FastReport 4 Programmer's Manual

(C) 1998-2013 Fast Reports Inc.

Manual v 1.2.0

目次

(C) 1998- 2013 Fast Reports Inc.

(C) 1998- 2013 Fast Reports Inc.

第 1 章

TfrxReport コンポーネントを使った作業

1.1 レポートの読み込みと保存

既定では、レポートはプロジェクトフォームと一緒にDFM ファイルなどに格納されます。ほとんどの場合、必要なことは何もなく、レポートを読み込むために何か処置を取る必要もありません。レポートフォームをファイルまたはデータベースのBLOB フィールドに格納する（プログラムを再コンパイルしなくてもレポートを変更できることから、柔軟性が向上します）ことに決めた場合は、そのレポートの読み込みと保存にTfrxReport メソッドを使用する必要があります。

```
function LoadFromFile(const FileName: String;
                      ExceptionIfNotFound: Boolean = False): Boolean;
```

指定した名前のファイルからレポートを読み込みます。第 2 パラメーターがTrue の場合、ファイルが見つからないと、例外が発生します。ファイルが正常に読み込まれた場合は、True を返します。

```
procedure LoadFromStream(Stream: TStream);
```

ストリームからレポートを読み込みます。

```
procedure SaveToFile(const FileName: String);
```

指定した名前のファイルにレポートを保存します。

```
procedure SaveToStream(Stream: TStream);
```

レポートをストリームに保存します。

レポートファイルには、既定の拡張子として"fr3" が与えられます。

例：

Pascal：

```
frxReport1.LoadFromFile('c:¥1.fr3');
frxReport1.SaveToFile('c:¥2.fr3');
```

C++：

```
frxReport1->LoadFromFile("c:¥¥1.fr3");
frxReport1->SaveToFile("c:¥¥2.fr3");
```

1.2 レポートのデザイン

レポートデザイナーは、TfrxReport.DesignReport メソッドを使用して開くことができます。プロジェクトにデザイナーを含める必要があります（TfrxDesigner コンポーネントをフォームに追加するか、またはfrxDesgn ユニットをuses リストに追加します）。

DesignReport メソッドには、既定のパラメーターが 2 つあります。

```
procedure DesignReport(Modal: Boolean = True; MDIChild: Boolean = False);
```

Modal パラメーターは、デザイナーがモーダルであるかどうかを決定します。MDIChild パラメーターは、デザイナーウィンドウをMDI 子ウィンドウにします。

例：

```
frxReport1.DesignReport;
```

(C) 1998- 2013 Fast Reports Inc.

1.3 レポートの実行

次の2 つのTfrxReport メソッドのうちいずれかを呼び出すと、レポートが開始 実行、作成）されます。

```
procedure ShowReport(ClearLastReport: Boolean = True);
```

レポートを開始し、その結果をプレビュー ウィンドウに表示します。ClearLastReport パラメーターがFalse の場合、レポートは、以前に構築されたレポートに追加されます。False 以外の場合、以前に構築されたレポートは消去されます 既定）。

```
function PrepareReport(ClearLastReport: Boolean = True): Boolean;
```

プレビュー ウィンドウを開かないで、レポートを開始します。パラメーターの機能は、ShowReport メソッドのパラメーターと同じです。レポートが正常に構築された場合は、True を返します。

ほとんどの場合、最初のメソッドを使用する方が便利です。これは、レポートが構築される間に、プレビュー ウィンドウを直ちに開きます。

ClearLastReport パラメーターは、バッチ レポートの印刷で起こるように、あるレポートを別のレポートに追加する場合に便利です。

例：

```
frxReport1.ShowReport;
```

1.4 レポートのプレビュー

レポートをプレビュー ウィンドウに表示する方法は2 つあります。TfrxReport.ShowReportメソッドを呼び出す方法 前述）と、TfrxReport.ShowPreparedReport メソッドを呼び出す方法です。2 番目のメソッドでは、レポートの構築を行わず、完成したレポートが表示されます。つまり、PrepareReport メソッドを使用してあらかじめレポートを構築しておくか、あるいは、ファイルに保存されている以前に構築されたレポートから読み込んでおく（「レポートの読み込みと保存」を参照）必要があるということです。

例：

Pascal:

```
if frxReport1.PrepareReport then
  frxReport1.ShowPreparedReport;
```

C++:

```
if(frxReport1->PrepareReport(true))
  frxReport1->ShowPreparedReport();
```

このコードを使用すると、まずレポートが構築されてから、それがプレビュー ウィンドウに表示されます。大きなレポートの構築には非常に時間がかかります。ShowReport メソッドはレポート構築の視覚的なフィードバックを提供するため、PrepareReport およびShowPreparedReport メソッドよりも、ShowReport を使用することをお勧めします。レポートプレビューの設定は、TfrxReport.PreviewOptions プロパティで行えます。

1.5 レポートの印刷

多くの場合、レポートはプレビュー ウィンドウから印刷されますが、レポートを手動で印刷するには、TfrxReport.Print メソッドを使用します。たとえば、次のようにします。

```
frxReport1.LoadFromFile(...);
frxReport1.PrepareReport;
frxReport1.Print;
```

(C) 1998- 2013 Fast Reports Inc.

Print が呼び出されると、印刷ダイアログが開くので、そこで印刷パラメーターを設定できます。印刷ダイアログを無効にし、TfrxReport.PrintOptions プロパティを使って印刷パラメーターを設定することもできます。

1.6 完成したレポートの読み込みと保存

完成したレポートの読み込みと保存は、プレビュー ウィンドウから実行できます。また、TfrxReport.PreviewPages メソッドを用いて手動で実行することもできます。

```
function LoadFromFile(const FileName: String;
                ExceptionIfNotFound: Boolean = False): Boolean;
procedure SaveToFile(const FileName: String);
procedure LoadFromStream(Stream: TStream);
procedure SaveToStream(Stream: TStream);
```

パラメーターの機能は、対応するTfrxReport メソッドと類似しています。完成したレポートを含むファイルの拡張子は、既定では"FP3" になります。

例：

Pascal:

```
frxReport1.PreviewPages.LoadFromFile('c:¥1.fp3');
frxReport1.ShowPreparedReport;
```

C++:

```
frxReport1->PreviewPages->LoadFromFile("c:¥¥1.fp3");
frxReport1->ShowPreparedReport();
```

留意すべき点は、完成したレポートが完全に読み込まれた後、ShowPreparedReport メソッドを呼び出すことによって、レポートがプレビューされるということです。

1.7 レポートのエクスポート

レポートのエクスポートは、プレビュー ウィンドウから実行できます。また、TfrxReport.Export メソッドを用いて手動で実行することもできます。使用したいエクスポート フィルターは、メソッドのパラメーターで渡す必要があります。

```
frxReport1.Export(frxHTMLExport1);
```

エクスポート フィルター コンポーネントが利用できるようになっており、正しく構成されている必要があります（コンポーネントは、プロジェクト フォームに配置されていなければなりません）。

1.8 カスタム プレビュー ウィンドウの作成

FastReport は、標準のプレビュー ウィンドウでレポートを表示します。何らかの理由でこれが期待どおりでない場合は、カスタム プレビュー フォームを作成することができます。FastReport コンポーネント パレットの TfrxPreview コンポーネントは、このために設計されました。レポートを表示するには、TfrxReport.Preview プロパティがこの TfrxPreview コンポーネントに向けられる必要があります。

TfrxPreview コンポーネントを使用する際の典型的な問題が 2 つあります。それは、キーが押されても必ずしも反応するとは限らない（矢印、PageUp、PageDown など）ことと、マウス ホイール（ある場合）に反応しないことです。TfrxPreview がキーの押下に反応するようにするには、TfrxPreview にフォーカスを渡します（たとえば、フォームの OnShow イベント ハンドラーで行います）。

```
frxPreview.SetFocus;
```

(C) 1998- 2013 Fast Reports Inc.

TfrxPreview がマウス スクロールに反応するようにするには、OnMouseWheel イベント ハンドラーを作成し、このハンドラーで TfrxPreview.MouseWheelScroll メソッドを呼び出す必要があります。

```
procedure TForm1.FormMouseWheel(Sender: TObject; Shift: TShiftState;
                          WheelDelta: Integer;
                          MousePos: TPoint; var Handled: Boolean);
begin
  frxPreview1.MouseWheelScroll(WheelDelta);
end;
```

1.9 複合レポートの構築（バッチ印刷）

レポートのグループを一度に印刷したり、1 つのプレビュー ウィンドウで表示したりする必要が生じることもあります。FastReport には、構築済みのレポートの終わりに2 つめのレポートを追加する手段が備わっています。TfrxReport.PrepareReport メソッドには、任意のブール値のパラメーター ClearLastReport があり、既定で True に設定されています。このパラメーターが True の場合は、前に構築されたレポートから出力を消去します。次のコードは、2 つのレポートのバッチ処理を構築する方法を示しています。

Pascal:

```
frxReport1.LoadFromFile('1.fr3');
frxReport1.PrepareReport;
frxReport1.LoadFromFile('2.fr3');
frxReport1.PrepareReport(False);
frxReport1.ShowPreparedReport;
```

C++:

```
frxReport1->LoadFromFile("1.fr3");
frxReport1->PrepareReport(true);
frxReport1->LoadFromFile("2.fr3");
frxReport1->PrepareReport(false);
frxReport1->ShowPreparedReport();
```

最初のレポートを読み込み、プレビュー ウィンドウで表示しないで構築を行います。次に、2 つめのレポートを同じ TfrxReport オブジェクトに読み込み、今度は ClearLastReport パラメーターを False に設定して構築します。これにより、最初に構築されたレポートに2 つめのレポートを追加することができます。最後に、完成したレポートをプレビュー ウィンドウに表示します。

1.9.1 複合レポートのページ数

システム変数の Page、Page#、TotalPages、および TotalPages# を使用して、ページ番号や総ページ数を表示することができます。複合レポートでは、これらの変数は次のような値を返します。

- Page　現在のレポートのページ番号
- Page#　バッチ内のページ番号
- TotalPages　現在のレポートの総ページ数
  （レポートはダブル パスの設定である必要があります）
- TotalPages#　バッチ内の総ページ数

1.9.2 複合レポートにおけるページの結合

前述したように、レポート デザイン ページの PrintOnPreviousPage プロパティを使用すると、印刷時に複数のページをつなぎ合わせることが可能になります。前のページの終わりに空いている領域も全部使うことによって、複合レポートが生成されます。このプロパティは、新しいレポートが、前のレポートの最終ページの空き領域から出力を開始できるようにします。これを実行するには、連続する各レポートの最初のデザイン ページの PrintOnPreviousPage プロパティをオンに設定する必要があります。

(C) 1998- 2013 Fast Reports Inc.

1.10 対話型レポート

対話型レポートは、プレビュー ウィンドウで特定のレポートオブジェクトのいずれかがマウス クリックされたとき、そのクリックに反応することができます。たとえば、データ行をクリックしたら、その選択した行の詳細なデータを含む新しいレポートを実行することができます。

どんなレポートでもTfrxReport.OnClickObject イベント ハンドラーを作成すれば、対話型になります。たとえば、次のようにします。

Pascal:

```
procedure TForm1.frxReport1ClickObject(Page: TfrxPage; View: TfrxView;
                                       Button: TMouseButton;
                                       Shift: TShiftState;
                                       var Modified: Boolean);
begin
  if View.Name = 'Memo1' then
    ShowMessage('Memo1 contents:' + #13#10 + TfrxMemoView(View).Text);
  if View.Name = 'Memo2' then
  begin
    TfrxMemoView(View).Text := InputBox('Edit', 'Edit Memo2 text:',
                               TfrxMemoView(View).Text);
    Modified := True;
  end;
end;
```

C++:

```
void __fastcall TForm1::frxReport1ClickObject(TfrxView *Sender,
                                    TMouseButton Button,
                                    TShiftState Shift,
                                    bool &Modified)
{
  TfrxMemoView * Memo;
  if(Memo =  dynamic_cast <TfrxMemoView *> (Sender))
  {
    if(Memo->Name == "Memo1")
      ShowMessage("Memo1 contents:¥n¥r" + Memo->Text);
    if(Memo->Name == "Memo2")
    {
      Memo->Text = InputBox("Edit", "Edit Memo2 text:", Memo->Text);
       Modified = true;
    }
  }
}
```

OnClickObject ハンドラーでは、次のことが行えます。
- ハンドラーに渡されるオブジェクトやページの内容を変更する（変更が実装されるように、Modified 出力パラメーターを設定する必要があります）
- TfrxReport.PrepareReport メソッドを呼び出して、レポートを再構築する

この例では、"Memo1" オブジェクトをクリックすると、そのオブジェクトの内容を示すメッセージが表示されます。"Memo2" オブジェクトをクリックすると、ダイアログが表示され、そこでこのオブジェクトの内容を変更できます。Modified フラグをTrue に設定することにより、変更は永続的になります。

同様にして、新しいレポートを実行するなどの、ほかの反応を作成できます。注意：FastReport v3 以降では、TfrxReport コンポーネントは、1 つのレポートしかプレビュー ウィンドウへ表示できません（FastReport v2.x とは異なります）。そのため、新しいレポートを別のTfrxReport オブジェクトで実行するか、または現在のレポートを現在のTfrxReport オブジェクトから削除する必要があります。

プレビュー ウィンドウでマウス カーソルがオブジェクト上を通過するときに、マウス カーソルの形状を変えることで、クリック可能

(C) 1998- 2013 Fast Reports Inc.

なオブジェクトを示す視覚的な手がかりをユーザーに与えることができます。これを行うには、レポートデザイナーで目的のオブジェクトを選択し、そのCursor プロパティをcrDefault 以外の値に設定します。

もう1 つ、クリック可能なオブジェクトの作成について懸念があります。シンプルなレポートでは、オブジェクトの名前やオブジェクトの内容をテストして、反応を起こすことができます。しかし、もっと複雑なレポートになると、そんな簡単にはいかない場合があります。たとえば、マスター/詳細レポートで、内容 '12' を含む "Memo2" をクリックした場合、そのデータはどこから来るのでしょうか？主キーがこの行を明確に識別できます。FastReport では、任意の有効な文字列データ（この場合は主キー）を格納しておくためのTagStr プロパティが提供されています。

FastReportDemo.exe に含まれる 'シンプル リスト' デモのレポートを例にとって、これを説明しましょう。これは、ある企業のクライアントの一覧で、クライアントの名前、住所、連絡担当者などのデータを含んでいます。データソースは、DBDEMOS デモデータベースのCustomer.db テーブルです。このテーブルはCustNo フィールドを主キーとしていますが、このフィールドはレポート出力に表示されません。我々の問題は、完成したレポートの任意のオブジェクトをクリックしたときに参照されるレコードを決定することです。主キーの値が必要となります。この値は、マスター データバンドにあるすべてのオブジェクトのTagStr プロパティに、次のような式として入力することができます。

```
[Customers."CustNo"]
```

レポートの構築中に、TagStr プロパティの内容は、テキストオブジェクトの内容が評価されるのと同様に評価されます。つまり、名前付き変数に値が代入され、式（角かっこで囲まれている）が評価されるということです。マスター データバンドに配置されているオブジェクトのTagStr プロパティには、レポート構築後、'1005' や '2112' などの値が入ります。文字列から整数への単純な変換をすれば、必要なレコードを見つけるための主キーの値が与えられます。

主キーが複数の要素から成る（つまり、2 つ以上のフィールドを含んでいる）場合、TagStr プロパティの内容は次のようになります。

```
[Table1."Field1"];[Table1."Field2"]
```

レポートの構築後、TagStr プロパティには '1000;1' のような文字列値が入っています。ここから個々のフィールド値を抽出するのはかなり簡単でしょう。

1.11 コードによるレポート オブジェクトのアクセス

FastReport のオブジェクト（レポートページ、バンド、テキストオブジェクトなど）は、コードから直接アクセスすることができません。これは、たとえば、フォーム上のボタンのアドレス指定をするとき、オブジェクトを名前で直接アドレス指定することはできないということです。オブジェクトをアドレス指定するには、まずTfrxReport.FindObject メソッドを使用してオブジェクトを探す必要があります。

Pascal：

```
var
  Memo1: TfrxMemoView;

Memo1 := frxReport1.FindObject('Memo1') as TfrxMemoView;
```

C++：

```
TfrxMemoView * Memo = dynamic_cast <TfrxMemoView *>
                      (frxReport1->FindObject("Memo1"));
```

オブジェクトが見つかったら、そのプロパティおよびメソッドにアクセスできます。

レポートのページのアドレス指定は、TfrxReport.Pages プロパティを使用して行います。

Pascal：

```
var
  Page1: TfrxReportPage;

Page1 := frxReport1.Pages[1] as TfrxReportPage;
```

(C) 1998- 2013 Fast Reports Inc.

C++:

```
TfrxReportPage * Page1 = dynamic_cast <TfrxReportPage *>
                    (frxReport1->Pages[1]);
```

1.12 コードによるレポート フォームの作成

通例、ほとんどのレポートはデザイナーを使用して作成します。しかし、レポートのフォームが不明な場合など、コードから手動でレポートを作成することが必要になる場合もあります。

レポートを手動で作成するには、次の作業を順に実行する必要があります。
- レポート コンポーネントをクリアする
- データソースを追加する
- "データ" ページを追加する
- レポートのページを追加する
- ページにバンドを追加する
- バンドのプロパティを設定し、そのバンドをデータに接続する
- 各バンドにオブジェクトを追加する
- オブジェクトのプロパティを設定し、そのオブジェクトをデータに接続する

リスト タイプのシンプルなレポートの作成を見てみましょう。次のようなコンポーネント、frxReport1: TfrxReport と frxDBDataSet1: TfrxDBDataSet があるとします (後者は、DBDEMOS の Customer.db テーブルのデータに接続されています)。レポートには、レポート タイトル バンドとマスター データ バンドを持つ 1 つのページが含まれます。レポート タイトル バンドには、"Hello FastReport!" というテキストを含むオブジェクトが入り、マスター データ バンドには、"CustNo" フィールドにリンクされたオブジェクトが入ります。

Pascal:

```
var
  DataPage: TfrxDataPage;
  Page: TfrxReportPage;
  Band: TfrxBand;
  DataBand: TfrxMasterData;
  Memo: TfrxMemoView;

{ レポートをクリアする }
frxReport1.Clear;

{ レポート内でアクセス可能なデータセットの一覧にデータセットを追加する }
frxReport1.DataSets.Add(frxDBDataSet1);

{ "データ" ページを追加する }
DataPage := TfrxDataPage.Create(frxReport1);

{ ページを追加する }
Page := TfrxReportPage.Create(frxReport1);
{ 一意の名前を作成する }
Page.CreateUniqueName;
{ 各フィールドのサイズ、用紙のサイズと向きを既定値に設定する }
Page.SetDefaults;
{ 用紙の向きを変更する }
Page.Orientation := TPrinterOrientation.poLandscape;

{ レポート タイトル バンドを追加する }
Band := TfrxReportTitle.Create(Page);
Band.CreateUniqueName;
{ バンドの Top 座標と高さのみ設定する必要がある。どちらもピクセル単位 }
Band.Top := 0;
Band.Height := 20;

{ レポート タイトル バンドにオブジェクトを追加する }
Memo := TfrxMemoView.Create(Band);
```

(C) 1998- 2013 Fast Reports Inc.

```
  Memo.CreateUniqueName;
  Memo.Text := 'Hello FastReport!';
  Memo.Height := 20;
  { このオブジェクトはバンドの幅まで引き伸ばされる }
  Memo.Align := baWidth;

  { マスター データ バンドを追加する }
  DataBand := TfrxMasterData.Create(Page);
  DataBand.CreateUniqueName;
  DataBand.DataSet := frxDBDataSet1;
  { Top は、前に追加したバンドの Top + Hight よりも大きくなければならない }
  DataBand.Top := 100;
  DataBand.Height := 20;

  { マスター データにオブジェクトを追加する }
  Memo := TfrxMemoView.Create(DataBand);
  Memo.CreateUniqueName;
  { データに接続する }
  Memo.DataSet := frxDBDataSet1;
  Memo.DataField := 'CustNo';
  Memo.SetBounds(0, 0, 100, 20);
  { テキストをオブジェクトの右余白に揃える }
  Memo.HAlign := haRight;

  { レポートを表示する }
  frxReport1.ShowReport;
```

C++:

```
  TfrxDataPage * DataPage;
  TfrxReportPage * Page;
  TfrxBand * Band;
  TfrxMasterData * DataBand;
  TfrxMemoView * Memo;

  // レポートをクリアする
  frxReport1->Clear();

  // レポート内でアクセス可能なデータセットの一覧にデータセットを追加する
  frxReport1->DataSets->Add(frxDBDataset1);

  // "データ" ページを追加する
  DataPage = new TfrxDataPage(frxReport1);

  // ページを追加する
  Page = new TfrxReportPage(frxReport1);
  // 一意の名前を作成する
  Page->CreateUniqueName();
  // 各フィールドのサイズ、用紙のサイズと向きを既定値に設定する
  Page->SetDefaults();
  // 用紙の向きを変更する
  Page->Orientation = TPrinterOrientation->poLandscape;

  // レポート タイトル バンドを追加する
  Band = new TfrxReportTitle(Page);
  Band->CreateUniqueName();
  // バンドの Top 座標と高さのみ設定する必要がある
  // どちらもピクセル単位
  Band->Top = 0;
  Band->Height = 20;

  // レポート タイトル バンドにオブジェクトを追加する
  Memo = new TfrxMemoView(Band);
  Memo->CreateUniqueName();
  Memo->Text = "Hello FastReport!";
  Memo->Height = 20;
```

(C) 1998- 2013 Fast Reports Inc.

```
// このオブジェクトはバンドの幅まで引き伸ばされる
Memo->Align = baWidth;

// マスター データ バンドを追加する
DataBand = new TfrxMasterData(Page);
DataBand->CreateUniqueName();
DataBand->DataSet = frxDBDataset1;
// Top は、前に追加したバンドの Top + Hight よりも大きくなければならない
DataBand->Top = 100;
DataBand->Height = 20;

// マスター データにオブジェクトを追加する
Memo = new TfrxMemoView(DataBand);
Memo->CreateUniqueName();
// データに接続する
Memo->DataSet = frxDBDataset1;
Memo->DataField = "CustNo";
Memo->SetBounds(0, 0, 100, 20);
// テキストをオブジェクトの右余白に揃える
Memo->HAlign = haRight;

// レポートを表示する
frxReport1->ShowReport(true);
```

いくつかの詳細を説明しましょう。

レポートで使用されるすべてのデータソースは、データソースの一覧に追加する必要があります。追加しておかないと、レポートは動作しません。上記のケースでは、"frxReport1.DataSets.Add(frxDBDataSet1)" を使用しています。

"データ" ページは、TfrxADOTable などの内部データセットをレポートに挿入するときに必要です。そのようなデータセットは、"データ" ページにのみ配置することができます。

Page.SetDefaults の呼び出しは必須ではありません。この場合、ページは A4 の用紙サイズで余白 0 mm となります。SetDefaults は、余白を10 mm に設定し、用紙サイズと配置をプリンターの既定値に設定します。

ページにバンドを追加する際は、バンドが互いに重なっていないことを確認してください。確実にするには、Top プロパティと Height プロパティを設定します。Left プロパティや Width プロパティを変更しても意味がありません。バンドは常に、それが配置されているページと同じ幅を持っているからです。ただし、バンドが垂直バンドである場合は、これに該当しません。代わりに、Left プロパティと Width プロパティを設定します。Top や Height プロパティは気にしません。ここで留意すべきは、ページにおけるバンドの順序が重要であるということです。常に、デザイナーでバンドを配置したとおりの順序でバンドを配置します。

オブジェクトの座標およびサイズは、ピクセル単位で設定されます。すべてのオブジェクトの Left、Top、Width、および Height プロパティは Extended 型であるため、整数以外の値を設定できます。以下の定数が、ピクセルからセンチメートルまたはインチへの変換用に定義されています。

```
fr01cm = 3.77953;
fr1cm  = 37.7953;
fr01in = 9.6;
fr1in  = 96;
```

たとえば、バンドの高さを5 mm に設定するには、次のようにします。

```
Band.Height := fr01cm * 5;
Band.Height := fr1cm * 0.5;
```

1.13 コードによるダイアログ フォームの作成

ご存知のとおり、レポートにはダイアログ フォームを含めることができます。次の例は、［OK］ボタンのあるダイアログ フォームを作成する方法を示しています。

Pascal:

(C) 1998- 2013 Fast Reports Inc.

```
{ ダイアログ オブジェクトを使って作業するには、次のユニットを使用する必要があります }
uses frxDCtrl;

var
  Page: TfrxDialogPage;
  Button: TfrxButtonControl;

{ ページを追加する }
Page := TfrxDialogPage.Create(frxReport1);
{ 一意の名前を作成する }
Page.CreateUniqueName;
{ サイズを設定する }
Page.Width := 200;
Page.Height := 200;
{ 位置を設定する }
Page.Position := poScreenCenter;

{ ボタンを追加する }
Button := TfrxButtonControl.Create(Page);
Button.CreateUniqueName;
Button.Caption := 'OK';
Button.ModalResult := mrOk;
Button.SetBounds(60, 140, 75, 25);

{ レポートを表示する }
frxReport1.ShowReport;
```

C++:

```
// ダイアログ オブジェクトを使って作業するには、次のユニットを使用する必要があります
#include "frxDCtrl.hpp"

TfrxDialogPage * Page;
TfrxButtonControl * Button;

// ページを追加する
Page = new TfrxDialogPage(frxReport1);
// 一意の名前を作成する
Page->CreateUniqueName();
// サイズを設定する
Page->Width = 200;
Page->Height = 200;
// 位置を設定する
Page->Position = poScreenCenter;

// ボタンを追加する
Button = new TfrxButtonControl(Page);
Button->CreateUniqueName();
Button->Caption = "OK";
Button->ModalResult = mrOk;
Button->SetBounds(60, 140, 75, 25);

// レポートを表示する
frxReport1->ShowReport(true);
```

1.14 レポート ページのプロパティの変更

ときに、レポートページの設定（たとえば、用紙調整やサイズなど）をコードで変更することが必要になります。
TfrxReportPage クラスには、ページのサイズを定義する次のプロパティが含まれています。

```
property Orientation: TPrinterOrientation default poPortrait;
property PaperWidth: Extended;
property PaperHeight: Extended;
```

(C) 1998- 2013 Fast Reports Inc.

```
property PaperSize: Integer;
```

PaperSize プロパティは、用紙の書式を設定します。これは、Windows.pas に定義されている標準値のうちのいずれかになります（たとえば、DMPAPER_A4）。このプロパティに値が代入されている場合、FastReport は自動的に PaperWidth プロパティとPaperHeight プロパティを設定します（ミリメートル単位の用紙サイズ）。PaperSize を DMPAPER_USER（または256）に設定すると、カスタム用紙サイズが設定されることになります。この場合、PaperWidth プロパティとPaperHeight プロパティはコードで設定する必要があります。

次の例は、最初のページのパラメーターを変更する方法を示しています。レポートは既にあるものとします。

Pascal:

```
var
  Page: TfrxReportPage;

{ 最初のレポートページはインデックス[1] になります。インデックス[0] はデータページです }
Page := TfrxReportPage(frxReport1.Pages[1]);
{ サイズを変更する }
Page.PaperSize := DMPAPER_A2;
{ 用紙の向きを変更する }
Page.Orientation := TPrinterOrientation.poLandscape;
```

C++:

```
TfrxReportPage * Page;

// 最初のレポートページはインデックス[1] になります。インデックス[0] はデータページです
Page = (TfrxReportPage *)frxReport1.Pages[1];
// サイズを変更する
Page->PaperSize = DMPAPER_A2;
// 用紙の向きを変更する
Page->Orientation = TPrinterOrientation->poLandscape;
```

1.15 コードを使用したレポートの構築

FastReport エンジンは通常、レポートの構築を担当します。これは、レポートが完成するまで、レポートのバンドを指定された順序で、接続されているデータソースが必要とする回数だけ表示します。ときには、FastReport エンジンでは生成できない、非標準のフォームのレポートの作成が必要になることもあります。この場合は、TfrxReport.OnManualBuild イベントを使用して、レポートを手動で構築できます。このイベントのハンドラーが定義されると、FastReport エンジンは一部の管理機能をハンドラーに引き渡します。レポートの構築に対する責任の配分は、次のように変更されます。

FastReport エンジン：
- レポートの準備（スクリプト、データソースの初期化、バンドのツリーの形成）
- すべての計算（集計関数、イベントハンドラー）
- 新しいページ/列の作成（ページ、列ヘッダー、フッター、レポートタイトル、およびレポートの概要の自動表示）
- その他のルーチンワーク

ハンドラー：
- バンドの順序付け

OnManualBuild ハンドラーの目的は、FastReport エンジンに対して、特定のバンドを表示するためのコマンドを発行することです。残りの作業はエンジン自体が実行します。たとえば、現在のページに場所がなくなったらすぐに新しいページを作成するとか、スクリプトを実行するとかなどです。

エンジンは、TfrxCustomEngine クラスで表されます。このクラスのインスタンスへのリンクは、TfrxReport.Engine プロパティにあります。

```
procedure NewColumn;
```
新しい列を作成します。列がページの最後の列である場合、これは新しいページを作成します。

```
procedure NewPage;
```
新しいページを作成します。

(C) 1998- 2013 Fast Reports Inc.

procedure ShowBand(Band: TfrxBand); overload;
バンドを表示します。

procedure ShowBand(Band: TfrxBandClass); overload;
指定された種類のバンドを表示します。

function FreeSpace: Extended;
ページの空き領域の量を返します（ピクセル単位）。次のバンドが表示された後、この値は減少します。

property CurColumn: Integer;
現在の列の番号を返す、または設定します。

property CurX: Extended;
現在のX位置を返す、または設定します。

property CurY: Extended;
現在のY位置を返す、または設定します。次のバンドが表示された後、この値は増加します。

property DoublePass: Boolean;
レポートは2パス（ダブルパス）であるかどうかを定義します。

property FinalPass: Boolean;
現在のパスが最後のパスであるかどうかを返します。

property FooterHeight: Extended;
ページフッターの高さを返します。

property HeaderHeight: Extended;
ページヘッダーの高さを返します。

property PageHeight: Extended;
ページの印刷可能な領域の高さを返します。

property PageWidth: Extended;
ページの印刷可能な領域の幅を返します。

property TotalPages: Integer;
完成したレポートのページ数を返します（ダブルパスレポートの第2パスでのみ）。

単純なハンドラーの例を見てみましょう。レポートには2つのマスターデータバンドがあります。これらはデータに接続されていません。ハンドラーは、これらのバンドを交互に各6回表示します。バンドを6回繰り返した後、小さなすき間が作られます。

Pascal:

```
var
  i: Integer;
  Band1, Band2: TfrxMasterData;

{ 指定されたバンドを探す }
Band1 := frxReport1.FindObject('MasterData1') as TfrxMasterData;
Band2 := frxReport1.FindObject('MasterData2') as TfrxMasterData;

for i := 1 to 6 do
begin
  { バンドを順々に表示する}
  frxReport1.Engine.ShowBand(Band1);
  frxReport1.Engine.ShowBand(Band2);
  { 小さいすき間を作る}
  if i = 3 then
    frxReport1.Engine.CurY := frxReport1.Engine.CurY + 10;
end;
```

C++:

(C) 1998- 2013 Fast Reports Inc.

```
int i;
TfrxMasterData * Band1;
TfrxMasterData * Band2;

// 指定されたバンドを探す
Band1 := dynamic_cast <TfrxMasterData *> (frxReport1->FindObject("MasterData1"));
Band2 := dynamic_cast <TfrxMasterData *> (frxReport1->FindObject("MasterData2"));

for(i = 1; i <= 6; i++)
{
  // バンドを順々に表示する
  frxReport1->Engine->ShowBand(Band1);
  frxReport1->Engine->ShowBand(Band2);
  // 小さいすき間を作る
  if(i == 3)
    frxReport1->Engine->CurY += 10;
}
```

次の例では、バンドの 2 つのグループをお互い横に並べて表示します。

Pascal:

```
var
  i, j: Integer;
  Band1, Band2: TfrxMasterData;
  SaveY: Extended;

Band1 := frxReport1.FindObject('MasterData1') as TfrxMasterData;
Band2 := frxReport1.FindObject('MasterData2') as TfrxMasterData;

SaveY := frxReport1.Engine.CurY;
for j := 1 to 2 do
begin
  for i := 1 to 6 do
  begin
    frxReport1.Engine.ShowBand(Band1);
    frxReport1.Engine.ShowBand(Band2);
    if i = 3 then
      frxReport1.Engine.CurY := frxReport1.Engine.CurY + 10;
  end;
  frxReport1.Engine.CurY := SaveY;
  frxReport1.Engine.CurX := frxReport1.Engine.CurX + 200;
end;
```

C++:

```
int i, j;
TfrxMasterData * Band1;
TfrxMasterData * Band2;
Extended SaveY;

Band1 = dynamic_cast <TfrxMasterData *> (frxReport1->FindObject("MasterData1"));
Band2 = dynamic_cast <TfrxMasterData *> (frxReport1->FindObject("MasterData2"));

SaveY = frxReport1->Engine->CurY;
for(j = 1; j <= 2; j++)
{
  for(i = 1; i <= 6; i++)
  {
    frxReport1->Engine->ShowBand(Band1);
    frxReport1->Engine->ShowBand(Band2);
    if(i == 3)
      frxReport1->Engine->CurY += 10;
  }
  frxReport1->Engine->CurY = SaveY;
  frxReport1->Engine->CurX += 200;
```

(C) 1998- 2013 Fast Reports Inc.

```
}
```

1.16 配列の出力

この例のコードは、"FastReport Demos\PrintArray" (FastReport Demos\BCB Demos\PrintArray') フォルダーにあります。このコードについて、いくつか詳細を説明しましょう。

配列を出力するために、マスター データバンドを1 つ持つレポートを使用します。このバンドは、配列に存在する要素の数だけ表示されます。これを行うには、フォームにTfrxUserDataSet コンポーネントを配置して、そのプロパティを設定します (例に示すとおり、コード内でこれを行うことができます)。

```
RangeEnd := reCount
RangeEndCount := 配列内の要素の数
```

その後、データバンドをTfrxUserDataSet コンポーネントに接続します。配列要素を表すために、式 " [<要素>]" を含んでいるテキスト オブジェクトをマスター データバンドの中に配置します。"要素" 変数は、TfrxReport.OnGetValue イベントを使って入力されます。

1.17 TStringList の出力

この例のコードは、"FastReport Demos\PrintStringList" (FastReport Demos\BCB Demos\PrintStringList") フォルダーにあります。方法は、配列を出力する例の場合と同じです。

1.18 ファイルの出力

この例のコードは、"FastReport Demos\PrintFile" (FastReport Demos\BCB Demos\PrintFile") フォルダーにあります。このコードについて、いくつか詳細を説明しましょう。

出力する場合、レポートには、1 回だけ出力されるマスター データバンドが含まれている必要があります (これを行うには、レコードが1 件しか入っていないデータソースにバンドを接続します。一覧から"Single row" という名前のソースを選択します)。バンドの拡張 (Stretch) および分断 (Allow Split) が有効になっています。つまり、バンドは、その上にあるすべてのオブジェクトが入るだけの場所を作るために引き伸ばされる。ただし、ページ内にそのバンドが入る余地が十分にない場合、バンドは2 ページ以上に分割される、ということです。

ファイルの内容は、式 " [<ファイル>]" の変数を含んでいるテキスト オブジェクトを使って表示されます。前の例で見られるように、この変数はTfrxReport.OnGetValue イベントを使って入力されます。また、オブジェクトの拡張も有効になっています (コンテキスト メニューの [拡張可能] 項目をオンにするか、StretchMode プロパティ= smActualHeight とします)。

1.19 TStringGrid の出力

この例のコードは、"FastReport Demos\PrintStringGrid" (FastReport Demos\BCB Demos\PrintStringGrid") フォルダーにあります。このコードについて、いくつか詳細を説明しましょう。

TStringGrid コンポーネントは、複数の行と列を含む表を表します。つまり、レポートは、高さだけでなく幅も同様に拡張するということです。このコンポーネントを出力するために、クロス集計オブジェクトを使ってみましょう (これは、プロジェクトにTfrxCrossObject コンポーネントが追加されているときに利用できます)。クロス集計オブジェクトは、含まれる行数および列数がわかっていない表のデータを出力する場合にのみ使用されます。オブジェクトには2 つのバージョンがあります。コード中に書かれたカスタム データを出力するためのTfrxCrossView と、DB テーブルから独特のデータを出力するためのTfrxDBCrossView です。

TfrxCrossView を使ってみましょう。オブジェクトを初期化する必要があります。それにはまず、レポート デザイナーを開き、オブジェクトをダブルクリックしてオブジェクト エディターを開きます。行および列の数を設定し、同様に表のセル内の値の数も設定する必要があります。この例では、これらの値すべてに'1' を使用し、行および列のタイトルと、行および列の合計を無

(C) 1998- 2013 Fast Reports Inc.

効にします。

TfrxReport.OnBeforePrint イベントで、StringGrid の値によってオブジェクトを埋める必要があります。値は、TfrxCrossView.AddValue メソッドを使って追加されます。このメソッドのパラメーターは、行値、列値、およびセル値の複合インデックスです（オブジェクトは1 つのセルに複数の値を含むことができるため、セル値も同様に各種の要素から成ります）。

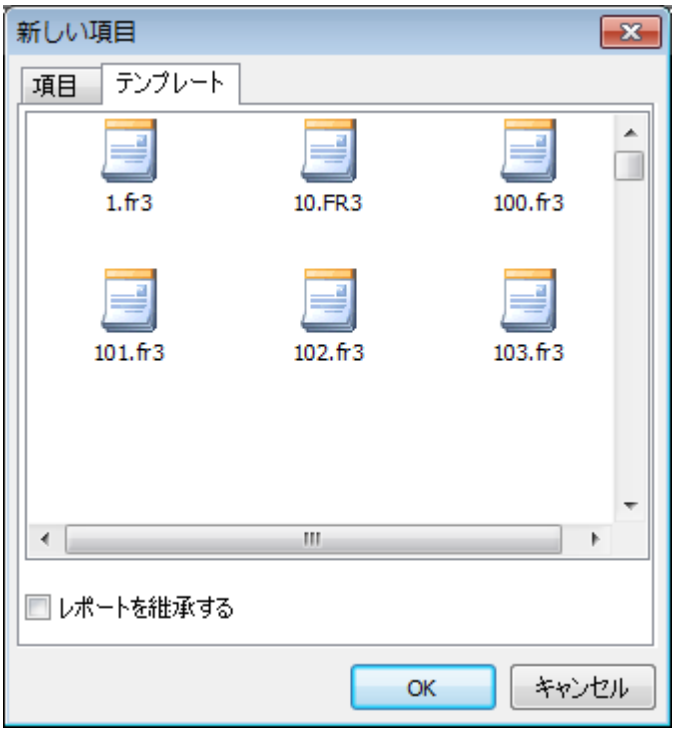
1.20 TTable または TQuery の出力

この例のコードは、"FastReport Demos\PrintTable"（"FastReport Demos\BCB Demos\PrintTable"）フォルダーにあります。原則は、TStringGrid を出力する例の場合と同じです。このケースでは、行インデックスは連続番号で、列インデックスはテーブル フィールドの名前、セル値はテーブル フィールドの値です。以下の点を留意することが重要です。クロス集計オブジェクト エディターで、セル要素に対する機能を無効にする必要がある（セルにはさまざまな種類のデータを格納できるため、これによって表の作成時にエラーが生じる）ことと、表のタイトルの並べ替えも無効にする必要がある（そうしないと、列がアルファベット順に並べ替えられる）ことです。

1.21 レポートの継承

レポートの継承については、『User's Manual』で説明しています。ここでは、いくつかの重要なポイントについて述べます。

レポートがファイルに保管されている場合は、基本レポートを検索するフォルダーをFastReport に伝える必要があります。このフォルダーの内容は、［ファイル｜新規作成］および［レポート｜オプション］メニューから開くダイアログに表示されます。



TfrxDesigner.TemplateDir プロパティがこの目的のために使用されます。既定ではこれは空なので、FastReport はプロジェクトの実行可能ファイル（.exe）と同じフォルダーで基本レポートを検索します。このプロパティには、絶対パスまたは相対パスを設定することができます。

レポートがデータベースに保管されている場合は、データベースから使用可能な基本レポートの一覧を取得し、データベースから基本レポートを読み込むためのコードを記述する必要があります。TfrxReport.OnLoadTemplate イベントを使用して基本レポートを読み込みます。

```
property OnLoadTemplate: TfrxLoadTemplateEvent read FOnLoadTemplate
```

(C) 1998- 2013 Fast Reports Inc.

```
                                    write FOnLoadTemplate;

  TfrxLoadTemplateEvent = procedure(Report: TfrxReport;
                          const TemplateName: String) of object;
```

このイベントハンドラーで、指定されたTemplateName の基本レポートをReport オブジェクトに読み込む必要があります。ハンドラーの例を以下に示します。

```
  procedure TForm1.LoadTemplate(Report: TfrxReport;
                        const TemplateName: String);
  var
    BlobStream: TStream;
  begin
    ADOTable1.First;
    while not ADOTable1.Eof do
    begin
      if AnsiCompareText(ADOTable1.FieldByName('ReportName').AsString,
                  TemplateName) = 0 then
      begin
        BlobStream := TMemoryStream.Create;
        TBlobField(ADOTable1.FieldByName('ReportBlob'))
                .SaveToStream(BlobStream);
        BlobStream.Position := 0;
        Report.LoadFromStream(BlobStream);
        BlobStream.Free;
        break;
      end;
      ADOTable1.Next;
    end;
  end;
```

使用可能なテンプレートの一覧を取得するには、TfrxDesigner.OnGetTemplateList イベントを使用します。

```
  property OnGetTemplateList: TfrxGetTemplateListEvent
                              read FOnGetTemplateList
                              write FOnGetTemplateList;

  TfrxGetTemplateListEvent = procedure(List: TStrings) of object;
```

このイベントハンドラーは、使用可能なテンプレートの一覧をList パラメーターに返します。ハンドラーの例を以下に示します。

```
  procedure TForm1.GetTemplates(List: TList);
  begin
    List.Clear;
    ADOTable1.First;
    while not ADOTable1.Eof do
    begin
      List.Add(ADOTable1.FieldByName('ReportName').AsString);
      ADOTable1.Next;
    end;
  end;
```

FastReport では、以前に作成したレポートから継承することができます。このために次の関数を使用します。

```
   TfrxReport.InheritFromTemplate(const templName: String;
                          InheritMode: TfrxInheritMode
                                  = imDefault): Boolean
```

この関数は、現在読み込まれているレポートを、指定したテンプレートから継承されるようにします。最初のパラメーターは親テンプレートの名前とパスで、2 番目のパラメーターには継承モードを設定します。次のいずれかになります。

imDefault 既定値） - 重複項目について名前を変更するか削除するか意向を示すためのダイアログを開く
imDelete - 重複するすべてのオブジェクトを削除する
imRename - 重複するすべてのオブジェクトの名前を変更する

(C) 1998- 2013 Fast Reports Inc.

注意：
親テンプレートの検索は、現在のテンプレートを参照して行われます。通常はアプリケーションの移動について心配する必要がないため、FastReport は相対パスを使用しています。唯一の例外は、現在のレポートと親テンプレートが異なるフォルダーに置かれているか、またはネットパスが使用される場合です。

1.22 マルチスレッド

FastReport は複数のスレッドで独立して操作することができますが、考慮すべき事柄がいくつかあります。

- TfrxDBDataSet を異なるスレッドで作成することはできません。これは、検索に"グローバル一覧" が使用されるためと、データセットは常に、最初に作成されたTfrxDBDataSet から取り出されるためです（グローバル一覧の使用をオフに切り替えることができます。既定では使える状態になっています）。

- レポートの実行中にオブジェクトのプロパティに変更がある（たとえば、スクリプトで Memo1.Left := Memo1.Left + 10 とする）場合は、次のことを覚えておいてください。TfrxReport.EngineOptions.DestroyForms = False である場合、次の操作時、レポートテンプレートは既に変更されており、再読み込みする必要はありません。変更がない場合は、TfrxReport.EngineOptions.DestroyForms := True を使用します。スクリプトのオブジェクトは更新後に削除されるため、更新中、スレッドから対話型レポートを使用することはできません。そういう訳で、場合によっては、TfrxReport.EngineOptions.DestroyForms := False を使用するか、または次の構築サイクル時に自分でテンプレートを更新することをお勧めします。

必要であれば、TfrxDBDataSet のコピーを検索するグローバル一覧の使用をオフに切り替えられます。

```
{ 毎回、レポートがファイルから、または現在のレポートから更新される場合は、DestroyForms をオフにできます }

FReport.EngineOptions.DestroyForms := False;
FReport.EngineOptions.SilentMode := True;

{ このプロパティは、グローバル一覧の検索をオフにします }
FReport.EngineOptions.UseGlobalDataSetList := False;

{ EnabledDataSets がローカル一覧の役割を果たすには、
  テンプレートが読み込まれる前にそれを設定しておく必要があります }
FReport.EnabledDataSets.Add(FfrxDataSet);
FReport.LoadFromFile(ReportName);
FReport.PrepareReport;
```

1.23 レポート キャッシュ

レポートとそのデータは、メモリ（高速化目的）とディスク上のファイル（RAM 使用量の節約目的）のいずれにもキャッシュすることができます。FastReport におけるキャッシュにはいくつかの種類があります。

- TfrxReport.EngineOptions.UseFileCache - True の場合、構築したレポート内のテキストとオブジェクトはすべて、ディスク上の一時ファイルに保存されます。TfrxReport.EngineOptions.MaxMemoSize には、RAM 内にレポート用として予約されるMB 量を設定します。

- TfrxReport.PreviewOptions.PagesInCache - キャッシュメモリに保持できるページ数によって、プレビューの速度は大幅に向上しますが、多くのメモリを使います（レポート内に図がある場合は特にそうです）。

- TfrxReport.PreviewOptions.PictureCacheInFile - True の場合、構築したレポート内の図はすべて、ディスク上の一時ファイルに保存されます。多数の図を含むレポートでは、メモリ使用量は大幅に減少しますが、速度は落ちます。

1.24 MDI アーキテクチャ

FastReport では、プレビュー用とデザイナー用の MDI アプリケーションの作成例を用意しています。例のコードは、"FastReport Demos\MDI Designer" にあります。

言及すべき点は、プレビュー ウィンドウやデザイナーごとにTfrxReport を作成することが望ましいということです。作成しな

(C) 1998- 2013 Fast Reports Inc.

いと、すべてのウィンドウが 1 つのレポートを参照することになります。

(C) 1998- 2013 Fast Reports Inc.

第 2 章

変数の一覧を使った作業

変数の概念については、対応する章で詳細に説明されています。簡潔に要点を思い浮かべてみましょう。

ユーザーは、レポートで 1 つまたは複数の変数を指定することができます。どの変数にも値または式を割り当てることができます。これらは、変数を参照するときに自動的に計算されます。変数は、「データツリー」ペインからレポートへ視覚的に挿入することができます。レポートでよく使われる複合式のエイリアスとして変数を使用すると便利です。

変数を使って作業するときは、プロジェクトで frxVariables ユニットを使用する必要があります。変数は、TfrxVariable クラスで表されます。

```
TfrxVariable = class(TCollectionItem)
published
  property Name: String;
  { 変数の名前 }

  property Value: Variant;
  { 変数の値 }
  end;
```

変数の一覧は、TfrxVariables クラスで表されます。これに、一覧を取り扱うために必要なすべてのメソッドが含まれています。

```
TfrxVariables = class(TCollection)
public
  function Add: TfrxVariable;
  { 一覧の最後に変数を追加します }

  function Insert(Index: Integer): TfrxVariable;
  { 一覧の指定された位置に変数を追加します }

  function IndexOf(const Name: String): Integer;
  { 指定された名前の変数のインデックスを返します }

  procedure AddVariable(const ACategory, AName: String;
                  const AValue: Variant);
  { 指定されたカテゴリに変数を追加します }

  procedure DeleteCategory(const Name: String);
  { 1 つのカテゴリとそのすべての変数を削除します }

  procedure DeleteVariable(const Name: String);
  { 1 つの変数を削除します }

  procedure GetCategoriesList(List: TStrings; ClearList: Boolean = True);
  { カテゴリの一覧を返します }

  procedure GetVariablesList(const Category: String; List: TStrings);
  { 指定されたカテゴリ内の変数の一覧を返します }

  property Items[Index: Integer]: TfrxVariable readonly;
  { 変数の一覧 }

  property Variables[Index: String]: Variant; default;
  { 指定された変数の値 }

  end;
```

変数の一覧が長い場合は、カテゴリ別に変数をグループ化すると便利です。たとえば、次のような変数の一覧があるとします。

```
Customer name
Account number
in total
total vat
```

(C) 1998- 2013 Fast Reports Inc.

これを次のように表すことができます。

```
プロパティ
 Customer name
 Account number
集計
 in total
 total vat
```

いくつかの制限があります。

- 少なくとも1 つはカテゴリを作成する必要がある
- カテゴリはデータツリーの第 1 レベルを形成し、変数は第 2 レベルを形成する
- カテゴリを入れ子にすることはできない
- 変数の名前は、カテゴリ内だけでなく一覧全体の中でも一意でなければならない

2.1 変数の一覧の作成

レポート変数へのリンクは、TfrxReport.Variables プロパティに格納されます。一覧を手動で作成するには、次の手順を踏む必要があります。

- 一覧をクリアする
- カテゴリを作成する
- 変数を作成する
- 2 番目と3 番目の手順を繰り返して、別のカテゴリの変数を作成する

2.2 変数の一覧のクリア

変数の一覧をクリアするには、TfrxVariables.Clear メソッドを使用します。

Pascal :

```
frxReport1.Variables.Clear;
```

C++ :

```
frxReport1->Variables->Clear();
```

2.3 カテゴリの追加

少なくとも1 つはカテゴリを作成する必要があります。カテゴリと変数の両方が1 つの一覧に格納されます。カテゴリは、名前の最初の文字が"空白" である点で変数と異なります。カテゴリの後の一覧にあるすべての変数は、そのカテゴリに属するものと見なされます。

一覧にカテゴリを追加する方法は2 とおりあります。

Pascal :

```
frxReport1.Variables[' ' + 'My Category 1'] := Null;
```

C++ :

```
frxReport1->Variables->Variables[" My Category 1"] = NULL;
```

または

(C) 1998- 2013 Fast Reports Inc.

Pascal:

```
var
  Category: TfrxVariable;

Category := frxReport1.Variables.Add;
Category.Name := ' ' + 'My category 1';
```

C++:

```
TfrxVariable * Category;

Category = frxReport1->Variables->Add();
Category->Name = " My category 1";
```

2.4 変数の追加

変数は、カテゴリが既に追加された後にのみ追加することができます。カテゴリの後の一覧にあるすべての変数は、そのカテゴリに属するものと見なされます。変数の名前は、カテゴリ内だけでなく一覧全体の中でも一意でなければなりません。

一覧に変数を追加する方法はいくつかあります。

Pascal:

```
frxReport1.Variables['My Variable 1'] := 10;
```

C++:

```
frxReport1->Variables->Variables["My Variable 1"] = 10;
```

この方法では、まだ存在しない変数の場合は変数を追加し、既存の変数の場合は値を変更します。

Pascal:

```
var
  Variable: TfrxVariable;

Variable := frxReport1.Variables.Add;
Variable.Name := 'My Variable 1';
Variable.Value := 10;
```

C++:

```
TfrxVariable * Variable;

Variable = frxReport1->Variables->Add();
Variable->Name = "My Variable 1";
Variable->Value = 10;
```

どちらの方法も一覧の最後に変数を追加します。そのため、変数は最後のカテゴリに追加されます。変数を一覧内の特定の位置に追加する場合は、Insert メソッドを使用します。

Pascal:

```
var
  Variable: TfrxVariable;

Variable := frxReport1.Variables.Insert(1);
Variable.Name := 'My Variable 1';
Variable.Value := 10;
```

C++:

(C) 1998- 2013 Fast Reports Inc.

```
TfrxVariable * Variable;

Variable = frxReport1->Variables->Insert(1);
Variable->Name = "My Variable 1";
Variable->Value = 10;
```

特定のカテゴリに変数を追加する場合は、AddVariable メソッドを使用します。

Pascal:

```
frxReport1.Variables.AddVariable('My Category 1', 'My Variable 2', 10);
```

C++:

```
frxReport1->Variables->AddVariable("My Category 1", "My Variable 2", 10);
```

2.5 変数の削除

Pascal:

```
frxReport1.Variables.DeleteVariable('My Variable 2');
```

C++:

```
frxReport1->Variables->DeleteVariable("My Variable 2");
```

2.6 カテゴリの削除

カテゴリとそれに含まれるすべての変数を削除するには、次のコードを使用します。

Pascal:

```
frxReport1.Variables.DeleteCategory('My Category 1');
```

C++:

```
frxReport1->Variables->DeleteCategory("My Category 1");
```

2.7 変数の値の変更

変数の値を変更する方法は2 つあります。

Pascal:

```
frxReport1.Variables['My Variable 2'] := 10;
```

C++:

```
frxReport1->Variables->Variables["My Variable 2"] = 10;
```

または

Pascal:

```
var
  Index: Integer;
  Variable: TfrxVariable;

{ 変数を検索する}
Index := frxReport1.Variables.IndexOf('My Variable 2');
```

(C) 1998- 2013 Fast Reports Inc.

```
{ 見つかった場合は、値を変更する}
if Index <> -1 then
begin
  Variable := frxReport1.Variables.Items[Index];
  Variable.Value := 10;
end;
```

C++:

```
int Index;
TfrxVariable * Variable;

// 変数を検索する
Index = frxReport1->Variables->IndexOf("My Variable 2");
// 見つかった場合は、値を変更する
if(Index != -1)
{
  Variable = frxReport1->Variables->Items[Index];
  Variable->Value = 10;
}
```

留意すべき点は、レポート変数にアクセスするとき、その変数が文字列型である場合には、値が計算されているということです。つまり、値が'Table1."Field1"' の変数は、文字列 'Table1."Field1"' ではなく、データベース フィールドの値を返します。レポート変数に文字列型の値を割り当てる場合には注意が必要です。たとえば、次のコードにより、レポートの実行時に"変数 'test' が不明です" という例外が発生します。

```
frxReport1.Variables['My Variable'] := 'test';
```

なぜなら、FastReport が変数 'test' の値を計算しようとするからです。文字列値を渡す正しい方法は次のとおりです。

```
frxReport1.Variables['My Variable'] := '''' + 'test' + '''';
```

この場合、変数の値である文字列 'test' は、エラーなしで表示されます。ただし、以下の点に留意してください。
- 文字列に一重引用符を含めてはいけない。一重引用符はすべて二重引用符にする必要があります。
- 文字列に#13 または#10 記号を含めてはいけない。

場合によっては、スクリプトを使用して変数を渡す方が簡単です。

2.8 スクリプト変数

レポート変数の代わりに、TfrxReport.Script にはスクリプト変数が存在します。これらは、FastScript メソッドを使用して定義することができます。レポート変数とスクリプト変数との違いを見てみましょう。

	レポート変数	スクリプト変数
配置	レポート変数一覧では、 TfrxReport.Variables	レポートスクリプトでは、 TfrxReport.Script.Variables
変数の名前	任意の記号を含むことができます。	任意の記号を含むことができます。 ただし、レポートスクリプトの内部で使用する場合、名前はPascal の識別子の要件に準拠している必要があります。
変数の値	任意の型を指定できます。 文字列型の変数は、アクセスされるたびに計算されます。また、それ自体が式です。	任意の型を指定できます。 計算は実行されません。 動作は、標準の言語の変数と同様です。
アクセシビリティ	プログラマは、[データツリー] ペインでレポート変数の一覧を見ることができます。	変数は表示されません。プログラマは、存在する変数を知っている必要があります。

スクリプト変数を扱うのは簡単です。次のように、単に変数に値を代入するだけです。

(C) 1998- 2013 Fast Reports Inc.

Pascal:

```
frxReport1.Script.Variables['My Variable'] := 'test';
```

C++:

```
frxReport1->Script->Variables->Variables["My Variable"] = "test";
```

このとき、FastReport は、変数が存在しない場合は変数を作成し、変数が既存であれば値を代入します。変数に文字列を代入するとき、余分な引用符を使用する必要はありません。

2.9 TfrxReport.OnGetValue で変数の値を渡す

レポートに値を渡すための最後の方法は、TfrxReport.OnGetValue イベントハンドラーを使用するものです。これは、動的な値をレポートに渡す（レコードごとに値が変わる可能性がある）場合に便利です。前述の 2 つの方法は、静的な値を渡す場合に適しています。

このイベントハンドラーを使用する例を見てみましょう。レポートを作成し、そこに"テキスト" オブジェクトを配置します。このオブジェクトに次のようなテキストを入力します。

```
[My Variable]
```

次に、TfrxReport.OnGetValue イベントハンドラーを作成します。

```
procedure TForm1.frxReport1GetValue(const VarName: String;
                                    var Value: Variant);
begin
  if CompareText(VarName, 'My Variable') = 0 then
    Value := 'test'
end;
```

レポートを実行して、変数が正しく表示されるか確認します。TfrxReport.OnGetValue イベントハンドラーは、FastReport が不明な変数を見つけるたびに呼び出されます。イベントハンドラーは、その変数の値を返す必要があります。

(C) 1998- 2013 Fast Reports Inc.

第 3 章

スタイルを使った作業

まず第一に、「スタイル」、「スタイルのセット」、および「スタイルのライブラリ」とは何であるかを思い出しましょう。

スタイルは、名前と特性を持ち、色、フォント、枠線など、いくつかのデザイン属性を決定する要素です。スタイルは、レポートオブジェクトの表示方法を決定します。TfrxMemoView などのオブジェクトには、スタイル名を保持する、Style プロパティがあります。このプロパティに値が渡されると、スタイルのデザイン属性がそのオブジェクトに適用されます。

スタイルのセットは、レポートで使用されるいくつかのスタイルで構成されます。TfrxReport コンポーネントには、TfrxStyles スタイルのオブジェクトを指し示す、Styles プロパティがあります。スタイルのセットも名前を持ちます。スタイルのセットは、レポート全体の外観のデザインを決定します。

スタイル ライブラリには、スタイルのセットが複数含まれています。レポートについて、スタイル ライブラリの中から特定のスタイル セットを便利に選択できます。

TfrxStyleItem は 1 つのスタイルを表します。

```
TfrxStyleItem = class(TCollectionItem)
public
  property Name: String;
  { スタイル名 }

  property Color: TColor;
  { 背景色 }

  property Font: TFont;
  { フォント }

  property Frame: TfrxFrame;
  { 枠線 }
end;
```

スタイルのセットは、TfrxStyles クラスで表されます。これは、スタイルの読み取り、保存、追加、削除などの設定操作だけでなく、スタイルの検索を実行するためのメソッドを備えています。スタイルのセットを含むファイルの拡張子は、既定では "fs3" になります。

```
TfrxStyles = class(TCollection)
public
  constructor Create(AReport: TfrxReport);
  { スタイルのセットを作成します。
    "AReport" の代わりに "nil" を指定することができます。
    ただし、その結果、ユーザーは "Apply" を使用できなくなるということもあり得ます }

  function Add: TfrxStyleItem;
  { 新しいスタイルを追加します }

  function Find(const Name: String): TfrxStyleItem;
  { 指定された名前のスタイルを返します }

  procedure Apply;
  { レポートにセットを適用します }

  procedure GetList(List: TStrings);
  { スタイル名の一覧を返します }

  procedure LoadFromFile(const FileName: String);
  procedure LoadFromStream(Stream: TStream);
  { セットを読み取ります }

  procedure SaveToFile(const FileName: String);
  procedure SaveToStream(Stream: TStream);
  { セットを保存します }

  property Items[Index: Integer]: TfrxStyleItem; default;
  { スタイルの一覧 }
```

(C) 1998- 2013 Fast Reports Inc.

```
  property Name: String;
  { セットの名前 }

  end;
```

最後に、TfrxStyleSheet クラスはスタイル ライブラリを表します。これには、ライブラリの読み取り、保存のほかに、スタイル セットの追加、削除、検索を行うためのメソッドが含まれています。

```
TfrxStyleSheet = class(TObject)
public
  constructor Create;
  { ライブラリを構築します }

  procedure Clear;
  { ライブラリをクリアします }

  procedure Delete(Index: Integer);
  { 特定のインデックス番号を持つセットを削除します }

  procedure GetList(List: TStrings);
  { スタイル セットの名前の一覧を返します }

  procedure LoadFromFile(const FileName: String);
  procedure LoadFromStream(Stream: TStream);
  { ライブラリを読み込みます }

  procedure SaveToFile(const FileName: String);
  procedure SaveToStream(Stream: TStream);
  { ライブラリを保存します }

  function Add: TfrxStyles;
  { ライブラリに新しいスタイルのセットを追加します }

  function Count: Integer;
  { ライブラリ内のスタイル セットの数を返します }

  function Find(const Name: String): TfrxStyles;
  { 指定された名前のセットを返します }

  function IndexOf(const Name: String): Integer;
  { 指定された名前のセット番号を返します }

  property Items[Index: Integer]: TfrxStyles; default;
  { スタイル セットの一覧 }

  end;
```

3.1 スタイル セットの作成

次のコードは、スタイル セットを作成し、そのセットに2 つのスタイルを追加する方法を示しています。これらの操作が完了した後、スタイルがレポートに適用されます。

Pascal:

```
var
  Style: TfrxStyleItem;
  Styles: TfrxStyles;

Styles := TfrxStyles.Create(nil);

{ 1 番目のスタイル }
```

(C) 1998- 2013 Fast Reports Inc.

```
  Style := Styles.Add;
  Style.Name := 'Style1';
  Style.Font.Name := 'Courier New';

  { 2 番目のスタイル }
  Style := Styles.Add;
  Style.Name := 'Style2';
  Style.Font.Name := 'Times New Roman';
  Style.Frame.Typ := [ftLeft, ftRight];

  { レポートにセットを適用する }
  frxReport1.Styles := Styles;
```

C++:

```
  TfrxStyleItem * Style;
  TfrxStyles * Styles;

  Styles = new TfrxStyles(NULL);

  // 1 番目のスタイル
  Style = Styles->Add();
  Style->Name = "Style1";
  Style->Font->Name = "Courier New";

  // 2 番目のスタイル
  Style = Styles->Add();
  Style->Name = "Style2";
  Style->Font->Name = "Times New Roman";
  Style->Frame->Typ << ftLeft << ftRight;

  // レポートにセットを適用する
  frxReport1->Styles = Styles;
```

別の方法でセットを作成し、使用することができます。

Pascal:

```
  var
    Style: TfrxStyleItem;
    Styles: TfrxStyles;

  Styles := frxReport1.Styles;
  Styles.Clear;

  { 1 番目のスタイル }
  Style := Styles.Add;
  Style.Name := 'Style1';
  Style.Font.Name := 'Courier New';

  { 2 番目のスタイル }
  Style := Styles.Add;
  Style.Name := 'Style2';
  Style.Font.Name := 'Times New Roman';
  Style.Frame.Typ := [ftLeft, ftRight];

  { レポートにセットを適用する }
  frxReport1.Styles.Apply;
```

C++:

```
  TfrxStyleItem * Style;
  TfrxStyles * Styles;

  Styles = frxReport1->Styles;
  Styles->Clear();
```

(C) 1998- 2013 Fast Reports Inc.

```
// 1 番目のスタイル
Style = Styles->Add();
Style->Name = "Style1";
Style->Font->Name = "Courier New";

// 2 番目のスタイル
Style = Styles->Add();
Style->Name = "Style2";
Style->Font->Name = "Times New Roman";
Style->Frame->Typ << ftLeft << ftRight;

// レポートにセットを適用する
frxReport1->Styles->Apply();
```

3.2 スタイルの変更/ 追加/ 削除

指定された名前のスタイルを変更します。

Pascal:

```
var
  Style: TfrxStyleItem;
  Styles: TfrxStyles;

Styles := frxReport1.Styles;

{ スタイルを検索する}
Style := Styles.Find('Style1');

{ フォントサイズを変更する}
Style.Font.Size := 12;
```

C++:

```
TfrxStyleItem * Style;
TfrxStyles * Styles;

Styles = frxReport1->Styles;

// スタイルを検索する
Style = Styles->Find("Style1");

// フォントサイズを変更する
Style->Font->Size = 12;
```

レポートのスタイル セットにスタイルを追加します。

Pascal:

```
var
  Style: TfrxStyleItem;
  Styles: TfrxStyles;

Styles := frxReport1.Styles;

{ 追加する}
Style := Styles.Add;
Style.Name := 'Style3';
```

C++:

```
TfrxStyleItem * Style;
```

(C) 1998- 2013 Fast Reports Inc.

```
TfrxStyles * Styles;

Styles = frxReport1->Styles;

// 追加する
Style = Styles->Add();
Style->Name = "Style3";
```

指定された名前のスタイルを削除します。

Pascal:

```
var
  Style: TfrxStyleItem;
  Styles: TfrxStyles;

Styles := frxReport1.Styles;

{ 削除する}
Style := Styles.Find('Style3');
Style.Free;
```

C++:

```
TfrxStyleItem * Style;
TfrxStyles * Styles;

Styles = frxReport1->Styles;

// 削除する
Style = Styles->Find("Style3");
delete Style;
```

変更後、Apply メソッドを呼び出す必要があります。

```
{ 変更を使用する}
frxReport1.Styles.Apply;
```

3.3 スタイルのセットの保存/ 復元

Pascal:

```
frxReport1.Styles.SaveToFile('c:¥1.fs3');
frxReport1.Styles.LoadFromFile('c:¥1.fs3');
```

C++:

```
frxReport1->Styles->SaveToFile("c:¥¥1.fs3");
frxReport1->Styles->LoadFromFile("c:¥¥1.fs3");
```

3.4 レポート スタイルのクリア

レポートスタイルは2 つの方法でクリアできます。

```
frxReport1.Styles.Clear;
```

または

```
frxReport1.Styles := nil;
```

(C) 1998- 2013 Fast Reports Inc.

3.5 スタイル ライブラリの作成

次の例は、ライブラリを作成し、それに2 つのスタイル セットを追加する方法を示しています。

Pascal:

```
var
  Styles: TfrxStyles;
  StyleSheet: TfrxStyleSheet;

StyleSheet := TfrxStyleSheet.Create;

{ 1 番目のセット}
Styles := StyleSheet.Add;
Styles.Name := 'Styles1';
{ ここで、スタイルを Styles セットに追加できます }

{ 2 番目のセット}
Styles := StyleSheet.Add;
Styles.Name := 'Styles2';
{ ここで、スタイルを Styles セットに追加できます }
```

C++:

```
TfrxStyles * Styles;
TfrxStyleSheet * StyleSheet;

StyleSheet = new TfrxStyleSheet;

// 1 番目のセット
Styles = StyleSheet->Add();
Styles->Name = "Styles1";
// ここで、スタイルを Styles セットに追加できます

// 2 番目のセット
Styles = StyleSheet->Add();
Styles->Name = "Styles2";
// ここで、スタイルを Styles セットに追加できます
```

3.6 スタイル セットの一覧、および選択したスタイルのアプリケーションの表示

スタイル ライブラリは、ComboBox や ListBox などのコントロールで、利用可能なスタイル セットを表示するためによく使われます。ユーザーによって選択されたスタイルをレポートに適用することができます。

一覧を表示する：

```
StyleSheet.GetList(ComboBox1.Items);
```

選択されたスタイルをレポートで使用する：

```
frxReport1.Styles := StyleSheet.Items[ComboBox1.ItemIndex];
```

または

```
frxReport1.Styles := StyleSheet.Find[ComboBox1.Text];
```

(C) 1998- 2013 Fast Reports Inc.

3.7 スタイル セットの変更/ 追加/ 削除

特定の名前のセットを変更するには、次のようにします。

```
var
  Styles: TfrxStyles;
  StyleSheet: TfrxStyleSheet;

{ 必要なセットを検索する}
Styles := StyleSheet.Find('Styles2');

{ 見つかったセットの中から、Style1 という名前のスタイルを変更する}
with Styles.Find('Style1') do
  Font.Name := 'Arial Black';
```

ライブラリにセットを追加します。

```
var
  Styles: TfrxStyles;
  StyleSheet: TfrxStyleSheet;

{ 3 番目のセット}
Styles := StyleSheet.Add;
Styles.Name := 'Styles3';
```

ライブラリからセットを削除します。

```
var
  i: Integer;
  StyleSheet: TfrxStyleSheet;

{ 3 番目のセットを検索する}
i := StyleSheet.IndexOf('Styles3');
{ 見つかった場合は、削除する}
if i <> -1 then
  StyleSheet.Delete(i);
```

3.8 スタイル ライブラリの保存と読み込み

スタイル ライブラリのファイル拡張子は、既定では"fss" です。

```
var
  StyleSheet: TfrxStyleSheet;

StyleSheet.SaveToFile('c:¥1.fss');
StyleSheet.LoadFromFile('c:¥1.fss');
```

(C) 1998- 2013 Fast Reports Inc.