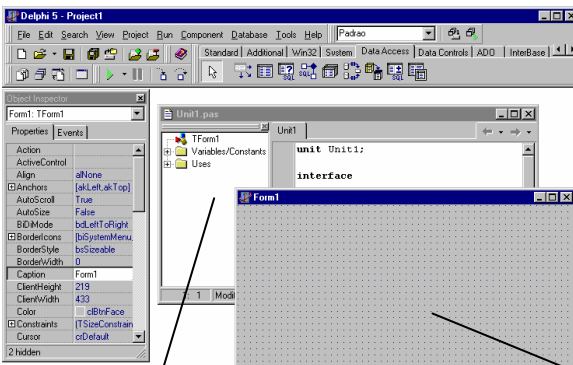
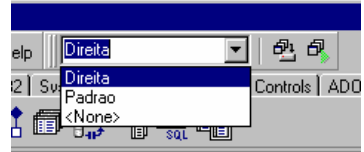


### Delphi 5 – Novo Visual

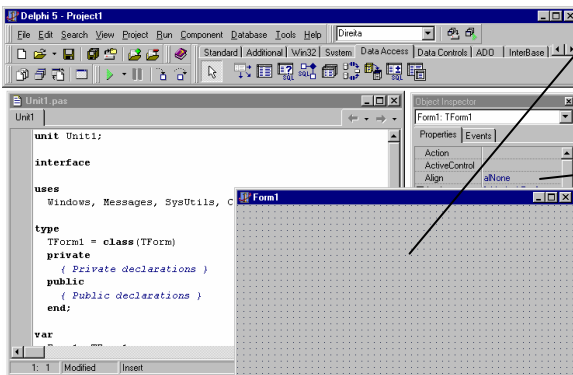
#### Desktop

O recurso Desktop permite organizar as janelas do Delphi da maneira ideal para o seu trabalho. Você posiciona as janelas de acordo com sua necessidade e pode gravar esta configuração como um Desktop. Os únicos itens que não sofrem alteração com a mudança de desktop são o formulário e a palheta de componentes. Veja as figuras abaixo em diferentes tipos de Desktop:



*Somente neste modo o Explorer é exibido.*

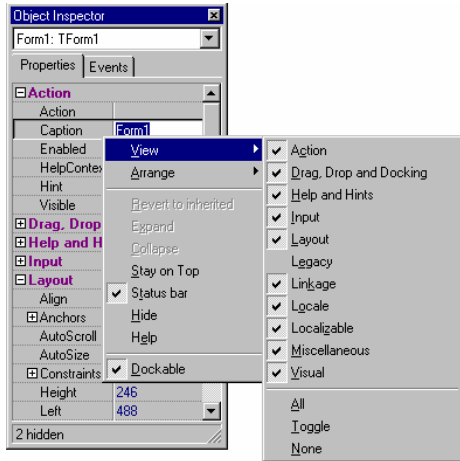
*O Form se mantem estável em diferentes Desktop's*



*O Object Inspector encontra-se à direita.*

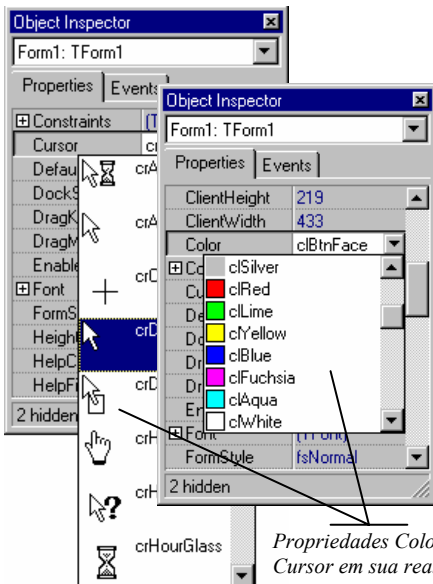
## O Novo Object Inspector

Novidades visuais foram adicionadas ao Object Inspector que se manteve o mesmo desde a versão 2.0 do Delphi. Agora é possível agrupar as propriedades por categorias pré-definidas pelo Delphi. Podemos ainda exibir ou ocultar estes grupos de propriedades (*Bastando clicar com o botão direito do mouse no Object Inspector e selecioná-los no menu View. A ordenação fica por conta do menu Arrange, onde selecionamos o modo convencional ou por grupo*).

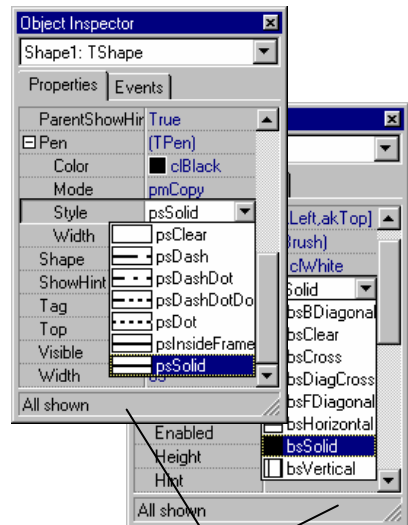


Também foi adicionada uma barra de status que é utilizada para indicar a quantidade de itens ocultos.

Propriedades de Lista suspensa como *Color* e *Cursor* ganharam novo visual na escolha dos itens. As propriedades do tipo *TBrush* e *TPen* também exibem seus desenhos nas listas.



*Propriedades Color e Cursor em sua real representação.*



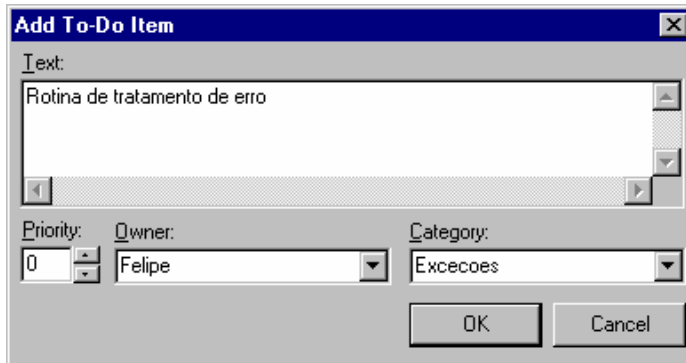
*Barra de Status no Object Inspector.*

## To-Do List

---

Esta ferramenta é de grande valia para auxiliá-lo no desenvolvimento de seus projetos. Com ela é possível criar uma listagem de tarefas que precisam ser implementadas no sistema.

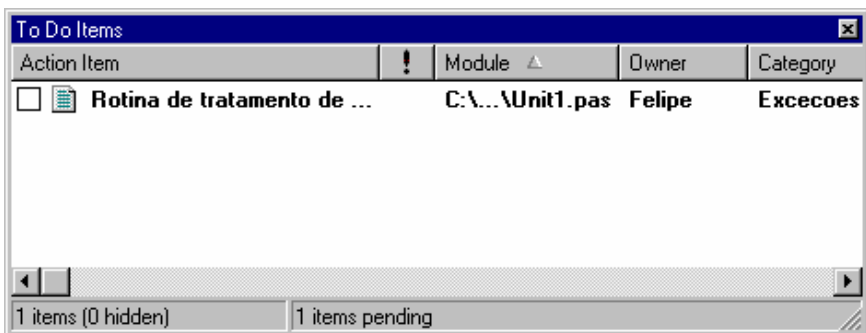
Sua utilização é bem simples, basta clicar com o botão direito do mouse sobre a janela 'Code Editor' e selecionar a opção 'Add To-Do Item...', para que a janela Add TO-DO seja exibida:



Preencha corretamente as informações nos campos e, em seguida, você verá uma linha de comentário, com uma codificação especial, adicionada a sua *Unit*:

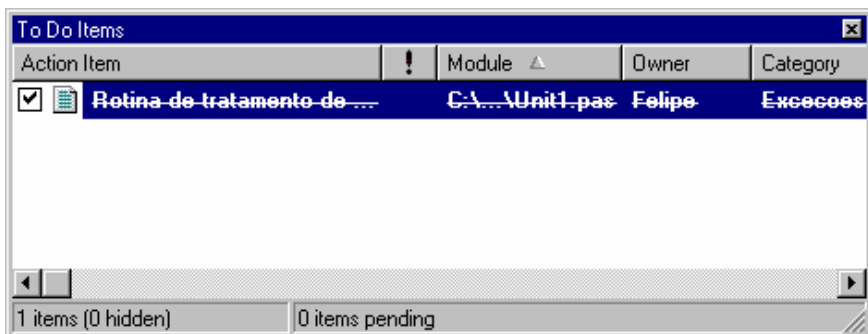
```
{ TODO -oFelipe -cExcecoes : Rotina de tratamento  
de erro }
```

A janela **To-Do List** pode ser exibida clicando no menu 'View | To-Do List'.



Ao excluir os itens contidos nesta janela, automaticamente a linha de código será removida. Marcando 'Done' para o item, o comentário na *Unit* sofrerá a seguinte alteração:

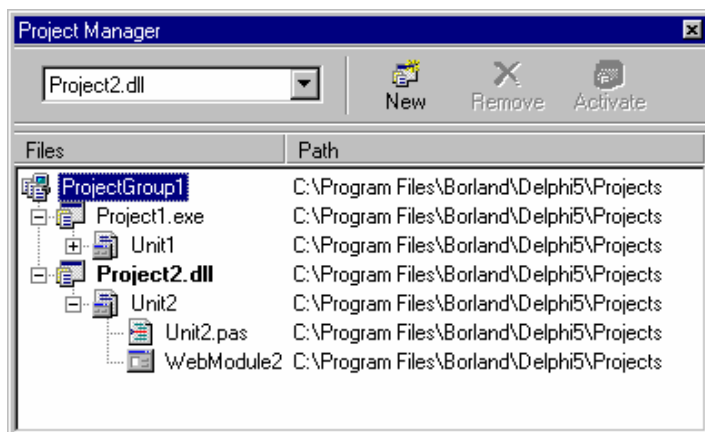
```
{ DONE -oFelipe -cExcecoes : Rotina de tratamento de erro }
```



## Project Manager

---

A principal novidade do Project Manager, é a interatividade com o Windows Explorer, permitindo o 'Drag And Drop' (Clicar e arrastar) entre arquivos para adicioná-los ao seu projeto.

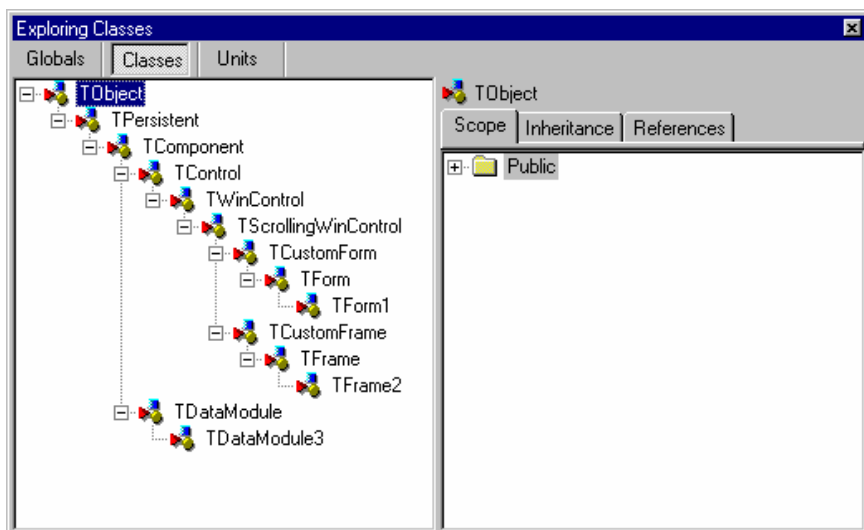


## Object Browser

---

O Object Browser é uma ferramenta antiga, utilizada por programadores mais experientes. O objetivo básico desta ferramenta é listar hierarquicamente as classes contidas em seu projeto. Nas versões

anteriores do Delphi, para acessar esta janela era necessário compilar o projeto, mas agora isto é feito automaticamente.

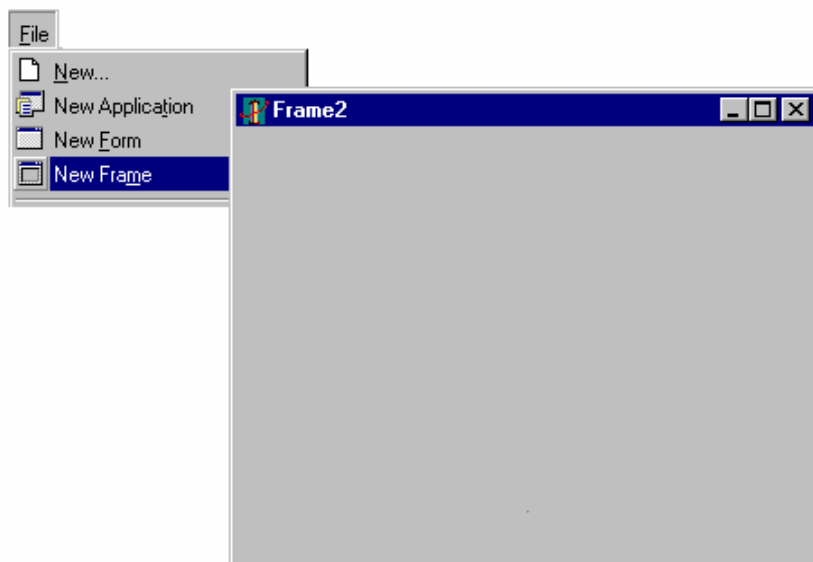


## Objeto TFrame

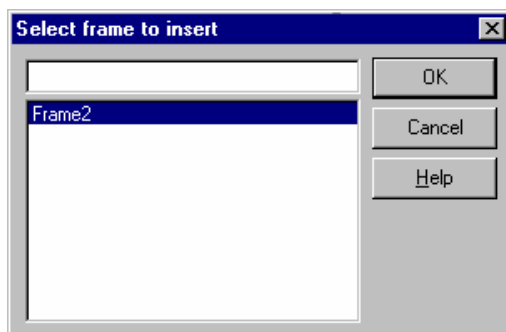
O objeto **TFrame** é semelhante ao componente *TPanel*. O propósito deste componente é criar uma classe incorporada de vários outros componentes. O mais interessante é que este novo *Frame* pode também ser adicionado a sua palheta de componentes, sendo assim instanciado futuramente em suas outras aplicações de forma independente da original.

As alterações feitas em objetos *TFrame* são automaticamente refletidas nas instâncias que a referenciam. O código é mantido na *Unit* do *Frame*.

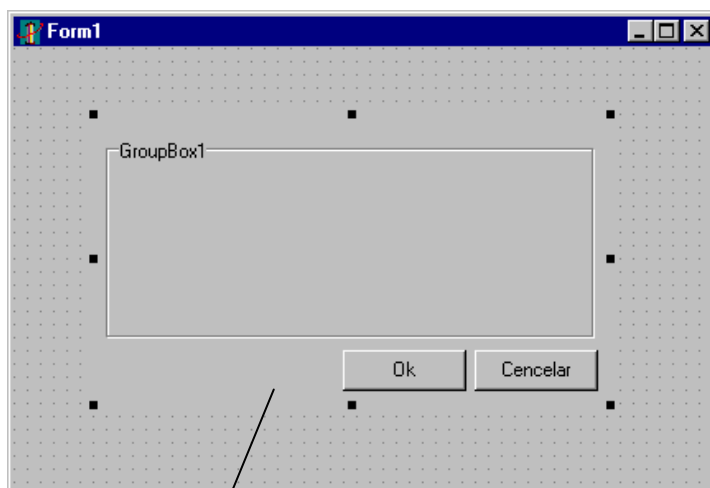
A utilização de *Frames* é muito simples, basta clicar em '*File | New Frame...*'.



Um novo *Frame* será criado em forma de um formulário. Inclua os objetos que deseja utilizar, e adicione ao *Form* principal um objeto *TFrame* da palheta de componentes.



Selecione o *Frame* criado na janela exibida, e clique em *Ok*. Será adicionado um objeto parecido com o *TPanel* no *Form* contendo os objetos incluídos no *Frame*.



O objeto *TFrame*  
inserido dentro de um  
formulário.

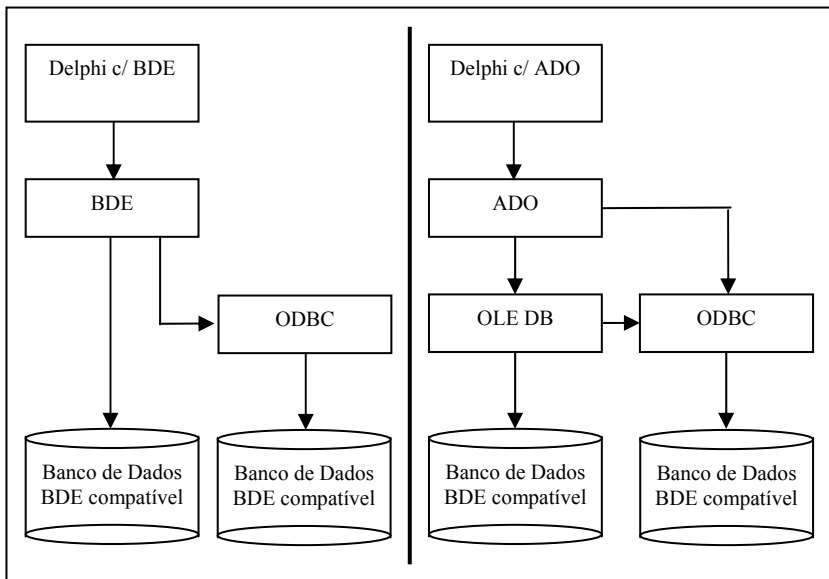
## Palheta ADO

O Delphi 5 incorpora uma biblioteca de 7 componentes para acesso à camada ADO. A camada



ADO é um novo padrão Microsoft para acesso a banco de dados. Antes do ADO, isto era feito com a camada ODBC, mas a Microsoft percebeu que este camada não serviria para aplicações Internet. Surgiu, então, a OLE DB que além de possuir drivers nativos para acessar banco de dados é altamente integrada à Internet. A camada ADO são classes de alto nível para acesso ao OLE DB.

O Delphi, até sua versão 4, só acessava banco de dados através da BDE. Mesmo quando era necessário utilizar o ODBC, a camada BDE servia como intermediária de comunicação. Com o advento da programação para Internet e a difusão do desenvolvimento em múltiplas camadas, a Inprise viu a necessidade de estender o padrão de acesso, visto que nem todos os provedores ou camadas Middletier poderão Ter o BDE instalado. Sendo assim, o Delphi 5 permite o acesso também via ADO (ActiveX Data Objects), de forma muito parecida com o acesso via BDE. Veja a ilustração abaixo, que representa a conexão via BDE ou ADO:



## **Exemplo1: Criando um DataSet com TADOTable:**

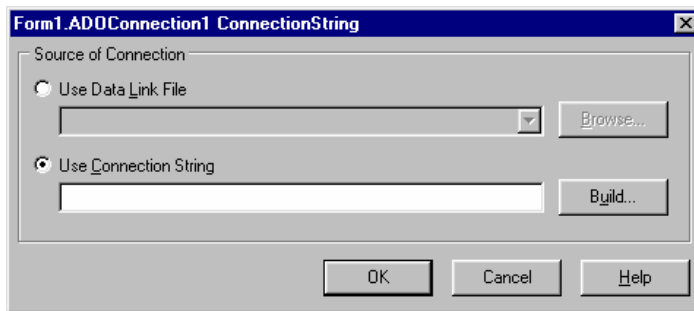
---

Este exemplo demonstra o uso mais simples dos objetos ADO. Aqui, um objeto DBGrid será preenchido com o conteúdo de uma tabela, fornecido através de um objeto TADOTable:

1) Crie uma nova aplicação, e no formulário 1, insira um objeto TADOConnection;

O objeto ADO Connection é equivalente ao objeto TDatabase, e como este, representa a conexão com o banco de dados. As transações e os parâmetros de conexão serão realizados através deste objeto.

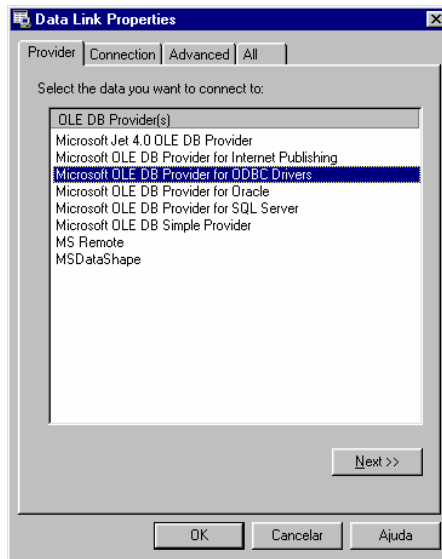
2) Em seguida, devemos configurar o objeto ADOConnection1. Dê um clique duplo sobre o objeto, para que a janela Connection String seja exibida:



Esta janela irá indicar os parâmetros de conexão com o banco de dados. As opções representam:

Use Data Link File:	Arquivo .UDL, que contém os parâmetros de conexão pré formatados.
Use Connection String:	Permite criar um novo arquivo .UDL, com todos os parâmetros de conexão.

Neste caso, selecione Use Connection String e clique no botão Build. A janela Data Link Properties será exibida:



Cada Item exibido nesta janela representa um mecanismo de conexão aos dados. Lembrando, o ADO não acessa o banco de dados sozinho, ele se conecta a um motor de acesso para realizar a conexão. Este motor pode ser uma conexão ODBC, uma conexão OLEDB nativa, uma conexão JET Engine, ou algum outro tipo de motor instalado. No nosso caso, usaremos o motor Jet Engine, para acessar o banco de dados Access.

Selecione o portanto, o item "Jet Engine 4.0 OLE DB Provider" e clique em Next.

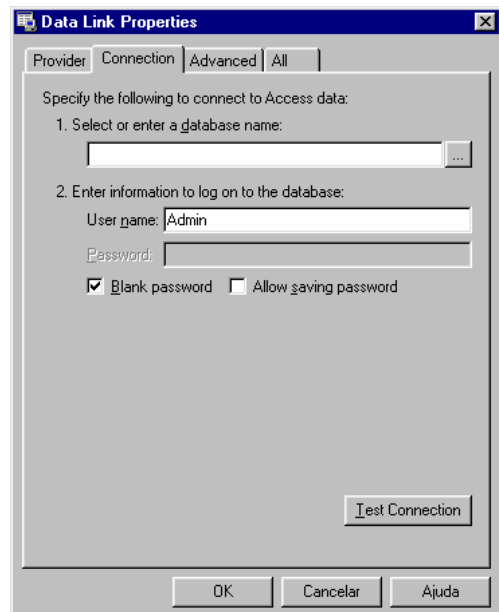
A aba connection será exibida:

**Nota:** Esta aba irá possuir uma interface diferente para cada banco de dados.

Na opção nº 1, selecione o arquivo .MDB do banco de dados que será acessado por nossa aplicação. Abra o arquivo DBDEMOS.MDB.

Após clique em OK, para voltar ao IDE do Delphi.

Altere a propriedade Connected para TRUE, a fim de conectar com o banco de dados.



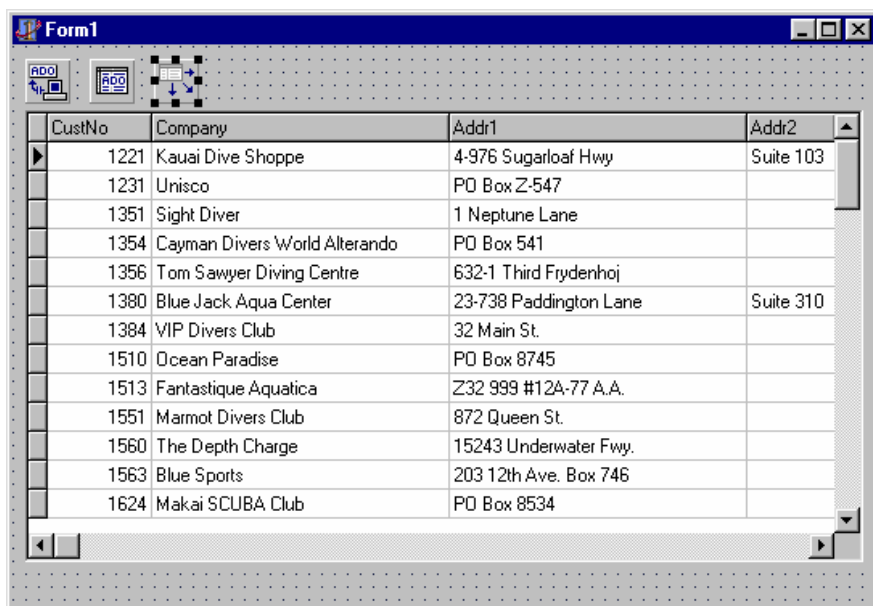
2) Insira um objeto TADOTable no formulário.

Este objeto representa a conexão com a tabela e é equivalente ao objeto TTable. Primeiramente, devemos conectar este objeto ao TADOConnection. Faça isto alterando a propriedade Connection, indicando o objeto ADOConnection1. Em seguida, indique o nome da tabela na propriedade TableName. Selecione Customer.

3) Insira um objeto TDataSource no formulário, e linke-o com o objeto ADOTable1.

4) Insira um objeto TDbGrid, e conecte-o com o objeto DataSource1.

Ok! Basta alterar a propriedade Active do objeto AdoTable1 para TRUE, e seu formulário será parecido com a figura abaixo:



Uma descrição detalhada de todas as propriedades, métodos e eventos relevantes aos objetos da palheta ADO encontra-se disponível a seguir:

## **TADOConnection**

---

Este componente é utilizado para criar uma nova conexão com a camada ADO. É bastante semelhante ao componente TDatabase, sua utilização também é opcional.

## Principais Propriedades

CommandTimeout	Esta propriedade permite especificar um determinado tempo de limite para obter uma resposta de um comando executado.
Connected	Especifica se existe ou não uma conexão.
ConnectionString	Define a String de conexão entre a camada intercessora do banco de dados com o ADO. Utilizamos esta propriedade para definir qual será o banco de dados e a camada para o <b>ADO</b> ( <i>Jet, ODBC, OLE DB</i> ).
ConnectionTimeout	Esta propriedade permite especificar um determinado tempo de limite para ser feita a conexão com o banco de dados.
ConnectOption	Esta propriedade define o tipo de conexão. <b>CoConnectUnspecified</b> Este e o modo padrao do ADO. <b>CoAsyncConnect</b> Este modo e utilizado Quando o servidor for muito lento.
CursorLocation	Define onde estará o cursor para suas tabelas.
DefaultDatabase	Define um banco de dados padrão para a conexão quando uma conexão via <i>ConnectionString</i> não for bem sucedida.
IsolationLevel	Define o nível de isolamento entre do banco de dados. Esta propriedade é semelhante à propriedade <i>TransIsolation</i> do objeto <i>Database</i> .
Mode	Define um modo de conexão: Somente leitura, leitura e gravação e etc.
Provider	Especifica o método de conexão que será utilizado pela camada ADO: <b>Jet, OLEDB, ODBC</b> , etc.

## Principais Eventos

AfterConnect	Ocorre após efetuar uma conexão.
AfterDisconnect	Ocorre após desconectar.
BeforeConnect	Ocorre antes de efetuar uma conexão.
BeforeDisconnect	Ocorre andes de desconectar.
OnBeginTrasComplete	Ocorre quando é iniciada uma nova transação.
OnCommitComplete	Ocorre quando o comando 'Commit' foi executado.
OnConnectCompleted	Ocorre quando é feita uma conexão.

OnDisconnect	Ocorre quando é terminada uma conexão.
OnExecuteComplete	Ocorre quando uma conexão com o ADO executou um comando(Command).
OnInfoMessage	Ocorre quando uma conexão é bem sucedida e informações adicionais são enviadas pelo <i>Provider</i> .
OnLogin	Ocorre quando é aberta a janela <i>'LoginPrompt'</i> .
OnRollbackTransComplete	Ocorre quando é executado um <b>RollBack</b> .
OnWillConnect	Ocorre quando uma conexão está prestes a ser efetuada.
OnWillExecute	Ocorre quando uma execução está prestes a ser efetuada.

### Principais métodos

Open	Permite ativar uma conexão
Close	Termina uma conexão

### TADOTable

---

Objeto ADOTable de uma conexão ADO é equivalente ao objeto TTable da camada BDE.

### Principais Propriedades

CacheSize	Indica quantos registros serão pedidos para o servidor por vez, e armazenados na memória local. Novos pedidos são automaticamente controlados pelo objeto.
Connection	Objeto TADOConnection, que representa a instancia do banco de dados atual.
ConnectionString	Pode ser usado no lugar de um objeto <i>TADOConnection</i> . Permite configurar uma string de conexão com o servidor.
CursorLocation	Indica se o objeto ira trabalhar em modo ChachedUpdates. Caso esta propriedade esteja configurada para clUseClient, os dados serao armazenados no cliente, e todas as operacoes realizadas (como insert, update e delete) permanecerao no mesmo. Caso a propriedade seja clUseServer, as alteracoes serao automaticamente enviadas para o servidor.

<p>CursorType</p>	<p>Indica o tipo de cursor utilizado pelo cliente. Os possiveis valores são:</p> <p><b>ctUnspecified</b> Um tipo de cursor não pre-definido pelo objeto.</p> <p><b>ctOpenForwardOnly</b> Cursor uni-direcional. Com este valor, o usuario podera apenas navegar para frente, nunca retornar os registros.</p> <p><b>ctKeyset</b> Cursor bi-direcional, não concorrente. Aqui, o usuario pode navegar para frente e para tras no result set. As alteracoes feitas por transacoes concorrentes não estarao acessiveis, e os registros deletados, apesar de visiveis, não estarao disponiveis.</p> <p><b>ctDynamic</b> Bi-direcional e concorrente. Este tipo de cursor e o que apresenta a pior performance e o que gasta mais recursos do sistema. Aqui, o usuario visualiza as alteracoes, insercoes e delecoes de outras transacoes.</p> <p><b>ctStatic</b> Uma simples copia dos registros. Qualquer tipo de alteracao concorrente não sera visualizada neste cursor.</p>
<p>LockType</p>	<p>Especifica como o lock de registros sera executado. Os possiveis valores são:</p> <p><b>ItUnspecified</b> Um tipo de lock que não foi pre-definido pelo objeto;</p> <p><b>itReadOnly</b> Somente leitura;</p> <p><b>itPessimistic</b> Impede que outras sessoes alterem o registro que estiver locado pelo usuario;</p> <p><b>itOptimistic</b> Todos podem alterar o mesmo registro;</p> <p><b>itBatchOptimistic</b> Impede que outras sessoes alterem quando estiver em modo update Batch.</p>
<p>MarshalOptions</p>	<p>Útil apenas quando a tabela está em Client-Side. Indica quais registros serão devolvidos para o servidor ou para a camada middleware. Se o valor for <b>moMarshalAll</b>, todos os registros serão enviados para o servidor. Caso o valor seja <b>moMarshalModifiedOnly</b>, apenas os registros modificados serão devolvidos.</p>

MaxRecords	Indica a quantidade máxima de registros exibidos pelo objeto. 0 (zero) indica todos os registros.
TableDirect	Quando <b>False</b> , os dados serão sempre requisitados via instruções <b>SQL Select</b> . Neste caso, a instrução é criada automaticamente pelo objeto. O valor Default e False.
TableName	Nome da tabela dentro do banco de dados.

## Principais Eventos

OnEndOfRecordset	Ocorre quando o ponteiro avança após o último registro carregado pelo <b>RecordSet</b> . É importante notar que esta ação não representa o final da tabela, e sim, o final do <i>buffer</i> . O parâmetro <b>MoreData</b> , quando <b>True</b> , irá indicar a requisição dos próximos registros que não foram carregados para o buffer. O parâmetro <b>EventStatus</b> indica se a operação que gerou o evento foi ou não bem sucedida.
OnFetchComplete	Ocorre quando uma requisição de dados para o <i>buffer</i> é terminada.
OnFetchProgress	Ocorre durante a leitura dos dados para o <i>buffer</i> . O parâmetro <b>Progress</b> indica a quantidade de registros lidos, <b>MaxProgress</b> a quantidade a ser lida e <b>EventStatus</b> se a operação foi bem sucedida.
OnfieldChangeComplete	Ocorre após a gravação de um valor para algum campo da tabela.
OnMoveComplete	Ocorre após o ponteiro de registro realizar alguma movimentação.
OnRecordChangeComplete	Ocorre após um ou vários registros serem atualizados.
OnWillChangefield	Ocorre antes da edição do valor de um campo.
OnWillChangeRecord	Ocorre antes da edição/inserção de um registro.
OnWillChangeRecordset	Ocorre antes da alteração de algum valor da tabela.
OnWillMove	Ocorre antes da movimentação do ponteiro para algum lugar.

## Principais Métodos

```
function Seek(const KeyValues: Variant; SeekOption: TSeekOption = soFirstEQ): Boolean;
```

Faz a pesquisa utilizando o índice atual. O primeiro parâmetro é um array OLE (VarArrayOf) de valores de pesquisa. O segundo indica como a pesquisa será realizada. Os possíveis valores para *SeekOption* são:

**soFirstEQ** Caso exista, o ponteiro é posicionado na primeira ocorrência.

**soLastEQ** Caso exista, o ponteiro é posicionado na última ocorrência.

**soAfterEQ** Record pointer positioned at matching record, if found, or just after where that matching record would have been found.

**soAfter** O ponteiro é posicionado no registro encontrado +1.

**soBeforeEQ** Record pointer positioned at matching record, if found, or just before where that matching record would have been found.

**soBefore** O ponteiro é posicionado no registro encontrado -1.

## TADOQuery

---

Representa o objeto TQuery da palheta DataAccess. O mais indicado para conexões com um banco de dados Cliente-Servidor.

## Principais Propriedades

Connection	Objeto TADOConnection que representa a instância do banco de dados atual.
CursorLocation	Indica se o objeto irá trabalhar em modo ChachedUpdates. Caso esta propriedade esteja configurada para clUseClient, os dados serão armazenados no cliente, e todas as operações realizadas (como insert, update e delete) permanecerão no mesmo. Caso a propriedade seja clUseServer, as alterações serão automaticamente enviadas para o servidor.

<p>CursorType</p>	<p>Indica o tipo de cursor utilizado pelo cliente. Os possiveis valores são:</p> <p><b>ctUnspecified</b> Um tipo de cursor não pre-definido pelo objeto.</p> <p><b>ctOpenForwardOnly</b> Cursor uni-direcional. Com este valor, o usuario podera apenas navegar para frente, nunca retornar os registros.</p> <p><b>ctKeyset</b> Cursor bi-direcional, não concorrente. Aqui, o usuario pode navegar para frente e para tras no result set. As alteracoes feitas por transacoes concorrentes não estarao acessiveis, e os registros deletados, apesar de visiveis, não estarao disponiveis.</p> <p><b>ctDynamic</b> Bi-direcional e concorrente. Este tipo de cursor e o que apresenta a pior performance e o que gasta mais recursos do sistema. Aqui, o usuario visualiza as alteracoes, insercoes e delecoes de outras transacoes.</p> <p><b>ctStatic</b> Uma simples copia dos registros. Qualquer tipo de alteracao concorrente não sera visualizada neste cursor.</p>
<p>LockType</p>	<p>Especifica como o lock de registros sera executado. Os possiveis valores são:</p> <p><b>ItUnspecified</b> Um tipo de lock que não foi pre-definido pelo objeto;</p> <p><b>itReadOnly</b> Somente leitura;</p> <p><b>itPessimistic</b> Impede que outras sessoes alterem o registro que estiver locado pelo usuario;</p> <p><b>itOptimistic</b> Todos podem alterar o mesmo registro;</p> <p><b>itBatchOptimistic</b> Impede que outras sessoes alterem quando estiver em modo update Batch.</p>
<p>MarshalOptions</p>	<p>Útil apenas quando a tabela está em Client-Side. Indica quais registros serão devolvidos para o servidor ou para a camada middleware. Se o valor for <b>moMarshalAll</b>, todos os registros serão enviados para o servidor. Caso o valor seja <b>moMarshalModifiedOnly</b>, apenas os registros modificados serão devolvidos.</p>

MaxRecords	Indica a quantidade máxima de registros exibidos pelo objeto. 0 (zero) indica todos os registros.
ParamCheck	Se <b>True</b> , o objeto irá considerar as expressões ":variavel" como um nome de parâmetro.
Parameters	Propriedade da classe <b>TParameters</b> , que serve para a passagem de valores para os parâmetros da SQL. A sintaxe é diferente da propriedade <b>Params</b> do objeto <i>TQuery</i> . Ex:  ADOQuery1.Parameters.Params.ParamByName('codigo_cliente').Value:=1;
SQL	Instrução SQL que será executada pelo objeto.

## Principais Eventos

OnEndOfRecordset	Ocorre quando o ponteiro avança após o último registro carregado pelo <b>RecordSet</b> . É importante notar que esta ação não representa o final da tabela, e sim, o final do <i>buffer</i> . O parâmetro <b>MoreData</b> , quando <b>True</b> , irá indicar a requisição dos próximos registros que não foram carregados para o buffer. O parâmetro <b>EventStatus</b> indica se a operação que gerou o evento foi ou não bem sucedida.
OnFetchComplete	Ocorre quando uma requisição de dados para o <i>buffer</i> é terminada.
OnFetchProgress	Ocorre durante a leitura dos dados para o <i>buffer</i> . O parâmetro <b>Progress</b> indica a quantidade de registros lidos, <b>MaxProgress</b> a quantidade a ser lida e <b>EventStatus</b> se a operação foi bem sucedida.
OnfieldChangeComplete	Ocorre após a gravação de um valor para algum campo da tabela.
OnMoveComplete	Ocorre após o ponteiro de registro realizar alguma movimentação.
OnRecordChangeComplete	Ocorre após um ou vários registros serem atualizados.
OnWillChangefield	Ocorre antes da edição do valor de um campo.
OnWillChangeRecord	Ocorre antes da edição/inserção de um registro.

OnWillChangeRecordset	Ocorre antes da alteração de algum valor da tabela.
OnWillMove	Ocorre antes da movimentação do ponteiro para algum lugar.

### Principais Metodos:

Open	Abre a query
Close	Fecha a Query
ExecSQL	Executa uma intrução SQL

### TADODataSet

---

Este objeto é a classe pai dos objetos **TADOTable**, **TADOQuery** e **TADOStoredProc**. Suas propriedades, métodos e eventos são os mesmo destes objetos.

### TADOCommand

---

Não possui equivalente na palheta DataAccess. Este componente representa o objeto ADO Command, da biblioteca ADO. Serve apenas para enviar instruções SQL para o servidor, e não permite a ligação com objetos DataControl. Possui uma performance otimizada para instruções que não necessitem retornar dados para a aplicação. Operações como backup, deleção de tabelas e *update batchs* são mais rápidos quando executados através deste componente. Aplicações SQL também podem ter um desempenho melhorado com este componente.

### Principais Propriedades

CommandText	Instrução SQL que será executada pelo servidor.
CommandTimeOut	Tempo que o objeto irá esperar pela resposta do servidor.
CommandType	Indica o tipo de instrução contida na propriedade CommandText.
Connection	Objeto TADOConnection, que representa a instância do banco de dados atual.
ParamCheck	Se <b>True</b> , o objeto irá considerar as expressoes ":variavel" como um nome de parâmetro.

Parameters	Propriedade da classe <i>TParameters</i> , que serve para a passagem de valores para os parâmetros da SQL. A sintaxe é diferente da propriedade <i>Params</i> do objeto <i>TQuery</i> . Ex:  ADOCommand1.Parameters.Params.Param Byname('codigo_cliente').Value:=1;
------------	--

Este objeto não possui eventos. Para tratar os possíveis resultados que este objeto pode gerar, utilize os eventos do objeto *TADOConnection*.

### Principais Métodos

Execute	Envia a intrucao SQL para o servidor
---------	--------------------------------------

### **TADOStoredProc**

---

Equivalente ao objeto *TStoredProc*, da palheta *DataAccess*. Permite acionar uma procedure armazenada no servidor de banco de dados.

### Principais Propriedades:

Connection	Objeto <i>TADOConnection</i> que representa a instância atual do banco de dados.
Parameters	Define os parâmetros que serão passados ou recebidos da <i>Stored Procedure</i> .
ProcedureName	Nome da ' <i>StoredProcedure</i> '.

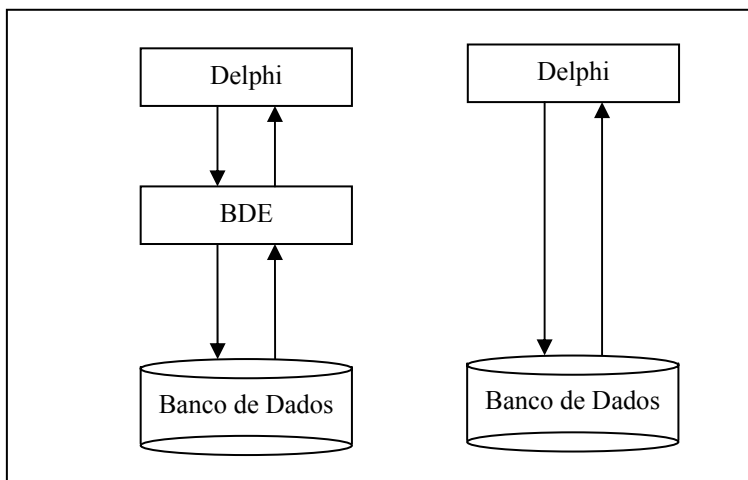
### **TRDSConnection**

---

Objeto para criação de aplicações multi-camadas. Este objeto serve como Middletier para conexões ADO em várias camadas. Existem outras alternativas e soluções mais interessantes para Delphi, como **MIDAS** e **CORBA**. Portanto, não daremos ênfase ao *Remote Data Space*.

## Palheta Interbase

A palheta Interbase fornece e o melhor método para criar uma aplicação Delphi acessando base de dados Interbase. Uma aplicação compilada em Delphi 5, utilizando os componentes da palheta Interbase apresentará uma performance superior do que as aplicações compiladas em versões anteriores do Delphi. Isto porque os novos componentes Interbase dispensam a camada BDE – ou qualquer outra camada. Uma aplicação construída com estes componentes terá o código de acesso ao banco “*built-in*”, ou seja, dentro do próprio executável. Repare na figura 1 a exemplificação desta diferença:



Além de maior performance, o desenvolvedor passa a ganhar também em recursos. Com os novos objetos é possível extrair várias informações do servidor, como memória disponível, espaço em disco, banco de dados utilizado, porcentual de recursos disponíveis, e outras. Um maior controle do SQL e das transações realizadas também foram adicionadas aos componentes. Apesar de todas estas vantagens, os componentes Interbase oferecem um óbvio ponto negativo: Uma aplicação construída com estes componentes deverá ser totalmente reconstruída caso o cliente resolver mudar de banco de dados.

## Componentes da palheta Interbase

A seguir encontra-se uma breve descrição dos principais componentes da palheta. Eles estão listados por ordem de importância dentro do projeto:

## TIBDatabase

---

Equivalente ao componente *TDatabase*, da palheta *DataAccess*. Representa a instância do banco de dados da aplicação, e realiza a conexão ao banco, através da propriedade *Connected*.

### Principais Propriedades:

Connected	Quando <b>True</b> , inicia a conexão com o servidor.
Databasename	Nome do arquivo de banco de dados.
DefaultTransaction	Indica um objeto <i>TIBTransaction</i> como objeto de transação <i>Default</i> .
IdleTimer	Especifica quanto tempo o cliente irá esperar caso o servidor não envie nenhuma resposta. Se o tempo for ultrapassado e o servidor não responder, a conexão será desfeita.
LoginPrompt	Se <b>True</b> , o objeto pedirá nome e senha ao usuário.
SQLDialect	Indica o código de dialeto SQL utilizado pelo cliente.
TraceFlags	Indica quais serão as ações monitoradas pelo objeto <i>TIBSQLMonitor</i> .

### Principais Eventos

AfterConnect	Ocorre após a conexão ser realizada.
AfterDisconnect	Ocorre após a conexão ser terminada.
BeforeConnect	Ocorre ao pedido de conexão.
BeforeDisconnect	Ocorre ao pedido de término da conexão.
OnIdleTimer	Ocorre enquanto o cliente espera por uma resposta do servidor.
OnLogin	Este evento pode ser usado para substituir o pedido de <i>username</i> e <i>password</i> do objeto. Ao definir alguma rotina neste evento, automaticamente a janela de login não será exibida e o nome e a senha deverão ser passados via código.

## TIBTransaction

---

Um dos componentes mais interessantes da nova palheta. O componente IBTransaction representa a instância da transação atual e simplifica muito o controle de transações. Com este objeto, é possível controlar transações concorrentes originárias da mesma conexão com o banco de dados, ou em Threads independentes.

### Principais Propriedades:

Active	Quando <b>True</b> , executa o método <i>StartTransaction</i> no servidor.
DefaultAction	Indica qual sera o metodo executado quando a aplicacao exceder o tempo de idle time out, ou seja quando nenhum comando for mais enviado para o servidor. Seus possiveis valores são:  <b>taRollback</b> Executa a instrução <i>RollBack</i> . <b>taCommit</b> Executa a instrução <i>Commit</i> . <b>taRollbackRetaining</b> Executa a instrução <i>RollBack</i> , e mantém o <i>handle</i> da transação aberto(os dados continuam na tela). Disponível apenas na versão 6.0 ou superior do <b>Interbase</b> . <b>TaCommitRetaining</b> Executa a instrução <i>Commit</i> e mantém o <i>handle</i> da transação aberto(os dados continuam na tela).
DefaultDatabase	Indica a instância do banco de dados no qual a transação será iniciada.
IdleTimer	Especifica quanto tempo o objeto irá esperar para executar o método especificado na propriedade <i>DefaultAction</i> . O tempo começa a ser contado a partir que nenhuma instrução for enviada para o servidor.

### Principais Eventos

OnIdleTimer	Ocorre enquanto a aplicação não envia nenhuma instrução SQL para o servidor.
-------------	--

## Principais Métodos:

Commit	Confirma a transação no servidor.
CommitRetaining	Confirma a transação e mantém o <i>handle</i> aberto. Neste caso, os dados permanecem na tela.
Rollback	Cancela a transação atual.
RollbackRetaining	Cancela a transação e mantém o <i>handle</i> aberto. Neste caso, os dados permanecem na tela. Disponível apenas a partir da versão 6 do Interbase.
StartTransaction	Inicia a transação.
Call	Retorna a mensagens de erro baseado no <i>'Error Code'</i> passado como parâmetro.

## **TIBTable**

---

Representa uma conexão com alguma tabela do banco de dados. Equivalente ao objeto TTable da palheta DataAccess. O uso do objeto TIBTable apresenta desvantagens em relação ao uso do objeto TIBQuery, visto que este componente envia uma quantidade de instrução muito maior ao servidor. É recomendável o uso do objeto TIBQuery.

## Principais Propriedades

Active	Abre a conexão com a tabela.
Database	Objeto TIBDatabase.
TableName	Nome da tabela dentro do banco de dados.
Transaction	Objeto TIBTransaction, que representara a transação ativa para as ações executadas através deste objeto.

Os demais métodos e eventos são os mesmos encontrados no objeto TTable, da palheta DataAccess.

## **TIBQuery**

---

Representa uma conexão SQL com o banco de dados. Praticamente toda a linguagem SQL(DDL/DML) suportada pelo banco de dados pode ser utilizada através deste objeto. Para utilizar o objeto TQuery como um objeto de edição, e recomendável o seu uso juntamente com o objeto TUpdateSQL.

## Principais Propriedades

Database	Objeto <b>TIBDatabase</b> , que representa a instância do banco de dados.
SQL	Código SQL a ser enviado para o servidor.
Transaction	Objeto TIBTransaction, que representará a transação ativa para as ações executadas através deste objeto.

Os demais eventos e métodos deste objeto são os mesmos encontrados no objeto TQuery, da palheta DataAccess.

## TIBStoredProc

---

Permite executar uma procedure armazenada no servidor. Equivalente ao objeto TStoredProc, da palheta DataAccess.

## Principais Propriedades

DataBase	Objeto TIBDatabase, que representa a instância do banco de dados
StoredProcName	Nome da procedure armazenada.

Os demais eventos e métodos são os mesmos do objeto TStoredProc.

## TIBUpdateSQL

---

Permite configurar o objeto TIBQuery *ReadOnly* para trabalhar com **Live Result Sets**. Com este objeto, é possível definir instruções SQL para cada método *Append*, *Edit* ou *Delete*. O objeto UpdateSQL garante maior performance para editar uma tabela, pois a query trabalha em modo *ReadOnly*.

## Principais Propriedades

InsertSQL	Define a instrução SQL a ser executada quando o método <i>Append</i> for chamado.
ModifySQL	Define a instrução SQL a ser executada quando a tabela for editada.
DeleteSQL	Define a instrução SQL a ser executada quando algum registro for deletado.

RefreshSQL	Define a instrução SQL a ser executada quando o método Refresh for executado.
------------	---

Os demais eventos e métodos são os mesmos do objeto TUpdateSQL, da palheta DataAccess.

## **TIBDataSet**

---

Este componente é o objeto ancestral para os componentes de acesso a dados da palheta Interbase. Sua utilização não é essencial, pois o objeto TQuery possui todos os seus recursos.

### **Principais Propriedades**

Active	Indica se a query sera aberta.
BufferChunks	Numero de registros no buffer.
Database	Objeto TIBDatabase, que representa o banco de dados atual.
DeleteSQL	Instrução SQL que será executada quando algum registro for deletado.
InsertSQL	Instrução SQL que será executada quando algum registro for inserido.
ModifySQL	Instrução SQL que será executada quando algum registro for alterado.
RefreshSQL	Instrução SQL que será executada quando o metodo <i>Refresh</i> for chamado.
SelectSQL	Instrução SQL que será executada quando a query for aberta.
UpdateRecordTypes	Indica quais registros estarão visíveis quando a propriedade <i>CachedUpdates</i> estiver setada para True. Por default, as propriedades <b>cusModified</b> , <b>cusInserted</b> , e <b>cusUnmodified</b> estão setadas para True, o que significa que os registros atuais e inseridos estarão visíveis.

### **Principais Eventos**

OnAfterDatabaseDisconnect	Ocorre após o término da conexão com o banco de dados.
---------------------------	--

OnAfterTransactionEnd	Ocorre após a execução de uma transação (com COMMIT ou ROLLBACK). Este evento captura apenas as transações "hard", ou seja, terminadas com o método <b>Commit</b> ou <b>RollBack</b> . Os métodos <b>CommitRetaining</b> e <b>RollbackRetaining</b> não são capturados por este evento.
BeforeDatabaseDisconnect	Ocorre antes do término da conexão com o banco de dados.
OnBeforeTransactionEnd	Ocorre antes da execução de um comando <b>Commit</b> ou <b>RollBack</b> .
DatabaseFree	Ocorre após a liberação dos <i>handle's</i> alocados pelo objeto TIBDatabase.
TransactionFree	Ocorre após a liberação dos <i>handle's</i> alocados pelo objeto TIBTransaction.

Os demais métodos deste objeto são mesmos encontrados na classe TDataSet, pai de vários componentes como TTable e TQuery.

## **TIBSQL**

---

Componente para executar instruções SQL. Sua diferença para com o objeto TQuery, é que este apresenta a maior performance de execução, e aloca o mínimo de recursos possíveis. Em contrapartida, este objeto não oferece interface para conexão com objetos DataControl – ele apenas executa as instruções. É recomendável o seu uso em operações que exijam uma rápida performance, como por exemplo, rotinas de backup. Este componente também pode ser utilizado para criação de aplicações DSQL (Dynamic SQL), ou seja, aplicações em que o próprio usuário cria as instruções SQL.

### **Principais Propriedades**

Database	Objeto TIBDatabase que representa o banco de dados atual.
GoToFirstRecordOnExecute	Se <b>True</b> , vai para o primeiro registro após a execução de uma instrução <b>SQL Live Result Set</b>

ParamCheck	Esta propriedade deve ser True quando o objeto TIBSQL possuir uma instrução SQL DDL com parâmetros. Se <b>True</b> , o objeto irá prevenir de não misturar os parâmetros definidos na propriedade SQL com os parâmetros contidos dentro da instrução SQL. Por exemplo, se a instrução criar uma Stored Procedure com um parâmetro, setar esta propriedade para True irá prevenir o objeto de tentar substituir o parametro da instrução <b>CREATE PROCEDURE</b> por um valor da propriedade <i>Params</i> .
SQL	Instrução que será executada.
Transaction	Objeto TIBTransaction que irá representar a transação atual.

## Principais Eventos

OnSQLchanging	Ocorre quando a instrução SQL é modificada.
---------------	---

## Principais Metodos

Call	Function Call(ErrCode: ISC_STATUS; RaiseError: Boolean): ISC_STATUS;  Retorna a mensagem de erro correspondente ao código de erro.
CheckClosed	Gera uma exceção quando a query está aberta.
CheckOpen	Gera uma exceção quando a query está fechada.
CheckValidStatement	Gera uma exceção se a query não possuir uma cláusula SQL válida.
Close	Elimina os <i>handles</i> gerados pela query.
Current	function Current: TIBXSQLDA;  Retorna o tipo IBXSQLDA atualmente alocado pela query. Este <i>descriptor</i> representa uma área de transferência de dados entre o banco de dados e aplicação cliente. Toda execução de uma instrução SQL cria uma instância deste <i>Record</i> .
ExecQuery	Executa a query SQL.
FieldByName	Utilizado para acessar um campo da tabela através de seu nome.

Next	Move o ponteiro para o próximo registro.
Prepare	Utilizado antes de passar parâmetros para o objeto. Permite preparar uma Query para execução.

## **TIBDatabaseInfo**

---

Retorna varias informações a respeito do banco de dados especificado. Basta inserir um componente TIBDatabaseInfo, configurar sua propriedade Database para o objeto TIBDatabase correspondente, e capturar os valores de suas propriedades.

### **Principais Propriedades:**

Allocation	Retorna o número de páginas alocadas.
BackoutCount	Indica o número de vezes que um registro do banco de dados foi removido.
BaseLevel	Retorna o número de versão do banco de dados.
CurrentMemory	Retorna a quantidade de memória atualmente alocada pelo servidor.
DataBase	Objeto TIBDatabase, que indicará o banco de dados provedor das informações.
DBFilename	Retorna o nome do arquivo de banco de dados.
DbImplementationClass	Retorna o número de classe do banco de dados.
DbImplementationNo	Retorna o número de implementação do banco de dados.
DbSiteName	Retorna o nome do site do banco de dados.
DbSQLDialect	Retorna o número <b>SQL Dialect</b> atualmente utilizado pelo servidor.
DeleteCount	Retorna o número de <i>Deletes</i> executados desde a última conexão com o servidor.
ExpungeCount	Retorna o número de registros removidos.
Fetches	Retorna o número de <i>Fetchs</i> realizados.
ForcedWrites	Indica como a gravação de dados está sendo realizada. 0 para assíncrono e 1 para síncrono.
InsertCount	Retorna o número de <i>Inserts</i> executados desde a última conexão com o servidor.
Marks	Retorna o número de gravações efetuadas no buffer
MaxMemory	Retorna, em bytes, a quantidade de memória utilizada desde a primeira conexão realizada

NoReserve	Indica se espaço reservado para backup de registros modificados no banco
NumBuffers	Indica o número de buffers alocados
ODSMajorVersion	Retorna a parte superior da versão do ODS
ODSMajorVersion	Retorna a parte inferior da versão do ODS
PageSize	Retorna o número de bytes alocados por página
PurgeCount	Retorna o número de <i>Purgings</i> realizados
ReadIdxCount	Retorna o número de leituras realizadas através de índices.
ReadOnly	Indica quando o banco de dados é ou não read-only
Reads	Retorna o número de páginas lidas no banco
ReadSeqCount	Retorna o número de leituras seqüenciais realizadas nas tabelas do banco
SweepInterval	Retorna o número de transações confirmadas entre "sweeps"
UpdateCount	Retorna o numero de <i>Updates</i> executados desde a última conexão com o servidor.
UserNames	Retorna em uma lista de String, os usuários conectados no banco de dados.
Version	Retorna a versão do Interbase.
Writes	Retorna o número de páginas de dados escritas no banco.

## **TIBSQLMonitor**

---

Permite criar um log de todas as operações e instruções SQL que foram enviadas ao servidor. Como o aplicativo SQL Monitor não pode monitorar as intruções SQL enviadas pelos componentes Interbase (pois trabalha no nível da camada BDE), o desenvolvedor precisa usar este componente para monitorar e otimizar a aplicação cliente.

O uso deste objeto é muito simples. Não existem propriedades a serem configuradas, basta inserir um componente IBSQLMonitor na aplicação (ou em outro aplicativo). Após, o usuário deve configurar a propriedade **TraceFlags** do objeto Database da aplicação cliente, indicando quais serão os comandos filtrados pelo objeto IBSQLmonitor.

As instruções SQL podem ser capturadas através do evento **OnSQL** do objeto. Este evento passa um parâmetro **EventText**, do tipo string, indicando o comando enviando para o servidor. Por exemplo, a aplicação poderia ter um ListBox para receber as instruções:

```
procedure TForm1.IBSQLMonitor1SQL(EventText: String);
```

**begin**

```
ListBox1.Items.Add(EventText);
```

**end;**

Com este código, todos os comando enviados ao servidor (definidos na propriedade *TraceFlags* do objeto Database) seriam exibidos no ListBox.

## **TIBEvents**

---

Componente para utilizar os “eventos” do Interbase. O conceito de eventos no Interbase é muito interessante: o desenvolvedor pode programar eventos no banco de dados, que serão executados em instruções como **INSERT**, **UPDATE** ou **DELETE**. Estes eventos podem ser programados em *Triggers* ou *Stored Procedures*, e as aplicações que registrarem os eventos através do objeto TIBEvents, receberão uma notificação toda vez que algum registro for inserido ou deletado, por exemplo. Este objeto é muito útil para criar telas de refresh automático. Por exemplo, um evento poderia ser criado toda vez que um registro for inserido na tabela. Todas as máquinas clientes iriam receber este evento, e o software poderia ser programado para dar um *Refresh* em suas Query’s abertas, criando o efeito de *Refresh* automático sem causar *Overhead* de CPU. Este objeto roda em um Thread separado, e não degrada a performance do sistema.

### **Principais Propriedades**

Database	Objeto TIBDatabase, que representa o banco de dados atual.
Events	Nome dos eventos que serão recebidos pelo objeto.
Registered	Se <b>True</b> , indica que o objeto estará ativo.

### **Principais Eventos**

OnEventAlert	Ocorre quando algum evento é recebido pelo objeto. Setar o parâmetro <b>CancelAlerts</b> para <b>True</b> , indica que o evento recebido deverá ser ignorado. Para retornar e receber determinado evento, basta chamar o método <b>QueueEvents</b> . Nem todas as operações envolvendo objetos da VCL poderão ser executadas neste evento, visto que ele roda em um Thread separado.
--------------	--

## Principais Metodos

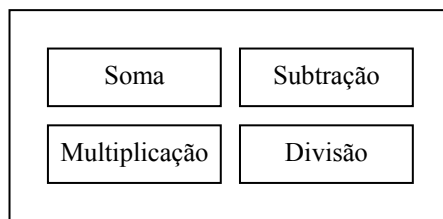
CancelEvents	Cancela o recebimento dos eventos pendentes.
QueueEvents	Indica ao objeto para iniciar o recebimento dos eventos.
RegisterEvents	Registra os eventos listados na propriedade <b>Events</b> . Este método já executa o método <b>QueueEvents</b> .
UnRegisterEvents	Cancela o recebimento dos eventos listados na propriedade <i>Events</i> .

## Palheta Servers

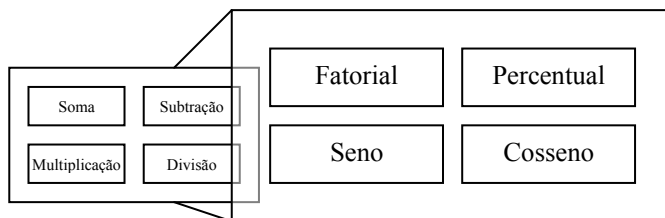
Esta palheta permite uma conexão COM(Components Object Model) com aplicativos da família Office. O Controle destes aplicativos já estavam disponíveis no Delphi 4, através de OLE Automation, porém, a interface COM é o mais novo padrão da Microsoft e, além de oferecer mais poder ao desenvolvedor, será a tendência dos novos aplicativos para Windows.

A arquitetura COM se baseia em pequenos objetos, com funcionalidade própria, que podem se conectar a outros objetos e formar o aplicativo. Nesta visão, podemos dizer que o Word e o Excel são, na verdade, um grande conjunto de objetos COM interfaceados e fornecendo uma solução integrada. Este objetos podem ser acessados por objetos COM de outros aplicativos, e a reutilização de funções pode ser conseguida de uma maneira muito prática.

Vamos imaginar, por exemplo, que estivéssemos construindo uma aplicação calculadora científica em C++. Primariamente, deveríamos construir quatro objetos COM: um para cada operação matemática básica (somar, diminuir, multiplicar e dividir). Com estes quatro objetos, poderíamos prosseguir e criar mais quatro: fatorial, porcentual, seno e cosseno. Teríamos uma família de objetos COM parecida com a ilustração abaixo:



Passo 1: Criação dos objetos básicos.



Passo 2: Criação de novos objeto a partir dos objetos básicos.

Com a calculadora criada, imaginemos agora que iniciaremos a construção de um aplicativo contábil em Delphi. Certamente iremos precisar das funções imbutidas na calculadora, mas não queremos reescrever o código. Teríamos três opções:

- Criar uma DLL em C com as funções da calculadora exportadas;
- Criar um objeto .OCX contendo as funções da calculadora;
- Utilizar os objetos COM do aplicativo calculadora diretamente da aplicação DELPHI;

Neste caso, a opção 3 oferece as vantagens:

No caso da DLL, se alguma função fosse incluída, deveríamos nos preocupar em atualizar tanto a DLL quanto a aplicação cliente, visto que esta precisa de um header para importar as funções. Outro problema se refere a memória, pois a DLL seria carregada uma vez para cada processo. Isto também vale para a OCX, já que cada processo cria uma instância completa da OCX em questão.

Já ao utilizarmos o objeto COM diretamente do aplicativo calculadora, teríamos maior facilidade para manutenção, já que não precisaríamos declarar "headers" nas aplicações clientes. Outro ponto positivo se deve ao fato de que apenas os objetos necessários seriam instanciados, e não todo o aplicativo (a DLL e a OCX seriam carregadas por completo na memória, mesmo que estivéssemos utilizando apenas uma parte do todo).

Podemos também oferecer a calculadora como um objeto distribuído (DCOM – Distributed COM), e instalar o aplicativo apenas no servidor. As máquinas clientes iriam se conectar ao objeto COM através da LAN, e desta forma, ao atualizar alguma das funções do objeto, deveríamos apenas atualizar a calculadora no servidor, nada mais.

**A palheta Servers oferece a funcionalidade COM** dos aplicativos da família Office. Entendemos isto como: MSWord, MSEExcel, MSPowerPoint, MsBrinder, MSOutlook e MSAccess. É importante notar que qualquer outro aplicativo ou objeto COM pode ser acessado através do Delphi, porém via código.

A funcionalidade COM dos aplicativos Office é muito extensa, e o objetivo desta apostila não é o de ser uma referência sobre o assunto. Segue abaixo alguns exemplos práticos sobre o uso desta palheta:

## **Inicializando o Word**

---

Insira um componente TWordApplication e utilize o código:

```
WordApplication1.Connect;  
WordApplication1.Visible := True;
```

## **Abrindo um documento em branco no Word:**

---

Insira um objeto TWordDocument, e execute o metodo NewInstance:

```
WordDocument1.Activate;           //Exibe o aplicativo Word
                                   (não e necessario)
WordDocument1.NewInstance;        //Cria um novo documento
```

## **Inserindo texto no final do arquivo:**

---

```
var
  Texto :WideString;
begin
  Texto := 'WorkShop Delphi 5 Novos Eecursos'
+chr(13)+chr(10)+ 'www.clubedelphi.com.br';
WordDocument1.Content.InsertAfter(t);
```

## **Substituindo uma palavra:**

---

Neste caso, a palavra de indice 2 sera substituida pela string Borland Delphi.

```
WordDocument1.Words.Item(2).Text:= 'Borland Delphi';
```

## **Alterando a formatação de uma palavra:**

---

Insira um objeto TWordFont e um TWordDocument. Em seguida, digite o codigo:

```
Var
  T: WideString;
begin
  WordFont1.ConnectTo(WordDocument.Content.Font);
  WordFont1.Bold:=1;
  WordFont1.Italic:=1;
  WordDocument1.Content.InsertAfter(t); //o texto sera
  inserido com a formatacao negrito e italico
End;
```

## **Inserindo texto no meio de um arquivo :**

---

```
var
  T: WideString;
begin
  T := 'Clube Delphi';
  WordDocument1.Words.Item(2).InsertAfter(t); //Este
  comando ira inserir a string 'Clube Delphi' apos a
  Segunda palavra
end;
```

## **Inserindo uma figura dentro do arquivo:**

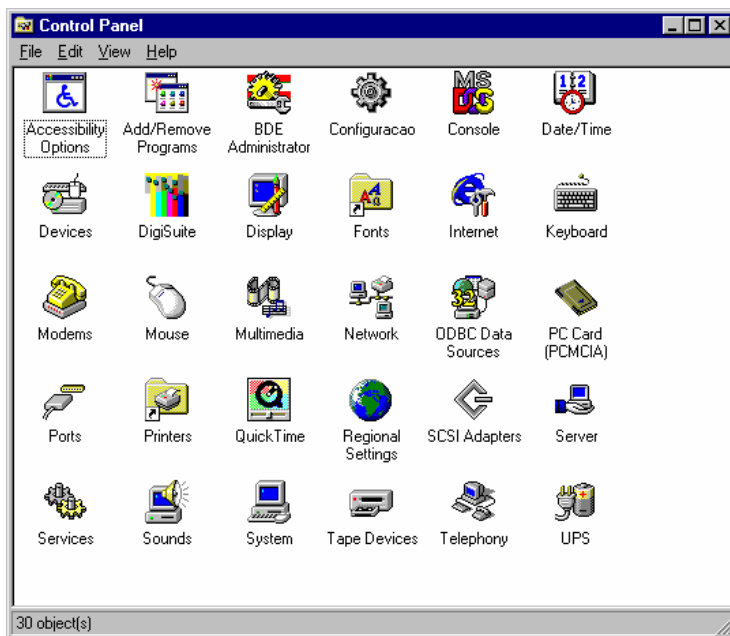
---

```
var
  formato : Word;
  Data : THandle;
  Palette: HPALETTE;
begin

  Image1.Picture.SaveToClipboardFormat (Formato, Data, Palett
  e);
  //copia o conteudo do objeto image para o clipboard
  Clipboard.SetAsHandle (Formato, Data);
  WordDocument1.Content.Paste;
  //cola a area de transferencia dentro do documento
end;
```

## Control Panel Applet

Arquivos .CPL agora podem ser facilmente criados com o Delphi. Arquivos CPL (Control Panel Extension) nada mais são do que os aplicativos reconhecidos pelo Windows como conteúdo do Painel de Controles. Um arquivo CPL possui a mesma estrutura de um arquivo executável, porém, seu cabeçalho é diferente.

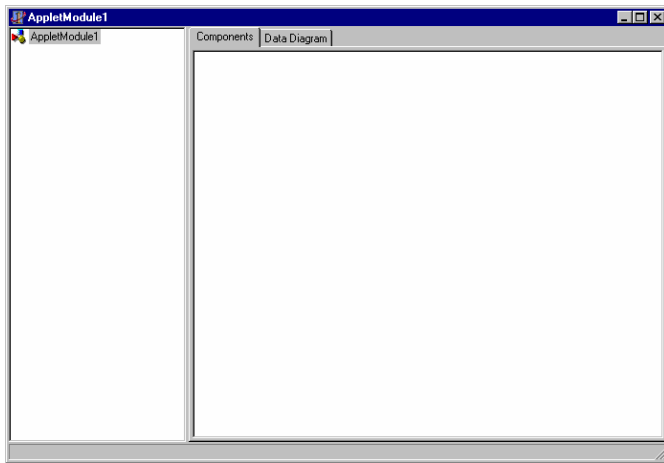


Cada ícone representa um arquivo .CPL, localizado no diretório Windows/System

## **Vejamos como criar uma aplicação CPL:**

Vá no *Object Repository* e selecione **Control Panel Application**;

Repare que um objeto TAppletModule será exibido:



Crie agora o formulário que será chamado pela aplicação, através do menu File, New form.

Selecione o objeto AppletModule1 e altere o seu evento OnActivate:

```
procedure TAppletModule1.AppletModuleActivate(Sender:
TObject;
  Data: Integer);
begin
  Form1.ShowModal;
end;
```

Altere as propriedades do objeto AppletModule1 de acordo com a tabela abaixo:

AppletIcon	Ícone que será exibido no Painel de Controle. Selecione algum arquivo .ICO do diretório de imagens do Delphi.
Caption	Texto que irá aparecer abaixo do ícone, representando o nome da aplicação. Digite Teste.
Help	Texto que será exibido na barra de status do painel de controle. Digite "Aplicação de teste".

Em seguida, salve o projeto. Clique com o botão direito sobre o objeto AppletModule1 e selecione a opção **Install Control Panel Applet**, para que a aplicação seja copiada para o diretório System do Windows.

Abra o Painel de Controle e teste o seu aplicativo!

**Nota:** O processo de instalação do aplicativo no Painel de Controle se limita apenas à cópia do arquivo .CPL para o diretório System do

Windows. Nenhum tipo de alteração no registro é necessário. O mesmo vale para a desinstalação do aplicativo.

## **Objeto TAppletModule**

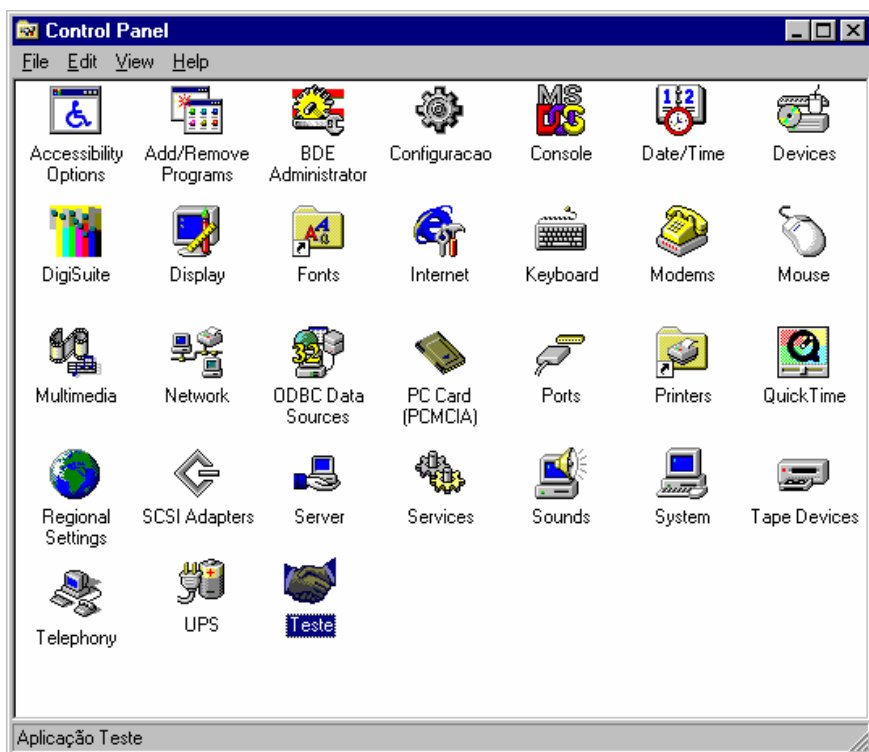
---

### **Principais Propriedades:**

AppletIcon	Ícone que será exibido no Painel de Controle.
Caption	Texto que irá aparecer abaixo do ícone, representando o nome da aplicação.
Help	Texto que será exibido na barra de status do painel de controle.
ResidIcon	Indica o número do ícone a ser exibido. Lembre-se que um aplicativo pode possuir vários ícones.
ResidInfo	Indica o número do <i>Resource</i> da string de help. Várias strings de Help só podem ser adicionadas com um editor de recursos.
ResidName	Indica o número do <i>Resource</i> da string referente ao <i>Caption</i> . Vários Captions só podem ser adicionados com um editor de recursos.

### **Principais Eventos**

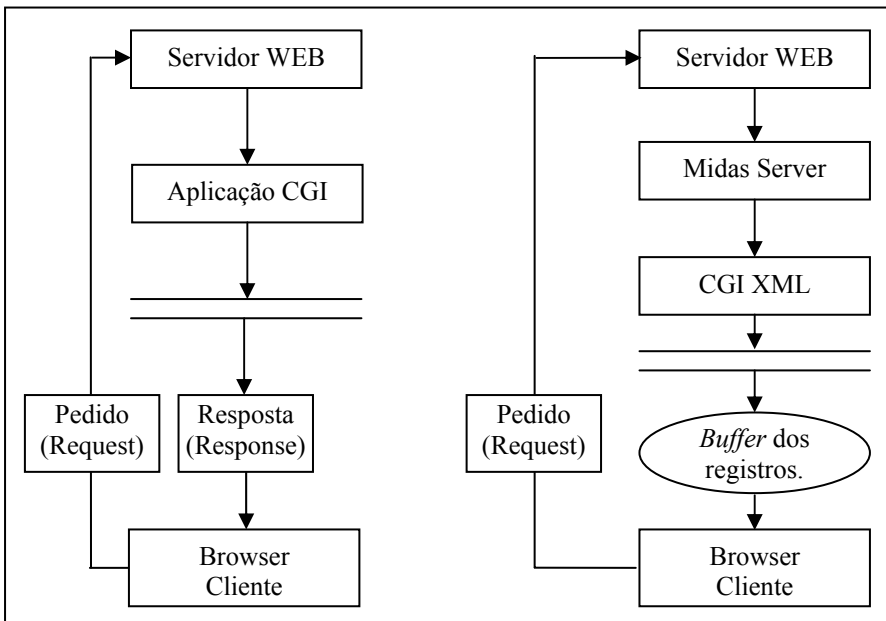
OnActivate	Ocorre quando o usuário inicia a aplicação através do painel de controle.
OnInquiry	Ocorre quando o Windows (Painel de Controle) lê as informações de recurso do aplicativo, como ícones, textos e outros.
OnStop	Ocorre quando o Windows (Painel de Controles) inicia o processo de <i>ShutDown</i> do aplicativo.
OnStartWParams	Ocorre quando o aplicativo é iniciado por outro aplicativo, e não pelo usuário.
OnCreate	Ocorre quando o <i>AppletModule</i> é instanciado.
OnDestroy	Ocorre quando o <i>AppletModule</i> é removido da memória.



## Internet Express

Esta palheta vem adicionar um recurso muito interessante para o desenvolvedor Delphi: o fato de poder exportar dados em formato **XML**. A XML (*Extended Markup Language*) é uma extensão da HTML, e traz várias melhorias no sentido de performance e flexibilidade. Uma das principais diferenças para quem trabalha com banco de dados, é que a XML "bufferiza" os dados no *Browser*, não necessitando executar um pesado *Request* a cada movimentação de registro feita pelo usuário.

Enviar e receber dados em pacotes XML é uma tarefa que também necessita de uma camada *MIDAS* para captura da informação no banco de dados. Note a diferença de camadas em uma aplicação *ISAPI* normal e uma aplicação *ISAPI XML*:



## **Vamos criar passo a passo uma aplicação para prover informações em pacotes XML:**

---

Primeiramente, devemos criar o pequeno servidor MIDAS que irá buscar as informações no banco de dados.

Crie um novo projeto no Delphi:

Dentro da nova aplicação, abra o Repositório de Objetos;

Selecione a aba MultiTier;

Clique duas vezes sobre o item Remote Data Module, que a janela a seguir irá aparecer;



Confira na tabela abaixo o significado das opções:

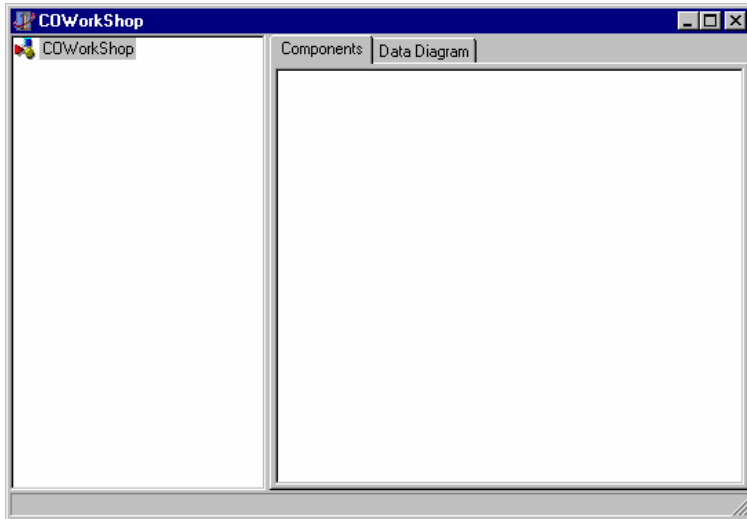
CoClassName	Nome da classe que será registrada no Windows;
Instancing	Indica como o Datamodule será instanciado. Os possíveis valores são: <b>Internal</b> Cria a instância do DataModule dentro do próprio processo. Recomendável quando o servidor que estiver sendo construído for uma DLL; <b>Single Instance</b> Uma instância da aplicação será criada para cada cliente conectado; <b>Multiple Instance</b> Apenas uma instância da aplicação será criada. Uma instância do DataModule será criada para cada cliente conectado, porém, dentro do espaço de processo da aplicação.

Threading Model	<p>Utilizado apenas quando o Remote Data Module está sendo criado dentro de uma DLL. Confira os possíveis valores:</p> <p><b>Single</b> Somente uma requisição será processada por vez. As demais permanecerão em fila;</p> <p><b>Apartment</b> O Datamodule1 é instanciado para cada requisição do cliente. Nesta situação, os dados estão seguros, pois todos os acessos serão safe-thread, ou seja, livre de conflitos de memória. Esta opção é recomendável para aplicações que utilizem o BDE;</p> <p><b>Free</b> Este thread Terá um comportamento parecido com Apartment, porém, neste caso, os dados não estarão Thread-Safe. As concorrências deverão ser tratadas para que um conflito de acesso a memória não ocorra;</p> <p><b>Both</b> Igual a opção <i>Free</i>, porém, o retorno para o cliente será feito em fila, e não em threads.</p>
-----------------	--

Baseado nisto, vamos configurar a caixa de diálogo conforma a tabela abaixo:

CoClassName	COMidasServer
Instancing	Multiple Instance
Tread Model	Apartment

Após dar OK na caixa de diálogo, o objeto TremoteDataModule irá aparecer:



Primeiramente, devemos colocar um objeto TSession, que irá controlar as ações de vários usuários concorrentes dentro da aplicação. É importante também configurar a propriedade AutoSessionName para TRUE. Esta propriedade irá criar um nome de seção diferente para cada usuário conectado na aplicação.

Após, um objeto Database deve ser inserido no DataModule1. Configure corretamente o objeto database para que aponte para o alias DBDEMOS.

E, em seguida, coloque um objeto Ttable no DataModule. Linke o table com o objeto Database e aponte-o para a tabela Animals.

Devemos agora inserir o objeto que ira prover os dados para a aplicacao XML. Este objeto será da classe TdataSetProvider, e ele se encontra na palheta MIDAS. Coloque um objeto DataSetProvider e configure-o de acordo com a tabela abaixo:

DataSet	Table1
Name	ProviderAnimals

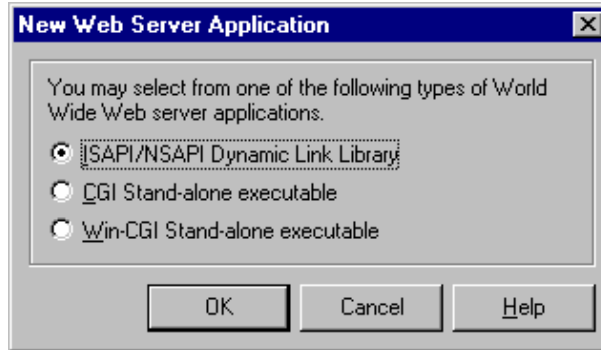
E OK! Salve a aplicação e gere o executável. A partir deste momento teremos um servidor midas instalado na maquina.

Devemos agora criar a aplicacao XML, que irá recuperar os dados através do servidor MIDAS e fornecê-los para a máquina cliente através do protocolo HTTP.

Crie uma nova aplicação no Delphi.

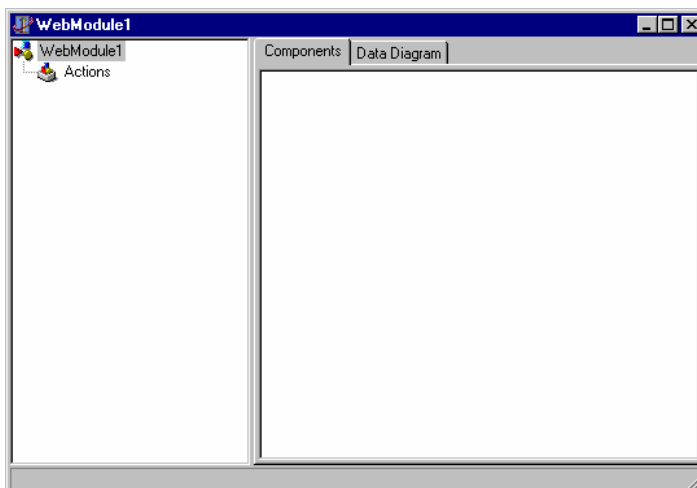
Abra o repositório de objetos e selecione a opção Web Server Application.

A janela de seleção Web Server Application será exibida:



ISAPI/NSAPI	A aplicação será gerada em forma de DLL. Esta opção irá oferecer melhor desempenho e menos consumo de memória.
CGI	A aplicação será gerada em forma de executável (.EXE). Esta opção, apesar de ser bem prática, oferece desempenho inferior ao ISAPI.
Win/CGI	Variação do CGI criada para ser utilizada em servidores Windows 16 bit.

Selecione a opção ISAPI e dê OK. Um novo objeto TwebModule será exibido:



Primeiramente, coloque um objeto TDCOMConnection (palheta MIDAS). Este objeto irá realizar a conexão com nosso servidor MIDAS. Configure suas propriedades de acordo com a tabela abaixo:

ServerName	Nome do servidor Midas a ser conectado
ComputerName	Deve ser preenchido apenas quando o servidor midas não estiver local. Neste caso, o nome da máquina onde o servidor está instalado deve ser assinalado nesta propriedade
Connected	TRUE

Em seguida, insira dois objetos TXMLBroker (palheta MIDAS) dentro do WebModule. Este objeto irá recuperar os dados das tabelas contidas dentro do servidor. Configure suas propriedades de acordo com a tabela abaixo:

Name	XMLBroker1
RemoteServer	Objeto TDCOMConnection, que representa a conexão com o servidor MIDAS. Selecione DCOMConnection1
ProviderName	Nome do objeto DataSetProvider que representa as tabelas exportadas pelo servidor MIDAS. Selecione ProviderAnimals.

Name	XMLBroker2
RemoteServer	DCOMConnection1
ProviderName	ProviderClients

Em seguida, devemos inserir a última classe requerida, TmidasPageProducer. Este objeto irá criar a página XML baseada nos dados da tabela. Insira um objeto TmidasPageProducer no formulário.

Selecione o objeto MidasPageProducer1 e dê um clique duplo sobre a propriedade WebPageItems do objeto. O Editor HTML será exibido:



Clique com o botão direito sobre o item `MidasPageProducer1` e selecione a opção `New Component`. Em seguida, escolha `DataForm`. Após, clique com o botão direito sobre o novo item, `DataForm`. Desta vez, selecione `Datagrid`. Selecione o novo item, `DataGrid1` e configure sua propriedade `XMLBroker` para o nome do objeto `XMLBroker` que irá prover os dados convertidos em XML. Neste caso, selecione `XMLBroker1`. Repare que o `DbGrid` já irá exibir o título das colunas.

O último passo é criar uma ação para executar este `MidasPageProducer`. Clique com o botão inverso no objeto `WebDataModule` e selecione a opção `actions editor`. No editor de ações, crie uma nova ação através do botão `New Action`. Um objeto `TwebAction` será criado. Configure suas propriedades de acordo com a tabela abaixo:

Default	TRUE. Isto irá executar está quando o usuário digitar o nome do aplicativo no browser, sem nenhum parâmetro. Ex: <a href="http://www.clubedelphi.com.br/project1.exe">www.clubedelphi.com.br/project1.exe</a>
Producer	Nome do objeto responsável pela reposta ao cliente. Neste caso, será o objeto <code>MidasPageProducer1</code> , que retornará uma página XML para o cliente.

PathInfo	Parâmetro que o usuário deverá digitar na URL para executar esta ação. Como esta é a ação default, não é necessário informar esta propriedade
----------	---

E Ok! Compile o projeto, e repare que o aplicativo project1.dll será adicionado no diretório da aplicação.

O próximo passo é copiar o aplicativo para algum diretório de aplicações do servidor WEB. No diretório da aplicação, alguns arquivos JavaScript deverão ser copiados. Estes arquivos estão no diretório <Delphi>\Source\WebMidas:

Xmlldb.js  
Xmldisp.js  
Xmldom.js  
Xmlerrdisp.js  
Xmlshow.js

É importante notar que o servidor WEB deverá estar corretamente configurado e ajustado para que a aplicação funcione de forma estável.

## **Utilizando o DBNavigator com XML**

---

Vamos criar um outro tipo de interface para a nossa Homepage: Os dados serão exibidos em campos e o usuário poderá navegar entre eles através de uma barra de navegação. Para a construção deste exemplo, abra o cliente XML construído anteriormente.

Insira um novo MidasPageProducer. Com este, iremos contruir a nova página de visualização dos dados.

Dê um clique duplo sobre a propriedade WebPageItems, para que o editor de páginas seja exibido.

Selecione o item MidasPageProducer1 e clique com o botão inverso do mouse. Selecione a opção New Component...

Na janela de seleção, selecione a opção DataForm.

Clique sobre o item dataform1 e pressione o botão inverso do mouse. Selecione a opção FieldGroup.

Clique novamente sobre o item Dataform1 e adicione o item DataNavigator.

Ok! Será necessário criar uma ação para este PageProducer. Clique com o botão inverso sobre o WebDataModule e selecione a opção Action Editor.

Crie uma nova ação e configure sua propriedade Producer para o objeto MidasPageProducer2, que acabara de ser criado.

Compile o projeto e instale-o no servidor Web. Para testar a aplicação basta digitar a url:

[http://servidorweb/diretorio\\_da\\_aplicação/project1.exe/MidasPageProducer2](http://servidorweb/diretorio_da_aplicação/project1.exe/MidasPageProducer2)

E o resultado será parecido com a ilustração abaixo:

