Utilisation de TComPort sous Delphi 2005 (Win 32)

par Nono40

Date de publication : 17/02/2005

Dernière mise à jour : 07/06/2005

Utilisation des composants TComPort sous Delphi 2005. Principes de base d'utilisations de ces composants

I - Introduction
II - Installation des composants
    II-A - Installation du paquet d'exécution
    II-B - Installation du paquet de conception
    II-C - Ajouter le chemin dans les options
III - Présentation des composants ComPort
    III-A - TComPort
    III-B - TComDataPacket
    III-C - TComComboBox et TComRadioGroup
    III-D - TComLed
    III-E - TComTerminal
IV - Applications
    IV-A - Un terminal simple
    IV-B - Envoyer et recevoir des caractères avec le TComPort
    IV-C - Synchrone / Asynchrone
        IV-C-1 - Envoi synchrone
        IV-C-2 - Envoi asynchrone
        IV-C-3 - Réception synchrone
        IV-C-4 - Réception asynchrone
    IV-D - Réception de trames avec TComDataPacket
V - Conclusion

I - Introduction

L'objectif de ce cours est de montrer l'utilisation des composants TComport pour utiliser des liaisons séries dans une application. Il suppose que vous avez déjà une expérience de Delphi. Les exemples présentés ici utilisent Delphi 2005 mais ils sont applicables facilement aux versions antérieures de Delphi.

Il suppose de même que vous avez une connaissance du principe d'une liaison série, même limitée.

Ces composants peuvent être téléchargés ici : Comport Library

Ce document est prévu pour les versions Pro, Entreprise et Architecte.

Pour la version Personnel veuillez lire ceci en premier.

II - Installation des composants

Il n'y a pas de paquets prévus pour Delphi 2005. Comme le projet n'est plus suivi pour le moment, il n'y en aura sans doute pas avant un moment.

Cependant, comme les sources sont livrés, la mise à niveau des paquets prévus à l'origine pour Delphi 7 s'opère sans problème.

Il y a deux paquets à compiler et installer : paquet d'exécution et paquet de conception.

La méthode suivie sera directement celle donnée dans le fichier README.TXT situé dans le zip.

Dans un premier temps, il faut décompresser le fichier *source.zip* et le placer dans un répertoire de votre choix.

II-A - Installation du paquet d'exécution

Ouvrir le paquet CPortLib.dpk. A l'ouverture de ce paquet, Delphi vous propose le choix de la conversion. Choisissez Delphi pour Win32 :

Compiler le projet, ceci se passe sans aucun problème.

Puis à l'aide d'un clic droit dans le gestionnaire de projet, choisissez *Installer*.

Delphi va automatiquement placer le paquet d'exécution dans un répertoire accessible par toutes les applications.

II-B - Installation du paquet de conception

Ouvrir le paquet DsgnPortLib.dpk. A l'ouverture de ce paquet, Delphi vous propose le choix de la conversion. Choisissez de même Delphi pour Win32.

Si vous essayez de compiler le paquet tel-quel, Delphi va sortir une erreur sur l'unité **DsgnIntf**. Ceci vient du fait que la version 2005 n'est pas reconnue dans la compilation conditionnelle.

Ouvrez le fichier *source\CPort.inc* et ajoutez à la fin le code suivant :

```
  {$IFDEF VER170}     { Delphi 2005 }
    {$DEFINE DELPHI_4_OR_HIGHER}
    {$DEFINE DELPHI_5_OR_HIGHER}
    {$DEFINE DELPHI_6_OR_HIGHER}
    {$DEFINE DELPHI_7_OR_HIGHER}
    {$DEFINE DELPHI_2005_OR_HIGHER}
    {$DEFINE DELPHI_2005}
    {$IFDEF BCBNOTDELPHI}
      {$ObjExportAll On}
```

```
    {$ENDIF}
    {$WARN UNSAFE_TYPE OFF}
    {$WARN UNSAFE_CODE OFF}
    {$WARN UNSAFE_CAST OFF}
  {$ENDIF}
```

Vous pouvez maintenant compiler et installer le paquet comme précédemment. Delphi va alors recenser les composants du paquet :

II-C - Ajouter le chemin dans les options

Afin que Delphi puisse compiler vos futurs projets avec ComPort, il faut définir le chemin dans les options de Delphi.

Allez dans *Outils->options*, puis *Options D'environnement->Options Delphi->Bibliothèque Win32*.

Et enfin ajouter le chemin d'accès aux sources du paquet :

III - Présentation des composants ComPort

Le paquet de conception va installer une nouvelle palette de composants comportant :

- **TComPort** : composant d'accès au port série lui-même.
- **TComDataPacket** : Composant de gestion de trames reçues.
- **TComComboBox** : Composant visuel d'aide à la configuration du TComport
- **TComRadioGroup** : Composant visuel d'aide à la configuration du TComport
- **TComLed** : Composant permettant de visualiser l'activité du composant ComPort associé
- **TComTerminal** : Composant de visualisation des trames de la liaison série

Nous allons maintenant détailler l'utilisation de chacun de ces composants.

III-A - TComPort

C'est le composant principal de ce paquet. C'est lui qui va permettre de configurer et de gérer le port série.

Ce composant comporte toutes les propriétés de configuration de la liaison série. Je ne vais pas toutes les présenter dans le détail, mais seulement les plus importantes et surtout celles que l'on modifie le plus souvent dans la pratique.

- Propriété **Port**

  Cette propriété définit le port à utiliser sur le PC. Dans la liste déroulante seuls les ports existant physiquement apparaissent.

  Notez que ComPort est basé sur les APIs Windows, donc il supporte parfaitement les drivers passerelle, style USB--RS232.
- Propriété **BaudRate**

  Définit la vitesse de la liaison série en Bauds.
- Propriété **DataBits**

  Définit le nombre de bits de chaque caractère émis/reçu sur la liaison série.
- Propriété **Parity**

  Définit la type de bit de parité de chaque caractère. Il est possible ici de définir si le contrôle de parité doit être effectué en lecture, et dans le cas où il est effectué de remplacer dans la trame reçue les caractères ayant un défaut de parité par un caractère spécifique.

  Pour les essais, je vous conseille de ne pas activer le contrôle de parité, il n'est pas indispensable au bon fonctionnement et peut facilement être ajouté par la suite si vous voulez renforcer le contrôle.

De toute manière, dans le cas d'envoi/réception de trames il est préférable de gérer un checksum plus évolué de style CRC.

- Propriété **StopsBits**

  Nombre de bits de stops de chaque caractère.
- Propriété **FlowControl**

  Cet ensemble de propriétés permet de définir la gestion du flux de caractères sur la liaison série.

  Par expérience et dans la mesure du possible il est préférable de ne pas utiliser le contrôle de flux, car cela ne fait que compliquer le gestion et le cablâge de la liaison. Il y a cependant des exceptions, comme la gestion d'un modem ou la gestion d'un convertisseur RS232-RS485 multipoints.

  Dans les premiers exemples donnés dans ce cours, aucun contrôle de flux ne sera utilisé.
- Propriété **Timeouts**

  Cet ensemble de propriétés permet de définir les différents timeouts en émission et en réception. Leur utilisation dépend des applications.
- Propriété **Events**

  Permet de définir quels événements liés à l'activité de la liaison série sont utilisés.

  Par défaut ils sont tous activés, mais vous pouvez supprimer tous ceux qui ne vous sont pas utiles afin de ne pas surcharger le dialogue avec le TComport.
- Propriété **EventChar**

  Cette propriété permet de définir le caractère déclenchant **OnRxChar** quand il est reçu. Il faut bien sur que l'événement soit activé dans la propriété **Events**.
- Propriété **Connected**

  Cette propriété permet d'ouvrir/fermer la liaison série et aussi d'en vérifier son état.

III-B - TComDataPacket

Ce composant permet de gérer de façon très simple la réception de trames de caractères, ou d'une certaine longueur de caractères.

- Propriété **Port**

  Permet de choisir à quel composant TComPort celui-ci doit être associé.

  Cette propriété doit impérativement être définie.
- Propriété **Size**

  Longueur de la trame à recevoir, si la longueur n'est pas fixe, positionner cette propriété à zéro.
- Propriété **StartString**

Chaîne définissant le début de la trame.

- Propriété **StopString**

Chaîne définissant la fin de la trame.

- Propriété **CaseInsensitive**

Permet de choisir la sensibilité à la casse sur les caractères reçus.

- Propriété **IncludeStrings**

Permet de définir si les chaînes **StartString** et **StopString** doivent être incluses dans la trame ou non.

III-C - TComComboBox et TComRadioGroup

Ces composants permettent de configurer de façon visuelle la liaison série. Ils seront utilisés pour créer une interface de configuration du port COM en quelques clics.

La liste des propriétés est la suivante :

- Propriété **ComPort**

Permet de choisir à quel composant TComPort celui-ci doit être associé.

Cette propriété doit impérativement être définie.

- Propriété **ComProperty**

Permet de définir quel paramètre de la liaison sera affecté par le composant. Comme par exemple la vitesse, la parité, etc.

- Propriété **AutoApply**

Permet de définir si les modifications apportées dans la sélection sont automatiquement portées sur la configuration de la liaison.

Si cette propriété est à True, tout changement sera directement reporté dans la configuration du port série du TComport associé.

Si cette propriété est à False, il faut appeler par code la méthode ApplySettings de ce composant pour que les modifications soient effectives.

III-D - TComLed

Composant permettant de réaliser une animation visuelle d'activité de la liaison série.

La liste des propriétés est la suivante :

- Propriété **ComPort**

Permet de choisir à quel composant TComPort celui-ci doit être associé.

Cette propriété doit impérativement être définie.

- Propriété **King**

Permet de définir le type de dessin du composant. Dans le cas de **lkCustom**, vous devez donner des images pour les états ON et OFF dans les propriétés **GlyphON** et **GlyphOFF**.

- Propriété **LedSignal**

Permet d'associer le composant à un état particulier de la liaison série.

- Propriété **State**

Permet de définir, mais surtout de connaître l'état ON/OFF du composant. Quand la liaison est active, cette propriété est en lecture seule.

III-E - TComTerminal

Ce composant visuel permet de définir une zone de visualisation des caractères émis/reçus sur la liaison série.

D'un intérêt limité en production, il peut être d'un grand secours lors de la mise au point d'une liaison série.

Une autre application est la simulation d'une console ASCII de type VT100 ou VT52.

La liste des propriétés est la suivante :

- Propriété **ComPort**

Permet de choisir à quel composant TComPort celui-ci doit être associé.

Cette propriété doit impérativement être définie.

- Propriété **WrapLines**

Définit si le retour à la ligne est automatique en cas de dépassement sur la droite.

- Propriété **LocalEcho**

Définit si les caractères tapés au clavier sont aussi affichés sur la console.

- Propriété **SendLF**

Définit si l'envoi d'un retour chariot (touche entrée, caractère #13) doit être suivit d'un caractère saut de ligne (Line-Feed, #10).

- Propriété **AppendLF**

Définit si la réception d'un retour chariot (caractère #13) doit être interprétée comme un caractère composé CR LF (ajout d'un LF, #10).

- Propriété **Force7bit**

Force l'émission/réception de caractères sur 7 bits seulement. Tous les caractères étendus (8 bits) seront réduits à 7.

- Propriétés **Columns** et **Rows**

Définissent la taille en caractères de la console.

- Propriété **Connected**

Est équivalente à la propriété Connected du composant ComPort associé. La modification de l'une entraîne la modification de l'autre.

- Propriété **Caret**

Définit la forme du caret de la console.

- Propriété **Emulation**

Définit le type d'émulation de la console dans le cas de réception d'une séquence d'échappement.

Cette propriété est surtout utile dans le cas d'utilisation de ce composant en remplacement d'une console ASCII.

- Propriété **ArrowsKeys**

Définit si les touches de directions doivent être envoyés sous forme de caractère (**akTerminal**) ou utilisées pour déplacer le caret (**akWindows**).

IV - Applications

IV-A - Un terminal simple

Afin de montrer la simplicité de gestion d'un port série avec ComPort, nous allons réaliser un terminal simple comme celui d'HyperTerminal.

Bien que simpliste, ce programme pourra être régulièrement utilisé pour "écouter" un port série et en afficher les trames.

Créez une nouvelle application VCL Win32 et placez un composant TComport

Placez un TPanel en haut de fiche et placez-y cinq composants TComComboBox pour les propriétés Port, Baudrate, DataBits, Parity et Stopbits du composant ComPort. Les propriétés **AutoApply** seront mises à **True**.

Placez de même trois leds pour visualiser les états "connecté", "RX" et "TX"

Placez en dessous du panneau un TComTerminal dont la propriété **Emulation** sera fixée à **teNone**. Placez un bouton dont le code est :

```
    Comport1.Connected := Not Comport1.Connected;
```

Vous devez obtenir une fiche ressemblant à :

Voila, c'est tout !

A l'exécution, choisissez un port et un format de données puis cliquez sur connecter.

Tous les caratères reçus seront affichés, et si la console détient le focus, tous les caractère tapés au clavier seront envoyés sur la liaison série.

Pour tester cette application vous pouvez vous servir d'HyperTerminal sur un autre port série ou un autre PC, mais vous pouvez aussi utiliser la même application lancée deux fois sur le même PC ou sur chacun des deux PCs.

Le cable utilisé pour ce test sera un simple câble trois fils croisé :

IV-B - Envoyer et recevoir des caractères avec le TComPort

Ici nous allons réaliser un programme de capture/envoi de caratères sur la liaison série. C'est un exemple simple d'utilisation du TComPort et de ses événements.

A partir du premier exemple, supprimez le TComTerminal et ajoutez les composants tels que ;

Ajouter le code sur les événements AfterOpen et OnRxChar, ainsi que sur le bouton btnEnvoyer.

```
procedure TForm1.ComPort1AfterOpen(Sender: TObject);
begin
  btnEnvoyer.Enabled := True;
  btnConnecter.Enabled := False;
end;

procedure TForm1.ComPort1RxChar(Sender: TObject; Count: Integer);
Var Chaine:String;
begin
  ComPort1.ReadStr(Chaine,Count);
  Memo1.Lines.Text := Memo1.Lines.Text + Chaine;
end;

procedure TForm1.btnEnvoyerClick(Sender: TObject);
begin
  If eCRLF.Checked
    Then ComPort1.WriteStr(eEnvoi.Text+#13#10)
    Else ComPort1.WriteStr(eEnvoi.Text);
end;
```

Notez ici que l'événement **OnRxChar** est activé sur chaque réception de caractère. La valeur de **Count** donnée lors de l'événement est le nombre de caractères actuellement disponibles sur le port série. Il ne faut pas considérer que l'on reçoit un événement par caractère, et donc il faut traiter l'ensemble des caractères en une seule fois.

La fonction **ReadStr** permet ici de lire de façon simple les caractères disponibles et de les ajouter au Memo.

L'envoi d'une chaine est ici aussi assez simple à l'aide de la fonction **WriteStr**.

*Les procédures utilisées (**ReadStr** et **WriteStr**) sont bloquantes, c'est-à-dire que l'exécution ne reprend qu'après l'envoi/réception du nombre de caractères voulus.*

Pour essayer vous pouvez procéder de la même manière que pour le premier exemple.

IV-C - Synchrone / Asynchrone

Le composant dispose de procédures d'émission/réception synchrone et asynchrone.

Les procédures Synchrones bloquent l'exécution du programme jusqu'à la fin du traitement. Ce sont les méthodes les plus simples à utiliser. Mais elles ont le gros inconvénient de laisser croire que l'application ne réagit pas bien. Surtout lors de réceptions de chaînes.

Les procédures Asynchrones ne bloquent pas l'exécution du programme, mais en contrepartie leur gestion est un peu plus compliquée. Ces méthodes sont néanmoins indispensables si l'application doit gérer autre chose que la liaison série ou si vous gérez plusieurs liaisons séries simultannément.

Le troisième programme montre l'utilisation des deux méthodes en lecture et en écriture.

IV-C-1 - Envoi synchrone

C'est très simple à mettre en oeuvre...

```
procedure TForm1.btnEnvoyerSyncClick(Sender: TObject);
begin
  btnEnvoyerSync.Enabled := False;
  ComPort1.WriteStr(eEnvoiSync.Text);
  btnEnvoyerSync.Enabled := True;
end;
```

Il suffit d'appeler la méthode **WriteStr()** pour envoyer une chaîne.

IV-C-2 - Envoi asynchrone

Pour l'envoi asynchrone, il faut utiliser une structure intermédiaire permettant de garder une trace de l'envoi en cours.

```
Var
  // Structure de trace de l'envoi
  WriteASyncPtr:PAsync = Nil;

procedure TForm1.btnEnvoyerASyncClick(Sender: TObject);
begin
  btnEnvoyerASync.Enabled := False;
  // Mise à jour de la structure d'envoi
  InitAsync(WriteAsyncPtr);
  // Envoi différé
  ComPort1.WriteStrAsync(eEnvoiSync.Text,WriteAsyncPtr);
end;

procedure TForm1.ComPort1TxEmpty(Sender: TObject);
begin
  // Quand la file d'envoi est vide on teste si un envoi est en cours
  If Assigned(WriteAsyncPtr) Then
  Begin
    // On supprime la structure de trace
    DoneAsync(WriteAsyncPtr);
    btnEnvoyerASync.Enabled := True;
  End;
end;
```

Dans cet exemple on utilise l'événement **OnTxEmpty** du TComport pour savoir quand l'envoi est terminé. En effet cet événement se produit juste après l'envoi du dernier caractère sur le port série.

IV-C-3 - Réception synchrone

La réception synchrone est basée sur la méthode **ReadStr**. Cette méthode attend deux paramètres : une chaîne servant de stockage, et une longueur à recevoir. Elle finit si le nombre de caractères reçus est suffisant ou si un time-out intervient. Il convient donc de bien mettre à jour le timeout avant de lancer cette méthode.

```
procedure TForm1.btnRecevoirSyncClick(Sender: TObject);
Var
  Chaine:String;
  Long:Integer;
begin
```

```
    // MAJ composants
    btnRecevoirSync.Enabled := False;
    Memo1.Lines.Add('Début envoi synchrone');
    // Mise à jour du time out demandé
    Comport1.Timeouts.ReadTotalConstant := seTimeOut.Value * 1000;
    // Supression de ce qui traine dans le buffer d'entrée
    Comport1.ClearBuffer(True,False);
    // Lecture bloquante de la longueur demandée
    Long:=Comport1.ReadStr(Chaine,seLongueur.Value);
    // MAJ composants
    Memo1.Lines.Add('Fin envoi synchrone '+IntToStr(Long)+' caractère(s)
reçu(s):'+Chaine);
    btnRecevoirSync.Enabled := True;
end;
```

Notez que dans cet exemple on n'utilise que le time-out global, mais bien sûr on peut combiner différents types de time-out suivant le résultat voulu.

De même, on prend soin de vider le buffer de réception avant la demande afin de purger les caractères qui pourraient éventuellement s'y trouver.

IV-C-4 - Réception asynchrone

De même que pour l'émission asynchrone, il faut utiliser une structure de trace afin de connaitre l'état de la réception en cours.

```
Var
  ReadASyncPtr:PAsync = Nil;

procedure TForm1.btnRecevoirAsyncClick(Sender: TObject);
Var
  Chaine:String;
  Long:Integer;
begin
  // MAJ composants
  btnRecevoirASync.Enabled := False;
  Memo1.Lines.Add('Début envoi asynchrone');
  // Mise à jour du time out demandé
  Comport1.Timeouts.ReadTotalConstant := seTimeOut.Value * 1000;
  // Supression de ce qui traine dans le buffer d'entrée
  Comport1.ClearBuffer(True,False);
  // Init de la structure de suivi
  InitAsync(ReadAsyncPtr);
  Try
    // Lecture non-bloquante de la longueur demandée
    // Attention, la chaine doit rester accessible hors de l'événement...
    Chaine:='';
    Comport1.ReadStrAsync(Chaine,seLongueur.Value,ReadAsyncPtr);
    Repeat
      // Ici on n'est pas bloqué, on peut donc traiter les messages Windows
      Application.ProcessMessages;
      // Attente de la fin de traitement
    Until Comport1.IsAsyncCompleted(ReadAsyncPtr);
    // récupération du résultat de la lecture de la chaine
    Long:= Comport1.WaitForAsync(ReadAsyncPtr);
    SetLength(Chaine,Long);
  Finally
    DoneAsync(ReadAsyncPtr);
  End;
  // MAJ composants
  Memo1.Lines.Add('Fin envoi asynchrone '+IntToStr(Long)+' caractère(s)
reçu(s):'+Chaine);
  btnRecevoirASync.Enabled := True;
end;
```

Pour gérer la structure de trace, on dispose de deux méthodes du TComPort.

IsAsyncCompleted() permet de savoir si la réception en cours est terminée ou non. Elle retourne True si le nombre de caractères demandés est bien reçu ou si un time-out intervient.

WaitForAsync() permet d'attendre la fin de la réception. Cette méthode est bloquante, il convient donc de ne l'appeler que si IsAsyncCompleted a retourné True avant. Elle permet en outre de connaître le nombre de caractères réellement reçus.

A noter qu'il y a plusieurs façons d'implémenter cette réception asynchrone. L'exemple donné ici est basé sur une boucle d'attente avec un appel de ProcessMessages afin de gérer la file des messages windows. C'est la solution la plus simple pour cet exemple. Par contre, au niveau de l'utilisation CPU ce n'est pas une bonne méthode.

Une autre solution consiste à tester la structure de trace dans un timer. Il faut juste penser dans ce cas que le chaîne passée en paramètre de ReadStrAsync doit être accessible en permanence pendant la réception.

IV-D - Réception de trames avec TComDataPacket

Dans le cas de réception de paquets, vous pouvez utiliser de façon très simple le composant **TComDataPacket**. Il permet de gérer de façon événementielle l'arrivée de trames particulières sur le port série.

Il suffit de le lier à un composant TComPort actif et de définir ses propriétés. L'événement principal de ce composant OnPacket sera appelé à chaque fois qu'une trame respectant les conditions sera reçue.

Il n'y a pas de code à écrire pour effectuer la gestion des paquets. Essayez l'exemple 4 donné dans le fichier joint afin de bien comprendre l'utilisation des propriétés de ce composant. Vous pouvez par exemple lancer sur un autre PC l'exemple 2 afin d'envoyer des trames et voir le résultat.

Il est possible aussi de connaître tous les caractères ignorés par le ComDataPacket. L'événement OnDiscard permet d'être averti de toutes les caractères rejetés dans le tri du paquet.

V - Conclusion

La gestion d'une liaison série à l'aide des composants TComPort n'est donc pas si compliquée que ça. Notez que ce cours n'a présenté que les mécanismes simples de ces composants. Suivant vos besoins vous aurez sans doute besoin de les pousser un peu plus loin...

Fichiers sources de cet article :

Miroir 1 : Sources des exemples [22Ko]

Dans le cas où le miroir 1 ne fonctionne pas :

Miroir 2 : Sources des exemples [22Ko]

Version PDF de cet article :

Miroir 1 : Version PDF

Dans le cas où le miroir 1 ne fonctionne pas :

Miroir 2 : Version PDF

Merci à Olivier Lance et Laurent Dardenne pour leurs remarques et la correction orthographique.