宮坂 優大

株式会社ミガロ.

システム事業部 システム1課

[Delphi/400]

FastReportを活用した電子帳票作成テクニック

略歴
1982 年 11 月 19 日生まれ
2006 年 近畿大学 理工学部卒業
2006 年 4 月 株式会社ミガロ. 入社
2006 年 4 月 システム事業部配属

現在の仕事内容
主に Delphi/400 を利用したシステムの受託開発と MKS サポートを担当。Delphi および Delphi/400 のスペシャリストを目指して精進する毎日である。

1.はじめに

基幹システムの構築では帳票機能が不可欠であるが、最近ではプリンタに紙で出力する従来の帳票機能ではなく、電子帳票での開発も増えてきた。

電子帳票化を実現する場合、一番の目的はコスト削減であることが多い。電子化によって用紙はもちろん、トナーなどプリンタ関連の消耗品にかかるコストや、その運用・保守費用を削減できる。帳票の保管や閲覧がシステム上で行えると、物理的なスペース制約や帳票紛失のトラブル解決にもつながるので、業務的なメリットは大きい。

また電子帳票であれば、ネットワークを通じて取引先にデータとして渡せるので、BtoB などで必要とされることも多い。もちろん必要に応じて電子ファイルから紙帳票として印刷もできる。総じてデメリットは少ないといえる。

電子帳票の実現に際しては、専用のパッケージソフトを導入する場合も多いが、ソフトの仕様に帳票書式や業務を合わせる必要があったり、社内システムとの連携が難しいこともある。

そのため自社用のシステムを開発・運用している場合は、そのシステムのなかで電子帳票を実装する要望が多い。電子帳票の開発には技術的にハードルの高いイメージがあるが、Delphi/400 ではツールの機能を活用して容易に実現できる。

本稿では電子帳票化テクニックとして、帳票を画像ファイルで出力する方法と、電子帳票でとくに要望が多い電子データ印を押印する手法について説明していく。

なお電子帳票には pdf 形式や画像形式があるが、本稿では画像加工のテクニックを扱うため画像形式を題材としている。もちろん Delphi/400 では、pdf の出力も可能である。

2.FastReportを使用した電子帳票化

FastReport と は、Delphi/400 XE3 以降で新しくバンドルされた帳票ツールである。

本稿では、社内ワークフローで使用される購入申請書をテーマに、FastReport を使用して電子帳票（画像ファイル）を作成する。作成する購入申請書フォーマットは、【図 1】に示す。

Delphi/400 XE7 と FastReport を使用して、以下のように電子帳票を作成していく。

2-1. 帳票フォーマットの作成

電子帳票作成の準備として、印刷フォームに TfrxReport コンポーネントを貼り付ける。【図 2】

次に、レポートデザイナを使用して帳票フォーマットを作成する。レポートデザイナを起動するには、Delphi/400 の開発画面に貼り付けた TfrxReport コンポーネントをダブルクリックする。【図 3】

レポートデザイナを起動したあとは、自由に線（Line）、枠（Shape）、文字（Memo）を使用してデザインする。

まずは線を引き、レイアウトのイメージを作成する。

図1

購入申請書

申請日

購入予定

＜申請者＞

所属

氏名

下記の通り、備品購入を申請いたします。

品名	単価	数量	金額	摘要
		合計金額		

備考

承認者2印	承認者1印	申請者印

レポートデザイナの左側にあるコンポーネントパレットから描画を選択し、ポップアップから線オブジェクトを選択する。マウスでドラッグ＆ドロップするだけで、簡単に線を引ける【図 4】。線オブジェクトは縦もしくは横にしか引けないので、斜めの線を引く場合は対角線を使用する。

次に固定文字列を貼り付ける。固定文字列をレイアウトに貼り付けるには、テキストオブジェクトを使用する。

コンポーネントパレットからテキストオブジェクトを選択し、文字を貼り付けたい箇所をマウスでクリックする。そこでテキストを入力するダイアログが起動するので、テキストタブに出力したい文字列を入力する。【図 5】

日付や名前、データベースから取得する値を設定する場合も、固定文字列と同じようにテキストオブジェクトとして貼り付ける。

また、ダイアログ内で任意の変数を［変数名］と記述することで、文字列をプログラム内の変数として扱える（ソースは後述）。たとえば、［VALUE001］のように設定できる。【図 6】

フッター部備考の下ラインは、破線で設定する。破線で設定するには、該当の線オブジェクトを選択し、「Frame ｜ Style」プロパティを fsDot で設定する。【図 7】

最後に線オブジェクトとテキストオブジェクトを組み合わせて、フォーマットを作成していく。一通り帳票レイアウトの設計が完了すれば、そのままレポートデザイナを「×」ボタンで終了する。

これで帳票フォーマットは作成できたので、次にボタンを押下した際の出力ロジックについて説明する。

2-2. 帳票データの出力

前述のように作成した帳票フォーマットでは、変数として「VALUE001」を宣言しているが、帳票出力時には注意が必要である。変数を宣言している場合には、必ず値を設定しておかねば、実行時にエラーとなる。

そのため、初期化ロジックとして InitVariables という手続きを作成し、文字型であればブランクを設定し、それ以外の型であれば 0 を設定しておくと、値の設定漏れを防げる。【ソース 1】

次に、帳票で実際の変数値をセットする。帳票フォーマット作成時に変数として宣言した［VALUE001］に値をセットする場合は、Script.Variables［(変数名)］を指定する。【ソース 2】

もちろん IBM i や SQLServer などのデータベースから取得した値を設定することも可能である。

データベースや配列を扱う方法については、2014 年発行のミガロ .テクニカルレポート№ 7 にある『FastReport を使用した帳票作成テクニック』で詳しく説明しているので、参考にしていただきたい。

ここまでで、帳票への出力内容が完成したので、次に帳票を画像ファイルとして保存する手法を説明する。

2-3. 帳票画像ファイルの保存

通常、FastReport で作成した帳票は「frxReport1.Print」メソッドを使用し、プリンタへ印刷する。しかし本稿では電子帳票として出力するので、画像ファイルとしての保存方法を説明する。画像ファイルの形式は、一般的に JPEG の使用が多い。

FastReport で帳票を JPEG 画像ファイルとして出力するには、TfrxJPEGExport コンポーネントが使用できる。

まず、TfrxJPEGExport コンポーネントを画面に貼り付ける【図 8】。次に、TfrxJPEGExport コンポーネントにプロパティを設定する。保存する画像品質は、「Resolution」を指定する。【ソース 3】

帳票を出力するメインのロジックは、【ソース 4】のとおりである。画像ファイルは Stream 形式を使って出力する。ポイントは、TfrxJPEGExport の Stream に msJPEG として内部生成した「TMemoryStream」を割り当てる点である。

あとは、TfrxJPEGExport コンポーネントを TfrxReport コンポーネントの Export メソッドを使用して、msJPEG に画像ファイル情報を転送する（【ソース 4】の①）。最後に、転送された画像ファイルを SaveToFile で任意の場所に保存できる（【ソース 4】の②）。これで電子帳票としての出力・保存が完成である。

これにより、購入申請の情報を画面で入力し、「購入申請書を発行」ボタンを押すと、電子帳票として保存するプログラムが作成された。保存したあとは、帳票表示用の画面を起動し、購入申請書の電子帳票を画面表示できる（これも帳票が電子ファイルであるメリットといえる）。【図 9】

また電子帳票でも紙の帳票と同じように、押印を必要とすることが非常に多い。そこで次に、電子データ印を作成し、この電子帳票に押印するテクニックを説明する。

【図 9】のプログラム例では、表示された電子帳票の下にある「確認印」ボタンを押すことで、電子データ印を生成し、マウスで好きな位置に貼り付けられるようにしている。

3.電子データ印の作成

電子データ印といっても、内容としてはデータ印を画像として作成するだけである。

電子データ印作成の手順としては、まずデータ印の枠、文字列を画像ファイルとして作成する。そして 2 つの画像ファイルを 1 つに合成することで、電子データ印として完成する。この手法で電子データ印を作成すれば、電子帳票上での押印に使用できる。

3-1　電子データ印枠の作成

最初に、2 つの TBitmap コンポーネントを配置する。1 つは電子データ印の枠用（bmpBase1）、もう 1 つは電子データ印に出力する出力日、名前、所属部署など文字列用（bmpOver1）として使用する。

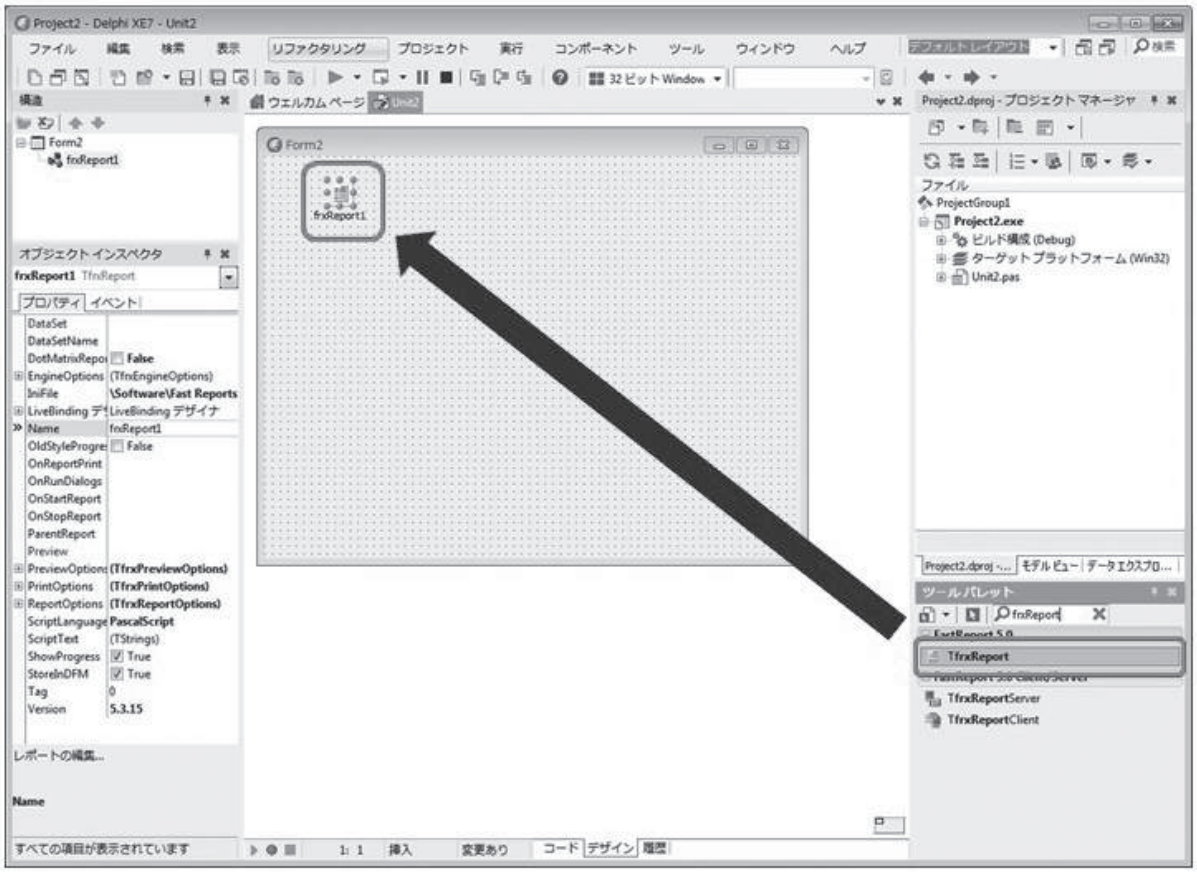
そして TImage コンポーネントを配置し、マウス操作で表示する電子データ印のプレビュー用（imgEditor）として使用する。

bmpBase1 と bmpOver1 はプログラム内で生成し、imgEditor は TImage コンポーネントを画面に貼り付ける。【図 10】

まずは電子データ印の枠色、線の太さを設定する。Ellipse メソッドを使用し、bmpBase1 に円を描画する。さらに、なかにある 2 本線を追加で描画する。【ソース 5】

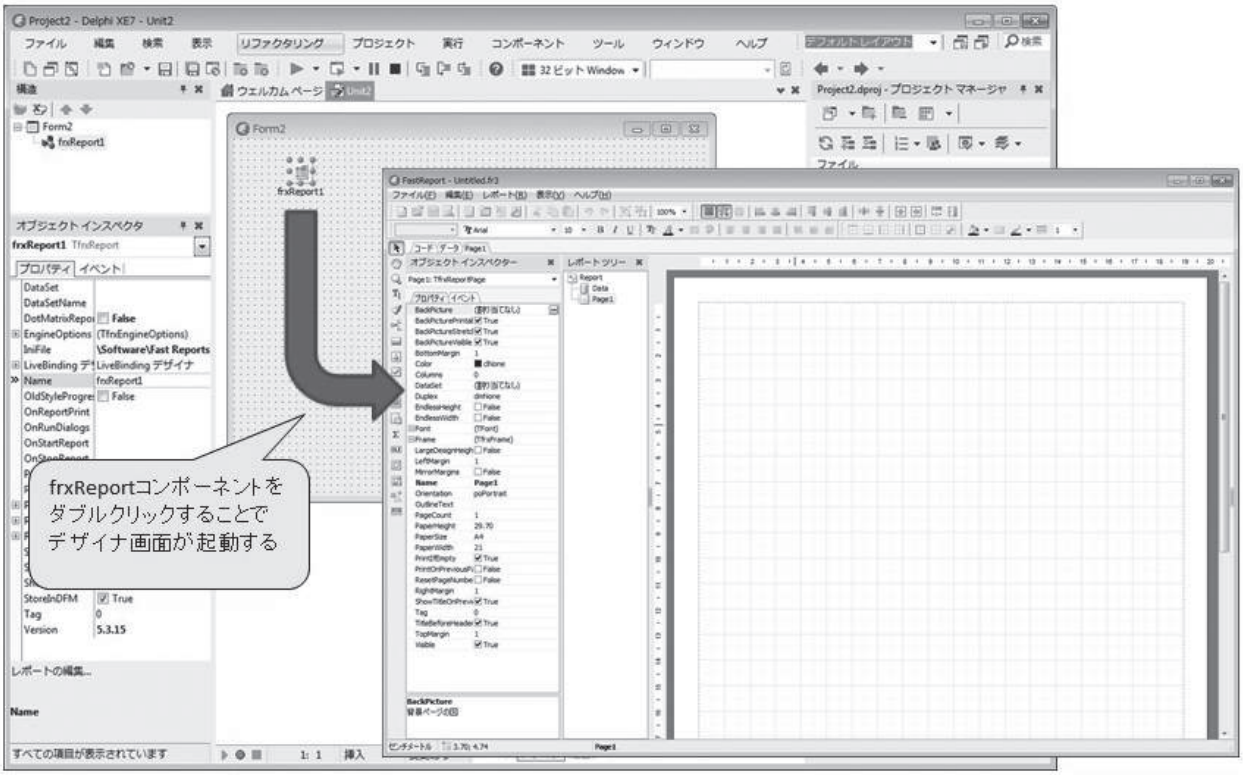
これで、電子データ印の枠が完成である。次に、この枠のなかに出力する文字列の作成ポイントを説明する。

図2



TfrxReportコンポーネントを貼り付ける

図3



レポートデザイナの起動

3-2　出力文字列の作成

電子データ印内に出力する文字列は、内容としてはただの文字であるが、画像化するため、フォントサイズや幅などの調整が重要になる。出力する文字列の内容は、上段に所属部署、中段に日付、下段に名前である。【図 11】

まず、上段に表示する所属部署の文字列を調整する。長い部署名になると、電子データ印の枠からはみ出すので、文字列の長さによってフォントサイズを変更する必要がある。

このプログラムでは、半角 8 文字まで出力できるように調整している。半角 1 ～ 6 文字ならフォントサイズ 15、半角 7 ～ 8 文字ならフォントサイズ 11 を設定する。これは電子データ印のサイズや、出力する文字列の想定によって変わってくる。

フォントサイズを決定したあとは、Canvas.TextOut で文字列を出力する。第 1 引数は X 座標の位置、第 2 引数は Y 座標の位置、第 3 引数は出力する文字列を指定する。X 座標の位置は、出力する文字列の長さによって調整する。【ソース 6】

次は、中段に表示する日付の文字列である。日付については、「YYYY/MM/DD」形式の半角 10 文字固定で出力するので、フォントサイズの調整は不要である。【ソース 7】

最後は、下段に表示する名前の文字列を調整する。電子データ印に表示する名前は、所属と同様にフォントサイズを調整し、bmpOver1 の Canvas に出力する。【ソース 8】

これで電子データ印に出力する文字列の調整が完了である。

3-3　電子データ印画像の作成

ここまでの作業で、 電子データ印の枠（bmpBase1）と電子データ印の文字列（bmpOver1）が準備できた。この 2 つを合成することで、電子データ印が完成する。【図 12】

画像を合成するには、StretchDraw メソッドを使用する。画像ファイルの合成は、単純に bmpBase1 の Canvas に bmpOver1 を描画するだけである。【ソース 9】

これで 2 つの画像ファイルを 1 つの画像ファイルに合成できた。電子データ印の画像ファイルの完成である。

補足として、この電子データ印を画面上で押印する際に、マウスでわかりやすく表示する方法を説明する。

まずプレビュー用 TImage コンポーネント（imgEditor）の Picture プロパティに電子データ印を読み込ませる。ここからはマウスが画面の帳票内にある場合のみ、マウス位置に電子データ印を表示する動作を実装していく。【図 13】

このマウスの動作は MouseMove、MouseLeave イベントを使うことで、簡単に実装できる。【ソース 10】

これで電子データ印の画像ファイルと、押印する際の画面動作プログラムを実装できた。

4.電子帳票への押印機能実装

2. で電子帳票の画像ファイルを作成し、3. では電子帳票に押印する電子データ印の画像ファイルを作成した。ここからは、この 2 つの画像ファイルを合成し、電子帳票上での押印を実現していく。

4-1 画像合成の準備

画面の帳票上でのマウス動作処理までを作成したが、さらにクリックした際、その電子帳票と電子データ印の画像ファイルを合成する処理を行う。【図 14】

この処理のために、2 つの TBitmap コンポーネントと 1 つの TJPEGImage コンポーネントを用意する。2 つの TBitmap コンポーネントのうち、1 つは電子帳票用（bmpBase2）、もう 1 つは電子データ印用（bmpOver2）、また TJPEGImage コンポーネントは JPEG ファイルとして取り扱うために使用する（jpgBase2）。

はじめに、電子帳票ファイルが JPEG 形式かどうかをチェックする。JPEG 形式だった場合には、Bitmap へ変換するために jpgBase2 に読み込み、その後、合成用に bmpBase2 へ再度読み込ませる。Bitmap 形式の場合は、直接 bmpBase2 に読み込ませる。【ソース 11】

JPEG、Bitmap 以外の画像ファイルを扱う場合は、いったん WICImage 型に読み込み直し、bmpBase2 にセットすることで対応する。

次に bmpBase2 の幅、高さを電子帳票ファイルの高さに設定し、押印する電子データ印を設定する。bmpOver2 は Assign メソッドを使用すれば、電子データ印画像を設定できる。【ソース 12】

4-2　電子押印機能の実装

最後に、bmpBase2 の StretchDraw メソッドを使って、bmpOver2 の押印を実装する。これは電子データ印自体を作成するときと同じ手法で、画像ファイル同士を合成する。

あとは押印した電子帳票を出力する TImage コンポーネントで読み込めば、処理が完了である。【ソース 13】【図 15】

これで電子帳票の作成、およびそれを応用した電子データ印の押印機能を実装できた。ここまでの処理ロジックは、【ソース 14】に実装ソース例をまとめているので、参考にしていただきたい。

社内システムでは、このような電子帳票を申請データとして保存し、ワークフローとして承認者がさらに押印する仕組みを構築することもできる。

5.まとめ

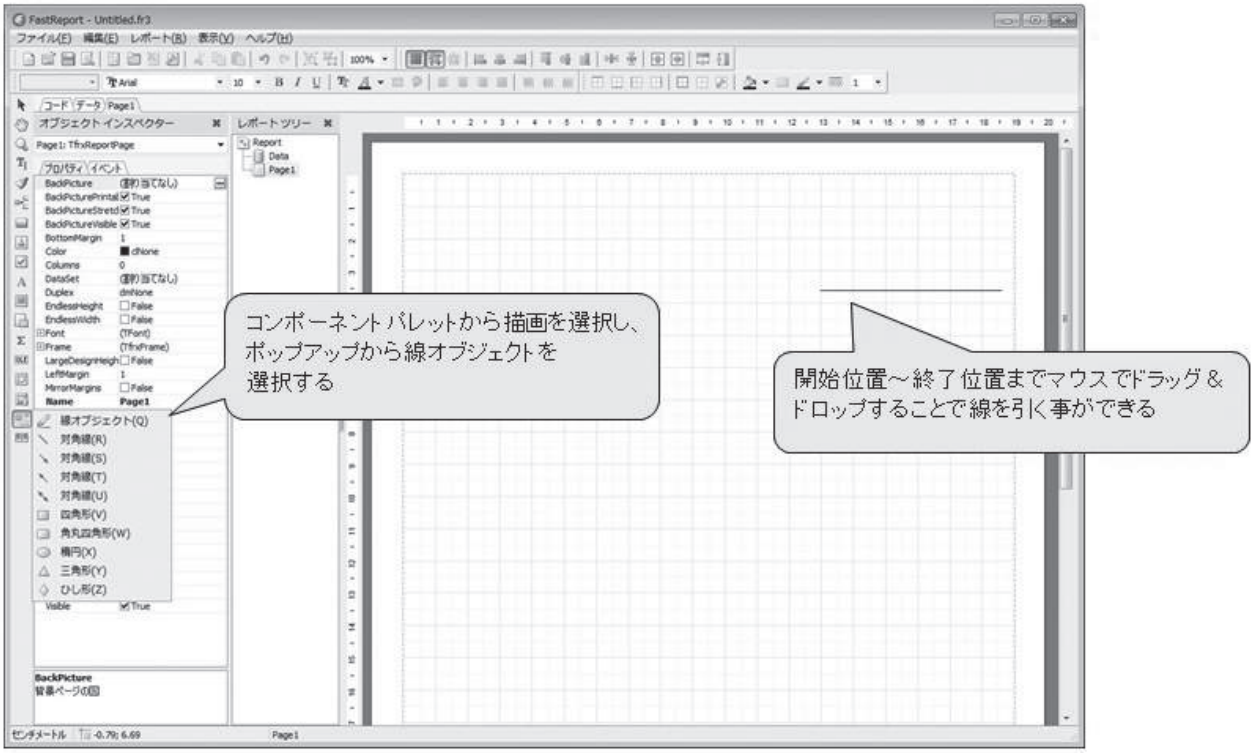
本稿では FastReport で電子帳票を作成する方法や、データ印画像の作成方法、電子帳票にデータ印を押印する方法を説明してきた。ここで紹介したテクニックを利用すれば、売上伝票や請求書、また商品画像等の電子帳票の作成も可能である。

画像や描画は一般には難しいプログラム分野であるが、FastReport が画像出力形式に対応しているので、Delphi/400 では簡単に電子帳票を実装できる。

冒頭でも述べたとおり、電子帳票化にはメリットも多く、FastReport では開発にも手間がかからない。帳票を開発する際には、電子帳票での出力も、主機能の 1 つとして組み込む価値が十分にある技術といえる。

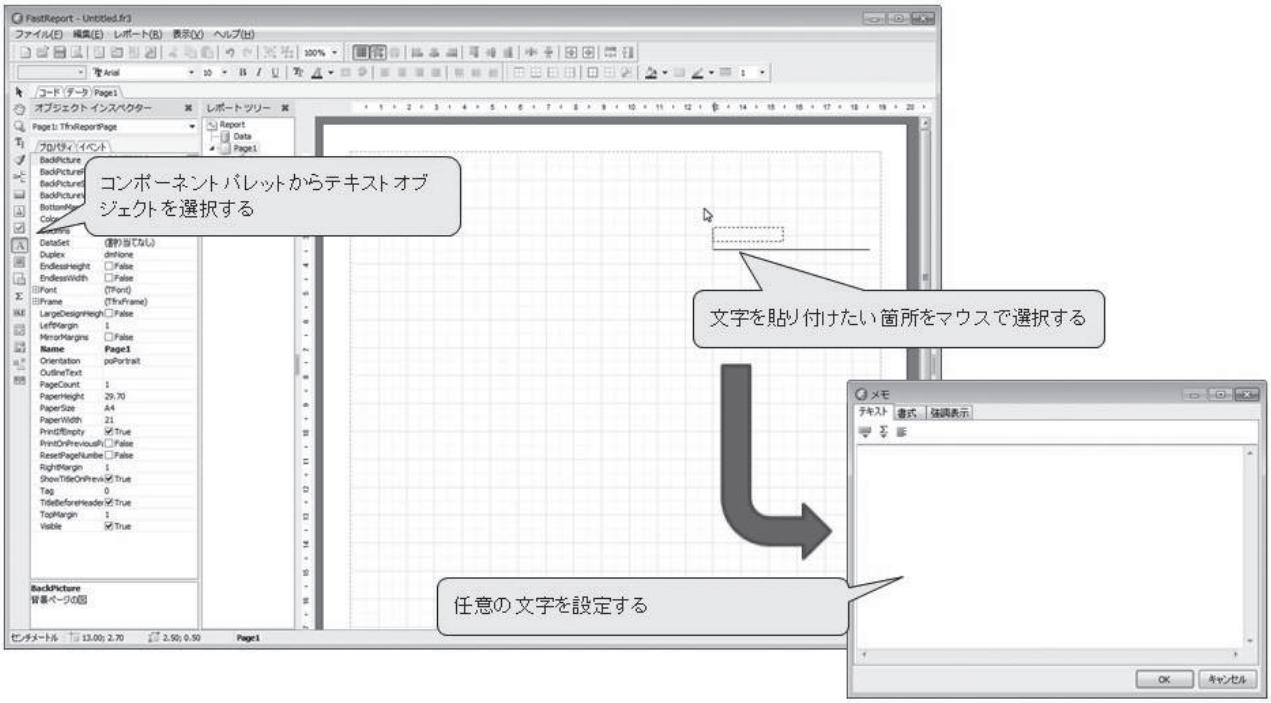
M

図4



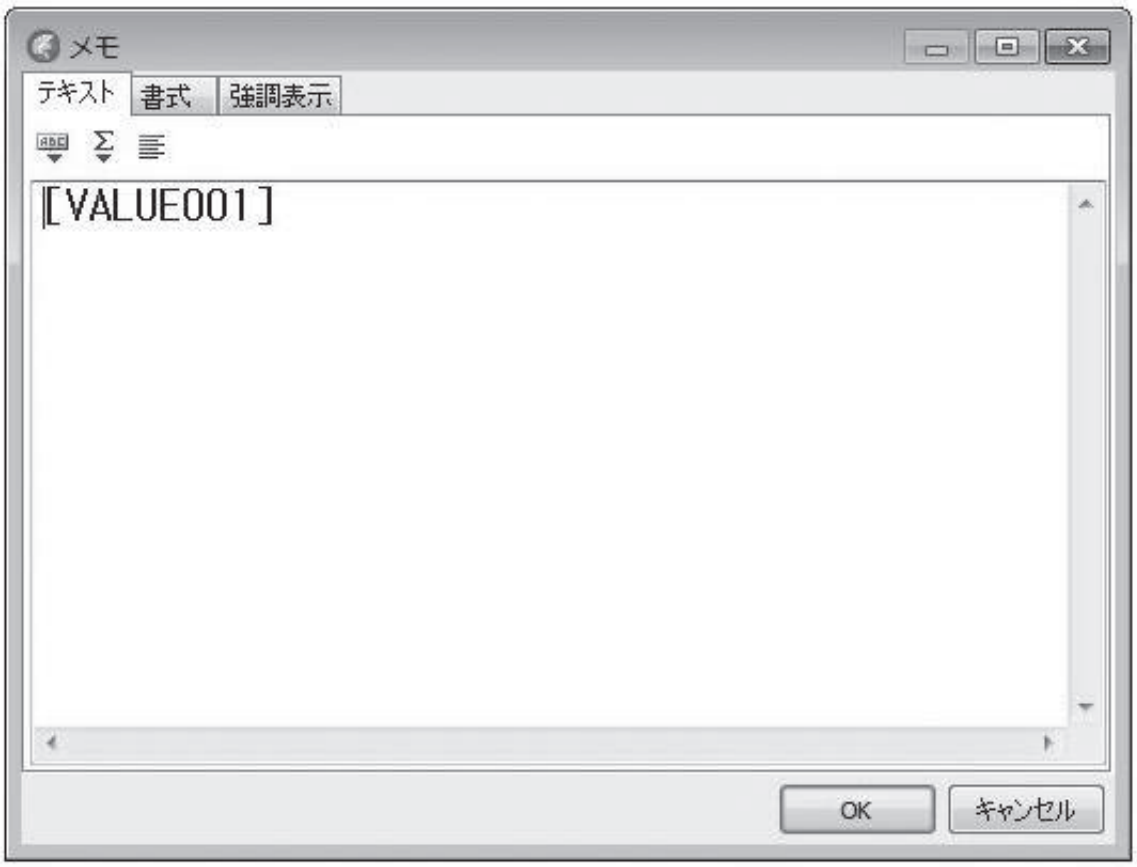
帳票フォーマットに線を引く

図5



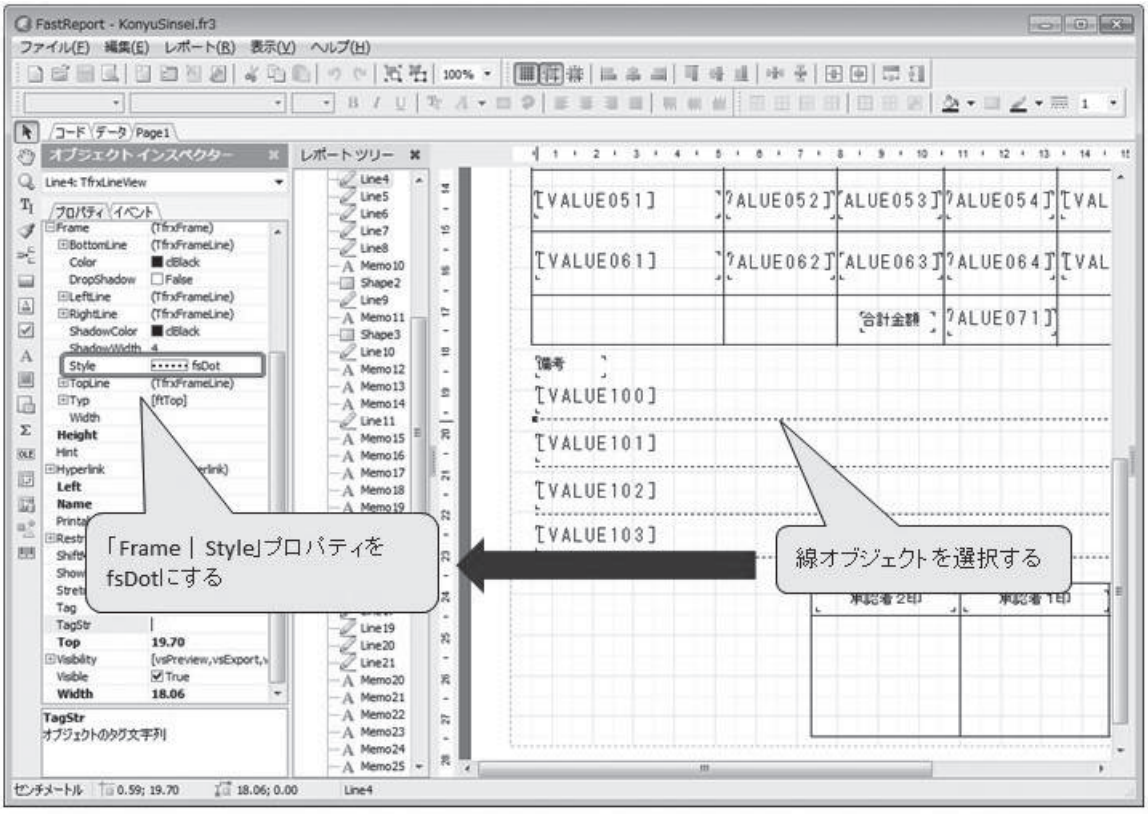
帳票フォーマットに文字を書く

図6



変数の定義

図7



破線の設定

ソース1

```
{*******************************************************************************
 目的：FastReportの変数の初期化
 引数：Afrx -  TfrxReportコンポーネント
 戻値：
*******************************************************************************}
procedure TfrmReportSample002.InitVariables(Afrx: TfrxReport);
var
  i: Integer;
begin
  //帳票フォーマット内で宣言した変数([VALUE001]等)に値をセットしない場合、
  //エラーとなるため、処理の最初にブランクもしくは0を初期セットする

  // 変数の初期化
  for i := 0 to Afrx.ComponentCount - 1 do
  begin
    // TfrxMemoviewだった場合
    if Afrx.Components[i] is TfrxMemoView then
    begin
      if (Pos('[', TfrxMemoView(Afrx.Components[i]).Memo.Text) > 0) and
         (Pos(']', TfrxMemoView(Afrx.Components[i]).Memo.Text) > 0) then
      begin
        // DisplayFormat.Kindが"fkText"だった場合、文字型と判断する
        if TfrxMemoView(Afrx.Components[i]).DisplayFormat.Kind = fkText then
        begin
          // 初期値としてブランクをセットする
          Afrx.Script.Variables[
          Copy(TfrxMemoView(Afrx.Components[i]).Memo.Text,
            Pos('[', TfrxMemoView(Afrx.Components[i]).Memo.Text) + 1,
            Pos(']', TfrxMemoView(Afrx.Components[i]).Memo.Text) -
            (Pos('[', TfrxMemoView(Afrx.Components[i]).Memo.Text) + 1) )] := '';
        end
        // DisplayFormat.Kindが"fkText"以外だった場合、数値型と判断する
        else
        begin
          // 初期値として"0"をセットする
          Afrx.Script.Variables[
          Copy(TfrxMemoView(Afrx.Components[i]).Memo.Text,
            Pos('[', TfrxMemoView(Afrx.Components[i]).Memo.Text) + 1,
            Pos(']', TfrxMemoView(Afrx.Components[i]).Memo.Text) -
            (Pos('[', TfrxMemoView(Afrx.Components[i]).Memo.Text) + 1) )] := 0;
        end;
      end;
    end;
  end;
end;
```

Initvariablesﾒｿｯﾄﾞを作成し、宣言した変数（[VALUE001]等）を初期化する

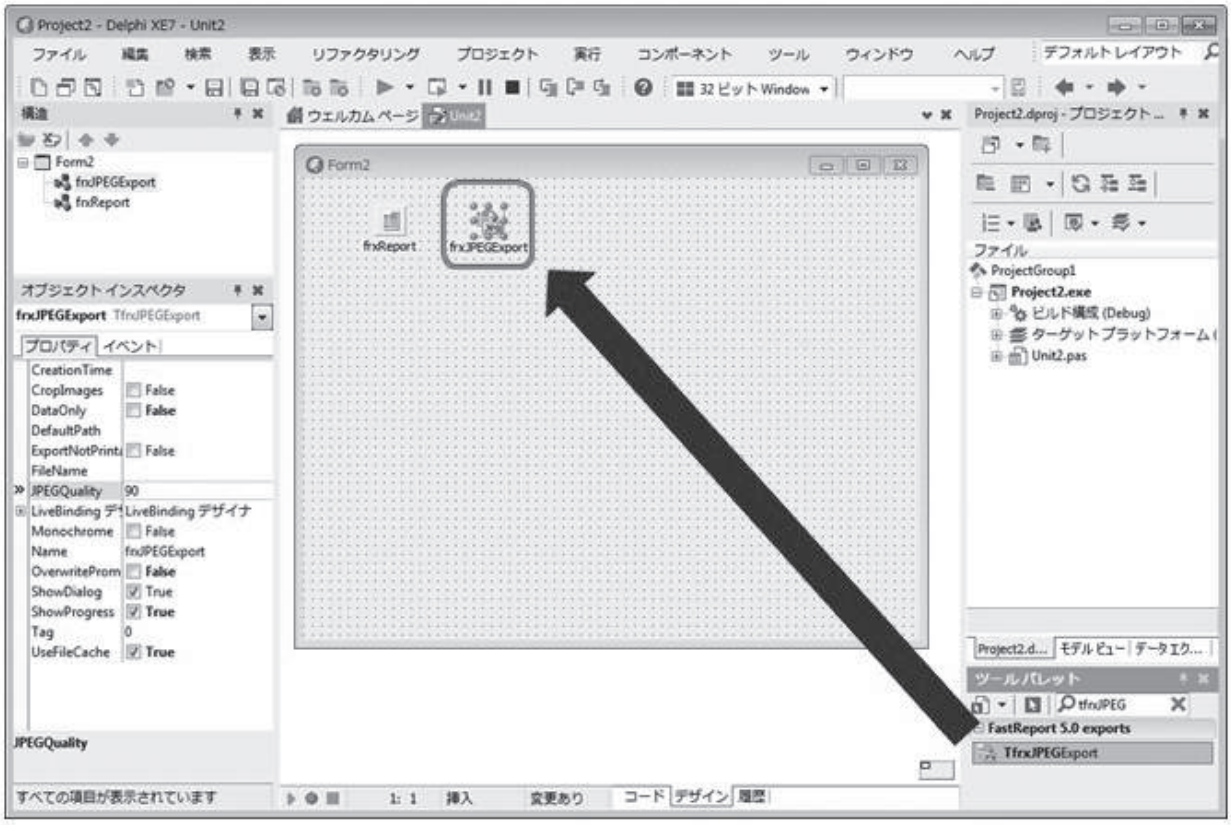
ソース2

```
{ ***** データを帳票にセット ***** }
//<ヘッダー>
frxReport1.Script.Variables['VALUE001'] := edtSinseibi.Text;    //申請日
frxReport1.Script.Variables['VALUE002'] := edtKonyuYotei.Text; //購入予定先
frxReport1.Script.Variables['VALUE003'] := edtSyozoku.Text;     //所属
frxReport1.Script.Variables['VALUE004'] := edtName.Text;        //氏名
```

各値は画面に配置したEditから取得する

変数に値をセットする

図8



TfrxJPEGExportコンポーネントを貼り付ける

ソース3

```
// JPEG変換用コンポーネントのプロパティを設定する
frxJPEGExport.Resolution    := 96;     // 解像度(dpi)
```

TJPEGExportコンポーネントのプロパティを設定

ソース4

```
{*******************************************************************************
 目的：購入申請書発行ボタン押下時
 引数：
 戻値：
*******************************************************************************}
procedure TfrmReportSample002.BitBtn5Click(Sender: TObject);
var
  sImagePath: String;      // 画像パス用
  msJPEG: TMemoryStream;   // JPEG作成用メモリストリーム
begin

  //帳票保存先フォルダの設定
  sImagePath := 'C:¥TechnicalReport¥Report¥';

  // 変数の初期化
  InitVariables(frxReport1);

  { ***** データを帳票にセット ***** }
  //<ヘッダー>
  frxReport1.Script.Variables['VALUE001'] := edtSinseibi.Text;    //申請日
  frxReport1.Script.Variables['VALUE002'] := edtKonyuYotei.Text; //購入予定先
  frxReport1.Script.Variables['VALUE003'] := edtSyozoku.Text;     //所属
  frxReport1.Script.Variables['VALUE004'] := edtName.Text;        //氏名

  //<フッター>
  frxReport1.Script.Variables['VALUE100'] := edtBiko1.Text;       //備考１行目
  frxReport1.Script.Variables['VALUE101'] := edtBiko2.Text;       //備考２行目
  frxReport1.Script.Variables['VALUE102'] := edtBiko3.Text;       //備考３行目
  frxReport1.Script.Variables['VALUE103'] := edtBiko4.Text;       //備考４行目

  { ***** 帳票画像作成 ***** }
  frxReport1.PrepareReport(True); // レポートに区切りを入れる
  // 保存先ディレクトリが存在しない場合、作成する
  if not(DirectoryExists(sImagePath)) then
  begin
    ForceDirectories(sImagePath);
  end;

  //帳票の画像保存メソッドの呼び出し
  try
    // JPEG作成用メモリストリーム作成
    msJPEG  := TMemoryStream.Create;
    try
      // JPEG変換用コンポーネントのプロパティを設定する
      frxJPEGExport1.Resolution    := 75;       // 解像度(dpi)

      // JPGに変換
      frxJPEGExport1.Stream := msJPEG;
      frxReport1.Export(frxJPEGExport1);

      // 保存
      msJPEG.SaveToFile(sImagePath+'SampleReport.jpg');

      // 読み込みボタン押下時用に保存先を内部変数に記憶
      sIMAGE := sImagePath+'SampleReport.jpg';

    finally
      // コンポーネントの解放
      FreeAndNil(msJPEG);
    end;
  except
    // 例外処理
    on E: Exception do
    begin
      raise;
    end;
  end;

  //データ印押印画面呼出
  frmReportSample001 := TfrmReportSample001.Create(Self);
  try
    frmReportSample001.FIMAGEPATH := sIMAGE;  // 帳票画像のパスを渡す
    frmReportSample001.ShowModal;             // データ印押印画面を起動する
  finally
    frmReportSample001.Release;
  end;


end;
```

InitVariables詳細は
【ソース2】参照

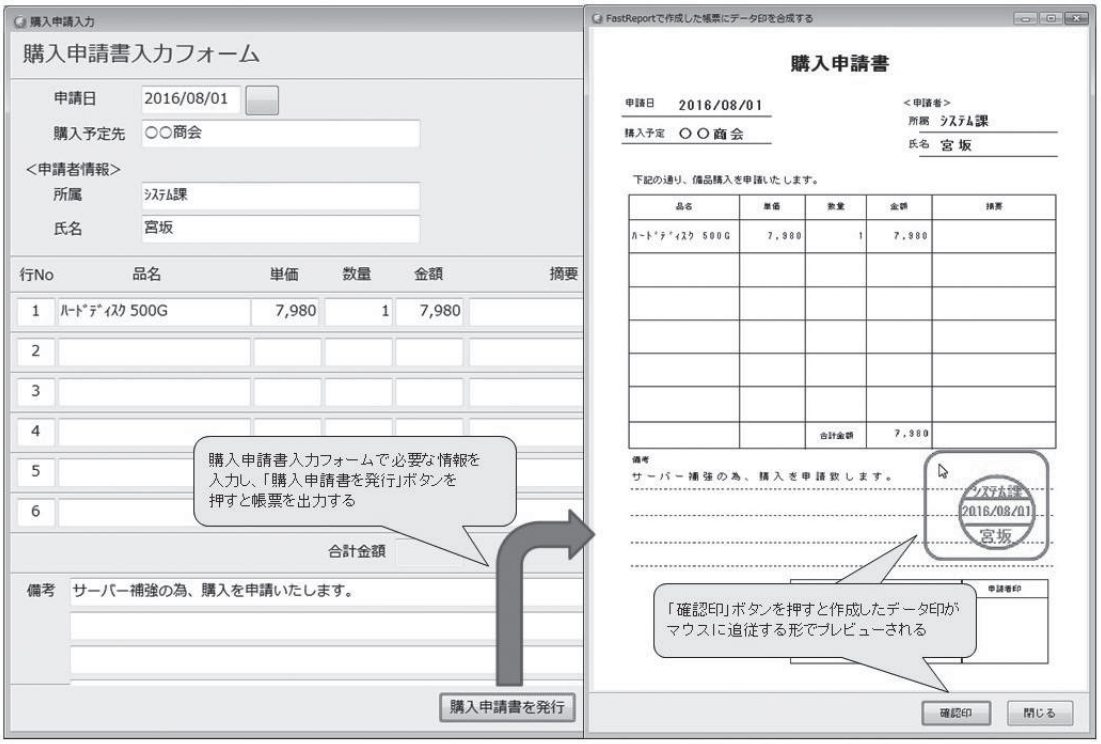
明細のデータセットについては省略

①

②

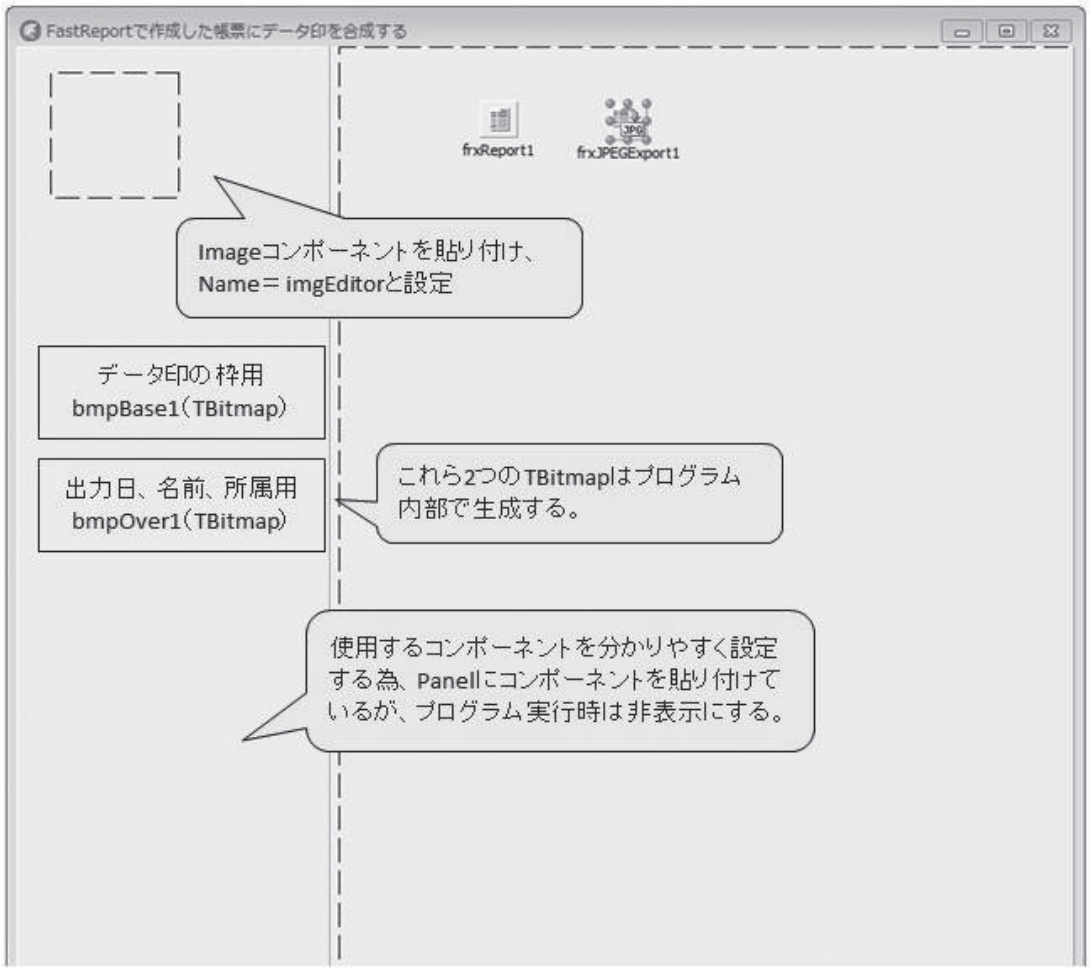
帳票画像ファイルを作成したら
データ印押印画面を呼び出す

図9



購入申請書入力フォーム

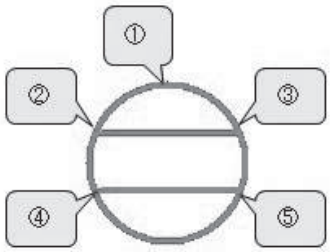
図10



購入申請書表示用子画面のコンポーネント配置

ソース5

```
bmpBase1.Canvas.Pen.Color := clRed;          // 枠線の色
bmpBase1.Canvas.Pen.Width := 4;              // 枠線の太さ
bmpBase1.Canvas.Ellipse(3, 3, 100, 100); // ①円のサイズ
bmpBase1.Canvas.MoveTo(10, 33);              // ②上の横線開始位置
bmpBase1.Canvas.LineTo(95, 33);              // ③上の横線終了位置
bmpBase1.Canvas.MoveTo(10, 68);              // ④下の横線開始位置
bmpBase1.Canvas.LineTo(95, 68);              // ⑤下の横線終了位置
```



データ印の枠(bmpBase1)の設定

図11

出力文字(bmpOver1)の設定

ソース6

```
//所属部署部分
//sSyozoku＝購入申請書入力フォームで入力した所属
iWidth := Length(AnsiString(sSyozoku));

if (iWidth <= 6) then
  bmpOver1.Canvas.Font.Size := 15
else
  bmpOver1.Canvas.Font.Size := 11;

bmpOver1.Canvas.Font.Color := clRed;
bmpOver1.Canvas.Font.Style := [fsBold];
case iWidth of // 文字列の長さによって位置を決定
  1: bmpOver1.Canvas.TextOut(48, 15, sSyozoku);
  2: bmpOver1.Canvas.TextOut(42, 15, sSyozoku);
  3: bmpOver1.Canvas.TextOut(36, 15, sSyozoku);
  4: bmpOver1.Canvas.TextOut(31, 15, sSyozoku);
  5: bmpOver1.Canvas.TextOut(25, 15, sSyozoku);
  6: bmpOver1.Canvas.TextOut(20, 15, sSyozoku);
  7: bmpOver1.Canvas.TextOut(14, 15, sSyozoku);
  else
    bmpOver1.Canvas.TextOut(10, 15, sSyozoku);
end;
```

文字列が1～6文字以下なら、FontSize＝15
文字列が7文字以上なら、FontSize＝11

文字列の長さによってフォントサイズを変更する(所属部分)

ソース7

```
//日付部分
//sToday＝購入申請書入力フォームで入力した申請日
bmpOver1.Canvas.Font.Size   := 12;
bmpOver1.Canvas.Font.Color  := clRed;
bmpOver1.Canvas.TextOut(7, 42, sToday);
```

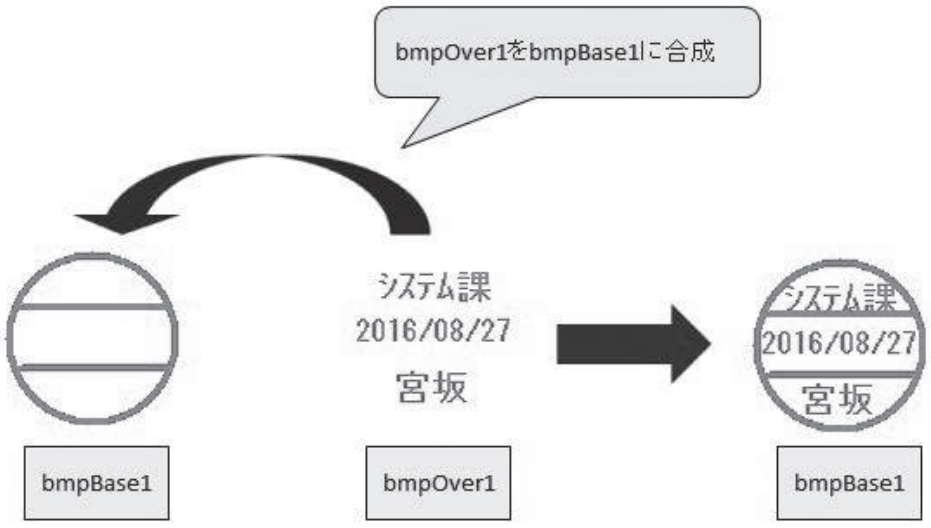
日付部分の設定

ソース8

```
//名前部分
//sMyName＝購入申請書入力フォームで入力した名前
iWidth := Length(AnsiString(sMyName));

if (iWidth <= 6) then
  bmpOver1.Canvas.Font.Size := 15
else
  bmpOver1.Canvas.Font.Size := 11;

bmpOver1.Canvas.Font.Color  := clRed;
bmpOver1.Canvas.Font.Style  := [fsBold];
case iWidth of // 文字列の長さによって位置を決定
  1: bmpOver1.Canvas.TextOut(63, 73, sMyName);
  2: bmpOver1.Canvas.TextOut(51, 73, sMyName);
  3: bmpOver1.Canvas.TextOut(39, 73, sMyName);
  4: bmpOver1.Canvas.TextOut(29, 73, sMyName);
  5: bmpOver1.Canvas.TextOut(19, 73, sMyName);
  6: bmpOver1.Canvas.TextOut(9,  73, sMyName);
  7: bmpOver1.Canvas.TextOut(5,  73, sMyName);
  else
    bmpOver1.Canvas.TextOut(-3, 73, sMyName);
end;
```

名前部分の設定

図12



データ印の合成イメージ

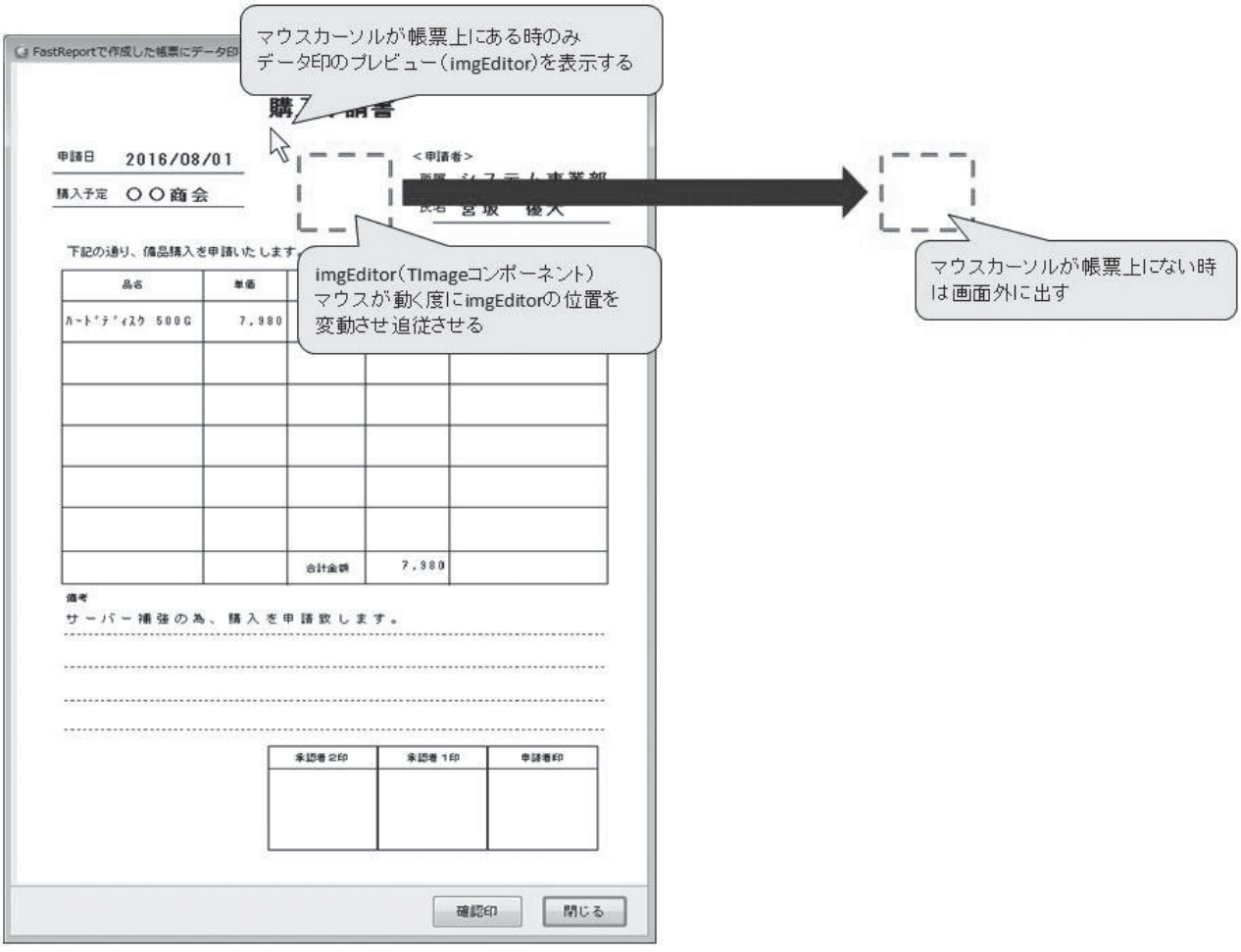
ソース9

StretchDraw関数を使用して
bmpOver1をbmpBase1に貼り付ける

```
//データ印の枠と文字列を融合する
bmpBase1.Canvas.StretchDraw(Rect(0, 0, bmpOver1.Width, bmpOver1.Height), bmpOver1);
```

画像の合成

図13



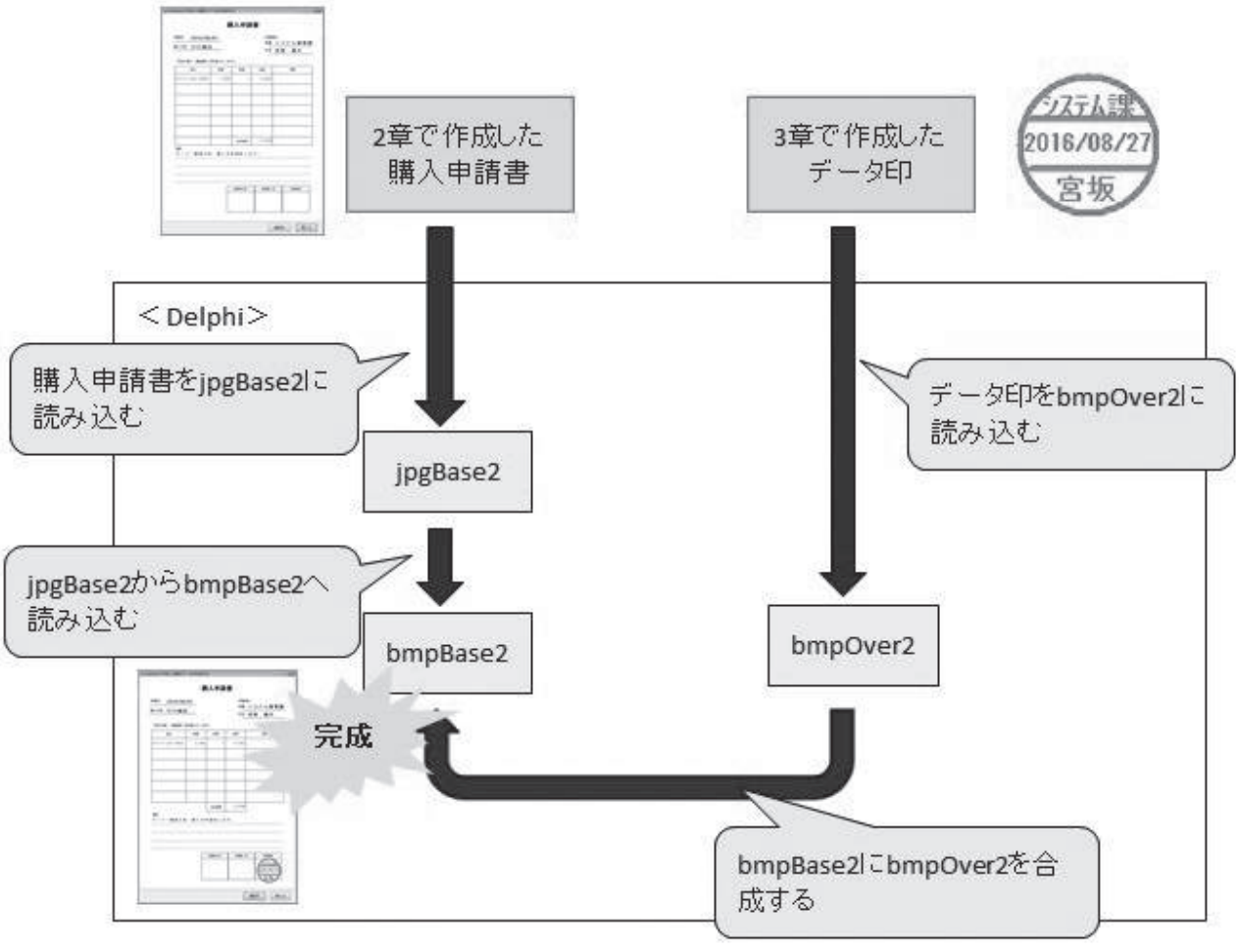
マウスカーソルに追従するプレビューイメージ

ソース10

```
{*************************************************************************************
 目的：画像マウス移動中処理
 引数：
 戻値：
*************************************************************************************}
procedure TfrmReportSample001.Image1MouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
  // bDateIn：プログラム内部データ印を作成したかどうかをBoolean型で保持
  if bDateIn then
  begin
    //データ印を作成した場合のみマウスに追従するImageコンポーネントを表示する
    imgEditor.Visible := True;

    //マウス移動時に追従するデータ印の位置調整
    imgEditor.Left    := X + 10;
    imgEditor.Top     := Y + 10;
  end;

end;

{*************************************************************************************
 目的：画像マウスフォーカス喪失時処理
 引数：
 戻値：
*************************************************************************************}
procedure TfrmReportSample001.Image1MouseLeave(Sender: TObject);
begin
  imgEditor.Visible := False;
end;
```

MouseMove、MouseLeaveイベント

図14



帳票とデータ印の画像合成イメージ

ソース11

```
// ベース画像の型を判別
if (Image1.Picture.Graphic is TJPEGImage) then
begin
  // ベース画像がJPEGの場合、一旦JPEG変数に読み込ませた上でbmpBase2に転送する
  jpgBase2.Assign(Image1.Picture.Graphic);
  bmpBase2.Assign(jpgBase2);
end
else
if (Image1.Picture.Graphic is TBitmap) then
begin
  // ベース画像がBitmapの場合、そのまま転送する
  bmpBase2.Assign(Image1.Picture.Graphic);
end;
```

帳票画像をjmpBase2を経由し、bmpBase2に読み込む

ソース12

```
// 上書きする画像を読み込む
bmpOver2.Assign(imgEditor.Picture.Graphic);
```

データ印画像をbmpOver2に読み込む

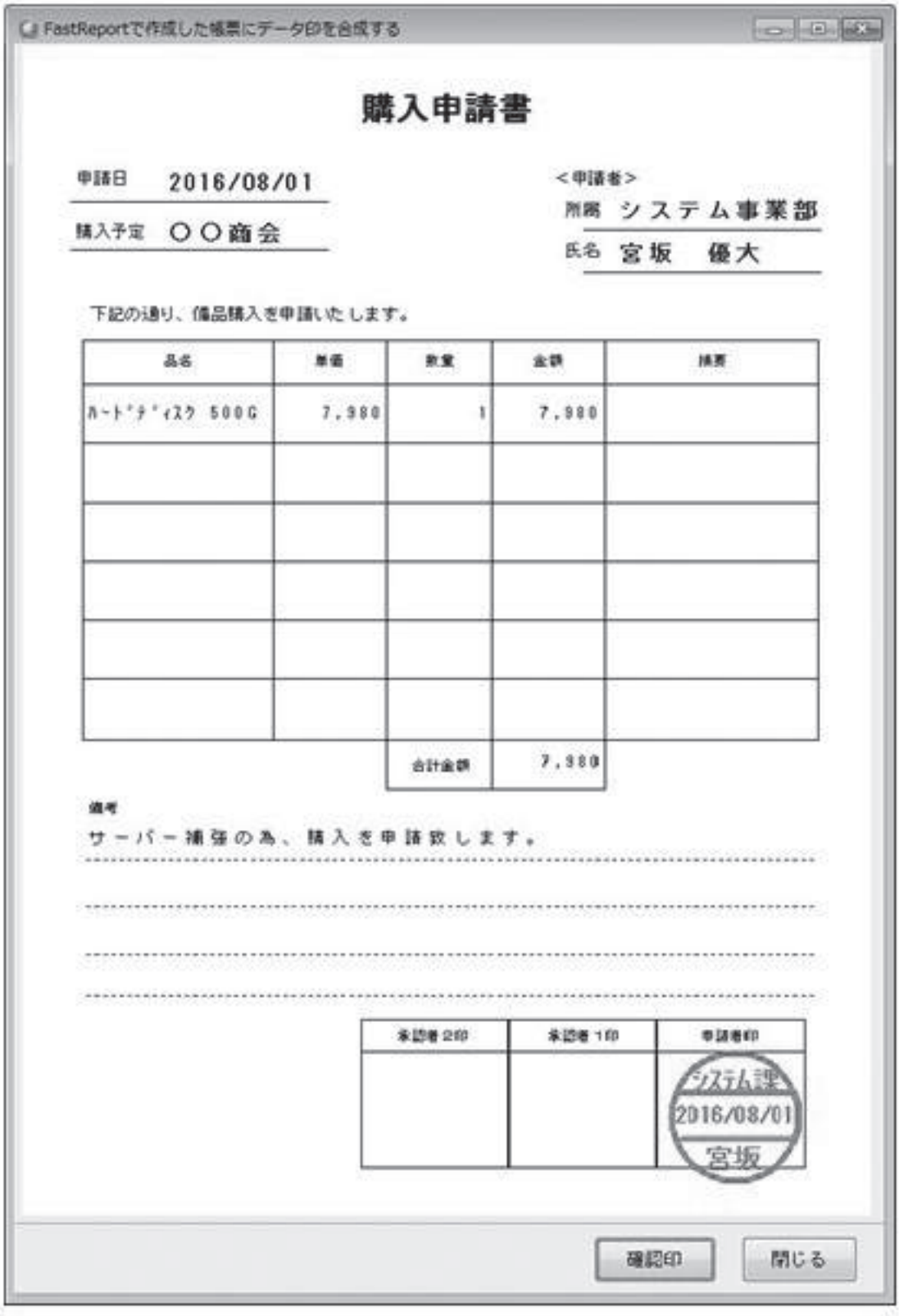
ソース13

```
// ベース画像に上書き画像を描画する
bmpBase2.Canvas.StretchDraw(Rect(X, Y, X+99, Y+102), bmpOver2);

// 合成したbmpBase2をImage1(購入申請書画像)に読み込む
Image1.Picture.Assign(bmpBase2);
```

マウスクリック位置と画像貼り付けサイズをRectの第3引数と第4引数で調整する。

帳票画像(bmpBase2)にデータ印画像(bmpOver2)を合成する

図15



完成した購入申請書

ソース14

```
{*****************************************************************************
 目的：画像マウス押下時処理
 引数：
 戻値：
*****************************************************************************}
procedure TfrmReportSample001.Image1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  sImagePath: String;      // 画像パス用
  bmpBase2: TBitmap;       // ベースとなる画像
  jpgBase2: TJPEGImage;    // ベース画像をjpg形式で保管
  bmpOver2: TBitmap;       // ベースに上書きする画像
begin
  // データ印を画像に合成する
//  ImageFusionEx(Image1, imgEditor, iHX, iHY);
  bmpBase2 := TBitmap.Create;      // ベース画像(Btimap)
  jpgBase2 := TJPEGImage.Create;   // ベース画像(JPEG)
  bmpOver2 := TBitmap.Create;      // 上書きする画像(Bitmap)
  try
    // ベース画像の縦・横幅を設定
    bmpBase2.Width  := Image1.Picture.Width;
    bmpBase2.Height := Image1.Picture.Height;
    // 上書きする画像の縦・横幅を設定
    bmpOver2.Width  := imgEditor.Picture.Width;
    bmpOver2.Height := imgEditor.Picture.Height;

    // ベース画像の型を判別
    if (Image1.Picture.Graphic is TJPEGImage) then
    begin
      // ベース画像がJPEGの場合、一旦JPEG変数に読み込ませbmpBase2に転送する
      jpgBase2.Assign(Image1.Picture.Graphic);
      bmpBase2.Assign(jpgBase2);
    end
    else
    if (Image1.Picture.Graphic is TBitmap) then
    begin
      // ベース画像がBitmapの場合、そのまま転送する
      bmpBase2.Assign(Image1.Picture.Graphic);
    end;

    //フルカラーを指定
    bmpBase2.PixelFormat := pf24bit;
    bmpOver2.PixelFormat := pf24bit;
    // CanvasにStretchDrawで描画する
    bmpBase2.Canvas.StretchDraw
      (Rect(0, 0, (bmpBase2.Width-1), (bmpBase2.Height-1)), bmpBase2);

    // 上書きする画像を読み込む
    bmpOver2.Assign(imgEditor.Picture.Graphic);

    // ベース画像に上書き画像を描画する
    bmpBase2.Canvas.StretchDraw(Rect(X, Y, X+99, Y+102), bmpOver2);

    // 合成したbmpBase2をImage1(購入申請書画像)に読み込む
    Image1.Picture.Assign(bmpBase2);

  finally
    // コンポーネントの解放
    FreeAndNil(bmpOver2);
    FreeAndNil(bmpBase2);
    FreeAndNil(jpgBase2);
  end;

  //帳票保存先フォルダの設定
  sImagePath := 'C:¥TechnicalReport¥Report¥';

  { ***** 帳票画像作成 印刷用 ***** }
  // 保存先ディレクトリが存在しない場合、作成する
  if not(DirectoryExists(sImagePath)) then
  begin
    ForceDirectories(sImagePath);
  end;

  Image1.Picture.SaveToFile(sImagePath+'Tyohyo.jpg');

end;
```