

Macoratti.net VB 2005 - Acessando e mantendo dados no Firebird Embarcado

Se você acha que já leu sobre este assunto em meu site, acertou , eu já tratei do acesso a dados no FireBird em diversos artigos. Confira:

- [VB - Acessando o FireBird com ADO\(VB6\)](#)
- [ASP - Acessando dados no FireBird 1.5](#)
- [VB.NET - Criando uma conexão genérica para qualquer banco de dados](#)
- [VB.NET - Acessando o FireBird.](#)

Então por que estou voltando ao assunto ? Bem , eu quero mostrar neste artigo como você pode acessar o FireBird embarcado usando o **Visual Basic 2005 Express Edition** de forma a efetuar a navegação pelos registros e também efetuar a manutenção de dados com inclusão, alteração e exclusão de registro de uma tabela no banco de dados FireBird. Creio que isto eu ainda não mostrei... 😊

Do que você vai precisar para acompanhar este artigo ? Veja a lista abaixo :

- [VB 2005 Express \(É gratis...\)](#)
- [Embedded Firebird for Windows 1.5.3 \[stable\] \(1.4 MB\)](#)
- [FirebirdClient 2.0.1 for .NET Framework 2.0 \(É grátis...\)](#)
- [IBExpert \(É quase grátis...\)](#)

E qual a vantagem em usar o FireBird embarcado ? Qual a diferença entre o FireBird embarcado e o Microsoft Access ?

Uma diferença fundamental e básica é que o FireBird embarcado é totalmente grátis e o Access não. 🙌😊

O que significa versão embarcada ? 😊

Bem , uma versão embarcada é aquela que você pode empacotar junto com sua aplicação e distribuir sem necessidade posterior de ter que instalar a base de dados. É isso mesmo, o FireBird possui uma versão na qual basta você incluir o arquivo **fbembed.dll** junto com sua aplicação, distribuir e pronto; o banco de dados já vai junto exatamente como você faz com o Access, com a vantagem de que o FireBird é muito mais do que um servidor de arquivos

veja abaixo uma comparação entre o Firebird e o Access 2000 (Microsoft Jet 4.0) copiada e colada do site:
<http://www.dotnetfirebird.org/blog/2005/01/firebird-and-microsoft-jet-feature.html>

Firebird and Microsoft Jet Feature Comparison

Incomplete and biased comparison of Firebird 1.5 and Microsoft Jet 4.0:

Feature	Firebird 1.5	Microsoft Jet 4.0
Deployment and Administration		
Single database file	✓	✓
Embeddable	✓ Details...	✓
XCOPY data deployment	✓ Details...	✓
Hot backup (24x7 availability is possible)	✓	✗
Can run on Windows 98	✓	✓
Scalability		
Database file size limit	✓ Unlimited*	⚠ 2 GB
Multiple concurrent users	✓	⚠ It doesn't scale
Engine Features		
Transações	✓	✗
Views	✓	✓
Autoincrement fields	✓	✓
Stored procedures and triggers	✓	✗
BLOB fields	✓	✓
Unicode support	✓	✓
National character sets support (including sorting)	✓	✓
ADO.NET Provider	✓	✓ (OLEDB)
Licensing		
Open-source license	✓ Details...	✗
License allows use in commercial applications	✓ Details...	✓

Nota:

O banco de dados Firebird Embedded Server é um cliente com um servidor de banco de dados completo, carregado

O Banco de dados FireBird Embarcado Server é um cliente com um servidor de banco de dados completo, carregado dentro de uma biblioteca dinâmica (**fbembed.dll**). Este provê os mesmos recursos que a distribuição SuperServer e exporta as entradas padrão da API do Firebird. (www.firebird.com.br)

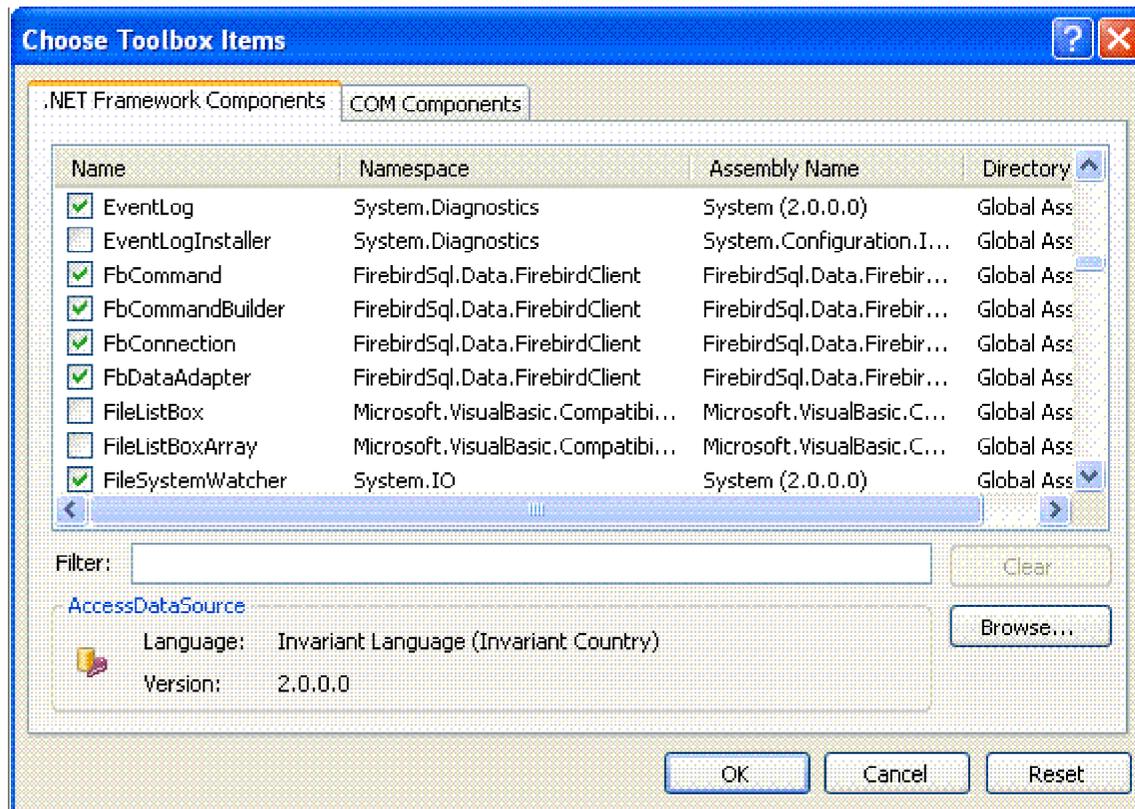
A seguir alguns detalhes desta versão :

- *Acesso ao Banco de Dados: Somente "acesso local" é permitido. O servidor não oferece suporte para protocolos remotos, o que implica que acessos via "localhost" ou "127.0.0.1" não irão funcionar.*
- *Autenticação e Segurança: O banco de dados de segurança (**security.fbd**) não é usado nesta distribuição como também não é requerido para funcionar. Qualquer usuário está habilitado para acessar qualquer banco de dados armazenado desta maneira. Desde que o cliente e o servidor rodem no mesmo local, a segurança passa a ser somente uma questão de acesso físico. Privilégios em instruções SQL ainda são verificadas, como nas outras distribuições do servidor.*
- *Compatibilidade: Podem ser executadas qualquer número de aplicações com o servidor sem nenhum conflito. Mesmo havendo servidores IB/FB em execução não haverá problemas.*
- *Deve-se tomar a precaução de não se acessar a mesma base de dados por múltiplos servidores, pois eles seguem a arquitetura SuperServer que mantém acesso exclusivo aos bancos de dados abertos.*

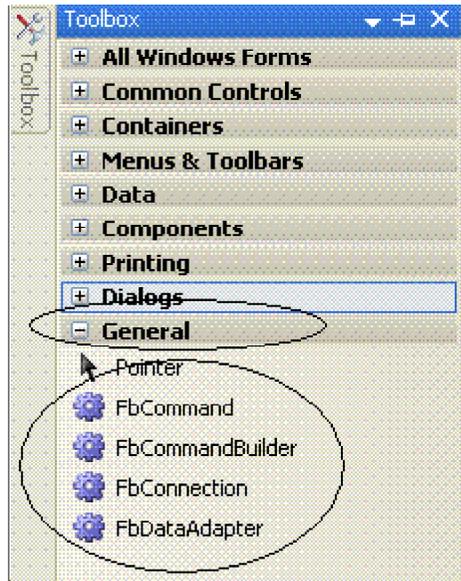
Após ter todos os componentes instalados vamos incluir os componentes na ToolBox do VB 2005 Express para podermos usá-los no projeto que iremos criar.

1- Abra o VB 2005 Express e no menu **Tools** , selecione **External Tools**

2- Na janela - **Choose Toolbox Items** - selecione os componentes: **FbCommand**, **FbCommandBuilder**, **FbConnection** e **FbAdapter** conforme a figura abaixo:



A seguir você já pode abrir a ToolBox e visualizar os componentes instalados na guia General:



Já temos tudo pronto para arregaçar as mangas e criar nossa aplicação. Só falta uma coisinha... Criar o banco de dados e a tabela que iremos usar. Para este artigo vamos criar um banco de dados chamado **Agenda.fdb** e uma tabela **Clientes** usando a ferramenta **IBExpert**.

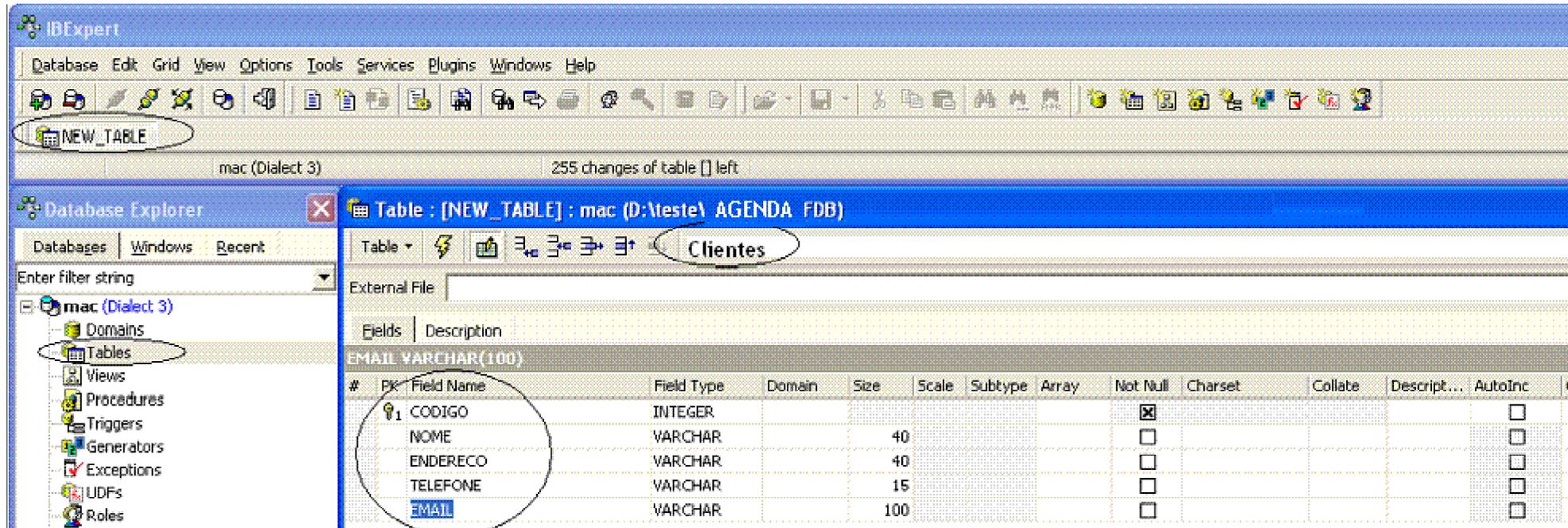
Para saber como usar a ferramenta **IBExpert** veja o artigo : [VB.NET - Acessando o FireBird](#).

A estrutura da tabela **Clientes** que devemos criar é a seguinte :

#	Key	FK	Fields	Type
1	1		CODIGO	INTEGER
2			NOME	VARCHAR(40)
3			ENDERECO	VARCHAR(40)
4			CIDADE	VARCHAR(30)
5			ESTADO	VARCHAR(3)
6			TELEFONE	VARCHAR(20)
7			EMAIL	VARCHAR(100)

Neste artigo eu vou colocar o banco de dados **AGENDA.FDB** na pasta [d:\dados](#). Você pode colocar o banco de dados em outra pasta mas se fizer isto deve alterar a string de conexão usada na aplicação.

Abaixo vemos a figura do banco de dados **AGENDA.FDB** e da tabela **Clientes** sendo criada no **IBExpert**:



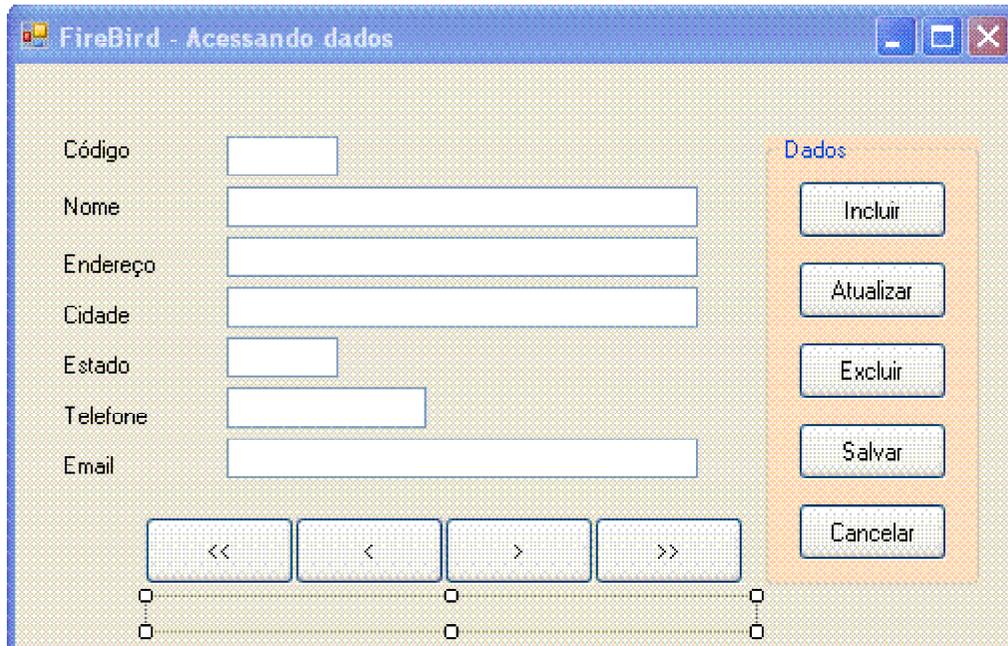
Agora sim, já temos tudo pronto para criar o projeto no VB 2005 Express. Vamos lá...

Acessando e mantendo dados no Firebird embarcado com o VB 2005 Express

O objetivo da aplicação será acessar a tabela Clientes do banco de dados Agenda.fdb e permitir a navegação pelos registros da tabela, incluir, alterar, atualizar e excluir registros usando o objeto DataSet sem a utilização de qualquer assistente do VB 2005 Express.

Abra o VB 2005 Express e crie um novo projeto chamado **agendaNET** alterando o nome do formulário padrão form1.vb para **frmConFB.vb**.

A seguir inclua no formulário controles **Label**, **TextBox**, **Button** e **GroupBox** conforme o leiaute da figura abaixo:



Nomes usados para os controles no formulário:

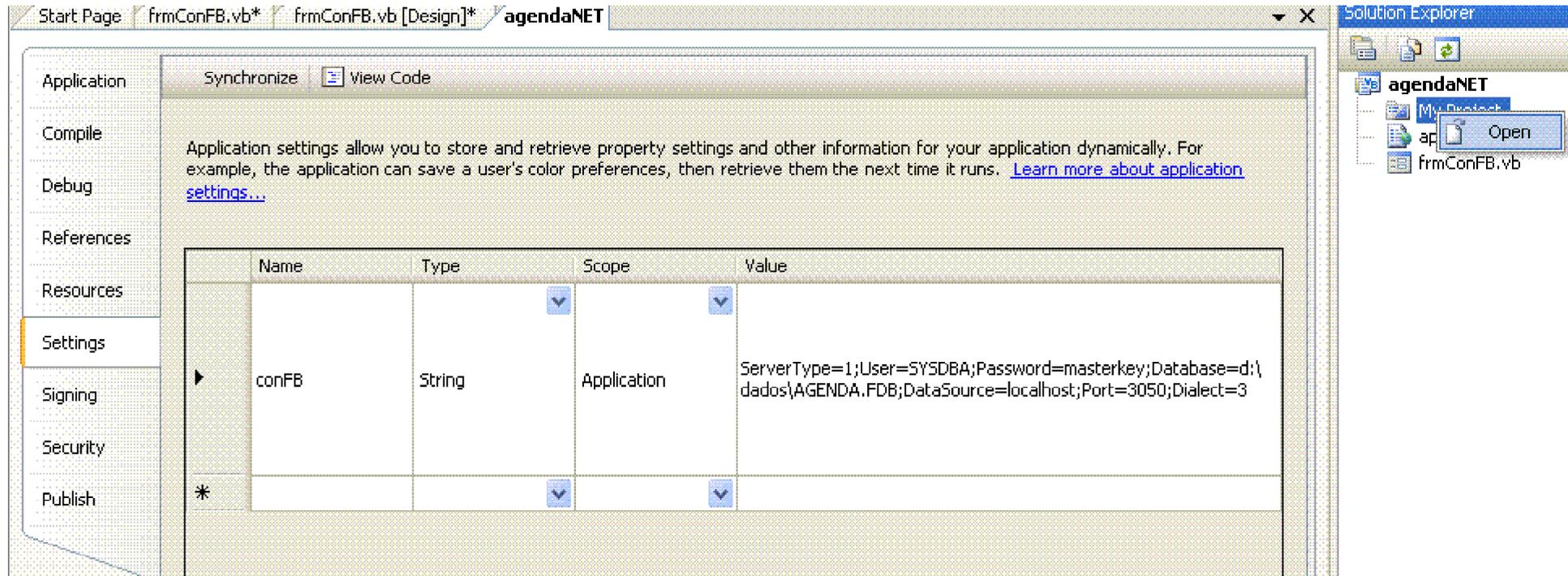
Controles TextBox	Botões de Navegação	Botões de Manutenção Dados
<ul style="list-style-type: none"> ▪ txtCodigo ▪ txtNome ▪ txtEndereco ▪ txtCidade ▪ txtEstado ▪ txtTelefone ▪ txtEmail 	<ul style="list-style-type: none"> ▪ btnInicio ▪ btnAnterior ▪ btnProximo ▪ btnUltimo 	<ul style="list-style-type: none"> ▪ btnIncluir ▪ btnAtualizar ▪ btnExcluir ▪ btnSalvar ▪ btnLimpar
<p>Controle Label - lblReg</p>		

Como sabemos qual a base de dados vamos usar e qual a localização do banco de dados vamos definir a string de conexão que será usada para realizar a conexão com a fonte de dados. A string de conexão usada para o nosso caso é a seguinte:

ServerType=1;User=SYSDBA;Password=masterkey;Database=d:\dados\AGENDA.FDB;DataSource=localhost;Port=3050;Dialect=3

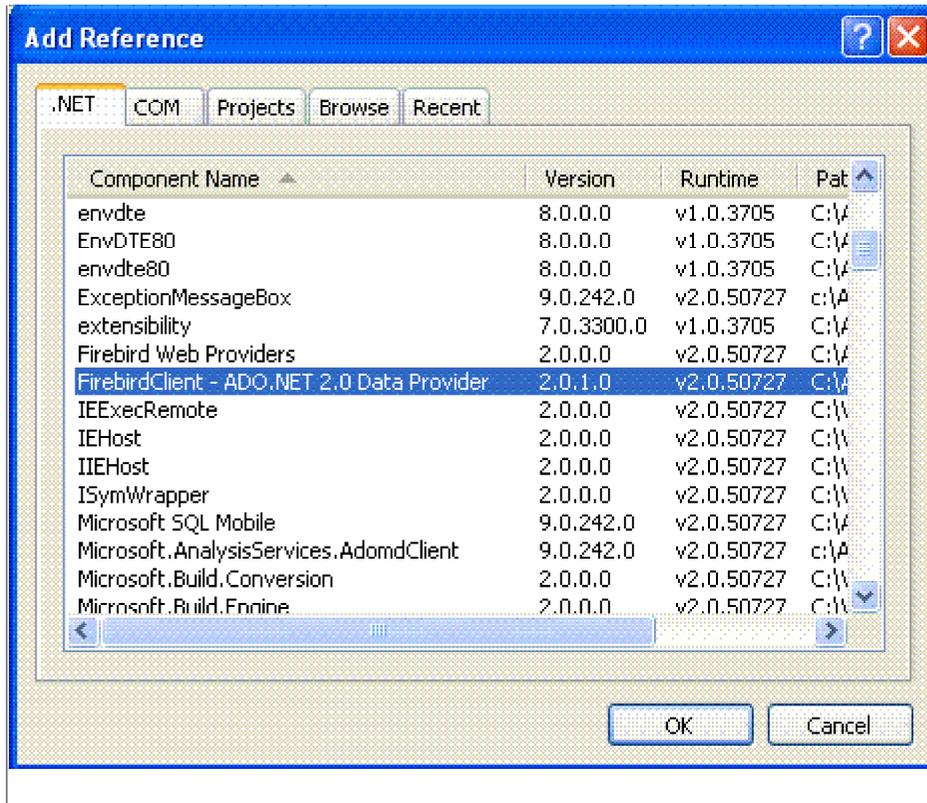
onde a declaração **ServerType=1** indica que estamos usando o FireBird embarcado.

Vamos armazenar a string de conexão no arquivo **web.config**. Para fazer isto clique com o botão direito do mouse sobre o **My Project** e selecione **Open**. A seguir na guia **Settings** altere o valores das propriedades **Name**, **Type**, **Scope** e **Value** conforme a figura abaixo:



Outra providência que devemos tomar para poder usar o FireBird embarcado é incluir o arquivo **fbembedd.dll** na pasta **\bin\Debug** da aplicação;

E, finalmente, antes de partirmos para o código devemos incluir uma referência ao provedor **FireBird ADO.NET 2.0** no projeto, clicando com o botão direito do mouse sobre o nome da solução e selecionando a opção **Add Reference**. Em seguida, na janela **Add Reference** selecione o provedor **FireBird - ADO NET 2.0 Data Provider**, conforme figura abaixo:



Nota: (www.firebird.com.br)

Uma aplicação para ser "embarcada" precisa ter as seguintes características :

- O caminho do banco de dados deverá ser identificado no componente de acesso como partindo-se do diretório onde se encontra seu aplicativo;
- O protocolo de comunicação deverá ser "local" e nunca remoto.

A primeira coisa a fazer é declarar o namespace usado no projeto:

Imports FirebirdSql.Data.FirebirdClient

A próxima etapa é declarar as variáveis para os objetos **DataAdapter**, **DataSet** para o FireBird e das variáveis **i** e **regs** que deverão ser vistas em todo o projeto:

```
Dim da As FbDataAdapter
Dim ds As New DataSet
Dim i As Integer
Dim regs As Integer
```

Quando o formulário principal da aplicação for carregado o evento **Load** deverá carregar os dados

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    carregadados()
End Sub
```

A rotina **carregaDados()** é definida assim :

```
Private Sub carregadados()
    Dim con As New FbConnection
    con.ConnectionString = My.Settings.conFB

    Try
        con.Open()
        Dim sql As String = "Select * from Clientes"
        da = New FbDataAdapter(sql, con)
        da.Fill(ds, "Clientes")
        con.Close()
        'armazena o total de registros da tabela
        regs = ds.Tables("Clientes").Rows.Count
        'define o índice do registro atual
        i = 0
        preenchedados()
    Catch ex As Exception
        MsgBox("Erro a efetuar a conexão com o FireBird embarcado " & vbCrLf & ex.Message)
    End Try
End Sub
```

A string de conexão esta sendo obtida do arquivo web.config via instrução : `My.Settings.conFB`

A conexão é aberta e usando um componente **Firebird Adapter** preenchemos o dataset com os dados da tabela Clientes: `da.Fill(ds, "Clientes")`

O total de registros existente na tabela é armazenado na variável **regs** e o índice para o registro atual é definido como sendo igual a zero e a rotina **preencheDados()** é chamada para exibir os dados no formulário.

Abaixo temos o código da rotina **preencheDados()** que atribui os valores atuais do registro definido pelo índice i as caixas de texto do o formulário;

```
Private Sub preenchedados()  
txtCodigo.Text = ds.Tables("Clientes").Rows(i).Item("Codigo")  
txtNome.Text = ds.Tables("Clientes").Rows(i).Item("Nome")  
txtEndereco.Text = ds.Tables("Clientes").Rows(i).Item("Endereco")  
txtCidade.Text = ds.Tables("Clientes").Rows(i).Item("Cidade")  
txtEstado.Text = ds.Tables("Clientes").Rows(i).Item("Estado")  
txtTelefone.Text = ds.Tables("Clientes").Rows(i).Item("Telefone")  
txtEmail.Text = ds.Tables("Clientes").Rows(i).Item("Email")  
  
End Sub
```

Cada campo é acessado pela instrução : `ds.Tables("Nome da tabela").Rows(indice do registro).Item(nome do campo)`

Para acessar o campo **Codigo** da tabela Clientes fazemos: `ds.Tables("Clientes").Rows(i).Item("Codigo")`

mas poderíamos também usar a seguinte instrução: `ds.Tables(0).Rows(i).Item(0)`

Onde o índice zero indica a primeira tabela no dataset e o primeiro item do registro.

Neste ponto quero chamar sua atenção para um detalhe : **Eu estou usando um dataSet não tipado**. E, existe uma certa polêmica sobre se é melhor usar ou não usar os datasets tipados. Como sempre não existe uma resposta absoluta e tudo vai depender do que se deseja obter e do ambiente que estaremos usando.

Um **DataSet tipado** é um dataset que é derivado de uma classe DataSet e que usa a informação contida em um arquivo de esquema XML (`.xsd`)

para gerar uma nova classe. Toda a informação do esquema (*tabelas* , *colunas* , *linhas* , *etc.*) é gerada e compilada neste nova classe DataSet. Como esta nova classe é derivada (herda) da classe DataSet ela assume toda a funcionalidade da classe DataSet.

Um **DataSet não tipado** não possui um corresponde arquivo de esquema , ele também possui tabelas , colunas , linhas , etc mas que são expostas somente como uma coleção. Você é quem deverá informar os nomes dos itens presentes nas coleções . Isto pode levar a que os erros sejam detectados somente na hora da compilação. Você pode criar um dataset tipado usando os assistentes do VB 2005 ou o utilitário **XSD.exe**.

Os DataSets tipados oferecem algumas vantagens :

- [intellisense dos campos;](#)
- [o código fica mais legível;](#)
- [fornece um sistema de dados mais fortes pois são definidos previamente;](#)
- [ganho de produtividade;](#)

Mas você pode usar coleções de objetos para fazer o mapeamento dos dados entre as camadas e obter um desempenho melhor. Para saber mais leia os artigos:

- [Como usar o objeto DataSet ?](#)
- [Tratamento de dados com DataSet](#)

Para navegar pelos registros da tabela temos 4 botões de comando que basicamente controlam o índice do registro atual chamando a rotina **preencheDados()** e atualizando a etiqueta **lblReg** com a informação pertinente. Segue abaixo o código associado ao evento **Click** para cada um dos botões :

```
Private Sub btnAnterior_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnAnterior.Click
If i > 0 Then
i -= 1
preenchedados()
lblreg.Text = ""
Elseif i = -1 Then
lblreg.Text = "Inicio do arquivo"
Else
lblreg.Text = "Primeiro registro"
End If
End Sub

Private Sub btnProximo_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
```

```

btnProximo.Click
If i <> regs - 1 Then
    i = i + 1
    preenchedados()
    lblreg.Text = ""
Else
    lblreg.Text = "Último registro"
End If
End Sub

Private Sub btnUltimo_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnUltimo.Click
If i <> regs - 1 Then
    i = regs - 1
    preenchedados()
    lblreg.Text = "Fim do Arquivo"
End If
End Sub

Private Sub btnInicio_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnInicio.Click
If i <> 0 Then
    i = 0
    preenchedados()
    lblreg.Text = "Inicio do Arquivo"
End If
End Sub

```

Vejamos agora o código de cada um dos botões que efetua o tratamento dos dados efetuando as operações de **Incluir, Atualizar, Excluir e Salvar**:

1- O Botão **Incluir** - inclui dados na tabela **Clientes**.

```

Private Sub btnIncluir_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnIncluir.Click
    desabilitaBotoes()
    limpacontroles()
    txtCodigo.Focus()
End Sub

```

Ao clicar neste o botão as rotinas **desabilitaBotes()** e **limpaControles()** são chamadas e a seguir o foco é colocado na caixa de texto **Codigo** do formulário.

A rotina **desabilitaBotoes()** juntamente com a rotina **habilitaBotoes** são mostradas abaixo e apenas habilitam/desabilitam os botões de comando do formulário:

```
Private Sub habilitaBotoes()
    btnIncluir.Enabled = True
    btnAtualizar.Enabled = True
    btnExcluir.Enabled = True
    btnSalvar.Enabled = False
    btnLimpar.Enabled = False
End Sub
```

```
Private Sub desabilitaBotoes()
    btnIncluir.Enabled = False
    btnAtualizar.Enabled = False
    btnExcluir.Enabled = False
    btnSalvar.Enabled = True
    btnLimpar.Enabled = True
End Sub
```

O código da rotina **limpaControles()** é dado a seguir e limpa todos os controles TextBox do formulário:

```
Private Sub limpacontroles()
    For Each ctrl As Control In Me.Controls
        If TypeOf ctrl Is TextBox Then
            CType(ctrl, TextBox).Text = ""
        End If
    Next
End Sub
```

O código do evento **Click** do botão **Atualizar** apenas chama a rotina **atualizaDados()** conforme mostrado abaixo:

```
Private Sub btnAtualizar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
    btnAtualizar.Click
```

```
atualizaDados()
```

```
End Sub
```

A rotina `atualizaDados()` efetua a atualização dos dados após qualquer alteração feita:

```
Private Sub atualizadados()
```

```
Try
```

```
Dim cb As New FbCommandBuilder(da)
```

```
ds.Tables("Clientes").Rows(i).Item("Codigo") = txtCodigo.Text
ds.Tables("Clientes").Rows(i).Item("Nome") = txtNome.Text
ds.Tables("Clientes").Rows(i).Item("Endereco") = txtEndereco.Text
ds.Tables("Clientes").Rows(i).Item("Cidade") = txtCidade.Text
ds.Tables("Clientes").Rows(i).Item("Estado") = txtEstado.Text
ds.Tables("Clientes").Rows(i).Item("Telefone") = txtTelefone.Text
ds.Tables("Clientes").Rows(i).Item("Email") = txtEmail.Text
```

```
da.Update(ds, "Clientes")
```

```
lblreg.Text = "Dados atualizados..."
```

```
Catch ex As Exception
```

```
MsgBox("Erro a efetuar a atualização de dados no FireBird embarcado " & vbCrLf &
ex.Message)
```

```
End Try
```

```
End Sub
```

A primeira linha de código é : `Dim cb As New FbCommandBuilder(da)`

Para atualiza o banco de dados você precisa do objeto **CommandBuilder**. O **CommandBuilder** irá criar uma instrução SQL para você. Você deve informar qual **DataAdapter** esta usando. Desta forma o **DataAdapter** pode atualizar o banco de dados com os valores do dataset. Sem o **CommandBuilder** o **DataAdapter** não pode fazer esta operação. *(isto vale também para a operação de excluir registros como veremos em seguida)*

A classe **CommandBuilder** faz parte do .NET Framework e tem como objetivo construir automaticamente comandos SQL **Insert**, **Update** e **Delete** para uma **DataTable** baseado em instruções SQL. Ele dá uma mãozinha ao desenvolvedor e ajuda o código ficar mais limpo e fácil de entender evitando que os comandos SQL fiquem ali embutidos. Para cada provedor existe uma classe **CommandBuilder**.

Em seguida os valores das caixas de texto informadas no formulário são atribuídos a cada campo da tabela **Clientes** e em seguida o comando **Update** para atualizar o dataset é usado.

O código do botão Salvar

```
Private Sub btnSalvar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnSalvar.Click

If i <> -1 And validaDados() Then
Try
Dim cb As New FbCommandBuilder(da)
Dim dsNovaLinha As DataRow

dsNovaLinha = ds.Tables("Clientes").NewRow()
dsNovaLinha.Item("Codigo") = txtCodigo.Text
dsNovaLinha.Item("Nome") = txtNome.Text
dsNovaLinha.Item("Endereco") = txtEndereco.Text
dsNovaLinha.Item("Cidade") = txtCidade.Text
dsNovaLinha.Item("Estado") = txtEstado.Text
dsNovaLinha.Item("Telefone") = txtTelefone.Text
dsNovaLinha.Item("Email") = txtEmail.Text

ds.Tables("Clientes").Rows.Add(dsNovaLinha)
da.Update(ds, "Clientes")

lblreg.Text = "Novo registro incluído com sucesso..."
habilitaBotoes()
carregadados()
Catch ex As Exception
MsgBox("Erro a efetuar ao salvar dados no FireBird embarcado " & vbCrLf & ex.Message)
End Try
End If
End Sub
```

Este código efetua uma validação nos dados informados pelo usuário e a seguir cria um novo registro através da instrução:

ESTE CÓDIGO EFETUA UMA VALIDAÇÃO NOS DADOS INFORMADOS PELO USUÁRIO E A SEGUIR CRIA UM NOVO REGISTRO ATRAVÉS DA INSTRUÇÃO.

Dim dsNovaLinha As DataRow

dsNovaLinha = ds.Tables("Clientes").NewRow()

Em seguida os valores são atribuídos a nova linha, a nova linha é incluída no dataset e o mesmo é atualizado:

```
ds.Tables("Clientes").Rows.Add(dsNovaLinha)
da.Update(ds, "Clientes")
```

O código da rotina validaDados() , dado a seguir, apenas verifica se algum valor não foi informado pelo usuário na interface e retorna True ou False:

```
Private Function validaDados() As Boolean
    'verifica cada campo
    If txtCodigo.Text = "" Or _
       txtNome.Text = "" Or _
       txtEndereco.Text = "" Or _
       txtCidade.Text = "" Or _
       txtEstado.Text = "" Or _
       txtTelefone.Text = "" Or _
       txtEmail.Text = "" Then
        MsgBox("Informe um valor válido para cada campo.", MsgBoxStyle.Exclamation, Me.Text)
        Return False
    Else
        Return True
    End If
End Function
```

Para excluir um registro o evento Click do botão Excluir possui o seguinte código:

```
Private Sub btnExcluir_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnExcluir.Click
    If MessageBox.Show("Deseja realmente excluir este registro?" "Excluir" MessageBoxButtons.YesNo
```

```

If MessageBox.Show("Deseja realmente excluir este registro.", "Excluir", MessageBoxButtons.YesNo, MessageBoxIcon.Warning) = Windows.Forms.DialogResult.No Then
    MsgBox("Operação Cancelada...")
    Exit Sub
Else
    Try
        Dim cb As New FbCommandBuilder(da)
        ds.Tables("Clientes").Rows(i).Delete()
        regs -= 1
        i = 0
        da.Update(ds, "Clientes")
        preenchedados()
    Catch ex As Exception
        MsgBox("Erro a efetuar a exclusão de dados no FireBird embarcado " & vbCrLf & ex.Message)
    End Try
End If
End Sub

```

Novamente usamos o objeto **CommandBuilder** e o método **Delete** do objeto **DataRow** para remover a linha da tabela. Se não houvesse a necessidade de enviar um comando de atualização (update) a fonte de dados poderíamos remover os registros acessando diretamente a coleção **Row : DataRowCollection.Remove**, mas este não é o caso.

A variável **regs**, que representa o total de registros do dataset, é diminuída de uma unidade, para atualizar a fonte de dados usamos o comando **update**.

O código do botão **Cancelar**, dado a seguir, apenas habilita os botões, define o índice como sendo igual a zero (primeiro registro) e preenche o formulário com os dados:

```

Private Sub btnLimpar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnLimpar.Click
    habilitaBotoes()
    i = 0
    preenchedados()
End Sub

```

Executando a aplicação temos a seguinte visualização de formulário de aplicação:

Executando o projeto temos a seguinte visualização do formulário da aplicação:

FireBird - Acessando dados

Código: 3

Nome: Jefferson Andre

Endereço: Rua XV Novembro 20

Cidade: Lins

Estado: SP

Telefone: 22-456465

Email: teste@teste.com

Dados

Incluir

Atualizar

Excluir

Salvar

Cancelar

<< < > >>

Acabamos de criar uma solução que gerencia uma base de dados no FireBird usando o VB 2005 Express. Embora simples, pois estamos tratando apenas com uma tabela, nela apresentamos as principais operações básicas usadas no dia a dia : navegação, inclusão, alteração e exclusão.

O projeto completo esta no [Super Cd .Net](#) e no [Super Dvd .Net](#).

Até o próximo artigo VB.NET ... 🏠

José Carlos Macoratti