

# Impression de Code-barres avec Quick-Report

par Bruno Guérangé ([nono40.developpez.com](http://nono40.developpez.com))

Date de publication : 20/04/2008

Dernière mise à jour :

Cet article présente comment créer un contrôle Quick-Report personnalisé pour l'impression de codes-barres.

- I - Introduction
- II - Dessin des codes-barres
- III - Intégration dans Quick-Report
  - III-A - Introduction
  - III-B - Création du composant TQRCodeBarre
  - III-C - Affichage en mode conception
  - III-D - Impression dans un rapport
  - III-E - Exportation dans un rapport non imprimé
- V - Conclusion

## I - Introduction

L'utilité des codes-barres n'est plus à démontrer aujourd'hui. Il permet une identification fiable et une relecture automatisée.

Nous allons donc voir comment les intégrer dans Quick Report sous forme d'un contrôle Quick-Report personnalisé.

## II - Dessin des codes-barres

Pour dessiner les codes-barres, nous allons utiliser le composant AsBarcode d'Andreas Schmidt.

Ce composant permet la gestion d'un grand nombre de types de code-barres et de les dessiner sur un canevas.

Les propriétés importantes sont les suivantes:

### Propriétés de définition du type de Code

- **Typ** : Propriété principale définissant le codage utilisé EAN13, Code128, etc
- **Checksum** : Propriété définissant si le checksum doit être ajouté au code-barres. Elle n'est utilisable que pour les code-barres contenant un checksum optionnel.
- **ChecksumMethod** : Propriété définissant si le format du checksum doit être ajouté au code-barres. Un seul type est supporté pour le moment.

### Propriétés d'affichage de code

- **Color** : Couleur de fond (barres claires) du code-barres.
- **ColorBar** : Couleur des barres. Je vous conseille de rester en noir sur fond blanc afin que le contraste soit au maximum entre les bandes claires et foncées.
- **Modul** : Nombre de points d'affichage pour le dessin d'une barre de largeur 1. Cette propriété est mise à jour en fonction de la largeur du code-barres. Il n'est donc pas utile de la modifier directement.
- **ShowText** : Définit si le code doit être affiché en superposition des barres.
- **ShowTextFont** : Définit la police d'affichage du code affiché en clair
- **ShowTextPosition** : Définit la position d'affichage du code par rapport au reste du dessin.
- **Angle** : Angle de dessin du code-barres de 0 à 360 degrés. 0 Correspond à un code-barres horizontal écrit de gauche à droite.


Bien que ce soit un composant non visuel, vous trouverez des propriétés de position et de taille. Ces propriétés sont utiles pour le dessin du code-barres. Il faut les mettre à jour afin qu'il soit dessiné sur le canevas à la bonne position et à la bonne taille.

Il n'y a que très peu de méthodes sur le composant. La plus utile est la méthode **DrawBarcode** dessinant le code-barres sur un canevas passé en paramètre. Voici un exemple d'appel pour le dessin du code sur un TImage en laissant 10 pixels de marge autour du dessin.

```
procedure TForm1.btnCodeClick(Sender: TObject);
begin
  // Exemple d'utilisation d'un AsBarcode
  AsBarcode1.Text := eCode.Text;

  // On met à jour la taille voulue, ici en fonction
  // de la taille du TImage de destination
  AsBarcode1.Top      := 10;
  AsBarcode1.Left    := 10;
  AsBarcode1.Width   := imgCode.Width-20;
  AsBarcode1.Height  := imgCode.Height-20;

  // On dessine le code-barres
  AsBarcode1.DrawBarcode(imgCode.Canvas);
end;
```

 Vous noterez que la marge de droite est souvent de taille supérieure à ce qui est voulu, le composant dessinant sur une largeur souvent plus réduite que celle demandée.

*Ceci est du au principe de dessin : en fonction de la largeur demandée, il calcule la taille de la barre de largeur 1 afin que le code-barres total tienne dans la largeur demandée. Il est possible de connaître la taille réelle du code dessiné à l'aide des propriétés en lecture seules **CanvasWidth** et **CanvasHeight**.*

Voici un exemple du résultat obtenu :

## III - Intégration dans Quick-Report

### III-A - Introduction

La méthode présentée ici est basée sur la version 4.07 de Quick-Report Professionnel et testé sous Delphi 2007 pour Win 32.

Tous les composants visuels déposés sur les bandes d'un état QuickReport descendent de la classe TQRPrintable. Ce composant introduit tous les comportements du composant pour son affichage en mode conception et son impression en mode rapport.

Un composant imprimable doit savoir s'afficher dans l'IDE afin de faciliter la mise en page, de s'imprimer de la manière la plus fidèle possible et enfin de s'exporter quand le rapport parent est exporté, par exemple en format PDF.



*Afin que l'impression des codes soient la plus précise possible, je vous conseille de pas utiliser une imprimante jet-d'encre. Surtout si les codes-barres doivent être petits sur la page.*

### III-B - Création du composant TQRCodeBarre

Nous allons donc dériver le nouveau composant du **TQRPrintable**. Ce composant va encapsuler un composant **AsBarcode** afin d'intégrer la gestion des codes-barres.

Le prototype du composant est alors le suivant :

```
TQRCodeBarre = class(TQRprintable)
private
  { Déclarations privées }
  FAsBarcode:TAsBarcode;
protected
  { Déclarations protégées }
public
  { Déclarations publiques }
  constructor Create(AOwner :TComponent); override;
  destructor Destroy; override;
published
  { Déclarations publiées }
end;
...
constructor TQRCodeBarre.Create(AOwner: TComponent);
begin
  inherited;
  FAsBarcode := TASBarcode.Create(Self);
  FAsBarcode.Height := 50;
  FAsBarcode.Width := 100;
  FImageID := 0;
end;

destructor TQRCodeBarre.Destroy;
begin
  fAsBarcode.free;
  inherited;
end;
```

Les propriétés utiles de définition du type de code et de son format d'affichage seront remontées au niveau du TQRCodeBarre afin d'être accessibles :

```
// Propriétés sotckées dans le composant AsBarcode
// reportées dans le composant QR pour accès.
Property BarAngle:Double           Read GetAngle           Write SetAngle;
Property BarChecksum:Boolean       Read GetChecksum        Write SetChecksum;
Property BarChecksumMethod:TChecksumMethod Read GetChecksumMethod Write
SetChecksumMethod;
Property BarColor:TColor           Read GetColor           Write SetColor;
Property BarColorBar:TColor        Read GetColorBar        Write SetColorBar;
Property BarModul:Integer           Read GetModul           Write SetModul;
Property BarRatio:Double           Read GetRatio           Write SetRatio;
Property BarShowText:TBarcodeOption Read GetShowText        Write SetShowText;
Property BarShowTextPosition:TShowTextPosition Read GetShowTextPosition Write
SetShowTextPosition;
Property BarText:String            Read GetText            Write SetText;
Property BarTyp:TBarcodeType       Read GetTyp              Write SetTyp;
```

Ces propriétés sont directement lues et écrites dans le composant AsBarcode sous-jacent. Comme par exemple la propriété **Text** :

```
function TQRCodeBarre.GetText: String;
begin
    Result := fAsBarcode.Text;
end;

procedure TQRCodeBarre.SetText(const Value: String);
begin
    fAsBarcode.Text := Value;
    Invalidate;
end;
```

Notez qu'à partir de là, les propriétés utiles sont en place, mais pour le moment il n'affiche rien et n'imprime rien. Dans la suite nous allons voir comment ce point sera changé.

### III-C - Affichage en mode conception

Ici, c'est très classique pour ceux qui ont déjà construit des composants visuels.

Comme tous les contrôles il suffit de surcharger la méthode virtuelle **paint** afin de répondre aux demandes de tracé.

Nous avons vu dans le chapitre II que le composant AsBarcode contient une procédure pour dessiner le code à barre sur un canevas. Or c'est précisément ce qu'il faut faire dans la méthode **Paint**. Nous allons donc demander simplement au composant AsBarcode de se dessiner à notre place !

```
protected
    { Déclarations protégées }
    procedure Paint; override;
    ...

procedure TQRCodeBarre.Paint;
begin
    inherited;
    FAsBarcode.Width := Width;
    FAsBarcode.Height := Height;
    FAsBarcode.DrawBarcode( Canvas );
```

```
end;
```

On ne peut plus simple pour afficher le code-barres en conception :

## Impression codes barres

3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

codes barres



[Nom]

[Racine]

[Identification]

[Nom]

[Racine]



reb de Nono40

### III-D - Impression dans un rapport

L'impression dans un rapport s'effectue de la même manière que l'affichage. Le composant TQRPrintable introduit la méthode **Print(OfsX, OfsY : integer); virtual**; qui est la méthode appelée quand le composant doit s'imprimer.

La différence est que pour l'affichage dans l'IDE nous utilisons un canevas **TCanvas** classique, ici il faut utiliser l'objet **QRPrinter** qui est une propriété du **TQRPrintable** et représente l'imprimante Quick-Report utilisée pour le rapport. Ce composant contient le canevas à utiliser pour imprimer.

Avant d'imprimer le composant, il faut le dimensionner à la taille de sortie sur l'imprimante. Vous avez certainement remarqué que la résolution d'une imprimante est bien supérieure à celle d'un écran. Il est donc important de dimensionner le composant AsBarcode à la taille voulue sans avoir à utiliser de **StretchDraw**, le fait d'imprimer le composant AsBarcode dans sa taille nominale garantit un tracé de bonne qualité.

La méthode Print() comporte des arguments donnant la position XY du coin supérieure gauche de la zone d'impression de notre composant dans le rapport. Contrairement à **Paint** où le canevas est placé en (0,0) pour le composant, ici il faut tenir compte de cette position.

D'autre part l'objet **QRPrinter** fournit les fonctions adaptées pour déterminer en pixel la taille du composant imprimé.

La propriété **Size** contient les dimensions de notre composant sur le rapport.

Voici alors le code déterminant en pixels la taille du code-barres imprimé :


```
// Ajustement du composant AsBarcode aux dimensions
// finales obtenues sur l'impression
FAsBarcode.Left := QRPrinter.XPos(OfsX + Size.Left);
FAsBarcode.Top := QRPrinter.YPos(OfsY + Size.Top);
FAsBarcode.Width := QRPrinter.XPos(OfsX + Size.Width + Size.Left) - QRPrinter.XPos(OfsX +
Size.Left);
FAsBarcode.Height := QRPrinter.YSize(Size.Height);
```


Une fois les dimensions en place, on dessine le code-barres comme on le fait sur l'écran :

```
protected
{ Déclarations protégées }
procedure Print(OfsX, OfsY : integer); override;
...

procedure TQRCodeBarre.Print(OfsX, OfsY : integer);
begin
  if ParentReport.FinalPass and IsEnabled then
    begin
      // Ajustement du composant AsBarcode aux dimensions
      // finales obtenues sur l'impression
      FAsBarcode.Left := QRPrinter.XPos(OfsX + Size.Left);
      FAsBarcode.Top := QRPrinter.YPos(OfsY + Size.Top);
      FAsBarcode.Width := QRPrinter.XPos(OfsX + Size.Width + Size.Left) - QRPrinter.XPos(OfsX +
      Size.Left);
      FAsBarcode.Height := QRPrinter.YSize(Size.Height);

      // On demande au composant de s'imprimer
      FAsBarcode.DrawBarcode( QRPrinter.Canvas );
    End;
  end;
```

 Notez le test de **ParentReport.FinalPass** et de **IsEnabled** avant le dessin. Ces tests évitent de dessiner le code-barres pendant les phases préparatoires (Quand il y a des composants liés à des champs texte et de taille automatique le premier appel permet de définir la taille, ici ce n'est pas notre cas) ou si le composant n'est pas actif.

 Vous remarquerez que nous n'appelons pas la méthode **Print** héritée, ceci pour éviter que le cadre autour du composant soit dessiné. Le cadre peut altérer les barres situées aux extrémités.

*Dans le cas d'un autre type de composant, vous pouvez appeler la méthode héritée pour assurer automatiquement le dessin du cadre.*

### III-E - Exportation dans un rapport non imprimé

Quick-report permet d'exporter un rapport dans divers formats reconnus comme le PDF, l'html ou le wmf.

Nous allons voir ici comment gérer cette fonctionnalité.

Pour exporter un rapport c'est aussi la méthode **print** qui est appelée, le code ne sera donc pas changé pour cela.

Mais, tous les formats de sortie ne supportent pas l'écriture sur un canevas. Il faut alors gérer l'exportation du composant dans le rapport en appelant les méthodes spécialisées dans l'exportation.

Notre composant étant plus proche d'une image que d'un texte, il faut appeler la méthode

### **TQRExportFilter.AcceptGraphic( Xoff, Yoff : extended; GControl : TControl).**

Cette méthode attendant un composant en paramètre, nous n'allons pas lui passer **Self** comme on pourrait le penser. La gestion interne de **AcceptGraphic** teste le type de composant pour gérer au mieux l'exportation en fonction du format.

Nous allons créer temporairement un composant **TQrGrImage** qui sera rempli avec notre code-barres et qui sera passé en paramètre.

```
// Création d'un TQrGrImage
GrImage := TQRGrImage.Create( Nil );
Try
  // On donne un nom unique au composant créé
  Inc( FImageID );
  GrImage.Name := 'QRCEXport'+IntToHex( FImageID, 8 );
  // On adapte le QrGrImage à la taille de l'AsBarcode
  GrImage.Picture.Bitmap.Width := FAsBarcode.Width;
  GrImage.Picture.Bitmap.Height := FAsBarcode.Height;
  GrImage.AutoSize := True;
  // On demande à l'AsBarcode de se dessiner sur le composant
  FAsBarcode.DrawBarcode( GrImage.Picture.Bitmap.canvas );
  // Et c'est le composant que l'on exporte
  TQRExportFilter( ParentReport.ExportFilter ).acceptgraphic(
    QRPrinter.XPos( ofsX + self.Size.Left ),
    QRPrinter.YPos( ofsY + self.size.top ), GrImage );
Finally
  GrImage.Free;
End;
```

Vous remarquerez dans le code ci-dessus que l'on donne un nom unique au composant, cette astuce est utilisée par le filtre HTML pour créer les images avec des noms différents. Chaque image exportée doit avoir un nom qui lui est propre afin d'être incluse dans le rapport.

La variable **FImageID** doit être globale, elle ne peut pas être une propriété car le composant est créé/détruit dynamiquement en fonction du nombre de ligne du rapport.

Pour initialiser cette variable en début de rapport, nous allons surcharger la méthode **Prepare** du **TQRPrintable**. **Prepare** est toujours appelé avant le début de l'impression ou exportation.

```
protected
{ Déclarations protégées }
procedure Prepare; override;
...
procedure TQRCodeBarre.Prepare;
begin
  inherited;
  FImageID:=0;
end;
```

## V - Conclusion

Nous aurons vu comment créer un composant visuel Quick-Report, ce n'est guère plus compliqué qu'un composant visuel standard. Libre à vous de créer les composants répondant à votre besoin.

Le paquet contenant les sources du composant ainsi que les sources du composant AsBarcode sont disponibles ici [QrCodeBarre.zip](#) ( [Miroir](#) )

En ce qui concerne la licence d'utilisation et de diffusion du composant AsBarcode, veuillez lire l'entête du fichier Barcode.pas.

Je tiens à remercier Lut Mentz de **QBS Software**

ainsi que **Rayek** et **Pedro** pour la relecture et les corrections.

