Formation C# – Delphi.NET – Delphi Win32
Développement & Sous-traitance

Imprimer sous .NET

Une fois que les données sont gérées, distribuées, traitées, transmises et stockées, il subsiste encore un besoin non satisfait : les imprimer. Le framework .NET propose des classes permettant de concevoir rapidement des impressions simples, mais il faut convenir que ce processus non visuel peut devenir fastidieux et qu'il vient un moment où l'on ressent la nécessité d'outils mieux adaptés. C'est ici qu'interviennent les générateurs d'état comme Rave Reports inclus avec Delphi ou d'autres comme Report Manager, *freeware* pour Delphi Win32 et .NET. Cet article vous propose un tour d'horizon de ces méthodes d'impression.

Imprimer via le framework

Le framework .NET offre plusieurs classes dédiées aux impressions et au contrôle des imprimantes. Avant de nous lancer dans des outils plus visuels, il semble intéressant de faire le tour des possibilités de .NET en la matière. Imprimer via le framework assure la portabilité des développements et limite la taille des applications puisque les outils sont déjà intégrés à la plate-forme. Dans certains cas, cela peut s'avérer une option à ne pas négliger.

Les principales classes

Le framework propose plusieurs classes pour créer des documents destinés à l'impression. Ces classes sont stockées dans l'espace de nom `System.Drawing.Printing`. En voici les principales :

- `PrintDocument` – Cette classe est généralement celle qu'on utilise systématiquement pour créer une impression. Une fois l'instance créée et configurée via ses propriétés, il suffit d'appeler sa méthode `Print` pour débuter l'impression. Durant celle-ci, l'objet déclenche l'événement `PrintPage` pour chacune des pages de l'état. En programmant un gestionnaire pour cet événement, on crée le contenu à imprimer de chaque page.

- `PrinterSettings` – Cette classe, utilisée comme propriété de la classe `PrintDocument`, contient l'ensemble des réglages du document à imprimer et notamment le choix de l'imprimante.

- `PageSettings` – Cette classe est aussi utilisée par `PrintDocument`, notamment par la propriété `DefaultPageSettings`, qui permet de paramétrer les réglages des pages comme l'orientation du papier

- `PrintPageEventArgs` – Cette classe contient les paramètres de l'événement `PrintPage`. Sont passés un rectangle de détourage ainsi qu'un objet `Graphics` décrivant la surface d'impression.

- `PrintEventArgs` – Cette classe contient les paramètres des événements `BeginPrint` et `EndPrint` de la classe `PrintDocument`. Dans les versions actuelles du framework, seule l'annulation de l'impression est disponible dans ces événements.

- `PrintDialog` – Présente le dialogue d'impression `PrintDlg` de Win32 (ce n'est donc qu'un *wrapper* .NET utilisant P/Invoke).

- `PageSetupDialog` – Présente le dialogue des propriétés des pages de Win32 `PageSetupDlg` (n'est donc aussi qu'un wrapper utilisant P/Invoke).

- `PrintPreviewControl` – Cette classe définit un module de prévisualisation pour les impressions créées avec `PrintDocument`.

- `PrintPreviewDialog` – Présente un dialogue affichant en mode prévisualisation un document produit par `PrintDocument`.

- `PrintController` – Cette classe est la classe mère des contrôleurs d'impression qui permettent de diriger et de traiter le flux d'impression généré par `PrintDocument`. À l'heure actuelle, le framework propose deux sous-classes : `DefaultPrintController`, qui envoie les pages sur une imprimante, et `PreviewPrintController`, qui envoie les pages dans une instance de `PrintPreviewControl` pour une prévisualisation.

Imprimer un texte

Pour mettre en évidence les mécanismes et la logique d'impression du framework, nous allons commencer par un exercice très simple : imprimer un fichier texte.

L'application de test sera de type VCL .NET, mais pourrait être aussi bien une Windows Form, cela ne fait aucune différence. Elle sera de toute façon d'un grand dénuement, puisque la fiche sera vide à l'exception d'un bouton permettant de lancer l'impression.

La déclaration de la fiche ressemble au code ci-dessous. Nous y avons placé les commentaires explicatifs et mis en gras les parties importantes.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms,
  Dialogs, System.ComponentModel, Borland.Vcl.StdCtrls,
  // les deux espaces de nom sont nécessaires
  System.Drawing, System.Drawing.Printing;
```

```
type
  TForm1 = class(TForm)
    btnPrint: TButton;
    procedure btnPrintClick(Sender: TObject);
  private
   // Gestionnaire de l'événement PrintPage de PrintDocument
    procedure Doc_PrintPage(
              sender:System.Object; ev:PrintPageEventArgs);
  public
    // la fonte d'impression
    PrintFont : System.Drawing.font;
    // le texte à imprimer
    FileToPrint : Tstrings;
    // l'objet document du framework
    Doc : System.Drawing.Printing.PrintDocument;
  end;
```

Comme vous le voyez ici, nous utilisons un objet de type `PrintDocument` qui implémente l'ensemble du mécanisme d'impression. Nous déclarons aussi un gestionnaire d'événement, qui sera appelé par ce dernier à chaque page à imprimer. Concernant le fichier à imprimer, nous implémenterons ici quelque chose de rudimentaire, puisque nous chargerons le fichier dans un `TStrings` en mémoire (qui sera instancié en `TStringList`).

Voici maintenant le code du bouton qui déclenche l'impression :

```
procedure TForm1.btnPrintClick(Sender: TObject);
begin
 // création de la liste de chaînes
 FileToPrint := TStringlist.Create;
 // chargement en mémoire du texte à imprimer
 FileToPrint.LoadFromFile('texte.txt');
 // creation de l'objet fonte pour le texte imprimé
 PrintFont := System.Drawing.Font.Create('Tahoma',9);
 // creation de l'objet d'impression
 Doc := PrintDocument.Create;
 // ajout du gestionnaire d'événement
 Include(Doc.PrintPage,Doc_PrintPage);
 // lancement de l'impression
 Doc.Print;
end;
```

Vous ne trouverez rien de bien compliqué dans le code ci-dessus. Le fichier à imprimer est stocké dans un `TStringList` en mémoire, puis nous créons les objets indispensables comme la fonte et l'objet `PrintDocument`. Enfin nous ajoutons le gestionnaire d'événement à la propriété `PrintPage` (événement) et nous lançons l'impression.

Le code le plus technique se trouve bien entendu dans le gestionnaire d'événement, puisque c'est lui qui a la charge de dessiner le contenu de chaque page.

```
procedure TForm1.Doc_PrintPage(sender: TObject;
                                ev: PrintPageEventArgs);
// les variables qui nous permettront de gérer la position
// du texte dans la page
var lpp, ypos : double;
    count : integer;
    leftmargin, topmargin : double;
    Line : string;
     // StringFormat est une classe gérant la presentation du texte,
     // son alignement, les substitutions en fonction des locales, etc
    sf : StringFormat;
begin
  count := 0 ;
  // nous prenons les marges par défaut de l'imprimante par défaut
  LeftMargin := ev.MarginBounds.Left;
  TopMargin  := ev.MarginBounds.Top;
  // nous calculons le nombre de lignes disponibles par page
  lpp := ev.MarginBounds.Height  /
                      PrintFont.GetHeight(ev.Graphics);
  sf := StringFormat.Create;
  // boucle d'impression
  While (count < lpp) and (FileToPrint.Count>0) do
   Begin
    // nous "soutirons" les lignes stockées une par une
    // en vidant le conteneur jusqu'à ce qu'il ne contienne plus rien
    line:=FileToPrint[0];
    FileToPrint.Delete(0);
    // calcul de la position Y de la ligne à imprimer
    yPos := TopMargin +
            (succ(count) * PrintFont.GetHeight(ev.Graphics));
    // impression de la ligne sur la surface graphique
    ev.Graphics.DrawString (line, PrintFont, Brushes.Black,
                            LeftMargin, yPos, sf);
    inc(count);
   end;
  // Y a-t-il d'autres pages à imprimer ?
  ev.HasMorePages := FileToPrint.Count>0;
end;
```

Le code ci-dessus est un peu plus complexe que le reste de l'application, dans le sens où il fait directement utilisation de classes du framework qui ne vous diront peut être rien pour l'instant. Le framework s'apprend comme s'apprenaient les API Win32, à force de coder…

Toutefois, il n'y a rien de bien compliqué non plus. Le gestionnaire utilise les informations passées dans l'objet argument, notamment pour connaître les marges et calculer le nombre de lignes qui peuvent tenir sur une page. La boucle d'impression n'a rien de bien sorcier non plus, puisque le code est exécuté selon deux conditions : que le compteur de lignes imprimées soit inférieur au maximum calculé juste avant et qu'il reste des lignes à imprimer dans l'objet `TStringList`.

Ensuite, chaque ligne est lue en puisant dans l'objet liste de chaînes en position zéro et en supprimant cet emplacement. Progressivement, la liste se vide par le haut, ce qui nous permet facilement de tester la fin d'impression en fonction de sa propriété `Count`.

Chaque ligne est dessinée en utilisant la méthode `DrawString` de l'objet `Graphics` passé dans l'argument du gestionnaire d'événement. Il faut voir ici une équivalence de principe avec la méthode `TextOut` de la classe `TCanvas` sous VCL.

En fin de boucle, le code teste s'il reste des lignes à imprimer et le signale à la propriété `HasMorePages` de l'argument passé au gestionnaire d'événement. Si cette propriété est passée à `False`, le cycle d'impression s'arrêtera. S'il est passé à `True`, l'objet `PrintDocument` recommencera un nouveau cycle durant lequel les trois événements suivants seront appelés dans cet ordre : `BeginPrint`, `PagePrint` et `EndPrint`. Notre exemple n'implémente que celui du milieu, le plus important.

Code complet
Retrouvez le code complet des exemples sur le CD-Rom du livre. Les différentes étapes décrites ici pour l'impression par le framework font l'objet de projets séparés afin que vous puissiez mieux les étudier.

Le choix de l'imprimante

Utiliser l'imprimante par défaut comme nous venons de le faire, sans rien demander à l'utilisateur, n'est pas une pratique très courtoise… Nous allons remédier à ce manque de civilité en ajoutant à notre programme la possibilité de choisir l'imprimante à utiliser.

Le mécanisme de sélection d'imprimante est très simple, puisqu'il suffit de créer un objet `PrintDialog` (espace de nom `System.Windows.Forms`) juste après avoir créé l'objet `PrintDocument` et de passer celui-ci à la propriété `Document` du dialogue. Il ne reste alors plus qu'à appeler la méthode `ShowDialog` de `PrintDialog` et de vérifier si la réponse est `DialogResult.OK` pour continuer (ou stopper).

Pour cet exemple, nous reprenons le projet précédent et nous y ajoutons l'espace de nom `System.Windows.Forms`. Comme tous les espaces de nom ajoutés dans la clause `Uses`, il ne faut pas oublier non plus d'ajouter la `DLL` correspondante dans les `Références` du projet.

Enfin, nous modifions la méthode `BtnPrintClick` de la façon suivante :

```
procedure TForm1.btnPrintClick(Sender: TObject);
var Dlg : System.Windows.Forms.PrintDialog;
begin
 FileToPrint := TStringlist.Create;
 FileToPrint.LoadFromFile('texte.txt');
 PrintFont := System.Drawing.Font.Create('Tahoma',9);
 Doc := PrintDocument.Create;
 Dlg := System.Windows.Forms.PrintDialog.Create;
 Dlg.Document := Doc;
 If Dlg.ShowDialog <> DialogResult.OK then exit;
 Include(Doc.PrintPage,Doc_PrintPage);
 Doc.Print;
end;
```

Les lignes ajoutées sont en gras dans le code ci-dessus. À l'exécution, le dialogue de sélection d'imprimante de Windows sera affiché dès que l'utilisateur cliquera sur le bouton d'impression de l'application. Dans une application classique, nous créerions l'objet `PrintDocument` ailleurs et offririons dans un menu la possibilité de sélectionner l'imprimante de façon indépendante.

Paramétrer les pages

L'utilisateur peut maintenant sélectionner l'imprimante. C'est beaucoup mieux, mais pourquoi s'arrêter en si bon chemin sur le sentier de l'ergonomie ? Nous pouvons aussi laisser à l'utilisateur le choix du paramétrage des pages, comme l'orientation portrait ou paysage.

Pour ce faire, il suffit d'ajouter un dialogue `PageSetupDialog`, auquel nous passerons une instance de la classe `PageSettings`, que nous réutiliserons dans la séquence d'impression au lieu de prendre les paramètres par défaut comme nous l'avons fait jusqu'à maintenant.

Plusieurs petites modifications au code de l'exemple précédent sont nécessaires. Tout d'abord, nous allons déclarer un objet `PageSettings` au niveau de la fiche en ajoutant :

```
PageInfos : System.Drawing.Printing.PageSettings;
```

Puis nous allons ajouter un bouton pour appeler le dialogue de configuration des pages. Dans le gestionnaire d'événement `OnClick` de ce dernier, nous tapons le code suivant :

```
procedure TForm1.btnPageSetupClick(Sender: TObject);
var Dlg : System.Windows.Forms.PageSetupDialog;
begin
 if not assigned(PageInfos) then PageInfos := PageSettings.Create;
 Dlg := PageSetupDialog.Create;
 Dlg.PageSettings := PageInfos;
```

```
 Dlg.ShowDialog;
end;
```

Le code ci-dessus vérifie si l'objet des informations de page a déjà été créé ou non ; dans la négative, il crée l'instance. Ensuite, une instance du dialogue `PageSetupDialog` est aussi créée et l'objet précédent est passé à sa propriété `PageSettings`. Enfin, le dialogue est appelé.

Il faut maintenant utiliser l'objet `PageSettings` s'il a été créé. Pour cela, il faut l'affecter à la propriété `DefautPageSettings` de l'objet `PrintDocument`. Nous effectuons cette modification juste après l'appel au dialogue de sélection d'imprimante ajouté à l'exercice précédent (le code ajouté est en gras) :

```
procedure TForm1.btnPrintClick(Sender: TObject);
var Dlg : System.Windows.Forms.PrintDialog;
begin
 FileToPrint := TStringlist.Create;
 FileToPrint.LoadFromFile('texte.txt');
 PrintFont := System.Drawing.Font.Create('Tahoma',9);
 Doc := PrintDocument.Create;
 Dlg := System.Windows.Forms.PrintDialog.Create;
 Dlg.Document := Doc;
 If Dlg.ShowDialog <> DialogResult.OK then exit;
 if assigned(PageInfos) then Doc.DefaultPageSettings:=PageInfos;
 Include(Doc.PrintPage,Doc_PrintPage);
 Doc.Print;
end;
```

Prévisualiser un document

Dernière détail pour parachever cette démonstration, nous allons lui ajouter la capacité de prévisualisation des documents. Loin d'être une simple cerise sur le gâteau, la prévisualisation est un élément essentiel dans une application à l'ergonomie soignée. C'est aussi une démarche citoyenne, puisque cette fonction évite le gâchis du papier pour s'apercevoir qu'on a mal fait sa mise en page par exemple. Pensons à nos utilisateurs et aux précieuses forêts et implémentons maintenant un module de prévisualisation !

Techniquement, il faut avouer qu'il serait bête de s'en passer, étant donnée la trivialité de la chose… Le framework met en effet à notre disposition un dialogue `PrintPreviewDialog`, auquel il suffit de passer l'objet `PrintDocument` avant d'invoquer sa méthode `ShowDialog`, et c'est tout…

Pour ajouter cette fonction à notre application exemple, nous allons commencer par ajouter un nouveau bouton `Preview`. Nous pourrions coder l'alternative (impression ou prévisualisation) en un seul endroit et simplement ajouter une case à cocher pour le choix

de la prévisualisation. C'est ce que nous ferions dans une application réelle. Ici, nous allons nous contenter de dupliquer le code d'impression et, *mutando mutandis*, de modifier ce qui doit l'être, ce qui simplifie la mise en évidence du mécanisme.

Le code ci-dessous est celui du `OnClick` du bouton de prévisualisation. Nous avons mis en gras ce qui a été ajouté par rapport à la séquence d'impression déjà étudiée :

```
procedure TForm1.btnPreviewClick(Sender: TObject);
var Dlg : System.Windows.Forms.PrintDialog;
    Dlgp: System.Windows.Forms.PrintPreviewDialog;
begin
 FileToPrint := TStringlist.Create;
 FileToPrint.LoadFromFile('texte.txt');
 PrintFont := System.Drawing.Font.Create('Tahoma',9);
 Doc := PrintDocument.Create;
 Dlg := System.Windows.Forms.PrintDialog.Create;
 Dlg.Document := Doc;
 If Dlg.ShowDialog <> DialogResult.OK then exit;
 if assigned(PageInfos) then Doc.DefaultPageSettings:=PageInfos;
 Include(Doc.PrintPage,Doc_PrintPage);
 Dlgp := PrintPreviewDialog.Create;
 Dlgp.Document := Doc;
 Dlgp.ShowDialog;
end;
```

Dans cet exemple, nous avons laissé le dialogue du choix de l'imprimante, ce qui peut sembler incohérent. Il n'en est rien, car ce dialogue permet de choisir la mise en page (taille du papier, orientation notamment) et que l'effet de ces choix sera visible en prévisualisation (voir figure 14.1).

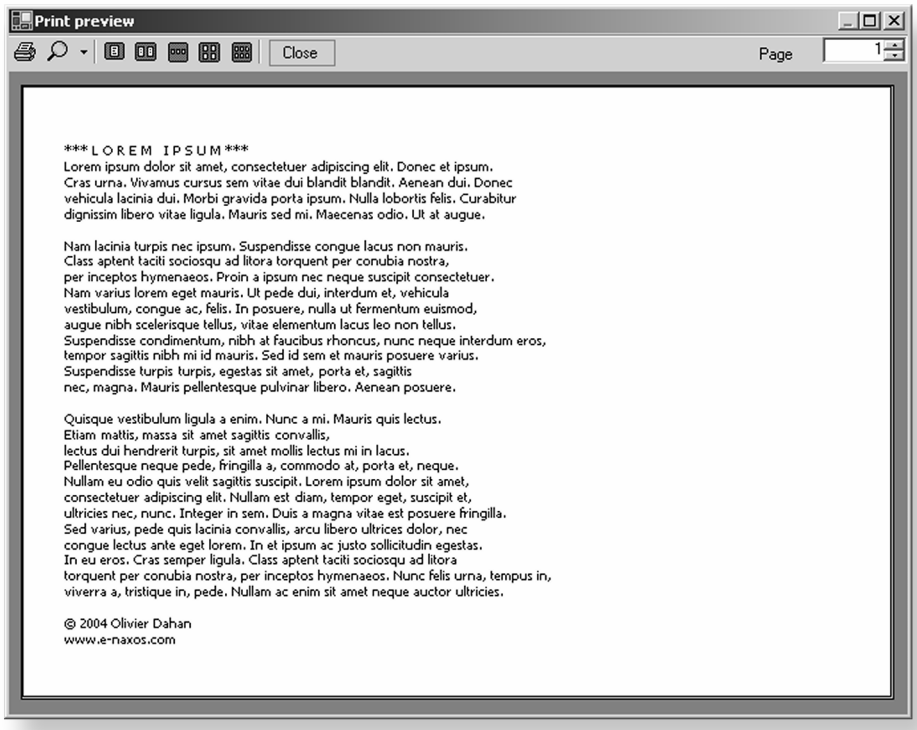

Figure 14.1

Prévisualisation d'une impression

Rave Reports

Delphi 7 a été l'occasion de nombreux changements et ajouts. Parmi ceux-ci, on notera l'abandon de Quick Report, remplacé par Rave Report de Nevrona. C'est ce produit que nous retrouvons sous Delphi .NET.

Qu'est-ce que Rave Reports ?

Rave Reports est un produit tiers conçu par la société Nevrona installée aux États-Unis et ayant une antenne en France. Il s'agit d'un générateur d'état fonctionnant sur le principe des bandes, comme Quick Report et de nombreux produits analogues. Toutefois, les possibilités de Rave Reports vont au-delà de la gestion traditionnelle par bande et le produit est plutôt basé sur le concept de page que de simple bande. Il est ainsi possible de concevoir des pages différentes dans un état ou bien de ré-exploiter d'autres états par le biais du *mirroring*, fonctionnalité propre à Rave Reports. Bien entendu, Rave Reports permet l'impression d'états aussi bien indépendants que liés aux données.

Il existe aussi une version exécutable de Rave Reports, pouvant s'installer sur un serveur afin de fournir des impressions aux utilisateurs connectés (en HTML, en PDF…). Enfin, le module de modification des états est distribuable aux utilisateurs en s'acquittant d'une licence raisonnable auprès de Nevrona.

Les possibilités du produit sont nombreuses. Nous allons ici en voir les principes essentiels.

Une première approche

Le menu « Outils/Rave Reports designer » donne accès au concepteur visuel de Rave Report. L'écran qui s'affiche est un peu chargé à première vue, mais tout a son utilité, comme les légendes que nous avons placées sur la figure 14.2 vous le prouveront. Lorsqu'un composant projet Rave est posé sur une fiche, un double clic sur celui-ci ouvre aussi le concepteur Rave.

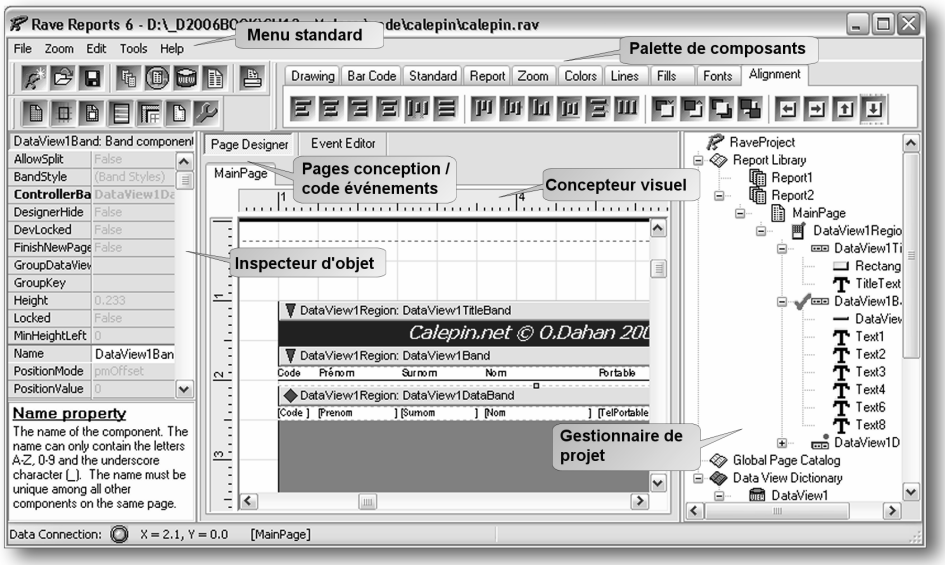

Figure 14.2

Le concepteur visuel Rave Reports

Sur cette figure, vous pouvez voir plusieurs zones.

Vous y trouvez le menu standard, comme dans toute application :

- La palette d'outils (qui ressemble plus à l'IDE de Delphi 7 qu'à celui de BDS) contient les composants à placer sur l'état et des outils comme la sélection de couleurs des objets.

- À gauche se trouve un inspecteur d'objets similaire à celui de Delphi, avec sa zone d'aide sur les propriétés dans sa partie inférieure.

- À droite se situe le gestionnaire de projet donnant accès à tous les états définis et aux composants posés sur ces derniers, même ceux qui n'ont pas de représentation visuelle (comme les composants d'accès aux données).

- Au centre, nous avons volontairement rétréci la zone de simulation de l'état pour rendre l'image lisible. Cette zone possède un onglet dans sa partie supérieure, qui permet de passer de la conception visuelle de l'état à la page de gestion des événements (*scripting*) ou à celle permettant de visualiser les données.

- Enfin, en bas d'écran se trouve la ligne d'état indiquant si la connexion à la base de données est active et affichant en permanence les coordonnées de la souris.

La zone centrale affiche par défaut le concepteur visuel. Ce dernier montre une page vide (sur notre figure, un état en cours de travail s'y trouve déjà). La page est le concept au cœur d'un état sous Rave Reports. Elle possède des propriétés comme une hauteur, une largeur, un type d'enchaînement avec les autres pages de l'état…

Le gestionnaire de projet possède trois nœuds importants qui sont :

- la bibliothèque d'états, qui contient tous les états définis dans le projet en cours,

- le catalogue des pages globales, qui liste les *templates* pouvant être réutilisés selon le principe de miroir (*mirroring*) propre à Rave Reports (en-têtes de lettres, formulaires…),

- le dictionnaire des vues de données, dans lequel apparaissent toutes les connexions aux données définies dans l'application hôte ou en local sous Rave Reports et qui sont disponibles pour la conception d'état.

La palette, qui ressemble beaucoup à celle de Delphi 7, possède des pages qui contiennent soit des composants, soit des outils de conception. Les pages proposées sont les suivantes :

- `Standard` – On trouve ici les composants ne nécessitant pas d'accès aux données : libellés, images…

- `État` – Cette palette regroupe les composants orientés données, ainsi que les composants spéciaux permettant par exemple de faire des calculs (somme d'une colonne…).

- `Code à barres` – Cette palette contient un ensemble de composants destinés à l'impression de code à barres (EAN, UPC…) avec ou sans connexion à une base de données.

- `Dessin` – Cette palette propose des lignes, des cercles et des rectangles pouvant être dessinés sur les états.

- `Zoom` – Cette palette permet de régler le zoom sur la page en cours.

- `Couleurs` – Cette palette permet de sélectionner la couleur d'un objet en cours d'édition dans le concepteur visuel.

- `Lignes` – Cette palette permet de sélectionner le style et l'épaisseur des lignes de l'objet en cours d'édition.

- `Remplissage` – Cette palette permet de sélectionner le style de remplissage de l'objet en cours d'édition.

- `Fontes` – Cette palette permet de sélectionner la fonte de l'objet en cours d'édition.

- `Alignement` – Cette palette permet de gérer l'alignement des objets entre eux comme la palette d'alignement de Delphi.

Si le concepteur visuel de Rave Reports ressemble tant à l'IDE de Delphi 7, c'est parce qu'il fonctionne de façon identique… et que le concepteur est lui-même toujours une application Win32. Cela revient à dire que vous pouvez concevoir des composants Rave Reports, ou en acheter à des éditeurs tiers, et les intégrer dans les palettes comme vous le feriez de composants sous Delphi. C'est un avantage notable.

Les composants (concepteur Rave)

Nous allons maintenant regarder plus en détail les différents composants que Rave Reports met à notre disposition pour créer des états. Il s'agit bien ici des composants disponibles dans les palettes du concepteur visuel de Rave, et non de ceux qui se trouvent dans Delphi et que nous verrons plus loin.

Les composants standards

Figure 14.3

Les composants standards de Rave Reports

Cette palette (figure 14.3) contient les différents composants de base utilisés dans la plupart des états. De gauche à droite on trouve :

- L'outil `Text`, qui permet de définir des libellés sur une ligne.
- L'outil `Memo`, qui permet de poser des zones de texte multi-lignes.
- L'outil `Section`, qui permet de regrouper des composants dans des sections, une page pouvant avoir plusieurs sections (ce qui permet la construction d'états assez sophistiqués).
- Les outils `Bitmap` et `Metafile`, qui permettent de poser des images sur les états (généralement utilisés pour les *logos* fixes : en-têtes de `mailing`, factures…).
- L'outil `Fontmaster`, qui permet de définir des fontes différentes pour les diverses parties d'un état (en-tête, corps, pieds…).
- L'outil `PageNumInit` qui permet de réinitialiser la numérotation des pages d'un état.

Les composants code à barres

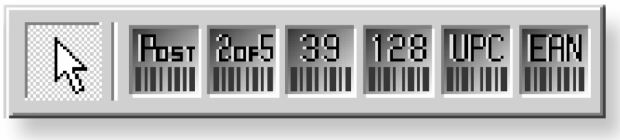

Figure 14.4

Les composants code à barres

Rave Reports contient de base une série de composants orientés code à barres. Ils s'utilisent comme des composants standard (affichage d'une valeur constante) ou bien couplés à une vue sur des données (affichage d'un champ).

Les codes gérés sont les plus courants, notamment l'EAN et le code 39.

Les composants de dessin

Figure 14.5

Les outils de dessin

Cette palette offre des lignes, des rectangles et des cercles qui peuvent être tracés sur les états. Pour faciliter la mise en page, les lignes existent en version libre ou strictement verticale ou horizontale. De même, il y a des rectangles ou des carrés parfaits et des cercles ou des ovales déformables.

Les composants de la palette État

Figure 14.6

La palette de composants État

Ces composants sont visuels et sont connectables à une vue de données (cette caractéristique est matérialisée par un point rouge en haut à droite de l'icône). Les quatre derniers composants à droite ont un fond vert (en plus clair ici) indiquant qu'ils ne sont pas visibles.

Les deux premiers composants (à partir de la gauche de la figure 14.6) permettent de créer des zones de texte à une ou plusieurs ligne(s) (mémos). Le composant `CalcText` portant le symbole sigma Σ permet d'afficher des calculs simples comme la somme, la moyenne, le compte, le minimum ou le maximum d'un champ.

Le quatrième composant est une zone miroir, nous reviendrons sur ce principe plus loin.

Le cinquième composant permet de définir une région. Une région peut contenir autant de bandes que nécessaire et une page peut contenir autant de régions qu'on le souhaite. Chaque région possède ses propres caractéristiques et les bandes qu'elle contient sont totalement indépendantes de celles définies dans les régions. Un état simple pour imprimer un listing de données comporte en général une seule région qui occupe toute la place disponible sur la page.

Le sixième et le septième composant représentent des bandes. L'une est constante, l'autre est conçue pour contenir des composants orientés données. Les bandes sont des conteneurs ayant éventuellement une position fixe (en-tête ou pied de groupe, haut ou bas de page…).

Le huitième composant, `DataCycle`, permet de définir un cycle sur des données pour former des relations maître/détail.

Le neuvième composant, `CalcOp`, est un outil de calcul permettant d'appliquer des opérations à des opérandes, qui peuvent être des variables ou des champs d'une vue de données. Le résultat peut être soit purement temporaire, soit stocké dans une variable ou un paramètre de l'état.

Le dixième composant, `CalcTotal`, ressemble en tout point à `CalcText` sauf que le résultat du calcul est destiné à traiter les données dans leur globalité sur tout l'état.

Le onzième et dernier composant, `CalcController`, permet de contrôler l'initialisation d'autres composants orientés calcul.

La conception d'un état, principes de base

Pour concevoir un état avec Rave Reports, il suffit d'appeler le concepteur visuel, de paramétrer l'état, puis de l'enregistrer sur disque. De fait, cela n'est pas couplé à Delphi directement et c'est même cet avantage qui permet de fournir le concepteur visuel aux utilisateurs finaux ou bien de se réserver la possibilité chez le client de modifier un état sans avoir à recompiler toute l'application. Il existe toutefois la possibilité de pouvoir charger le fichier de définition des états d'impression dans le composant projet de Rave sous Delphi, ce qui évite d'avoir à le déployer.

Une application qui utilise Rave Reports fait appel aux composants spécifiques se trouvant, eux, dans la palette de Delphi. Il s'agit en général d'un gestionnaire d'état contenant le fichier état ou une référence vers celui-ci et d'un ou plusieurs composant(s) annexe(s), notamment pour créer des filtres de sortie pour les formats `PDF`, `HTML`… Le concepteur visuel de Rave Reports devient alors l'éditeur de composant du gestionnaire d'état et peut être invoqué par un double-clic sur celui-ci.

Les composants Rave Reports dans Delphi

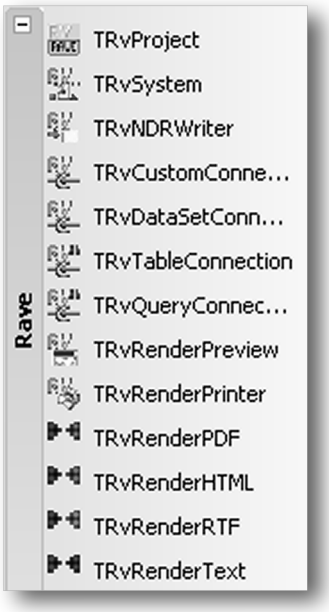

Figure 14.7

Les composants Rave Reports de Delphi .NET

La palette Rave dans Delphi propose un ensemble de composants assez réduit, puisque l'essentiel du travail de conception d'état se trouve intégré au concepteur visuel que nous avons vu plus haut. Il s'agit donc ici de s'occuper plutôt de la gestion globale des états (prévisualisation, impression, filtres d'exportation…) que de l'aspect même des états.

De haut en bas (figure 14.7), ces composants sont :

- `TRvproject` – C'est le gestionnaire d'état principal qui donne accès aux états conçus depuis le concepteur visuel de Rave. On utilise en général un seul composant de ce type dans un projet classique gérant des états. Le gestionnaire s'occupe de stocker les différents états dans un fichier externe portant l'extension `RAV`, qui est appelé aussi « projet », d'où le nom du composant. La méthode `Open` du composant charge le projet en mémoire, en vue de son exploitation dans l'application. Il est essentiel d'appeler la méthode `Close` avant de quitter l'application. Il est possible de modifier dynamiquement les états ; dans ce cas, il faut sauvegarder le projet par la méthode `Save`. Les autres méthodes et propriétés de ce composant permettent de gérer facilement la liste des états du projet, d'obtenir la description d'un état en particulier… À l'aide de ces outils, il est facile de concevoir une interface totalement intégrée à l'application hôte pour gérer les différents états que propose celle-ci. La propriété `Engine` permet d'utiliser un moteur de rendu différent de celui de base, par exemple pour créer un autre système de prévisualisation, pour créer

directement des fichiers `PDF` ou `HTML` sans passer par une étape visuelle…

- `TRvSystem` – Ce composant englobe trois fonctionnalités différentes (qui faisaient d'ailleurs l'objet de trois composants différents dans la version 4.0 de Rave Reports, vendue en dehors de Delphi). Il peut servir à envoyer une impression vers une imprimante ou une page de prévisualisation et peut afficher un dialogue de configuration comme un écran de statut. Ce composant est optionnel dans le cadre d'une exploitation de base de Rave Reports.

- `TRvNDRWriter` – Il s'agit là encore d'un composant réservé à des utilisations plus sophistiquées de Rave Reports. Sa tâche est de stocker une impression dans un format *tokenisé*, soit dans un fichier, soit en mémoire en vue de traitements ultérieurs.

Viennent ensuite les composants de connexion aux données. Lorsque vous les posez sur une fiche Delphi, et après les avoir configurés, ils sont reconnus par le concepteur visuel qui les utilisera comme sources de données potentielles. Nous verrons dans un exemple la façon de se servir de ces composants. Parmi les connexions disponibles se trouvent :

- `TRvCustomConnection` – Il est à la base des autres composants d'accès aux données. Dans cette version, l'application doit répondre aux événements pour fournir les données au générateur d'état. Cela permet d'utiliser des sources d'information qui ne sont pas forcément stockées dans une base de données, ou bien d'exploiter une base de données qui n'est pas couverte par les autres composants de connectivité.

- `TRvDatasetConnection` – Dans cette version, le composant est spécialisé pour se connecter à une source de données compatible avec la classe `TDataset`, c'est-à-dire la majeure partie des sources de données sous Delphi (qu'il s'agisse de dbExpress.NET, ADO.NET, BDE…). Ce composant est généralement celui que vous utiliserez le plus souvent pour créer des sources de données exploitables dans vos états.

- `TRvTableConnection` et `TRvQueryConnection` – Ces versions sont spécialisées pour une connexion avec les objets `Ttable` et `TQuery` du BDE. Ce dernier étant aujourd'hui *deprecated* et le `TRvDatasetConnection` étant plus universel, il n'y a guère de raison d'utiliser ces versions spécialisées, sauf pour créer des états sur une base Paradox.

Nous trouvons ensuite des composants spécialisés dans le traitement des flux d'impression, soit pour les stocker, soit pour les transformer :

- `TRvRenderPreview` – Ce composant utilise un fichier créé par `TRvNDRWriter` et l'envoie à l'écran pour une prévisualisation. Les événements de ce composant permettent ainsi de créer un système de prévisualisation personnalisé. Dans le cadre d'une utilisation de base, vous n'aurez pas besoin d'utiliser ce composant, Rave Reports proposant déjà un mode de prévisualisation par défaut.

- `TRvRenderPrinter` – Ce composant fonctionne comme le précédent, à la seule différence qu'il se destine aux impressions papier. Les mêmes remarques s'appliquent.

- `TRvRenderPDF`, `TRvRenderHTML`, `TRvRenderRTF` et `TRvRenderText` – Il s'agit de filtres d'exportation permettant de créer des fichiers `PDF`, `HTML`, `RTF` ou texte à partir d'un état Rave Reports. Si vous les posez sur une fiche de votre application, ils s'enregistreront automatiquement et l'utilisateur en verra la liste dans les dialogues d'impression et de prévisualisation. Vous n'avez donc rien d'autre à faire. Il est maintenant possible de les manipuler directement en tant que moteurs de rendu d'impression, pour automatiser la création de fichiers sans intervention de l'utilisateur.

Le conceptuel visuel de pages

Comme présenté à la figure 14.2, ce concepteur est un IDE à part entière comprenant sa palette, son inspecteur d'objets et sa zone de travail. L'autre outil indispensable est le gestionnaire de projet, car il permet de sélectionner les composants non visuels afin de faire apparaître leurs propriétés dans l'inspecteur d'objets.

L'objet de base de tout état est la page. Un état peut contenir plusieurs pages différentes, mais au minimum il en contient une. Elle est matérialisée dans l'espace central de travail par une surface blanche quadrillée (le quadrillage peut être modifié, voire supprimé). Comme tous les autres composants, la page possède des propriétés modifiables comme sa hauteur et sa largeur. Pour y accéder, il faut sélectionner la page dans l'arbre des objets. Les pages sont stockées dans l'arbre sous le nœud `Bibliothèque d'états`. Chaque page porte un nom ; par défaut, il s'agit de `Page1`, `Page2`... Lorsqu'on le crée, un projet contient au minimum une section `Report1` dotée d'une page nommée `Page1`. Les sections permettent de définir des états différents puisqu'un projet, au sens Rave Reports, est une collection d'états.

Dans le cas le plus simple, votre projet contient une section avec une page.

La manipulation des objets sur un état est identique à celle dont on dispose sous Delphi. Il n'y a donc ici rien de spécial à comprendre. La palette fonctionne elle aussi de façon similaire. Vous noterez que tous les composants possèdent une propriété `Locked`. Lorsqu'elle est à `True`, la sélection reste possible mais plus aucune propriété ne peut être modifiée (à l'exception de `Locked` pour déverrouiller le composant).

L'arbre des objets sert aussi à modifier ou copier des éléments. Les opérations de glisser/déposer, combinées avec les touches `ALT` ou `CTRL`, permettent respectivement de déplacer un élément ou de le copier de sa position de départ vers celle d'arrivée. Avec `ALT`-Glisser il est ainsi possible de changer la parenté d'un composant en le déplaçant dans l'arbre.

Pour exécuter un état, tâche répétitive lors de la conception, vous disposez de plusieurs méthodes : depuis le menu Projet en cliquant sur l'icône représentant une imprimante, par le raccourci F9, par le menu standard Fichier/Exécuter.

Quant on exécute un état, un dialogue s'affiche donnant la possibilité d'une prévisualisation, d'une impression directe ou d'un stockage dans un fichier dans l'un des formats disponibles. La fenêtre de prévisualisation fournie par défaut est assez classique, comme le prouve la figure 14.8.

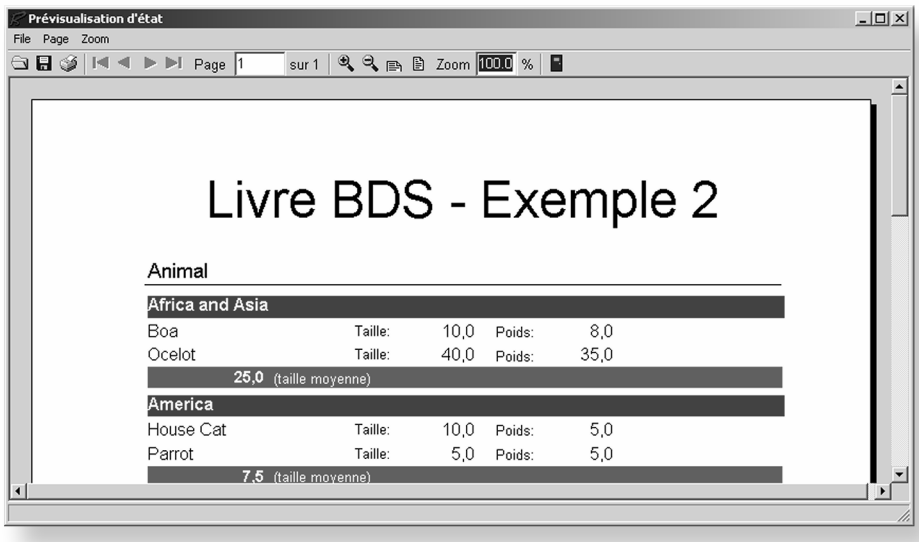

Figure 14.8

Prévisualisation d'un état

Les formats de sortie

Par défaut, Rave Reports offre deux formats de sortie sur fichier : `NDR` et `PRN`. D'autres formats peuvent être ajoutés grâce aux filtres `PDF`, `HTML`, `RTF` et `TEXT`.

Le format `NDR` est le format natif de Rave. C'est une sauvegarde de l'état sous un format *tokenisé* ne prenant pas trop de place. Un fichier `NDR` peut être chargé et relu dans la fenêtre de prévisualisation. Cette option se retrouve dans la plupart des générateurs d'états, et le format `NDR` peut être vu comme le strict équivalent du format `QRP` de Quick Report (dans le concept, pas au niveau binaire !).

Le format `PRN` est un standard indépendant de Rave. Il s'agit d'un format de fichier stockant toutes les commandes qui devraient être normalement envoyées à l'imprimante sélectionnée. Si le format `NDR` peut être imprimé sur toute imprimante (via le module de prévisualisation), le format `PRN` est généré pour satisfaire les exigences techniques d'une machine particulière. Les fichiers `PRN` peuvent être envoyés à l'imprimante de diverses façons, la plus simple étant par la console avec une commande de type :

```
TYPE lefichier.PRN >PRN
```

Les autres formats disponibles dépendent des filtres qui sont installés dans l'application hôte. Le format `HTML` engendre un fichier par page de l'état en ajoutant des commandes de navigation sur chacune. Le format `PDF` crée un fichier compatible Adobe Acrobat Reader. Le format `RTF` produit des fichiers lisibles par les traitements de texte et le format `TEXT` retourne un fichier texte standard.

La connexion aux données

L'une des utilisations principales d'un générateur d'état est l'impression de données, que celles-ci proviennent d'une base de données ou qu'elles soient produites par l'application. Rave Reports se plie à cette exigence en offrant un modèle d'accès aux données très souple permettant d'exploiter des sources de nature très différentes.

Principe de base

Rave Reports dispose de plusieurs méthodes de connexion aux données, la plus simple étant celle que vous utiliserez le plus souvent… Elle consiste, dans Delphi, à poser des composants

d'accès tout à fait standards, par exemple un `TClientDataset` pour accéder à une base `XML`, puis à placer un composant Rave `TRvDatasetConnection` qu'on relie au premier par sa propriété `Dataset`. Lorsque vous invoquerez le concepteur visuel de Rave, cette connexion sera reconnue et disponible pour créer une vue.

La création d'une connexion

Dans l'EDI de Rave, vous trouverez dans la palette Projet une icône en forme de cylindre marquée View. En cliquant sur cette icône, Rave créera une nouvelle vue. Un dialogue s'affiche immédiatement pour vous permettre de choisir comment cette vue sera créée, ou plutôt d'où elle puisera les données. Ce dialogue offre plusieurs possibilités. Celle qui s'appelle « Vue directe de données » permet d'exploiter les connexions qui ont été créées dans le projet Delphi par la méthode indiquée plus haut.

En cliquant sur le bouton Suivant, le dialogue propose la liste des connexions disponibles qu'il reconnaît.

L'autre choix possible sur la première page du dialogue est « Connexion de base de données ». Ce choix implique l'installation de fichiers `RVD` spécifiques à Rave qui se comportent comme des pilotes (appelés aussi *datalinks*). `RVD` est l'acronyme de *RaVe Driver*. Les fichiers de ce type sont chargés au démarrage de l'application.

Les autres sélections possibles concernent la gestion de la sécurité des accès. « Contrôleur de sécurité simple » permet de saisir des couples `utilisateur=mot de passe` dans un `TStrings` pour créer rapidement une gestion d'accès. « Contrôleur de sécurité lié aux données » autorise une gestion plus souple en se connectant à une vue de données définissant les droits d'accès, ce qui permet de les maintenir dans l'application hôte sans avoir à modifier la liste dans Rave.

Une fois la vue de données créée, celle-ci apparaît dans l'arbre des objets sous le nœud `Dictionnaire des vues de données`. En ouvrant le nœud correspondant à la vue, on a accès aux champs qui ont été ajoutés. En cliquant sur ceux-ci, on fait apparaître leurs caractéristiques dans l'inspecteur d'objet pour les modifier.

La vue de données possède des propriétés qui dépendent du type de connexion. Lorsqu'on utilise une connexion via les *drivers DIBL*, on peut saisir une authentification de conception et d'exécution différente. Cela est très pratique lorsque le *login* à la base de

données n'est pas le même sur la machine du développeur et sur la machine cliente de destination.

Vous noterez que la petite led affichée dans la barre de statut en bas à gauche change de couleur selon l'état de la connexion aux données. Elle est grise quand la connexion est inactive, jaune quand elle est active mais en attende de données, verte quand elle est active mais occupée et rouge quand elle est active mais que les délais de *time-out* ont été dépassés.

La création d'un état par les experts

L'IDE de Rave propose deux experts permettant la création rapide d'états simples. Ils sont accessibles par le menu Outils de Rave. Le premier expert crée un état basé sur une table, le second permet la création d'états mettant en jeu des relations maître/détail.

Une fois que vous avez placé un `TRvProject` et un `TRvDatasetConnection` sur votre fiche Delphi et que vous avez relié le second à un `TDataset` ouvert, double-cliquez sur le premier pour appeler le concepteur visuel de Rave.

Créez ensuite une vue de données selon le principe expliqué plus haut. Enfin, appelez l'expert pour table simple dans le menu Outils.

Un dialogue est alors affiché pour vous permettre de choisir la vue de données à utiliser. La page suivante du dialogue permet la sélection des champs à utiliser sur l'état ; celle qui vient ensuite permet de modifier l'ordre de ces derniers.

Une fois le dialogue validé, Rave crée une nouvelle page qui ressemble à ce que la figure 14.9 montre.

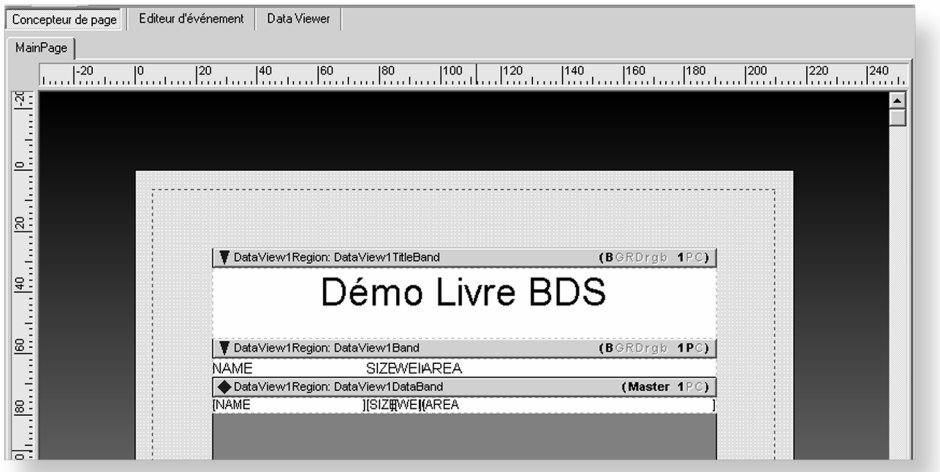

Figure 14.9

Un état simple produit par l'expert

L'état qui est créé comporte une seule région centrée dans la page. La région contient trois bandes, une de titre, une pour les titres au-dessus des colonnes, et une de données contenant les champs à imprimer. Le principe reste donc très proche de Quick Report et si vous aviez l'habitude de cet outil, vous ne devriez pas être trop dépaysé.

La création d'un état maître/détail n'est guère plus compliqué.

Partant de la trame créée par l'expert, vous pouvez dès lors apporter toutes les modifications que vous jugez nécessaires. Vous pouvez prévisualiser l'état en tapant sur F9.

Gérer les états dans un projet Delphi

L'essentiel du travail s'effectuant dans l'IDE de Rave qui ressemble, dans son principe, à celui de Delphi, il n'y a pas grand-chose à dire de plus sur la conception des états, sauf à copier servilement le contenu des documentations livrées avec le produit, ce que nous nous garderons bien de faire ici.

De fait, sous Delphi, l'opération la plus fréquente consiste à lancer l'impression d'un état, ce qui se fait en utilisant les méthodes du composant `RvProject`.

Pour illustrer cela, nous avons conçu un projet dont vous retrouverez le code source sur le CD-Rom du livre. Nous vous laissons le découvrir seul ; recopier ici ce que vous verrez sur votre écran sous Delphi n'apporterait pas grand-chose.

Utilisation avancée de Rave Reports

Rave Reports ne se limite pas aux seules possibilités exposées jusqu'à maintenant. Ce produit est réellement d'une grande richesse et il est même difficile d'en faire le tour sans bloquer une bonne journée pour tout tester et tout comprendre, ce que nous vous conseillons de faire.

Il serait donc illusoire de vous transférer ici tout ce savoir contenu dans les centaines de pages de documentation fournies avec le produit. Toutefois, après vous avoir donné les clés qui vous permettront de rapidement prendre en main Rave Reports, nous allons pointer quelques fonctionnalités de plus haut niveau dont l'intérêt semble évident.

Les astuces des composants d'état

DataText

Le composant `DataText` est utilisé pour afficher le contenu d'un champ d'une vue de données, comme nos exemples l'ont montré jusqu'ici. C'est en effet l'utilisation la plus courante.

Toutefois, ce composant offre d'autres possibilités. L'une des plus utiles, pour la traduction en clair des codes par exemple, est sa capacité à utiliser un *lookup* (propriétés `LookupDataView`, `LookupDisplay`, `LookupField`).

Ce n'est pas tout. `DataText` permet aussi d'utiliser des variables, des paramètres de l'état, et de mixer tout cela pour créer une valeur affichage. Dans l'inspecteur d'objets, la propriété `DataField` possède à la fois une flèche pour ouvrir la liste déroulante des noms de champs, mais aussi un bouton à ellipse ouvrant un dialogue d'édition. Dans ce dernier, vous pouvez construire des expressions complexes.

On retrouve dans le dialogue de création d'expression les champs des vues de données, mais aussi les variables d'état comme le nombre total de pages, les paramètres d'état et les *PI Vars*. Ces deux dernières options font référence à des variables définies librement au niveau du projet Rave. Les *PI Vars* sont des variables généralement utilisées dans les calculs, les paramètres étant sensiblement identiques au niveau fonctionnel, mais différents dans leur nature. On les utilise pour passer des constantes au projet avant son impression par exemple.

DataMemo et Memo

Ces composants sont utilisés pour afficher des textes de plusieurs lignes. Le `DataMemo` peut être connecté à un champ mémo d'une vue de données, c'est là la seule différence.

En dehors de cette utilisation de base, ce composant offre la possibilité de gérer les remplacements de parties de texte pour être utilisé dans un générateur de *mailing*. L'astuce se trouve dans la propriété `MailMergeItems`, qui permet de définir des marqueurs remplacés à l'impression par des champs, des variables…

FontMaster

Ce composant est accessible depuis la palette Standard de Rave.

Chaque composant visuel possède en général une propriété `Font`. En modifiant cette dernière, une police spécifique peut être définie pour le composant considéré en particulier. Si cela est pratique, il

s'avère que la cohérence graphique est parfois mise à mal en raison du fait que chaque état ou chaque objet sur une page possède un attribut de fonte qui peut différer totalement des autres. L'accumulation au fil du temps des différentes fontes au sein d'un état ou d'une série d'états n'est pas une bonne chose, tant du point de vue ressources que du point de vue visuel.

Le composant `FontMaster` permet de gérer les choses en mettant un peu d'unité dans le foisonnement des fontes. Chaque `FontMaster` permet de définir une fonte et on utilise généralement plusieurs `FontMaster` dans un projet. Le mieux est de leur donner des noms clairs comme `fmEntete`, `fmPied`, `fmTitre`...

Pour utiliser un composant `FontMaster`, il suffit ensuite, au lieu de modifier la propriété `Font` des composants visuels, d'affecter le `FontMaster` par la propriété `FontMirror`.

Tout simple, mais très utile…

PageNumInit

Ce composant non visuel permet de modifier la numérotation des pages.

Dans de nombreux cas, il n'est pas possible d'utiliser le compteur de page global car cela n'a pas de sens fonctionnel. L'exemple que donne Nevrona et que nous reprenons ici est celui de l'impression d'un relevé de compte bancaire. Ce dernier peut être défini sur plusieurs pages (n'oubliez pas que Rave Reports vous permet de définir plusieurs pages différentes au sein d'un même état). Par exemple, la première page peut comporter le résumé des informations du compte client (nom, adresse, numéros, options de gestion…), la seconde peut lister les dépôts et la troisième les retraits et prélèvements. Les deux premières pages peuvent ne comporter qu'une seule page, mais la troisième peut s'étendre sur plusieurs si les mouvements de retraits d'espèces, de chèques, de cartes bancaires sont nombreux. Si on désire que chaque section de l'état d'un client soit numérotée de façon unique, les deux premières pages devront être notées « 1 sur 1 » (si elles ne comportent qu'une page). Pour la troisième section, le nombre de page est variable, et si le compte impose la production de 3 pages, elles doivent être numérotées « 1 sur 3 », « 2 sur 3 » et « 3 sur 3 ».

Lorsque les besoins de numérotation des pages s'écartent d'un simple compteur, il faut alors utiliser un `PageNumInit`. En saisissant une valeur d'initialisation dans sa propriété `InitValue`, il est possible de forcer la numérotation comme on le désire. L'accès à la valeur du compteur se fait par le biais de la

variable d'état `RelativePage` (on y accède en posant un `DataText` et en appelant le dialogue expert d'expression).

La gestion de la sécurité d'accès

Rave Reports comporte un gestionnaire de sécurité. Actuellement, il n'est utilisé que par le serveur d'état qui se sert alors des services d'authentification de HTTP.

Un gestionnaire de sécurité se définit comme une vue de données. Le choix de la première page du dialogue de définition des vues offre deux options : soit un gestionnaire simple, soit un gestionnaire connecté à une base de données.

Dans le premier cas, le composant possède une liste `Tstrings` qui doit contenir des couples `utilisateur=motdepasse`. Dans le second cas, les couples sont recherchés dans une base de données.

Chaque état peut être sécurisé dans un projet par le biais de sa propriété `SecurityControl`, qui pointe sur un contrôleur de sécurité. Le projet lui-même peut être contrôlé de cette façon, en plus de posséder un mot de passe d'administration.

Il est possible de définir autant de contrôleurs de sécurité que nécessaire et de les affecter à chaque état ou au projet selon les besoins.

Au sein d'une application Delphi, la sécurité est généralement assurée par celle-ci, raison pour laquelle ce système n'est en fonction que dans la version serveur d'impression de Rave Reports, puisqu'il s'agit d'une application autonome.

Les états multi-pages/multi-états

Rave Reports permet de définir plusieurs pages par état et de gérer leur enchaînement. La première solution consiste à utiliser la propriété `GotoPage` de l'objet page. En jouant sur la propriété `GotoMode`, on peut indiquer comment se passe l'enchaînement (à la fin du traitement de la page, donc de ces n pages réelles éventuelles, après chaque page physiquement produite par la page…). Le projet de test utilisé par notre application exemple définit un tel enchaînement pour l'exemple numéro 1.

Cette méthode est généralement bien adaptée dans la majorité des cas, mais on notera une faille : la page enchaînée est toujours appelée, quoi qu'il arrive (de la façon dictée par le mode, mais elle

sera appelée). De plus, cette façon de faire est plutôt conçue pour créer une structure multi-page à l'intérieur d'un même état.

Il existe un autre besoin : celui de pouvoir regrouper dans un même *batch* une succession d'états indépendants. Ce cas est couvert par la propriété `PageList` du composant Report. Au lieu de définir plusieurs états, on définit chaque état comme une (ou plusieurs) page(s) au sein d'un même état, l'enchaînement étant alors réalisé par `PageList`. Chaque page sera traitée complètement (rappelons qu'une page peut donner naissance à un *listing* physique sur plusieurs pages) avant que le contrôle ne soit passé à la page suivante.

On notera qu'en utilisant la première méthode (`GotoPage` et `GotoMode`), il est possible de définir une stratégie d'enchaînement entre deux pages pour créer une impression différenciant les pages paires et impaires par exemple.

La gestion des miroirs

Rave permet de définir des groupes de composants comme étant des éléments « miroirs », dans le sens où ils peuvent être utilisés dans les pages des différents états sans qu'une copie physique ne soit réellement effectuée.

Cela permet de définir des en-têtes de lettres, des blocs prêts à l'emploi seront ensuite utilisés dans les pages réelles des états.

La gestion des événements

Le concepteur visuel de Rave possède un onglet sur la partie centrale qui permet de passer de l'édition de la page en cours à la modification des événements.

Cette possibilité est d'une grande richesse puisqu'elle permet d'ajouter du *scripting* au sein même d'un état. Ce sujet mériterait un autre article à lui seul et nous renvoyons le lecteur intéressé à la documentation fournie par Nevrona.

Les événements des composants Rave pour Delphi

Même si l'essentiel du travail se passe dans le concepteur Rave, les composants Delphi de Rave possèdent aussi certaines caractéristiques qui permettent de personnaliser les impressions. C'est le cas des événements disponibles dans les composants de connexion aux données.

Il est possible de contrôler le filtrage des données, d'effectuer des opérations particulières sur la base lorsqu'une session est ouverte ou fermée…

Créé directement des fichiers PDF, HTML…

L'application exemple de gestion de calepin extraite du dernier ouvrage de l'auteur chez Eyrolles (chapitre dédié à MyBase) offre un exemple de génération directe de fichier PDF (ou autre) de façon automatique.

Pour rappel le code est celui-ci :

```
procedure TForm1.ExportPDF1Click(Sender: TObject);
var p:System.Diagnostics.Process;
begin
 RvProject1.Open;
 RvSystem1.DefaultDest := rdfile;
 RvSystem1.DoNativeOutput := false;
 RvSystem1.RenderObject := RvRenderPDF1;
 RvSystem1.OutputFileName := 'calepin.pdf';
 RvSystem1.SystemSetups := RvSystem1.SystemSetups - [ssAllowSetup];
 RvProject1.Execute;
 RvProject1.Close;
… … …
end;
```

Un composant `TRvSystem` est connecté au projet rave `RvProject1` par sa propriété `Engine`. C'est grâce à ce composant système qu'on peut utiliser le moteur de rendering PDF `RvRenderPDF1` pour créer un fichier de sortie dans ce format directement sur disque. Cela fonctionne de la même façon avec les autres moteurs de rendering comme HTML, texte ou RTF.

Conclusion

La génération d'états est un point crucial dans bon nombre d'applications, il convient donc de bien choisir ses outils et de prendre le temps de les maîtriser pour en tirer le meilleur le plus rapidement possible. Le sujet est tellement vaste et les simples techniques présentées ici sont, finalement, tellement riches qu'il faudrait un livre entier pour en décrire toutes les arcanes et les astuces. Fidèle à notre mission d'éclaireur, nous avons surtout souhaité ici vous donner les clés qui vous permettront de prendre contact plus rapidement avec les outils et techniques présentés, voire de vous faire une première idée sans y passer des journées.