Manual de Engenharia de Software

Análise Detalhada do Gerador de Etiquetas

Documentação Técnica

2 de dezembro de 2025

Sumário

Capítulo 1

Introdução e Arquitetura

Este documento analisa as 600+ linhas de código do projeto **Gerador de Etiquetas** desenvolvido em Delphi. O objetivo é fornecer uma compreensão profunda para que qualquer desenvolvedor possa manter ou recriar o sistema.

1.1 O Problema Resolvido

O sistema resolve três problemas críticos de impressão de etiquetas:

- **Flexibilidade:** Permite alterar o tamanho do papel em tempo de execução (dinâmico).
- **Performance:** Evita múltiplas consultas ao banco de dados para gerar cópias.
- **Interface:** Garante que a tela funcione em monitores de qualquer tamanho (Responsividade).

1.2 A Estrutura da Unit (UnitPrincipal)

Todo arquivo `.pas` no Delphi é dividido em duas seções principais:

1. **Interface:** Onde declaramos "quem somos"(quais bibliotecas usamos, quais componentes existem na tela, quais funções estão disponíveis).
2. **Implementation:** Onde escrevemos a lógica real ("como funciona").

Capítulo 2

Análise da Seção Interface

2.1 Bibliotecas (Uses)

A cláusula `uses` importa funcionalidades do Windows e do Delphi.

- **System.IniFiles:** Essencial. Permite ler e gravar arquivos `.ini` para salvar as configurações do usuário (tamanho da etiqueta, última impressora usada).
- **FireDAC.*:** Conjunto de bibliotecas para conexão de alta performance com Banco de Dados (SQL Server).
- **ppReport, ppBands, etc.:** Bibliotecas do *ReportBuilder*, o motor de relatórios usado para desenhar a etiqueta.

2.2 A Classe do Formulário

A classe `TGeradorEtiquetas` representa a janela. Tudo o que está dentro dela é um componente ou um método.

2.2.1 Componentes de Dados (Invisíveis)

```
1    Conexao: TFDConnection;        // O 'cabo' que liga ao SQL Server
2    FDQuery2: TFDQuery;            // O carteiro que leva e traz dados
3    mtEtiquetas: TFDMemTable;      // A 'Mesa de Trabalho' na Memória RAM
```

Conceito Chave: A `mtEtiquetas` (MemTable) é o coração da performance. Ela atua como um ”banco de dados falso”na memória RAM. Em vez de perguntar ao servidor 1000 vezes sobre um produto, perguntamos 1 vez e copiamos o resultado 1000 vezes para esta tabela.

2.2.2 Métodos Personalizados (Private)

Aqui declaramos funções que não existem nativamente no Delphi, criadas especificamente para este projeto:

```
1    procedure PosicionarCampo(...);
2    procedure AjustarLayoutGeral;
```

```
3     procedure ConfigureReportForLabels(...);
```

Estas funções serão explicadas detalhadamente nos próximos capítulos.

Capítulo 3

Lógica de Inicialização e Layout

3.1 O Evento FormCreate

Este é o primeiro código a rodar quando a janela abre.

```
procedure TGeradorEtiquetas.FormCreate(Sender: TObject);
begin
  Self.WindowState := wsMaximized; // Força tela cheia

  // Desativa as ancoras automaticas
  for i := 0 to Panel2.ControlCount - 1 do
      Panel2.Controls[i].Anchors := [akLeft, akTop];
  ...
end;
```

Por que desativar as âncoras? O Delphi tenta alinhar componentes automaticamente usando a propriedade `Anchors`. Como nós queremos criar um layout matemático preciso (via código), as âncoras automáticas atrapalhariam, fazendo botões "voarem"para lugares errados. Nós assumimos o controle manual.

3.2 O Cérebro Visual: AjustarLayoutGeral

Esta procedure é responsável pela responsividade. Ela é chamada sempre que a tela muda de tamanho (`OnResize`).

3.2.1 Lógica do Grid

```
LarguraMedia := LarguraTotal div DBGrid1.Columns.Count;
for i := 0 to DBGrid1.Columns.Count - 1 do
    DBGrid1.Columns[i].Width := LarguraMedia;
```

Aqui usamos matemática simples: pegamos a largura total disponível na tela, descontamos a barra de rolagem (25px) e dividimos pelo número de colunas. O resultado é aplicado a cada coluna, garantindo que a tabela ocupe 100% da tela sem deixar buracos brancos.

3.2.2 Posicionamento Matemático de Campos

Em vez de arrastar componentes com o mouse, definimos posições `X` (Left) e `Y` (Top) fixas.

```
  Col1 := 20;   // Margem Esquerda
  Col2 := 180;  // Coluna da Direita
  Y    := 50;   // Altura inicial
  AlturaLinha := 55; // Pulo para a proxima linha
```

Isso cria um "grid invisível"onde encaixamos os componentes. Se precisarmos descer tudo, basta alterar a variável `Y` inicial.

3.3 A Função Auxiliar: PosicionarCampo

Para evitar repetir código de alinhamento para cada um dos 15 componentes, criamos esta função:

```
procedure TGeradorEtiquetas.PosicionarCampo(Lb: TLabel; Ctrl: TControl; X,
    Y, W: Integer);
begin
  Ctrl.Left := X; Ctrl.Top := Y; ...
  Lb.Left := X;
  Lb.Top  := Y - Lb.Height - 3; // O Pulo do Gato
end;
```

O Pulo do Gato: A linha `Y - Lb.Height - 3` calcula automaticamente onde o Rótulo deve ficar para estar exatamente 3 pixels acima da caixa de texto. Isso garante alinhamento perfeito, milimétrico, impossível de fazer na mão.

Capítulo 4

O Núcleo de Processamento (Gerar Etiqueta)

O botão `BtnGerarEtiquetaClick` executa a tarefa principal. Vamos dividir em 4 fases.

4.1 Fase 1: Tratamento de Strings (Input)

O usuário pode digitar códigos de forma "suja": `1, 2, 3` ou com quebras de linha. O SQL Server é rigoroso e falharia.

```
ListaDeCodigos := StringReplace(edtCodigo.Text, ' ', ',', [rfReplaceAll]);
...
ListaDeCodigosFormatada := '''' + StringReplace(..., ',', ''',''', ...) + '
    ''';
```

O código transforma `1, 2, 3` em `'1','2','3'`. Isso é crucial para a cláusula `IN` do SQL.

4.2 Fase 2: O SQL Otimizado

```
BaseSQL := 'SELECT ... FROM ... WHERE P.au_ite IN (' +
    ListaDeCodigosFormatada + ')';
```

Note que usamos `IN`. Isso busca todos os produtos de uma só vez.

4.3 Fase 3: A Multiplicação em Memória (TFDMemTable)

Este é o diferencial profissional do código.

```
mtEtiquetas.DisableControls; // Congela a tela (Performance)
try
  for I := 1 to Quantidade do // Loop das Copias
  begin
    FDQuery2.First;
    while not FDQuery2.Eof do // Loop dos Produtos
    begin
```

```
      mtEtiquetas.Append;
      mtEtiquetas.CopyFields(FDQuery2); // Clona o registro
      mtEtiquetas.Post;
      FDQuery2.Next;
    end;
  end;
```

Explicação: Se o usuário quer 10 cópias de 3 produtos:

1. Buscamos os 3 produtos no banco (Query).
2. Entramos num loop de 1 a 10.
3. Para cada volta, copiamos os 3 produtos para a Memória.
4. Resultado: 30 linhas na tabela de memória, prontas para impressão, sem sobrecarregar a rede.

4.4 Fase 4: Configuração Dinâmica do Relatório

O ReportBuilder (componente `Etiqueta`) normalmente tem tamanho de papel fixo. Nós "quebramos"isso via código:

```
Etiqueta.PrinterSetup.PaperName := 'Custom';
Etiqueta.PrinterSetup.PaperWidth := LarguraTotalPapel;
Etiqueta.PrinterSetup.PaperHeight := AlturaEtiqueta;
```

Isso força o motor de impressão a aceitar as medidas que o usuário digitou na tela, permitindo usar qualquer impressora térmica (Zebra, Argox, Elgin) sem recompilar o programa.

Capítulo 5

Exportação para Excel (O Truque do HTML)

Criar arquivos `.xls` binários é complexo. O código usa uma técnica inteligente de compatibilidade:

```
HTML.Add('<html><body><table border="1">');
...
HTML.Add('<td>' + FieldValue + '</td>');
...
TempList.SaveToFile(..., TEncoding.UTF8);
```

Geramos uma tabela HTML padrão. Ao salvar com a extensão `.xls`, o Excel detecta a tabela HTML e a converte automaticamente para planilha. **Vantagem:** Não precisa ter Excel instalado na máquina para gerar o arquivo, e funciona em qualquer versão do Windows.

Capítulo 6

Persistência de Dados (INI Files)

Para que o usuário não precise digitar ”4,0 cm”e ”2 colunas”toda vez que abre o programa, usamos `TIniFile`.

```
1 ArquivoIni.WriteString('ConfigEtiqueta', 'Largura', edtLargura.Text);
```

Isso cria um arquivo de texto ao lado do executável (`.ini`) que funciona como uma memória de longo prazo do sistema.

Capítulo 7

Conclusão

Este código é um exemplo de **programação defensiva e otimizada**:

1. **Defensiva:** Trata strings de entrada para evitar erros de SQL, usa `Try...Except` para capturar falhas, e `PosicionarCampo` para evitar erros visuais.
2. **Otimizada:** Usa memória RAM (`TFDMemTable`) em vez de processamento de banco de dados e congela a interface visual (`DisableControls`) durante processos pesados para ganhar velocidade.

Dominar estes conceitos (MemTable, Layout via Código, Manipulação de Strings SQL) eleva o nível do programador Delphi de iniciante para intermediário/avançado.