DISPOSITIVOS LÓGICOS MICROPROGRAMABLES	El componente TComPort	1

Índice / Introducción | Ampliación de contenidos

El componente TComPort

En construcción. Por verificar niveles de tensión

Índice de contenidos

Introducción

El componente TComPort (desarrollado por Dejan Crnila de Eslovenia) es un componente

gratuito para comunicaciones serie para Delphi y C++ Builder. Permite el acceso al puerto serie del PC en Windows XP, Vista y Windows 7. TComPort se basa en la API de Windows por lo que es totalmente compatible con los controladores o drivers pasarela tipo USB - RS232. Es fácil de utilizar para propósitos de comunicación serie básicos. En tcomport.rar está disponible la versión 4.11 y en la página oficial http://sourceforge.net/projects/comport/ estará disponible la última versión.

TComPort encapsula en un componente no visible el acceso al puerto serie del PC. Tiene métodos que permiten abrir y cerrar el puerto, así como escribir y recibir datos a través de el. Se puede descargar un archivo de ayuda de TComPort (de la versión 2.63) en inglés, muy útil aún en versiones posteriores.

En este tema se indica cómo instalar el componente TComPort en C++ Builder, como utilizarlo y algunos métodos del componente, como transmisión, recepción y nivel en entradas/salidas del puerto serie.

Existen otros componentes, la mayoría de pago, aunque hay otra potente alternativa gratis, TurboPower Async Professional (desarrollado por durkin10). Es una solución completa para Delphi, C++ Builder y entornos ActiveX. Proporciona acceso directo al puerto serie y al API de Microsoft, incluyendo el TAPI (Telephony Application Programming Interface). Soporta fax, emulación de terminal, VOIP y mas. Se encuentra disponible en http://sourceforge.net/projects/tpapro/

El puerto serie tiende a desaparecer en los equipos actuales para dar cabida a los puertos USB y TComPort funciona perfectamente en puertos Virtuales que usan como pasarela USB. Ver Conversores USB - RS232 y Puerto serie virtual VSPD y COMPIM de Proteus.

Instalación de TComPort en Borland C++ Builder 6

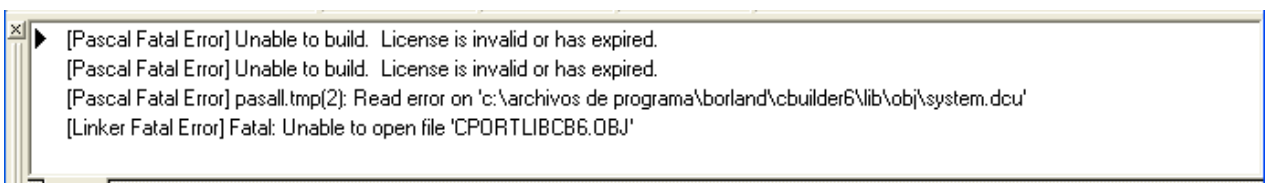
Los pasos se han detallado al máximo de manera que se pueda instalar el componente sin problemas.

Instalación del componente

Básicamente se trata de compilar e instalar los archivos que corresponden a la versión de CBuilder 6 que están en los archivos descargads de tcomport.rar o de http://sourceforge.net/projects/comport/, y que son; **CPortLibCB6.bpk** (el **Runtime Package** para compilar el paquete) y **DsgnCPortCB6.bpk** (el **DesingTime Package** para instalar el paquete en Builder).

También podemos descargarlo de tcomport.rar. En este rar encontraremos un ZIP como el que podemos descargar de SourceForge y el contenido de este ZIP descomprimido.

1. Antes de hacer nada, decir que si abriésemos **CPortLibCB6.bpk** en C++ Builder, aparecería el siguiente mensaje:



   Este error indica que la licencia de ComPort ha expirado.

2. Para evitar el error anterior debemos poner la fecha del sistema por ejemplo a

25/02/2002. Luego podremos cambiarla y no se producirán mas errores.

3. Descomprimir el archivo descargado de sourceforge o de tcomport.rar.

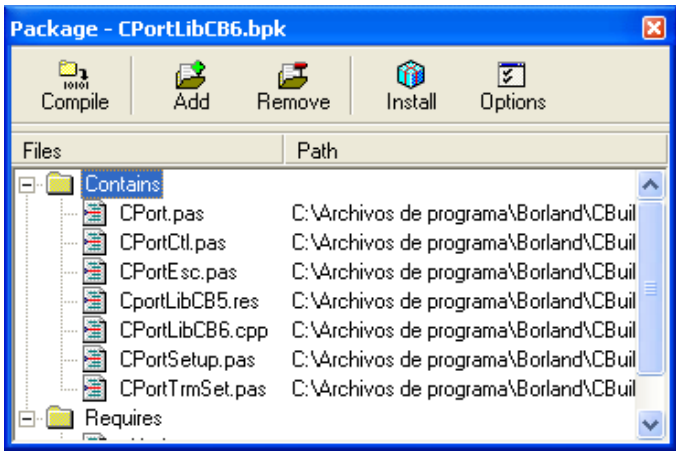
4. Copiar la carpeta "Source" en la carpeta "C:\Archivos de programa\Borland\CBuilder6 \Imports\". Una vez copiada renombrerla como "tcomport".

5. Abrir Borland C++ Builder 6.

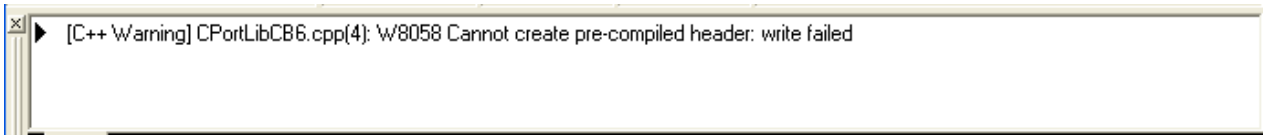
6. En Borland seleccionar ***File / Close All*** (Archivo / Cerrar todo).

7. Abrir el **Runtime Package** mediante ***File / Open Projects*** C:\Archivos de programa\Borland\CBuilder6\Imports\tcomport**CPortLibCB6.bpk**.

8. Al abrir el paquete aparece una ventana con las opciones para compilarlo o instalarlo.

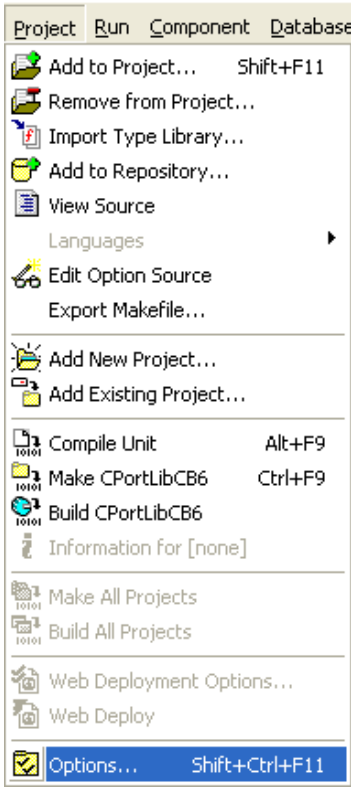


9. Si lo compiláramos sin más daría un error:

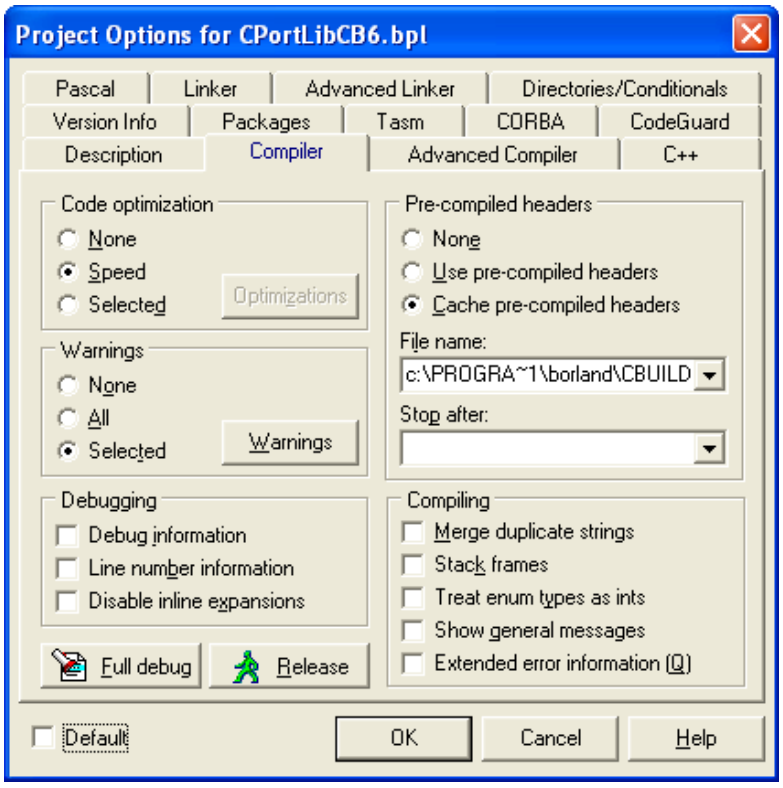


10. Para evitar este error en la compilación del paquete, antes de pulsar el botón **Compile**, debemos establecer que no vamos a utilizar cabeceras precompiladas.

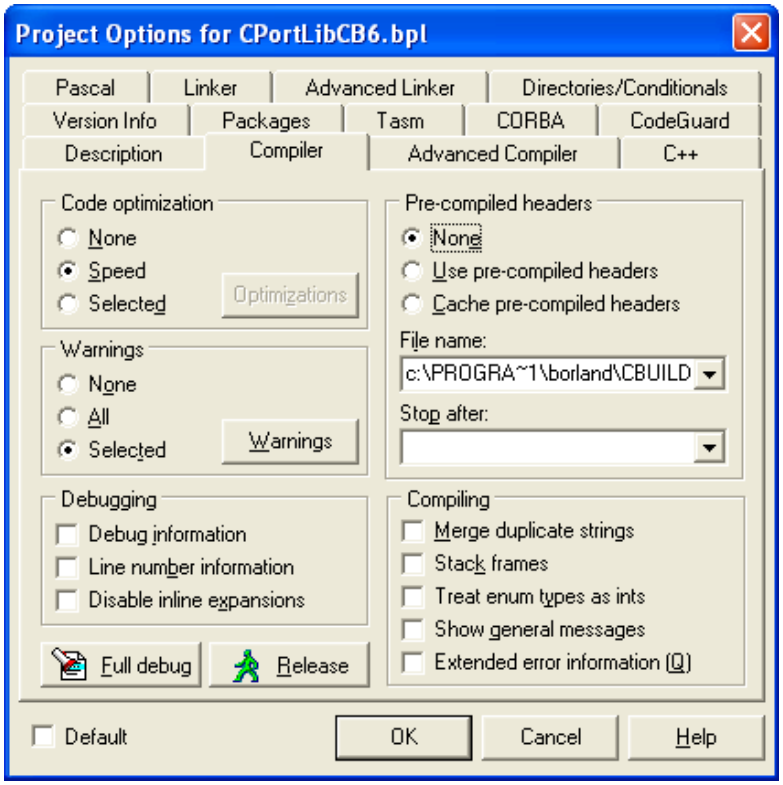
    Abrimos **Project / Options**:



Seleccionamos la solapa **Compiler** y nos aparece:

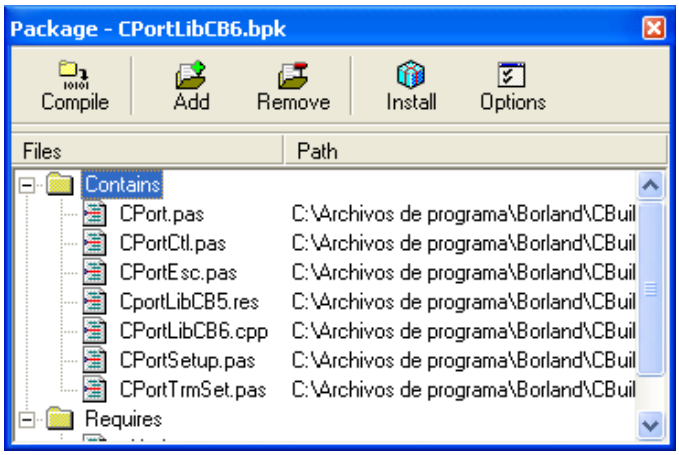


El grupo de opciones **Pre-compiled headers**, que estará en **Cache pre-compiled headers**, **c:\PROGRA~1\borland\CBUILD~1\lib\vcl60.csm**, debemos dejarlo en **None**.

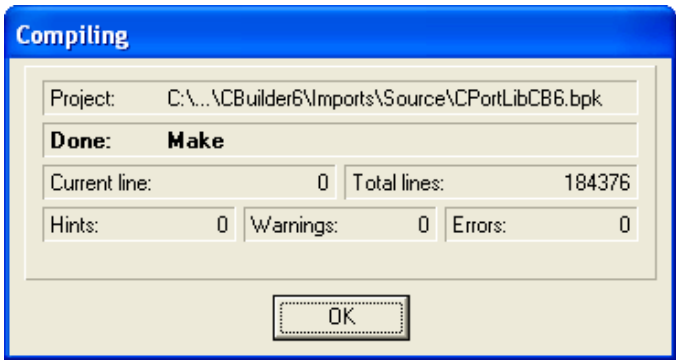


Pulsamos ***OK***.

11. Ahora en la ventana con las opciones para compilar o instalar el paquete hacemos click en el botón **Compile**.

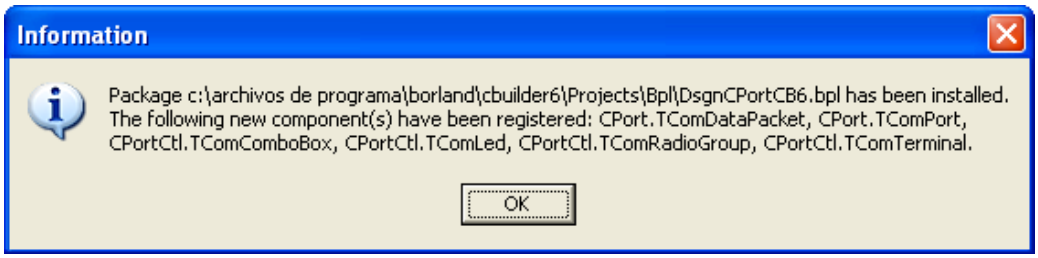


12. Una nueva ventana se abrirá cuando esté terminado. Pulsar OK.



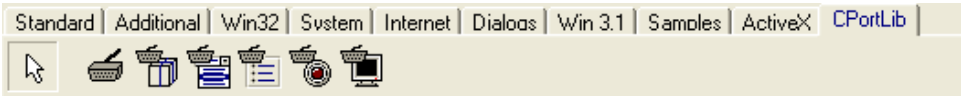
13. Abrir (sin cerrar nada) el **DesingTime Package** mediante ***File / Open Projects*** C:\Archivos de programa\Borland\CBuilder6\Imports\tcomport**DsgnCPortCB6.bpk**.

14. Aparece otra vez la ventana con las opciones compilar o instalar. Ahora le damos al botón **Install**. Al final del proceso debe aparecer un mensaje indicando que se instaló correctamente el paquete y nuevos componentes.



15. ***File / Close All*** (Archivo / Cerrar todo). Hacer clic en Sí cuando pregunte para guardar los cambios.

Si todo ha ido bien TComPort debe de aparecer en la paleta de componentes (al final) la solapa CPortLib y los componentes:



Cerramos Borland, pero aún no hemos terminado con la instalación.

Ubicación de archivos

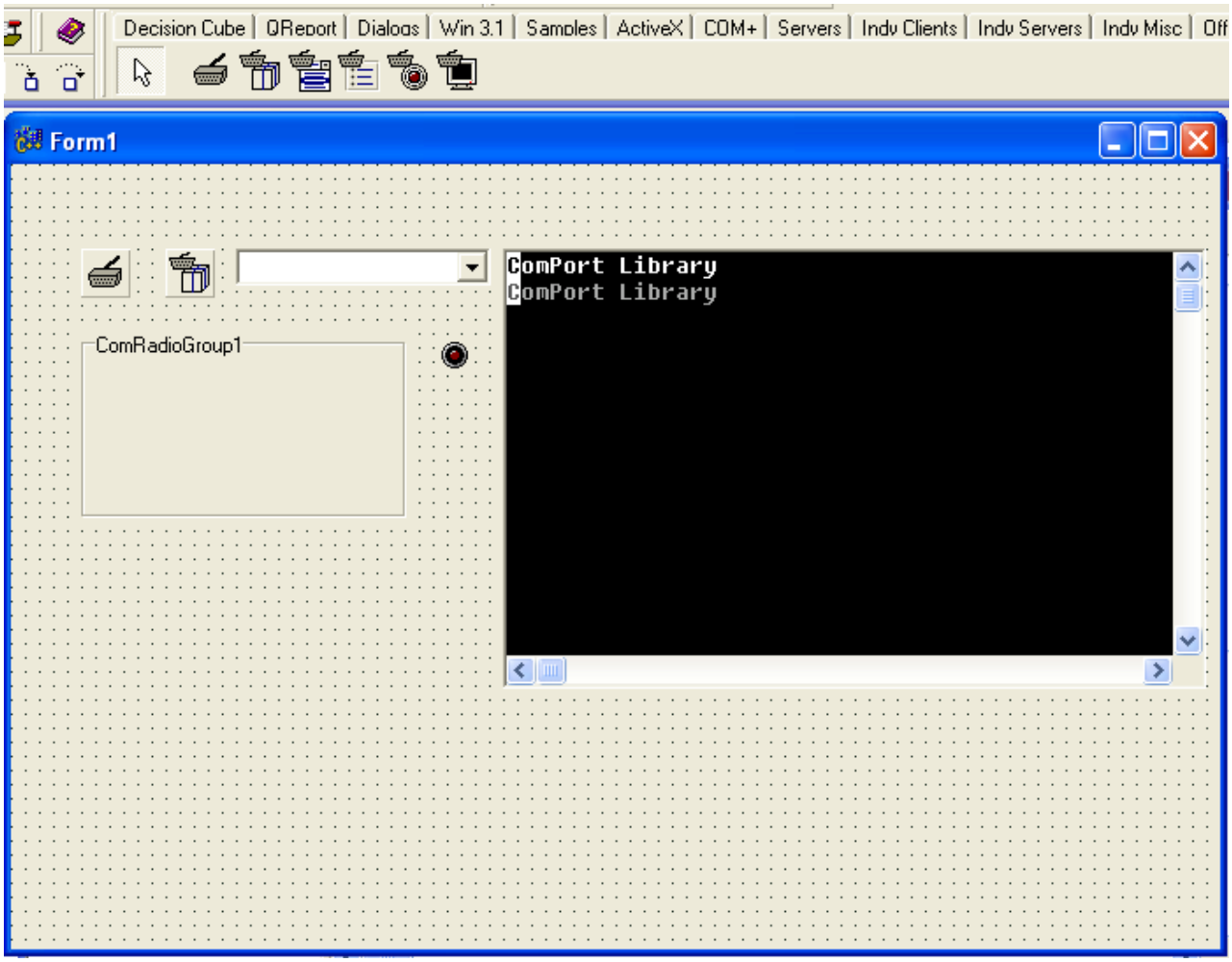
Es necesario reubicar los archivos generados en la carpeta "tcomport".

- Copiar los 7 archivos **.HPP** de "C:\Archivos de Programa\Borland\CBuilder6\Imports \tcomport\" a "C:\Archivos de Programa\Borland\CBuilder6\Include\Vcl\".
- Copiar los 9 archivos **.OBJ** de "C:\Archivos de Programa\Borland\CBuilder6\Imports \tcomport\" a "C:\Archivos de Programa\Borland\CBuilder6\Lib\Obj\".
- Copiar los 3 archivos **.DFM** de "C:\Archivos de Programa\Borland\CBuilder6\Imports \tcomport\" a "C:\Archivos de Programa\Borland\CBuilder6\Lib\Obj\".
- Copiar el archivo **CPortImg.res** de "C:\Program Files\Borland\CBuilder6\Imports \tcomport\" a "C:\Archivos de Programa\Borland\CBuilder6\Lib\Obj\".

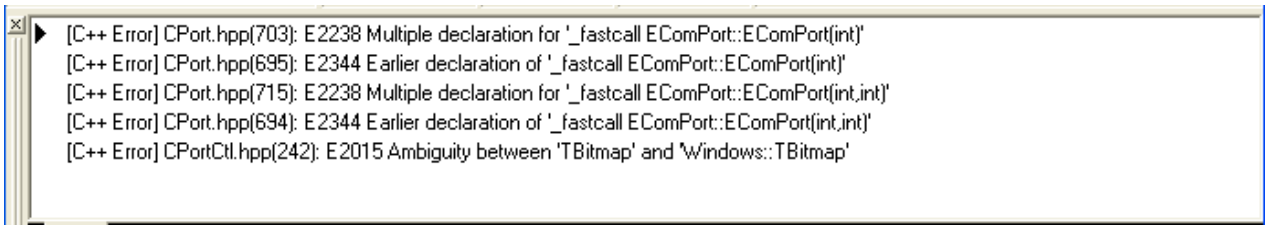
Correcciones de archivos

Es necesario corregir los archivos CPort.hpp y CPortCtl.hpp pues dan errores en Builder.

- Iniciar Borland C++ Builder.
- File / New / Aplication (Archivo / Nuevo / Aplicación).
- Colocar todos los nuevos componentes CPort en el form (son 6). Así se tendrá acceso a todo el código.



- Presionar F9 para compilar.
- Varios mensajes de error aparecerán, es normal y deben ser corregidos.



- Hacer doble clic en el primer mensaje de error:
  - /* Exception.CreateRes */ inline __fastcall EComPort(int Ident)/* overload */ : Sysutils::Exception(Ident) { }
- Comentar mediante //. Quedaría así:
  - // /* Exception.CreateRes */ inline __fastcall EComPort(int Ident)/* overload */ : Sysutils::Exception(Ident) { }
- Repetir con los otros 3 errores siguientes:
  - La línea:
    - __fastcall EComPort(int ACode);
  - Comentarla:
    - // __fastcall EComPort(int ACode);
  - La línea:
    - /* Exception.CreateResHelp */ inline __fastcall EComPort(int Ident, int AHelpContext)/* overload */ : Sysutils::Exception(Ident, AHelpContext) { }
  - Comentarla:

        - // /* Exception.CreateResHelp */ inline __fastcall EComPort(int Ident, int AHelpContext)/* overload */ : Sysutils::Exception(Ident, AHelpContext) { }
    - La línea:
        - __fastcall EComPort(int ACode, int AWinCode);
    - Comentarla:
        - // __fastcall EComPort(int ACode, int AWinCode);
- Hacer doble clic en el último error y transformar la línea:
    - typedef TBitmap TLedBitmap;
- por:
    - typedef Graphics::TBitmap TLedBitmap;
- Vamos a guardar **CPortCtl.hpp** con **File** / **Close** (Archivo / Cerrar).
- Hacer clic en Sí para guardar los cambios. Una vez corregido el archivo, al guardarlo todo quedará correctamente.
- Ahora vamos a guardar **CPort.hpp** con **File** / **Close** (Archivo / Cerrar).
- Hacer clic en Sí de nuevo para guardar los cambios. Una vez corregido el archivo, al guardarlo todo quedará correctamente.
- Presionar F9.
- La aplicación se ejecutará sin errores.
- Si nos equivocamos basta con copiar de nuevo los archivos **CPort.hpp** y **CPortCtl.hpp** de "tcomport" de "Import" y corregirlos de nuevo .
- No es necesario guardar la aplicación que creamos así que podemos cerrar Borland sin guardar los cambios.

Existe otro error a corregir cuando se pretende utilizar la asignación del puerto mediante un AnsiString:

```
ComPort1->Port="COM1";
```

Aparece el error:

```
[Linker Error] Unresolved external '__fastcall Cport::TCustomComPort::SetPortA(const
System::AnsiString)' referenced from I:\TRABAJO\PIC\PIC_PC\CPP_ENTERPRISE_TCOMPORT3\CPP05A\UNIT1.OBJ
```

Para solucionarlo es necesario modificar el archivo **CPort.hpp** de C:\Archivos de programa\Borland\CBuilder6\Include\Vcl

Cerramos Borland si estuviese abierto y abrimos el archivo **CPort.hpp** (vale cualquier editor de texto ASCII, como NotePad++). Buscamos la línea:

```
void __fastcall SetPort(const AnsiString Value);
```

Y la sustituimos por:

```
#ifdef SetPort
  #undef SetPort
  void __fastcall SetPort(const AnsiString Value);
  #define SetPort SetPortA
#else
  void __fastcall SetPort(const AnsiString Value);
#endif
```

Buscamos:

```
__property AnsiString Port = {read=FPort, write=SetPort};
```

Y la sustituimos por:

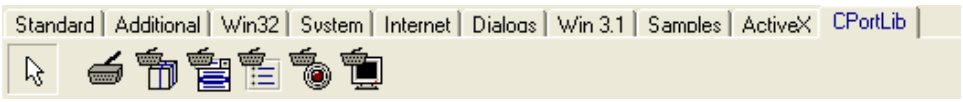
```
#ifdef SetPort
  #undef SetPort
  __property AnsiString Port = {read=FPort, write=SetPort};
```

```
  #define SetPort SetPortA
#else
  __property AnsiString Port = {read=FPort, write=SetPort};
#endif
```

Finalmente guardamos el archivo. Si el sistema no nos deja guardar el archivo en su ubicación, guardarlo en otra, reiniciar la máquina y colocar el archivo en su lugar (C:\Archivos de programa\Borland\CBuilder6\Include\Vcl).

Si todo ha ido bien, puede borrarse la carpeta "tcomport" de "Import" y volver a poner correctamente la fecha del sistema.

Componentes de TComPort



El paquete de diseño habrá instalado un nuevo conjunto de componentes que incluye:

- **ComPort**: Componente de acceso al puerto serie en sí.
- **ComDataPacket**: Componente para realizar operaciones de lectura de paquetes.
- **ComComboBox**: Componente visual tipo lista de texto desplegable para crear interface de configuración del puerto.
- **ComRadioGroup**: Componente visual tipo radio para crear interfaces de configuración del puerto.
- **ComLed**: Componente visual tipo LED para mostrar la actividad del puerto.
- **ComTerminal**: Componente que muestra datos serie en pantalla.

A continuación se detalla el uso de cada uno de estos componentes.

ComPort

Este es el principal componente de este paquete. Nos permitirá configurar y administrar el puerto serie. Contiene todas las propiedades de configuración de la conexión serie. Sin entrar en todos los detalles lo mas destacado es:

- Propiedad **Port**: Esta propiedad define el puerto a usar en el PC. En la lista desplegable de la propiedad aparecen sólo los puertos existentes (virtuales o físicos).

- Propiedad **Baudrate**: Establece la velocidad serie del enlace de transmisión.

- Propiedad **DataBits**: Establece el número de bits de cada carácter transmitido o recibido en el enlace serie.

- Propiedad **Parity**: Define el tipo de bit de paridad. Aquí es posible definir si la comprobación de paridad debe ser realizado en la lectura, y si se lleva a cabo para sustituir los carácteres con la falta de paridad en la trama recibida por un carácter específico. Para realizar pruebas es aconsejable no permitir la verificación de paridad, no es esencial para el buen funcionamiento y se puede añadir fácilmente más adelante si se desea reforzar el control. De todos modos, en caso de transmisión/recepción de tramas es mejor manejar una suma de comprobación de tipo CRC.

- Propiedad **StopsBits**: Número de bits de parada de cada caracter.

- Propiedad **FlowControl**: Este conjunto de propiedades se utilizan para definir la gestión

del flujo de caracteres en el enlace de serie. Normalmente es preferible no utilizar control de flujo, ya que esto sólo complica la gestión y el cableado del enlace. Además una gestión incorrecta puede hacer inestable al sistema operativo e incluso bloquearlo. Hay excepciones que obligan a su uso, como la gestión de un sistema multipunto con convertidores RS232-RS485.

- Propiedad **Timeouts**: Este conjunto de propiedades se utilizan para definir los tiempos de espera en la transmisión y recepción. Su uso depende de las aplicaciones y en principio se pueden dejar como están.

- Propiedad **Events**: Se utiliza para definir los eventos relacionados con la actividad de la conexión serie. De forma predeterminada están activadas, pero se pueden eliminar todos aquellos que no ayudan a evitar la sobrecarga del diálogo con el TComport.

- Propiedad **EventChar**: Esta propiedad establece el carácter de activación de OnRxChar. Por supuesto si el evento está habilitado en la propiedad **Events**.

- Propiedad **Connected**: Esta característica permite abrir/cerrar la conexión serie y también para comprobar su estado.

ComDataPacket

Este control permite gestionar de una manera muy sencilla la recepción de tramas de caracteres, o una cierta cantidad de caracteres.

- Propiedad **Port**: Selecciona a qué componente TComPort debe estar asociado. Esta propiedad siempre debe estar definida.

- Propiedad **Size**: Longitud de la estructura a recibir, si la duración no es fija, establecer esta propiedad en cero.

- Propiedad **StartString**: Cadena que define el inicio de la trama.

- Propiedad **StopString**: Cadena que define el final de la trama.

- Propiedad **CaseInsensitive**: Permite la selección de mayúsculas y minúsculas en los caracteres recibidos.

- Propiedad **IncludeStrings**: Establece si las cadenas StartString y StopString deben incluirse en la trama o no.

TComComboBox y TComRadioGroup

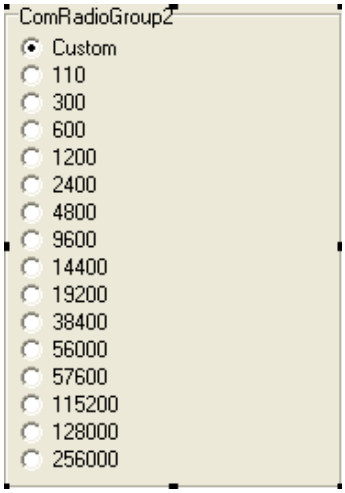
Estos componentes permiten configurar visualmente el enlace serie. Se utilizan como interfaz para la configuración del puerto COM.

La lista de propiedades es la siguiente:

- Propiedad **ComPort**: Selecciona a qué componente TComPort debe estar asociado. Esta propiedad siempre debe estar definida.

- Propiedad **ComProperty**: Se utiliza para definir los parámetros de la conexión que mostrará el componente. Tales como la velocidad, paridad, etc. Por ejemplo, en la imagen siguiente se muestra un ComRadioGroup con el valor de la propiedad ComProperty en

pBaudRate.



- Propiedad **AutoApply**: Establece si los cambios en la selección se introducen automáticamente en la configuración del enlace. Si esta propiedad es True, los cambios se aplicarán directamente a la configuración del puerto serie. Si esta propiedad es False, se debe llamar mediante código al método ApplySettings de este componente para que los cambios surtan efecto.

TComLed

TComLed es un indicador LED para las señales del puerto serie. Es un componente que permite mostrar de manera sencilla si el puero esta conectado y el estado de las señales CTS, DSR, RLSD, Ring, Tx y Rx . Podemos utilizar mapas de bits personalizados para mostrar el estado de las señales. TComLed se actualiza automáticamente ante los cambios de la señal de control, pero para que se muestre hay que refrescar la ventana de la aplicación. Está por determinar cual es el problema de fondo y como resolverlo.

La lista de propiedades es la siguiente:

- Propiedad **ComPort**: Selecciona a qué componente TComPort debe estar asociado. Esta propiedad siempre debe estar definida.
- Propiedad **Kind**: Establece el tipo de diseño del componente. En el caso de que queramos personalizarlo usaremos el valor ***lkCustom*** y además debemos proporcionar imágenes de ON y OFF en las propiedades ***GlyphON*** y ***GlyphOFF***.
- Propiedad **LedSignal**: Puede asociar el componente a un estado particular de la conexión serie.
- Propiedad **State**: Establece el estado ON/OFF del componente. Cuando el enlace está activo, esta propiedad es de sólo lectura.

Para la señal Ring, ver "Sobre la señal Ring".

TComTerminal

Este componente visual permite definir un área de visualización de caracteres enviados / recibidos en el enlace serie. Tiene un interés limitado en la aplicación final pero puede ser de gran ayuda en el desarrollo de un enlace serie. Puede utilizarse como consola ASCII tipo VT100

o VT52.

La lista de propiedades es la siguiente:

- Propiedad **ComPort**: Selecciona a que componente TComPort debe estar asociado. Esta propiedad siempre debe estar definida.
- Propiedad **WrapLines**: Define si la envoltura es automática al adelantar a la derecha.
- Propiedad **LocalEcho**: Especifica si los caracteres escritos en el teclado también se muestran en la consola.
- Propiedad **SendLF**: Establece la posibilidad de enviar un retorno de carro (CR, caráctecter ASCII 13) debe ir seguido de un carácter de nueva línea (LF, Line-Feed, ASCII 10).
- Propiedad **AppendLF**: Define si la recepción de un retorno de carro CR debe interpretarse como una combinación de caracteres CR LF.
- Propiedad **Force7bit**: Fuerza de la transmisión / recepción de caracteres de 7 bits. Todos los caracteres extendidos (8 bits) se reducirán a 7.
- Propiedades de **Columns** y **Rows**: Definen el tamaño de los caracteres de la consola.
- Propiedad **Connected**: Es equivalente a la propiedad Connected de los componentes asociados.
- Propiedad **Caret**: Define la forma del cursor de la consola.
- Propiedad **Emulation**: Define el tipo de emulación de la consola en el caso de recibir una secuencia de escape. Esta propiedad es especialmente útil cuando se utiliza el componente en lugar de una consola ASCII.
- Propiedad **ArrowsKeys**: Establece si las teclas de dirección debe ser enviada en carácter (akTerminal) o se utiliza para mover el cursor (akWindows).

Uso de TComPort

Puede ver como crear aplicaciones con Borland C++ Builder y TComPort en PIC-PC RS232 C++

Enumeración de los puertos

Antes de establecer el puerto serie a utilizar, es útil llamar al procedimiento EnumComPorts para enumerar los puertos serie en la máquina local. Nuestra aplicación puede asignar un miembro del resultado TStrings a la propiedad del puerto.

```
//---------------------------------------------------------
  EnumComPorts(ComboBox1->Items); //Se pasan los puertos al ComboBox.
  if( ComboBox1->Items->Count > 0) //Si hay items.
  {
    ComboBox1->ItemIndex=0; //Se elige el primer puerto disponible
    ComPort1->Port = ComboBox1->Items->Strings[ComboBox1->ItemIndex];
  }
//---------------------------------------------------------
```

Apertura del puerto

Se ha definido anteriormente, COM1, COM2, COM3 o COM4 en el Inspector de Objetos, (propiedad del Puerto).

```
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ComPort1->Connected=true; //Abrir el puerto (listo para comunicarse)
}
//---------------------------------------------------------
```

Otra forma:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  ComPort1->Open();
}
//---------------------------------------------------------------------------
```

El puerto ya está abierto y listo para comunicarse con un dispositivo externo.

Cierre del puerto

```
//---------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    ComPort1->Connected=false; //Cerrar el puerto ( el asunto de Comunicaciones
)
}
//---------------------------------------------------------
```

Otra forma:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
  ComPort1->Close();
}
//---------------------------------------------------------------------------
```

El puerto serie está cerrado, todos los intentos de lectura o escritura muestran un error.

Cambiar el nivel en las salidas RTS y DTR

Se pueden enviar señales simples todo/nada al circuito conectado al puerto del PC controlando las señales RTS y DTR, que son salidas en el PC. De esta manera podría por ejemplo controlarse un simple relé.

Por medio de RTS:

```
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```
    ComPort1->SetRTS(true);  // RTS = +15V
    ComPort1->SetRTS(false); // RTS = -15V
}
//---------------------------------------------------------
```

Por medio de DTR:

```
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    ComPort1->SetDTR(true);  //DTR = +15V
    ComPort1->SetDTR(false); //DTR = -15V
}
//---------------------------------------------------------
```

Estado de las entradas CTS, DSR y RLSD

Puede ser útil saber cuál es el estado en una patilla de entrada. Esto implica utilizar los eventos de estos pines . Veamos un ejemplo con CTS.

```
//---------------------------------------------------------
void __fastcall TForm1::ComPort1CTSChange(TObject *Sender, bool OnOff)
{
    if (OnOff == true)
        Button1->Caption = "Estado alto en CTS"; // CTS = +15V
    else
        Button1->Caption = "Estado bajo en CTS"; // CTS = -15V
}
//---------------------------------------------------------
```

Si el "caption" de Button1 muestra "Estado alto en CTS" es que el nivel que tiene aplicado es +15 V. Podemos hacer esto para los eventos DSR y RLSD (DCD).

Escribir una cadena de texto en TxD

Aquí se muestra un ejemplo de cómo enviar cadenas de caracteres ASCII por el puerto serie

```
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    AnsiString Frase;
    Frase = Edit1->Text;
    ComPort1->WriteStr(Frase); //Escribir toda la cadena "frase" en el puerto
serie
}
//---------------------------------------------------------
```

Escribir un valor en TxD

```
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    unsigned char tabla[7] = {'F', 'a', 'r', 'a', 'd', 'a', 'y'};
    ComPort1->Write(tabla, 7); //Escribe 7 bytes de "tabla" por el puerto
serie.
```

```
}
//---------------------------------------------------------
```

Lectura de una cadena de caracteres por RxD

Después de enviar una cadena de caracteres por la línea TxD , sería bueno saber cómo recibir datos. Para ello se utiliza el evento OnRxChar:

```
//---------------------------------------------------------
void __fastcall TForm1::ComPort1RxChar(TObject *Sender, int Count)
{
    AnsiString Frase2;
    ComPort1->ReadStr(Frase2, Count);
    //Leer los "Count" bytes presentes en el buffer de entrada y coloralos en
Frase2
}
//---------------------------------------------------------
```

La cadena recibida se almacena en el buffer de entrada del puerto, que por cierto es configurable en tamaño en el inspector de objetos, en la propiedad Buffer / Input e OutputSize. En este ejemplo la lectura es carácter por carácter, ReadStr borra el carácter del buffer y a continuación, pasa al siguiente y otra vez lo mismo. Una lectura puede realizarse así, también, fuera del evento OnRxChar .

Lectura de un valor por RxD

También en el evento OnRxChar:

```
//---------------------------------------------------------
void __fastcall TForm1::ComPort1RxChar(TObject *Sender, int Count)
{
    unsigned char *Buf = new unsigned char [Count];
    ComPort1->Read(Buf, Count);
    //Lee "Count" bytes presentes presentes en el buffer de entrada y los coloca
en "Buf"
    delete [] Buf;
    Buf = NULL;
}
//---------------------------------------------------------
```

La lectura también se puede hacer fuera de este evento.

Comprobar si hay errores en el puerto serie

Es posible, a través del evento OnError , saber si se ha producido un error en el puerto serie.

```
//---------------------------------------------------------
void __fastcall TForm1::ComPort1Error(TObject *Sender, TComErrors Errors)
{
    if (Errors.Contains(ceFrame))
    {
        // Error en la trama, mala velocidad de comunicación
    }
    if (Errors.Contains(ceOverrun))
    {
```

```
        // Error de desbordamiento. Los siguientes datos son erroneos.
    }
    // Utilizar "LastError" de  TComPort para continuar con la gestión de
errores
}
//---------------------------------------------------------
```

Número de bytes en el búfer de entrada

```
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    int NbrBits;
    NbrBits = ComPort1->InputCount();
}
//---------------------------------------------------------
```

Cada función de lectura Read, ReadStr etc... decrementa InputCount.

Vaciar los buffer de entrada y de salida

Podemos borrar el búfer, cuando los datos se consideren inservibles por las razones que sea.

```
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    ComPort1->ClearBuffer(true, false); // Buffer de entrada vacío , de salida
intacto
}
//---------------------------------------------------------
```

Esperar a un evento para realizar una acción

Durante un diálogo entre el PC y otro dispositivo, a veces, es necesario esperar una respuesta del dispositivo antes de enviarle un comando. Para ello disponemos del método WaitForEvent.

El uso de WaitForEvent es como sigue:

- Borramos todos los eventos de la propiedad de Eventos de TComPort . Esto implica que no se crea un hilo o subproceso.
- Abrimos el puerto serie.
- Declaramos el evento de activación.
- Llamamos al método WaitForEvent estableciendo el tiempo de espera en ms. Es posible sustituir el tiempo de espera por "WaitInfinite"
- Probamos si se ha producido el evento.

A continuación vemos lo dicho con un ejemplo de código:

```
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    ComPort1->Events = TComEvents(); // Eliminar todos los eventos
    ComPort1->Connected = true;

    TComEvents Ev;
```

```
    Ev << evRxChar;
    ComPort1->WaitForEvent (Ev, 0, 5000);
    // Espera un evento OnRxChar o 5 segundos antes de continuar

        if (Ev.Contains(evRxChar))
        {
            // Ejecutar una acción, un dato llegó
        }
        else
        {
            // Ejecutar otra acción. Los cinco segundos han transcurrido
        }

}
//---------------------------------------------------------
```

Cambiar la configuración del puerto

Una posibilidad es llamar al método ShowSetupDialog

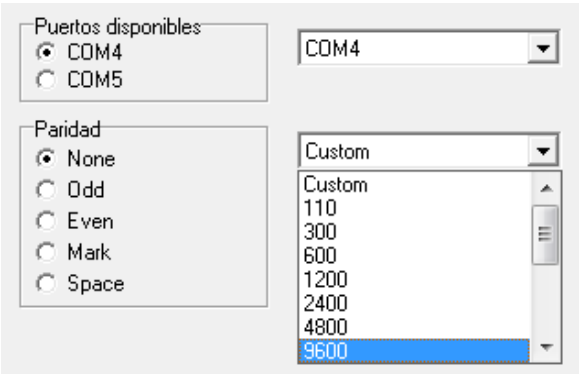
```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    ComPort1->ShowSetupDialog ();
}
//---------------------------------------------------------------------------
```

La segunda es utilizar, de la paleta de componentes, solapa CPortLib, uno o varios componentes TComComboBox, estableciendo en sus propiedades que aspecto de la configuración del puerto queremos que muestre:

- Propiedad "AutoApply " como **true**
- Propiedad "ComPort " apuntando al puerto que queremos configurar , ComPort1 , ComPort2, etc ...
- Propiedad "ComProperty " con el parámetro a configurar, cpPort para el puerto, cpBaudRate para la velocidad , etc ...

La misma técnica se puede utilizar para el componente TComRadioGroup.

Ejemplos:



Guardar la configuración del puerto en un archivo .INI

Al cierre de la solicitud se puede escribir.

```
//---------------------------------------------------------------------------
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
    ComPort1->StoreSettings (stIniFile, GetCurrentDir() + "\\configport.ini");
    ComPort2->StoreSettings (stIniFile, GetCurrentDir() + "\\configport.ini");
}
//---------------------------------------------------------------------------
```

Debe gestionarse la gestión de errores en caso de fallo de la escritura.

Leer la configuración del puerto desde un archivo .INI

Al crear la aplicación se puede escribir.

```
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    ComPort1->LoadSettings (stIniFile, GetCurrentDir() + "\\configport.ini");
    ComPort2->LoadSettings (stIniFile, GetCurrentDir() + "\\configport.ini");

    ComRadioGroup1->UpdateSettings (); // Si se ha colocado un TComRadioGroup
    // utilizar el méto explicado antes en "Cambiar la configuración del puerto"

}
//---------------------------------------------------------------------------
```

Podemos proceder de la misma manera para un TComComboBox .

Control de excepciones

Es posible controlar las excepciones causados por el componente TComPort.

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{

    AnsiString msg = "";

        try
        {
            ComPort1->Connected = true;
        }
        catch (EComPort &e)
        {
            switch (e.Code)
            {
                case CError_OpenFailed : msg = "El puerto serie ya está abierto"
; break ;
                case CError_WriteFailed : msg = "Error al escribir en el puerto
de serie" ; break ;
                // Ver la ayuda de  TComPort sobre "Error codes" para continuar
con la gestión de errores
            }
```

```
            ShowMessage(msg);
        }

}
//---------------------------------------------------------------------------
```

Información adicional

TComPort se muestra inactivo

Importante: Si el componente TComPort se muestra inactivo puede solucionarse colocando tres TComLed y asignando sus propiedades. Para cada uno de los tres TcomLed que coloquemos:

- Propiedad "ComPort" en el puerto adecuado, ComPort1, ComPort2, etc...
- Propiedad "LedSignal": uno para la apertura del puerto (IsConn), otro para la línea RxD (IsRx) y finalmente otro para TxD (IsTx).

Sobre la señal Ring

La señal Ring se trata de forma diferente a otras señales. TCustomComPort no informa si el estado de la señal es "activa" o "no activa". Sólo informa de la detección de la señal de Ring.

Por ejemplo, si utilizamos TComLed, hay una propiedad llamada RingDuration, que se utiliza de la siguiente manera; se detecta el Ring, State se establece en lsOn y después de RingDuration milisegundos, State se establece en lsOff.