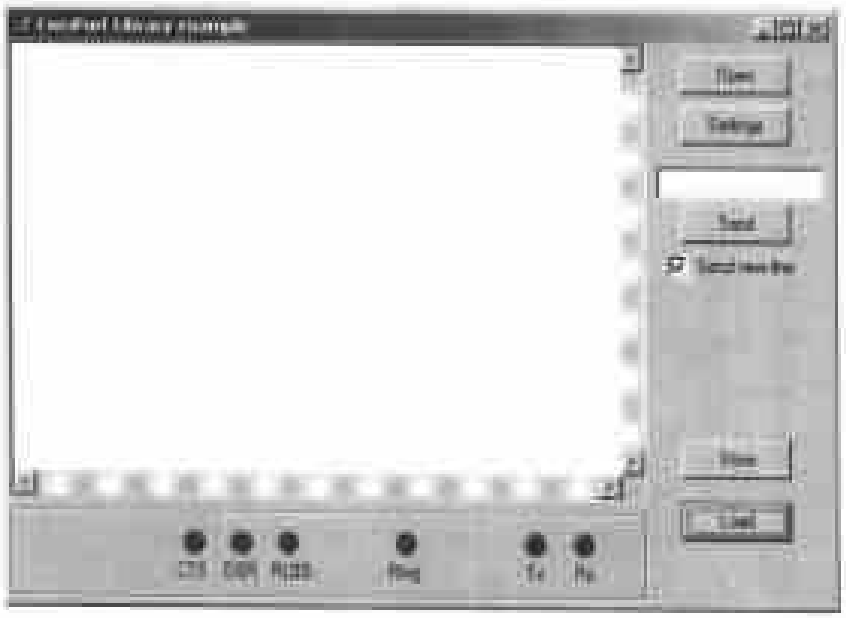

图 3-61　示例程序运行结果

程序清单如下：

```
unit ComMainForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls, ExtCtrls, CPort, CPortCtl;

type
  TForm1 = class(TForm)
    ComPort: TComPort;
    Memo: TMemo;
    Button_Open: TButton;
    Button_Settings: TButton;
    Edit_Data: TEdit;
    Button_Send: TButton;
    NewLine_CB: TCheckBox;
    Panel1: TPanel;
    Bt_Store: TButton;
    Bt_Load: TButton;
    ComLed1: TComLed;
    ComLed2: TComLed;
    ComLed3: TComLed;
```

```
    ComLed4: TComLed;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    ComLed5: TComLed;
    ComLed6: TComLed;
    Label1: TLabel;
    Label6: TLabel;
    procedure Button_OpenClick(Sender: TObject);
    procedure Button_SettingsClick(Sender: TObject);
    procedure Button_SendClick(Sender: TObject);
    procedure ComPortOpen(Sender: TObject);
    procedure ComPortClose(Sender: TObject);
    procedure ComPortRxChar(Sender: TObject; Count: Integer);
    procedure Bt_LoadClick(Sender: TObject);
    procedure Bt_StoreClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button_OpenClick(Sender: TObject);
begin
  if ComPort.Connected then
    ComPort.Close
  else
    ComPort.Open;
end;
```

```
procedure TForm1.Button_SettingsClick(Sender: TObject);
begin
  ComPort.ShowSetupDialog;
end;

procedure TForm1.Button_SendClick(Sender: TObject);
var
  Str: String;
begin
  Str := Edit_Data.Text;
  if NewLine_CB.Checked then
    Str := Str + #13#10;
  ComPort.WriteStr(Str);
end;

procedure TForm1.ComPortOpen(Sender: TObject);
begin
  Button_Open.Caption := 'Close';
end;

procedure TForm1.ComPortClose(Sender: TObject);
begin
  if Button_Open <> nil then
    Button_Open.Caption := 'Open';
end;

procedure TForm1.ComPortRxChar(Sender: TObject; Count: Integer);
var
  Str: String;
begin
  ComPort.ReadStr(Str, Count);
  Memo.Text := Memo.Text + Str;
end;

procedure TForm1.Bt_LoadClick(Sender: TObject);
begin
  ComPort.LoadSettings(stRegistry, 'HKEY_LOCAL_MACHINE\Software\Dejan');
end;
```

```
procedure TForm1.Bt_StoreClick(Sender: TObject);
begin
//  ComPort.StoreSettings(stIniFile, 'e:\Test.ini');
  ComPort.StoreSettings(stRegistry, 'HKEY_LOCAL_MACHINE\Software\Dejan');
end;
end.
```

3.4.3.2 范例二

下面是一个测试 Modem 的控制台程序，程序清单如下：

```
program ModTest;
{$APPTYPE CONSOLE}//打开控制台程序编译开关

uses
  SysUtils, CPort, Windows;

var
  ComPort: TComPort;
  Events: TComEvents;
  Answer, Data: string;
  Step: Integer;

begin
  try
    ComPort := TComPort.Create(nil);//创建 ComPort 实例
    try
      if ParamCount > 0 then
        ComPort.Port := ParamStr(1)
      else
        ComPort.Port := 'COM1';
      ComPort.Events := [];
      ComPort.FlowControl.ControlDTR := dtrEnable;
      ComPort.FlowControl.ControlRTS := rtsEnable;
      ComPort.Open; // open port
      ComPort.WriteStr('AT'#13#10); // 发送测试命令
      Answer := '';
      Step := 0;
```

```
    repeat
      Events := [evRxChar];
      ComPort.WaitForEvent(Events, 2000);  // 等待字符
      if evRxChar in Events then
      begin
        ComPort.ReadStr(Data, ComPort.InputCount);
        Answer := Answer + Data;
        if Pos('OK', Answer) > 0 then
          Break;
      end;
      Inc(Step)
    until (Events = []) or (Step = 20);
    if Pos('OK', Answer) > 0 then
      WriteLn('Modem found on ' + ComPort.Port)
    else
      WriteLn('Modem NOT found on ' + ComPort.Port);
  finally
    ComPort.Free;
  end;
except//异常处理
  on E: Exception do
    WriteLn('Error: ' + E.Message);
end;
end.
```