Índice

Introdução

Esta página pretende ensinar aos desenvolvedores do Borland Delphi 3 a construir aplicativos de servidor WEB. Se este material lhe serviu em algum propósito, se gostou ou até mesmo se possui críticas, por favor assine meu Livro de Visitas, deixe lá seu comentário. Com a sua ajuda poderemos melhorar cada vez mais.

Parto do princípio que, o leitor desta página, já possui conhecimentos de HTML e até mesmo em como funciona um Servidor WEB, seja ele em UNIX ou Microsoft Windows NT. Se desejar, poderá obter maiores esclarecimentos nestes assuntos seguindo os Links em Leituras Complementares.

É altamente recomendável que desenvolvamos nossos aplicativos WEB em servidores locais, e quando estiverem totalmente livres de erros, passarmos os mesmos para os servidores de produção.

Apesar dos exemplos aquí estarem todos feitos em ISAPI, sua conversão para o padrão CGI é bastante simples. Tudo que temos a fazer é criar um novo aplicativo do tipo CGI, excluir as *Units* que vêem por *default*, e então adicionar as nossas. Poderemos ter dois projetos, um CGI e outro ISAPI, compartilhando das mesmas *Units*.

Boa Sorte.

CGI

CGI - Commom Gateway Interface, nada mais é do que uma definição de como o servidor

WEB e o Browser se comunicam, para permitir aos autores de páginas HTML fornecerem conteúdo dinâmico e personalizado de acordo com informações enviadas por seus usuários. Uma página dinâmica é uma página que é gerada na hora a partir de modêlos existentes sem, contudo, existir fisicamente no disco do servidor.

O CGI possibilitou ao servidor WEB produzir um conteúdo que pode ser entendido por um Browser a partir de um conteúdo não acessível, como por exemplo um Banco de Dados. Dessa maneira, o CGI age como uma ponte entre dois produtos, permitindo ao usuário ter acesso à informações que antes só poderiam ser lidas por um sistema desenvolvido especificamente para este propósito, transformando o Browser WEB em um perfeito "Cliente Universal".

Um aplicativo CGI é um programa que é executado pelo servidor WEB, em resposta a uma solicitação do navegador, e que escreve uma página HTML como resultado. Esta página é então enviada de volta ao navegador.

A seguir os passos detalhados de um processo de CGI:

01 - O navegador pede ao servidor WEB para executar determinado aplicativo CGI, ou seja, gera uma requisição.
02 - O servidor WEB inicia um novo processo para rodar o aplicativo CGI
03 - O CGI é executado e recebe parâmetros, que podem estar localizados em propriedades diferentes, dependendo do tipo de requisição (*GET* ou *POST*). Neste ponto poderemos acessar banco de dados, fazendo consultas ou mesmo alterações.
04 - Após efetuar suas tarefas, o CGI deve escrever em *stdout*, que será enviado ao navegador.

ISAPI (Internet Server API)/ NSAPI (Netscape Server API)

Com a chegada do servidores WEB ao mundo Windows, alguns fabricantes, como Netscape e Microsoft, resolveram criar API's proprietárias de acesso ao serviço WEB, se utilizando de recursos existentes no Windows para contornarem o principal problema do CGI, que é justamente a natureza de todo programa executável. Um EXE tem que ser carregado na memória, executado em seu próprio espaço de endereçamento e, finalmente encerrado e retirado da memória, isto tudo para cada requisição cliente.

Através destas API's o servidor WEB pôde, então, tirar proveito do mecanismo de DLL's do Windows para carregar o ISAPI ou o NSAPI apenas uma vez, no seu próprio espaço de endereçamento. Dessa maneira cada requisição passou a gerar apenas uma nova "thread" ao invés de um processo inteiro.

As vantagens desta abordagem sobre o CGI é que os aplicativos se tornam mais rápidos, principalmente no seu tempo de carga, porém passam a depender de um único tipo de servidor WEB - Isso antes do Delphi 3, pois agora o mesmo aplicativo pode, facilmente, se transformar de CGI para ISAPI/NSAPI e vice-versa.

Existe um problema com os aplicativos ISAPI ou NSAPI. Como eles rodam no mesmo espaço de endereçamento do servidor WEB, os mesmos só poderão ser substituídos ou apagados quando o serviço www, ou até mesmo o servidor, estiver fora do ar. Desligar e ligar um servidor WEB não é nada difícil, más se for de produção pode ocasionar problemas.
O Microsoft Personal Web Server 4 não é recomendável para desenvolvimento, pois uma vez no ar, só poderá ser retirado após a inicialização da máquina. Se possuir o Windows 95 poderá instalar o Personal Web Server 1.0a, que é perfeito para desenvolvimento. O Windows 95 OSR2 já o possui, e em português.

Borland Delphi 3 Client/Server

Com o Delphi 3 Client/Server podemos desenvolver quaisquer destes aplicativos para WEB, sem termos que nos preocupar com os detalhes inerentes a cada uma destas API's.

Seu esquema é bem simples e poderoso, graças à sua "Orientação à Objetos". Um aplicativo WEB é na realidade um objeto descendente da classe *TWEBApplication*, que para cada chamada irá criar dois novos objetos descendentes de *TWEBRequest* e *TWEBResponse,* para tratar respectivamente das requisições e das respostas. Dessa forma ficamos completamente afastados de toda a complexidade que é a programação a nível de API de um servidor WEB, como tinha que ser feito com o Delphi 2.

Para aplicativos ISAPI ou NSAPI, que são DLL's, teremos que efetuar alterações no fonte do projeto gerado pelo Delphi. Teremos que incluir, nas primeiras linhas após o **begin** o seguinte:

IsMultiThread := True;
Application.CacheConnections := False;

Nosso projeto então deverá ter, aproximadamente, esta codificação:

```
begin
  // Duas linhas incluídas
  IsMultiThread := True;
  Application.CacheConnections := False;
  //
  Application.Initialize;
  Application.CreateForm(TWebModule1, WebModule1);
  Application.Run;
end.
```

Tipos de Aplicativos WEB

Nós podemos criar quatro tipos de aplicativos para WEB, chamados de "WEB Server Extensions", cada tipo usa um descendente específico de *TWEBApplication, TWEBRequest* e *TWEBResponse*.

Tabela de Tipos de Objetos Aplication, Request e Response por Tipo de Aplicativo:

Tipo de Aplicativo WEB	Objeto Application	Objeto Request	Objeto Response
Microsoft Server DLL (ISAPI)	TISAPIApplication	TISAPIRequest	TISAPIResponse
Netscape Server DLL (NSAPI)	TISAPIApplication	TISAPIRequest	TISAPIResponse
Console CGI Application	TCGIApplication	TCGIRequest	TCGIResponse
Windows CGI Application (Win-CGI)	TISAPIApplication	TWinCGIRequest	TWinCGIResponse

Todos os tipo de aplicativos WEB são criados da mesma maneira, selecionando na IDE do Delphi:

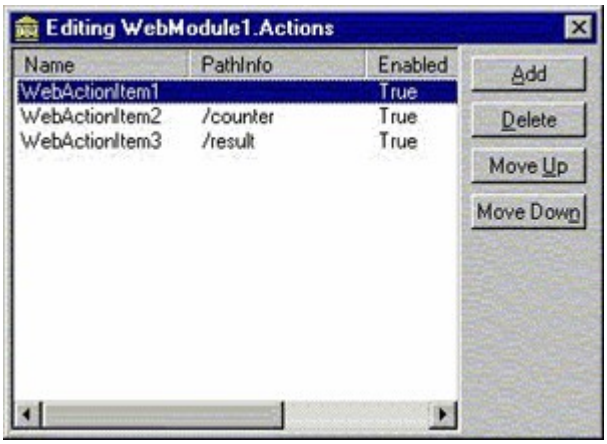
File -> New -> Web Server Application

A partir deste ponto escolhemos o tipo, de aplicativo WEB, que melhor se ajusta ao Servidor WEB no qual será implementado. O Delphi criará um novo projeto com um *TWEBModule* vazio. Este *TWEBModule* é um descendente direto de um *TDataModule* comum, e deve ser usado da mesma maneira, ou seja, podemos colocar nele componentes, principalmente os de Banco de Dados, sendo que a principal diferença é que ele age como um "Dispatcher", tratando as requisições dos clientes através de *Action Items*, que são os verdadeiros responsáveis pelo preenchimento e envio das respostas.

Como é comum a conversão de um sistema já em funcionamento em um aplicativo WEB, o Delphi 3 nos oferece um outro componente chamado *TWEBDispatcher*. Com ele nós podemos adicionar toda a funcionalidade de um *TWEBModule* à um *TDataModule* existente. Isso é feito da seguinte maneira: Após gerarmos um novo aplicativo WEB, retiramos do projeto o *TWEBModule* gerado e adicionamos nosso *TDataModule*, depois solta-se sobre ele um *TWEBDispatcher* e pronto, metade do trabalho foi poupado.

Action Items

Existe uma propriedade no *TWEBModule* chamada *Actions*, que uma vez acionada invoca o *Action Editor*, conforme podemos ver na figura abaixo:



A partir daí podemos adicionar quantos *Action Items* quisermos, dependendo claro, de quantos tipos de requisições desejamos tratar. Isto funciona da seguinte maneira: Quando o servidor WEB recebe uma requisição do tipo:

http://www.server.com/scripts/myappl.dll/counter	ISAPI ou NSAPI DLL
http://www.server.com/scripts/myappl.exe/counter	Executável CGI

ele executa o programa myappl, seja uma DLL ou um EXE. É neste ponto onde se dá a verdadeira diferença entre o ISAPI e o CGI, ou seja, no modo como o servidor WEB "executa" cada um deles. No caso da DLL, ela será carregada apenas na primeira chamada, permanecendo na memória como uma extensão do servidor até ocorrer um "shutdown" do serviço WEB. Nosso *TWEBModule* é um só, enquanto que os *Action Items* são criados a cada requisição, como uma nova *thread*. Por este motivo o evento do *TWEBModule* chamado *OnCreate* não será executado para toda requisição, más no EXE sim, este seria então o local ideal para uma conexão com um Banco de Dados Cliente/Servidor. Já o evento *BeforeDispatch* irá ocorrer para todas as requisições, independente do tipo de aplicativo WEB, sendo o local ideal para a abertura das tabelas ou uma configuração de alguma característica que deva ser reinicializada a cada execução.

Tabela de Ocorrências dos Eventos de *TWEBModule*:

Tipo de Aplicativo WEB	OnCreate	BeforeDispatch	AfterDispatch	OnDestroy
CGI	Toda	Toda Requisição	Toda	Toda Requisição

	Requisição		Requisição	
ISAPI / NSAPI	Só na Primeira	Toda Requisição	Toda Requisição	Desliga serviço WEB

A partir deste momento o *TWEBModule* procura na sua lista de *Action Items* por um que atenda às especificações da requisição. Isto é feito através da comparação das propriedades *PathInfo* e *MethodType* de cada *Action Item* com as propriedades de mesmo nome no objeto *TWEBRequest* (representa as informações da requisição).

O *PathInfo* no nosso exemplo de requisição anterior é o "/counter", ou seja, é a parte logo em seguida ao nome do aplicativo. Quanto ao *MethodType*, é a forma como o seu FORM HTML envia os dados, sendo que os mais comuns são *GET* e *POST*.

Obs.: Seria altamente aconselhável um breve conhecimento de HTML, sobretudo a parte de FORMs

Quando é encontrado um *Action Item* que atenda às condições, é disparado um evento deste *Action Item* chamado *OnAction*. Dentro deste evento é que ocorre todo o trabalho de um aplicativo WEB, agora é que vamos entrar no modo de programação mesmo.

Anatomia do Evento *OnAction*:

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender:
TObject;
Request: TWebRequest; Response: TWebResponse; var Handled:
Boolean);
begin

end;
```

Parâmetros:

Sender:	Representa o *TWEBModule.*
Request:	Informações sobre a requisição HTTP cliente.
Response:	Deve ser preenchida com a resposta a ser enviada de volta ao Browser do usuário.
Handled:	Se o evento encerrou a resposta deve ser *True*, se outros Action Items forem completar a resposta deve ser *False*. Se nada for informado age como *True.*

Um *Action Item* pode tratar sozinho de uma requisição ou apenas fazer uma parte do trabalho, deixando que outros a completem. Se o parâmetro *Handled* for setado como *False* o *TWEBModule* irá procurar por um outro *Action Item* que atenda às especificações e, se nenhum for encontrado, será acionado o *Action Item Default*. Se nenhum for executado nada será retornado ao Browser. Um *Action Item* é posto como *Default* colocando *True* em sua propriedade *Default*. Apenas um pode ser o *Action Item Default*, sendo que o valor de seu parâmetro *Handled* deve sempre ser setado como *True* ou então ignorado (trata como *True*), más nunca *False*.Após os *Action Items* terem sido executados e a resposta enviada ao servidor WEB, será executado o *AfterDispatch* do *TWEBModule*.

Uma observação muito importante sobre ISAPI e NSAPI apenas: Como o *TWEBModule* permanece na memória e é único a todas as *threads* geradas (*Action Items*), devemos tomar muito cuidado com variáveis e propriedades criadas no *TWEBModule*, pois as mesmas serão também únicas, ou seja, se tivermos uma propriedade no nosso *TWEBModule* e ela tiver seu valor alterado por uma requisição, as próximas irão exergá-la com este novo valor. Quando precisarmos gravar valores nestas propriedades e/ou variáveis de dentro de um *Action Item*, devemos fazer uso de uma ***Critical Section*** para garantir que uma outra *thread* espere até a atual ter terminado o trabalho. Devemos usar este recurso também quando quisermos acessar arquivos texto (*File-IO*) em nosso

disco, como veremos no exemplo de contador de páginas.

Revisando até aqui:

Um aplicativo WEB é basicamente um *WEBModule* com um *Action Item* para cada tipo de requisição que tratamos, dessa maneira se tivermos 4 requisições diferentes usaremos 4 *Action Items*. Veja a tabela abaixo:

Action do FORM HTML	Method do FORM HTML	Action Item que será disparado
/scripts/myappl.dll	GET ou POST	O *Default*
/scripts/myappl.dll/consulta	GET	Um que possua o *PathInfo* como /consulta e o *MethodType* como *mtGet* ou *mtAny*, ou então o *Default*
/scripts/myappl.dll/incluir	POST	Um que possua o *PathInfo* como /incluir e o *MethodType* como *mtPost* ou *mtAny*, ou então o *Default*
/scripts/myappl.dll/teste	POST	Um que possua o *PathInfo* como /teste e o *MethodType* como *mtPost* ou *mtAny*, ou então o *Default*

Cada Action Item possui um evento *OnAction*, que será executado quando chegar a requisição, recebendo dois objetos representando os dados do pedido e da resposta: *TWEBRequest* e *TWEBResponse*.

TWEBRequest

Quando uma requisição HTTP chega ao *TWEBModule*, os parâmetros referentes a esta requisição são convertidos em propriedades de um descendente de *TWEBRequest* (dependendo do tipo de aplicativo). Aquí estão algumas de suas propriedades mais importantes.

Propriedades	Significado
Authorization	Contém a informação de autenticação do cliente, se ele foi autenticado.
ContentFields	TStrings contendo os nomes e valores dos campos do FORM HTML passados via *POST*.
CookieFields	TStrings contendo os nomes e valores de cookies vindos do cliente.
From	Contém o e-mail do cliente (Comigo não funciona).
MethodType	Contém o *Method* do FORM HTML.
PathInfo	Contém o *PathInfo* da requisição.
QueryFields	TStrings contendo os nomes e valores dos campos do FORM HTML passados via *GET.*
RemoteAddr	Contém o IP do cliente.
UserAgent	Contém informações sobre o Browser cliente.

Se quiser obter mais detalhes ou até mesmo ver as outras propriedades, dê uma olhada no Help do Delphi 3.

TWEBResponse

Da mesma forma que uma requisição gera um objeto *TWEBRequest*, é criado também um objeto *TWEBResponse*, o qual deve ser preenchido e retornado ao Browser cliente. Vamos ver algumas propriedades e métodos mais importantes.

Propriedades	Significado
Content	O próprio conteúdo da resposta, podendo ser comandos HTML, arquivo HTML ou texto.
ContentStream	Quando a resposta precisa ser escrita diretamente de um Stram
ContentType	Usado para indicar o tipo de conteúdo (MIME-Type). O padrão é 'text/html'
WWWAuthenticate	Usado para autenticação de usuários.

Métodos	Significado
SendRedirect	Usado para redirecionar o cliente para uma outra página.
SendStream	Envia conteúdo de um Stream.
SetCookieField	Usado para adicionar Cookies à resposta.

Todas estas propriedades e métodos serão usados nos exemplos que vamos ver a partir de agora.

Exemplo 01 - VI_Request

Um ISAPI simples, com apenas um *Action Item*, o *Default*, retornando as propriedades do *TWEBRequest*.

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
Var
  slValores: TStringList;
begin
  slValores := TStringList.Create;
  Try
    slValores.Add('
');
    slValores.Add('
');
    slValores.Add('
```

Valores da sua Requisição:

```
');
    slValores.Add('Authorization = '+ Request.Authorization +'
' );
    ...
    ...
    slValores.Add('Date = '+ FormatDateTime('dd/mm/yyyy - hh:nn',
                  Request.Date ) +'
' );
    slValores.Add('
');
    //
    Response.Content := slValores.Text;
    // Ok - o WebModule vai enviar a resposta

  Finally
    slValores.Free;
  End;
end;
```

No *OnAction* nós criamos um TStringList e incluímos nele os comandos HTML, depois atribuímos a propriedade *Text* dele à propriedade *Content* do *TWEBResponse.* Pronto, a página foi enviada de volta ao cliente.

Para executá-lo basta digitar seu nome completo no campo de endereço do navegador.Assim:

Download dos fontes de vl_request

Nós vimos neste exemplo o uso da propriedade *Content*, do *TWEBResponse*, para o retorno de uma página HTML, más podemos usar também o *SendRedirect* para redirecionar uma requisição para uma outra página, ou *SendStream* para enviar o conteúdo de um Stream, estes, aliás, serão usados nos próximos 2 exemplos.

Exemplo 02 - WebCounter

Neste exemplo veremos como construir um contador de página HTML, usaremos o SendStream para retornar uma imagem no formato JPEG. Este é bem simples, más pode ser alterado para se obter uma versão mais configuirável. Veja abaixo o *OnAction* de nosso único *Action Item*.

```
procedure TWMCounter.WMCounterAICounterAction(Sender: TObject;
Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
Var
  sCount: String;
  imCount: TJPEGImage;
  stCount: TMemoryStream;
  sFile: String;
begin

  // Request.PathTranslated retorna o path físico do ROOT
  // no meu caso: c:\webshare\wwwroot
  sFile := Request.PathTranslated + '\fbn\scripts\fcount.cnt';
  // Entra na Sessão Crítica - Acesso a recursos por várias threads
  // Isso garante que só uma thread executará esta função por vez
  EnterCriticalSection( csFileIO );
  Try
    // Incrementa contador e retorna novo número
```

```
    sCount  := IncCounter( sFile );
  Finally
    LeaveCriticalSection( csFileIO );
  End;
  // Gera a Imagem do Contador
  imCount := BuildImage( sCount );
  Try
    // Uso de um MemoryStream
    stCount := TMemoryStream.Create;
    Try
      imCount.SavetoStream( stCount );
      stCount.Position := 0;
      Response.ContentType   := 'image/jpeg';
      Response.ContentLength := stCount.Size;
      Response.SendResponse;
      Response.SendStream( stCount );
    Finally
      stCount.Free;
    End;
  Finally
    imCount.Free;
  End;
end;
```

Criamos 2 funções, uma para incrementar o contador, que está em um arquivo texto, e outra para transformar o texto com o contador em uma imagem JPEG. Criamos então um *TMemoryStream* para carregar a imagem nele e o enviamos ao browser do cliente.

Download dos fontes de WebCounter

Este exemplo, funcionando, pode ser visto no final desta página e para executá-lo basta criar um tag de imagem, no HTML, apontando para a DLL.

```
<p>Esta página foi acessada <img
src="/scripts/webcounter.dll">
vezes.</p>
```

Para transformarmos este exemplo em CGI, no caso de termos que usar um outro servidor WEB não compatível com os da Microsoft, devemos iniciar um novo projeto de aplicativo WEB, escolher o tipo CGI, retirar as *Units* criadas, e adicionar a *Unit* deste exemplo. Retirar também todas as linhas de ***CriticalSection***. Esta conversão já foi feita e pode ser obtida aquí.

Download dos fontes de WebCounter (Versão CGI)

Da maneira que o WebCounter se encontra, só poderemos usá-lo em uma única página. Como poderemos então colocar nosso contador em várias páginas de nosso site ? - Simples, a única mudança necessária seria em passarmos o nome do arquivo como parâmetro para o aplicativo. Veja como ficaria nosso HTML.

```
<p>Esta página foi acessada <img
src="/scripts/webcounter.dll?fn=cont1.cnt">
vezes.</p>
```

Note que após o nome do aplicativo nós colocamos o ?fn=cont1.cnt, que seria a passagem de um parâmetro chamado *fn* com o valor de *cont1.cnt* no método *GET.* Este parâmetro estaria disponível ao aplicativo WEB na propriedade *QueryFields* do *TWEBRequest*, Veja o trecho de código modificado:

```
...

begin
  sFile := Request.PathTranslated + '\fbn\scripts\' + ;
                   Request.QueryFields.Values[ 'fn' ];
// Passa Nome do Arquivo
  ...
  imCount := BuildImage( sCount );
  ...
```

Não esquecer de alterar o código em '\fbn\scripts\' para apontar para seu diretório de scripts no servidor WEB.

Exemplo 03 - HitCounter

O *SendRedirect* pode ser usado da seguinte maneira:
Quando possuímos um link em nossa página para uma outra de um dos nossos patrocinadores, e desejamos saber quantas pessoas chegaram até lá via nosso link.. Por exemplo, suponha, que tenhamos um link para a página da Borland e que necessitamos saber quantos acessos foram feitos à Borland através de nossa página.

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  IncCounter( Request.PathTranslated );
  Response.SendRedirect( 'http://www.borland.com' );
end;
```

Isto é mais simples do que se pode imaginar. Só precisamos criar um procedure para incrementar o contador de acessos e, através do *SendRedirect*, enviamos o usuário para o *site* da Borland

Download dos fontes de HitCounter

Para executá-lo basta colocar um Link em sua página dessa maneira:

```
<p align="center"><a href="/scripts/hitcounter.dll">Borland WebSite</a></p>
```

Nexte exemplo não coloquei uma ***Critical Section***, se desejar usar este aplicativo, deve providenciar isto. Veja exemplo anterior

Geradores de Conteúdo

Ficar gerando página HTML através de uma *TStringList* não é nada agradável, más não se preocupe, o Delphi 3 possui alguns objetos que nos ajudam na tarefa de produzir uma resposta em forma de página HTML. Estes objetos derivam de TCustomContentProducer e, como eles, nós podemos escrever nossos próprios geradores de conteúdo derivados.

Os 3 geradores de conteúdo disponíveis são: *TPageProducer*, *TDataSetTableProducer* e *TQueryTableProducer*. O primeiro gera HTML através de alterações em Documentos HTML que serverm como *Templates.* Os outros 2 produzem comandos HTML baseados em informações de um Banco de Dados.

TPageProducer

Este componente é muito utilizado na geração de páginas HTML baseadas em uma outra página HTML padrão, este template pode existir no disco, como um arquivo, ou na memória apenas. Por exemplo, podemos possuir uma página de resposta padrão a uma transação efetuada pelo usuário, más gostaríamos de personalizá-la colocando o nome do mesmo.

Isto pode ser feito graças a *Tags* que são colocadas nesta página HTML padrão, e que serão substituídas por informações reais em nosso aplicativo WEB. Um *Tag* possui o seguinte formato:

<#TagName Param1="Value1" Param2="Value2" ...>

Os sinais de > e Tags transparentes ao navegadores, no caso da *Tag* não ser tratada pelo *TPageProducer*. O sinal # informa ao *TPageProducer* que esta construção é uma *Tag*, e que deve ser substituída por outro valor. Uma *Tag* pode opcionalmente possuir parâmetros,

sendo que eles devem ser no formato Nome=Valor, com nenhum espaço entre o nome, o sinal de = e o valor. Cada parâmetro deve ser separado dos demais por um espaço em branco.

O Delphi 3 nos oferece um número de *Tags* predefinidas, que estão associadas com valores do tipo *TTag*. São elas:

Nome da *Tag*	Valor de *TTag*	No que ela deve ser convertida
Um Nome qualquer	tgCustom	A seu critério
Link	tgLink	Em um Link A../A
Image	tgImage	Em uma Tag HTML de Imagem IMG SRC=...
Table	tgTable	Em uma tabela HTML TABLE.../TABLE"
ImageMap	tgImageMap	Em um mapa de imagens MAP.../MAP"
Object	tgObject	Em um Controle ActiveX OBJECT.../OBJECT
Embed	tgEmbed	Em um NetScape Add-In DLL EMBED.../EMBED

Vale ressaltar que o Delphi não oferece nenhum processamento especial para estas *Tags* predefinidas, elas servem apenas para nos ajudar no processo de conversão. Dessa maneira cabe a nós transformar um Tag do tipo tgLink em uma sequência do tipo 'href="www.borland.com">Borland WebSite</A>'.

O *TPageProducer* possui duas propriedades, chamadas *HTMLFile* e *HTMLDoc*, pelas quais podemos especificar qual será nossa página HTML padrão. *HTMLFile* é usada para armazenar o nome de um arquivo em disco, enquanto que *HTMLDoc* armazena um *TStrings* representando o *Template*, ou seja, se nossa página HTML padrão for uma página que está no disco, devemos colocar seu nome na propriedade *HTMLFile*, se não, podemos criá-la em memória e armazená-la na propriedade *HTMLDoc*, que é um *TStrings*. Nós podemos também armazenar nossas páginas padrões em campos MEMO de um banco de dados e usar o método *ContentFromStream* para obter este HTML diretamente do campo.

Quando especificamos que a propriedade Content do objeto *TWEBResponse* é igual ao *Content*, ou um correspondente, de *TPageProducer*, a página HTML padrão é avaliada e, para cada *Tag* HTML encontrado será disparado o evento *OnHTMLTag* de *TPageProducer*.

Anatomia do Evento *OnHTMLTag*:

```
procedure TWebModule1.PageProducer1HTMLTag(Sender:
TObject; Tag: TTag;
   const TagString: String; TagParams: TStrings;
var ReplaceText: String);
begin

end;
```

Parâmetros:

Sender:	Representa o *TPageProducer*
Tag:	O Tipo da *Tag* (tgCustom, tgLink, ...).
TagString:	O Nome da *Tag*.

TagParams: Descendente de *TStrings* com cada um de seus itens representando um parâmetro da *Tag* HTML.

ReplaceText: Deve ser preenchida com o Valor que substituirá o Tag HTML.

Exemplo 04 - Pg_Mensagem

Neste exemplo nós vamos fazer uso de um FORM HTML, onde iremos receber a entrada de dados do Usuário e submetê-las ao nosso aplicativo WEB para que ele possa gerar uma página de agradecimento personalizada. Vale ressaltar que nossa linha de definição do FORM HTML é a seguinte:

Quando o botão Enviar (que é do tipo Submit) for pressionado, os dados do FORM serão enviados, via método POST para o aplicativo representado pela propriedade Action, no nosso caso para pg_mensagem.dll.

Código do FORM HTML:

```
<body bgcolor="#FFFFFF">
<h1 align="center">FBN - Pg_Mensagem</h1>
<h3 align="center">Formulário de entrada de dados</h3>
<hr>
<form action="/scripts/pg_mensagem.dll" method="POST" name="FrmMensagem">
<pre>Seu Nome: <input type="text" size="29" name="txt_nome"></pre>
<pre>Seu e-mail: <input type="text" size="29" name="txt_email"></pre>
<pre>Sua Mensagem: <input type="text" size="51" name="txt_mensagem"></pre>
<p></p>
<p><input type="submit" name="Enviar" value="Enviar">
      <input type="reset" name="Reset" value="Limpar"></p>
</form>
<hr></body>
```

Podemos perceber que o nome do FORM HTML é *FrmMensagem*, seu método é *POST* e sua ação se dará em nosso aplicativo *pg_mensagem.dll*. Quando pressionarmos o controle do tipo *Submit*, cujo nome é *"Enviar"*, todos os campos do formulário serão enviados ao nosso aplicativo, e como estamos usando *POST*, o acessaremos através da propriedade de *TWEBRequest* chamada *ContentFields*. É bom lembrar que os campos serão identificados através de suas propriedades *name*, que são sensíveis à maiúsculas e minúsculas, portanto se quisermos obter o conteúdo do campo que identifica o nome, temos que codificar o seguinte: *Request.ContentFields.Values['txt_nome']*.

Evento *OnAction*:

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
          Request: TWebRequest; Response: TWebResponse; var Handled:
Boolean);
begin
  // Informa que mod_mensagem.htm será o Template
  PageProducer1.HTMLFile := 'mod_mensagem.htm';
  // Poderíamos gravar estes dados em Tabelas de BD
  Response.Content := PageProducer1.Content;
end;
```

Este evento nada mais faz do que associar o *Template* (mod_mensagem.htm) à propriedade *HTMLFile* de *TPageProducer*, e informar que o *Content* de *TWEBResponse* deve ser preenchido com o resultado do método *Content* de *TPageProducer*. Quando fazemos isso, o *Template* será varrido atrás de *Tags* HTML, e para cada uma delas será executado o evento *OnHTMLTag* de *TPageProducer*. Nós colocamos, em mod_mensagem.htm 3 *Tags*: nome, email e mensagem, sendo que esta última possui parâmetros.

Código de mod_mensagem.htm:

```
<body>
<h1 align="center">FBN - Exemplo 05</h1>
<hr>
```

```
<!-- Tag #nome  lembre-se de que é sensível à Maiúsculas e Minúsculas -->
<p>Obrigado <#nome>, pela sua participação</p>
<!-- Tag #email -->
<p>Seu e-mail: <#email></p>
<!-- Vamos usar um parâmetro para este Tag: mensagem -->
<p>Sua Mensagem ( tipo 1 ): <#mensagem cor="1"></p>
<p>Sua Mensagem ( tipo 2 ): <#mensagem cor="2"></p>
<hr>
```

Evento *OnHTMLTag*:

```
// Este evento será executado para cada Tag encontrada
procedure TWebModule1.PageProducer1HTMLTag(Sender: TObject;
Tag: TTag;
          const TagString: String; TagParams: TStrings; var
ReplaceText: String);
Var
  rParam: TWEBRequest;
begin
  rParam := PageProducer1.Dispatcher.Request;
  If CompareStr( TagString, 'nome' ) = 0 Then
    ReplaceText := rParam.ContentFields.Values[ 'txt_nome' ];
  If CompareStr( TagString, 'email' ) = 0 Then
    ReplaceText := rParam.ContentFields.Values[ 'txt_email' ];
  If CompareStr( TagString, 'mensagem' ) = 0 Then
    If TagParams.Values[ 'cor' ] = '1' Then
      ReplaceText := '' +

                     rParam.ContentFields.Values[ 'txt_mensagem' ] +
                     ''
    Else
      ReplaceText := '' +
                     rParam.ContentFields.Values[ 'txt_mensagem' ] +
                     ''
end;
```

Podemos observar que quando este evento for excecutado para a *Tag* nome, por exemplo, o valor de *ReplaceText* será obtido do conteúdo do campo *txt_nome* do Form HTML que chamou este aplicativo WEB. Como o Método foi *POST*, nós podemos usar o *ContentFields* de *TWEBRequest* para obter este valor.

Download dos fontes de Pg_Mensagem

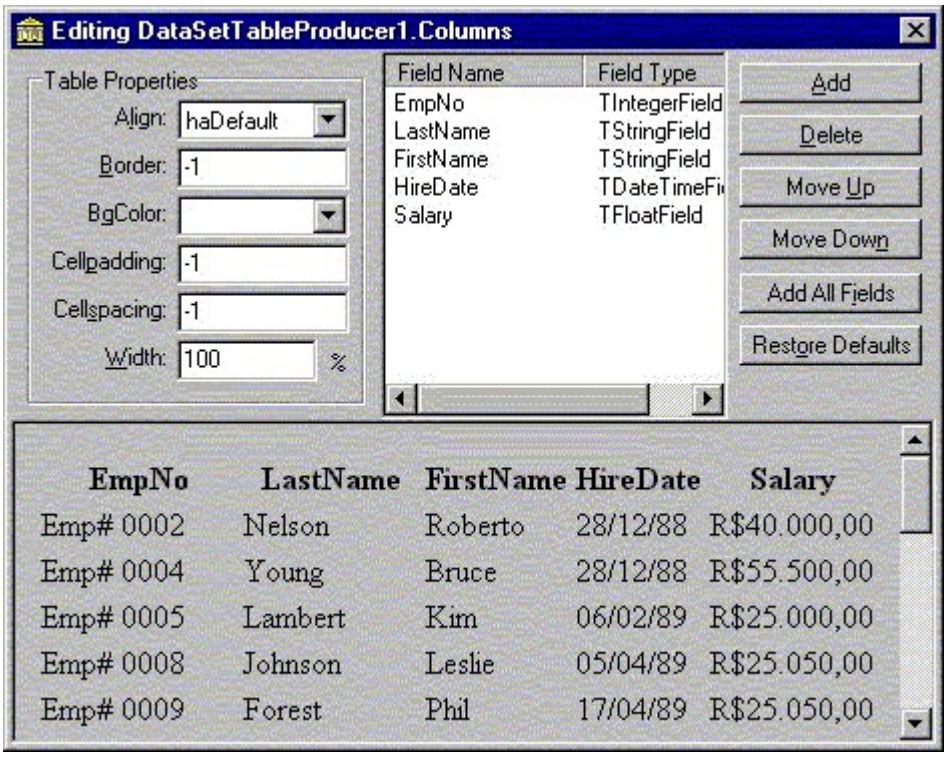
Este exemplo requer um pouco mais de trabalho para ser executado, já que ele é composto de 2 páginas HTML e um aplicativo ISAPI. Devemos copiar os HTM's para o diretório raiz do seu WEBServer, e a DLL para a pasta de scripts do mesmo. Se estiver usando o Microsoft Personal WebServer: WebShare\wwwroot e WebShare\scripts. Se for o Microsoft IIS: InetPub\wwwroot e InetPub\scripts. Então, pelo Browser, chame o arquivo pg_mensagem.htm e seguir adiante.

Antes de entrarmos no assunto de TDataSetTableProducer e TQueryTableProducer, faz-se necessário algumas colocações sobre o uso de componentes de banco de dados em aplicativos ISAPI e NSAPI, devido à sua natureza Multi-thread. Devemos usar sempre um objeto *TSession* configurado com sua propriedade *AutoSessionName* setada para *True*, isso para que cada thread tenha sua própria sessão com o BD, sem interferir com uma outra thread. Qualquer *TTable* ou *TQuery* deve estar com *Active* setada para *False*, e devem ser abertas quando precisar ou no evento *BeforeDispatch* do *TWEBModule*.

TDataSetTableProducer / TQueryTableProducer

Estes dois componentes são usados para gerar Tabelas HTML a partir de uma *TTable* e *TQuery* respectivamente. Eles são idênticos, diferindo apenas que o *TQueryTableProducer* usará o conteúdo de *QueryFields* ou *ContentFields*, de acordo com o *Methodtype* (*GET* ou *POST*), de *TWEBRequest* para preencher os parâmetros de uma *TQuery*. As propriedades *Header* e *Footer* permitem adicionar outros comandos HTML à página gerada. Clicando-se na

propriedade *Columns* nós poderemos ver o Editor da Tabela HTML.



Através deste editor poderemos personalizar nossa tabela. Existem várias propriedades que nos permitem aprimorar bastante a aparência, trocando cores do cabeçalho, de uma coluna ou mesmo da tabela inteira.

A manipulação destes componentes é bastante intuitiva, qualquer pessoa que já tenha configurado um TDBGrid vai se sentir à vontade, lembrando-se sempre que o HTML é bastante limitado.

Exemplo 05 - Vendas_Empregado

Este exemplo é um pouco mais elaborado que os anteriores, visto que usaremos dois *Action Items*. O *Default* vai nos mostrar uma tabela, gerada via um *TDataSetTableProducer*, dos empregados (tabela EMPLOYEE.DB do DBDEMOS). Note o uso de uma função chamada *MakeRef*, ela será chamada de dentro do evento *OnGetText*, do campo *EmpNO*, para colocar o mesmo como um *Link* para nosso próprio aplicativo, só que desta vez estaremos definindo como *PathInfo* o valor */vendas* e passando, via *GET*, o valor do próprio campo *EmpNO*. Então quando clicarmos em um destes *links*, nosso segundo *Action Item* entrará em ação para nos mostrar, via *TQueryTableProducer*, todas as vendas efetuadas por aquele determinado vendedor.

Veja nosso Action Item Default:

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender:
TObject;
          Request: TWebRequest; Response: TWebResponse;
var Handled: Boolean);
begin
  tbEmpregados.Open;
  Response.Content := DataSetTableProducer1.Content;
  tbEmpregados.Close;
end;
```

A primeira coisa a fazer é abrir a tabela de empregados, depois associamos o retorno do método *Content* de *TDataSetTableProducer* à propriedade *Content* de nossa Resposta e, após ela ser gerada, fechamos a tabela. Veja o evento OnGetText do campo EmpNO da

tabela tbEmpregados, a função MakeRef está no exemplo.

```
procedure TWebModule1.tbEmpregadosEmpNoGetText(Sender: TField;
          var Text: String; DisplayText: Boolean);
begin
  Text := MakeRef( Trim( Sender.AsString ) );
end;
```

O que esta função faz é uma coisa bem simples. Um *link* é composto do seguinte: texto. Como exemplo, quando o valor de EmpNO fosse 114 queríamos que o *link* apontasse para o seguinte:
< a href="scripts/vendas_empregado.dll/vendas?empno=114">114 . Note que após o nome da DLL está a informação de *PathInfo* /vendas, e após ela o nosso parâmetro *empno* com valor 114, sendo que o texto que aparece vem após o ">, ou seja, 114.

No nosso aplicativo, o segundo Action Item possui em sua propriedade PathInfo o valor /vendas, dessa forma quando clicarmos no link da tabela anterior, será este o evento a ser executado:

```
procedure TWebModule1.WebModule1WebActionItem2Action(Sender: TObject;
          Request: TWebRequest; Response: TWebResponse; var Handled:
Boolean);
begin
  qrVendas.Open;
  Response.Content := QueryTableProducer1.Content;
  // Não deixa passar pelo ActiomItem Default
  Handled := True;
  qrVendas.Close;
end;
```

Ele faz basicamente a mesma coisa que o *Action Item* anterior, só que usamos Query. É bom observar que, em momento algum, atribuímos valor ao *qrVendas.ParamByName('empno')*. Como já foi dito, isto será feito automaticamente pelo conteúdo do campo *QueryFields* ou *ContentFields*, cujos nomes correspodam ao nomes dos parâmetros da Query, e nós vimos no exemplo acima que existe o parâmetro **empno=114**, passado via *GET* (estará então em *QueryFields*). Más se quiser, isto pode ser feito na mão mesmo, como estamos acostumados.

Download dos fontes de Vendas_Empregado

Ele pode ser executado da mesma forma que o exemplo vl_request, ou seja, diretamente da linha de endereço do seu browser: /scripts/vendas_empregado.dll

Cookies

Cookies são pequenos arquivos texto que são enviados por um servidor WEB para nosso Navegador, e salvos em nossos discos. Eles não podem ser executados e nem conter vírus, podem apenas ser lidos pelo mesmo servidor, somente, que os colocou lá. São usados, geralmente, para guardar informações a respeito do usuário, de suas preferências ou mesmo informações pessoais que nos identificam quando voltamos naquele *site*. Várias empresas se utilizam deste recurso. Um bom exemplo pode ser visto no *site* da **Amazon** (livraria virtual em www.amazon.com). Quando compramos algum livro lá, damos informações pessoais como nome, endereço, telefone, etc. Estas informações são armazenadas em nosso sistema, e quando voltamos, eles sabem quem somos, ou seja, não teremos mais que informar os dados novamente. Além disso, nossa cesta de compras é armazenada via *Cookies*, dessa forma, não importa em que página daquele *site* estejamos, poderemos navegar direto até a página da compra e efetuá-la.

Tudo isto porquê quando uma requisição é atendida, pelo servidor WEB, e uma resposta nos é enviada, nem o servidor nem o Browser cliente se lembram do que ocorreu, e este é o maior desafio na programação de aplicativos WEB, ou seja, a separação das sessões. Cada requisição é tratada como se fosse um novo usuário, e não adianta identificar o usuário pelo seu número IP, pois algumas empresas usam **Proxy**, e nesse caso dezenas

de usuários irão possuir o mesmo IP.

Um *Cookie* nada mais é do que valor texto no formato CookieName=CookieValue, e não deve incluir aspas, vírgulas ou espaçoes em branco.

Para aprender mais sobre *Cookies* veja os *Links* em Leituras Complementares

Existe um método, do *TWEBResponse*, chamado *SetCookieField* que serve para enviar um *Cookie* junto com nosa resposta.

Anatomia do Evento *SetCookieField*:

```
procedure SetCookieField(Values: TStrings;
const ADomain, APath: string;
          AExpires: TDateTime; ASecure: Boolean);
begin

end;
```

Parâmetros:

Values:	Um *TStrings* (normalmente usaremos *TStringList*) onde cada item representa um *Cookie*, no formato Nome=Valor.
ADomain:	Domínio ao qual o *Cookie* pertence.
APath:	*PATH* deste domínio ao qual o *Cookie* pertence.
AExpires:	Determina por quanto tempo este *Cookie* deve permanecer válido.
ASecure:	Se um *Cookie* deve ou não ser passado apenas por uma conexão segura (HTPPS)

ADomain e *APath* são usados em conjunto para identificar o servidor e o Path à qual o *Cookie* pertence, ou seja, somente quando o usuário visitar este domínio e este path, é que o *cookie* poderá ser lido pelo servidor WEB.

Se, por exemplo, enviarmos um *Cookie* com 'grupoalianca.com.br' em *ADomain* e '/' em *APath*, então, o Navegador do cliente somente permitirá a leitura do *Cookie* por uma página ou aplicativo pertencente à este domínio. Já se passarmos o valor '/fbn' em *APath*, apenas as páginas ou aplicativos dentro deste Path poderão ler o *Cookie*.

O *AExpires* é usado para podermos colocar uma data na qual o *Cookie* deixará de existir. Se passarmo *Now+2* em *AExpires* nosso *Cookie* só poderá ser lido nos próximos 2 dias. Se enviarmos uma data passada, o nosso *Cookie* será apagado.

Para fazer um *Cookie* durar apenas enquanto o navegador do usuário estiver aberto, basta passar em *AExpires* o valor **0** (Zero) ou, no caso do Microsoft IE, **-1**. Use o *TWEBRequest.UserAgent* para saber qual o Navegador do cliente.

Os Cookies são lidos, por nossos aplicativos, através da propriedade *CookieFields* de *TWEBRequest*. Dessa maneira, para lermos o conteúdo de um *Cookie* chamado *usuario*, usaremos a seguinte sintaxe:
Request.CookieFields.Values['usuario'].

Devido a problemas com *Cookies* no Microsoft Internet Explorer, nós devemos colocar no evento *OnCreate* do nosso *TWEBModule* a seguinte codificação, para garantir que nosso *Cookie* funcione independente do Navegador

```
procedure TWebModule1.WebModule1Create(Sender: TObject);
begin
  //
```

```
      ShortDayNames[01] := 'Sun';
      ShortDayNames[02] := 'Mon';
      ShortDayNames[03] := 'Tue';
      ShortDayNames[04] := 'Wed';
      ShortDayNames[05] := 'Thu';
      ShortDayNames[06] := 'Fri';
      ShortDayNames[07] := 'Sat';
      //
      ShortMonthNames[01] := 'Jan';
      ShortMonthNames[02] := 'Feb';
      ShortMonthNames[03] := 'Mar';
      ShortMonthNames[04] := 'Apr';
      ShortMonthNames[05] := 'May';
      ShortMonthNames[06] := 'Jun';
      ShortMonthNames[07] := 'Jul';
      ShortMonthNames[08] := 'Aug';
      ShortMonthNames[09] := 'Sep';
      ShortMonthNames[10] := 'Oct';
      ShortMonthNames[11] := 'Nov';
      ShortMonthNames[12] := 'Dec';
    //
    end;
```

Devemos alterar os nomes dos dias e dos meses para qualquer aplicativo WEB que formos desenvolver, e no qual seja necessário a manipulação de *Cookies*. Isto é necessário devido a problemas do Delphi.

Exemplo 06 - Gera_Cookie:

Este é um exemplo bastante simples, temos uma página HTML chamada *Gera_Cookie.htm* que possui um *Form* com um campo chamado *txt_nome* e um botão do tipo *Submit*, que quando pressionado chamará nosso aplicativo WEB. Este aplicativo possui um só *Action Item*, cujo evento *OnAction* é:

```
procedure TWebModule1.WebModule1WebActionItem1Action
(Sender: TObject;
          Request: TWebRequest; Response: TWebResponse;
var Handled: Boolean);
Var
  slCookies: TStringList;
  sNome: String;
begin
  sNome := Request.ContentFields.Values[ 'txt_nome' ];
  slCookies := TStringList.Create;
  Try
    slCookies.Add( 'c_nome=' + sNome );
    slCookies.Add( 'c_data=' + DateTimeToStr( Now ) );
    Response.SetCookieField( slCookies,'','/',NOW+1,False );
    Response.Content := '
'+ sNome + ', O Cookie foi salvo no seu HD'+ '
';
  Finally
    slCookies.Free;
  End;
end;
```

A primeira coisa que fizemos foi criar uma variável (*sNome*) para armazenar o conteúdo do campo *txt_nome* do nosso *Form* HTML, passado via *POST* ele se encontra em *ContentFields*. Depois, criamos uma *TStringList* para armazenar nossos *Cookies*, e adicionamos a ela duas *strings* no formato nome=valor. Observe que não deve haver espaços em branco entre o nome e o valor.O que fizemos, na verdade, foi criar dois Cookies, um chamado *c_nome* e outro *c_data*. O *Cookie c_nome* irá armazenar o nome do usuário, informado no campo *txt_nome* do *Form* HTML, e o *Cookie c_data* terá o valor da data de hoje.

Usando *SetCookieField* nós enviamos os *Cookies* para o micro do usuário, e encerramos com uma resposta padrão. Note que os parâmetros *ADomain* e *APath* foram passados em branco, e cabe a você, se desejar, passar com suas informações de domínio e *path*.Quanto ao parâmetro *AExpires*, ele foi passado como *NOW+1*, o que significa que só vale por um dia.

Download dos fontes de gera_cookies

Exemplo 07 - Ler_Cookie:

Nós poderíamos ter criado, no exemplo anterior, um outro *Action Item* para ler os *Cookies*, más resolvemos criar um novo aplicativo só para isso, pois a intenção é mostrar que qualquer aplicativo WEB, dentro do nosso domínio, poderá ler estes Cookies. Este exemplo é bem mais simples. veja o evento *OnAction* do *Action Item Default*:

```
procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
Var
  sNome: String;
  sData: String;
begin
  sNome := Request.CookieFields.Values[ 'c_nome' ];
  sData := Request.CookieFields.Values[ 'c_data' ];
  //
  Response.Content := '
Olá ' + sNome + ', este '+
                      'Cookie foi gerado em: '+ sData +'
';
end;
```

A única coisa que fazemos é obter os valores dos dois *Cookies*, enviados pelo exemplo anterior, e mostrá-los ao usuário. Se tentarmos executar este exemplo após a validade do *Cookie* ter expirado, não vai haver erro, somente os valores estarão em branco.

Download dos fontes de ler_cookie

Autenticação de Usuários:

Se nosso servidor WEB possuir a opção de ***Basic Authentication*** habilitada, poderemos tornar nosso aplicativo acessível somente por pessoas cadastradas em nosso sistema.

O Microsoft Personal Web Server versão 4, quando rodando em Windows 95, não permite esta configuração de segurança.

A maneira correta de fazermos a verificação se uma requisição possui, ou não, os direitos de execução do nosso aplicativo seria através do uso da propriedade *Authorization* de *TWEBRequest*. Se estiver em branco, o usuário não foi autenticado pelo servidor, seja Windows 95 ou Windows NT. Neste momento poderemos retornar um *StatusCode* com valor de 401 de volta ao Navegador do usuário, o que fará aparecer uma janela *Pop-Up* pedindo o nome e a senha ao usuário. Se os dados baterem com os de algum usuário cadastrado, *Authorization* estará preenchida. Para resumir, só executaremos nosso aplicativo se *Request.Authorization* for diferente de uma *string* vazia.

Exemplo 08 - Autenticado:

Neste exemplo possuímos um *Action Item Default*, com o seguinte *OnAction*:

```
procedure TWebModule1.WebModule1WebActionItem1Action
```

```
(Sender: TObject;
          Request: TWebRequest; Response: TWebResponse;
var Handled: Boolean);
begin
  If Request.Authorization = '' Then
    Begin
      Response.StatusCode := 401;
      Response.WWWAuthenticate := 'basic';
    End
  Else
    Response.Content := '
Usuário Autenticado
';
end;
```

Nós começamos verificando se a propriedade *Authorization* de *TWEBRequest* está vazia, em caso positivo, preenchemos 2 propriedades de *TWEBResponse*, chamadas *StatusCode* e *WWWAuthenticate*, com os valores 401 e 'basic' respectivamente. Isto informa ao servidor WEB que apenas usuários cadastrados poderão continuar, e como a autenticação é no formato 'basic', o navegador do usuário pedirá um nome e uma senha através de uma janela *Pop-Up*, e após preenchimento nosso aplicativo torna a ser executado. Se *Authorization* estiver preenchida, o usuário foi autorizado e poderá prosseguir.

Download dos fontes de Autenticado

Debug

Aplicativos ISAPI: Antes de mais nada deveremos parar (ShutDown) o nosso servidor WEB. Se estiver usando o Windows NT, tenha certeza que possui o seguinte direito: *Act As Part Of Operating System*.

No menu (IDE do Delphi) ***Project -> Options ->Directories/Conditionals*** na caixa de texto ***Output Directory*** escreva o *Path* para seu diretório de scripts do servidor WEB.

```
No Microsoft IIS:
   OutPut Directory = C:\Inetpub\Scripts
No Microsoft Personal Web Server ( qualquer versão ):
   OutPut Directory = C:\WebShare\Scripts.
```

No menu ***Run -> Parameters***, em Host Application escreva o nome do executável, com *Path*, do seu servidor WEB, e em ***Run Parameters*** digite os parâmetros necessários para rodá-lo.

```
No Microsoft IIS:
   Host Application = C:\Winnt\System32\InetSrv\InetInfo.exe
   Run Parameters   = -e w3svc
No Microsoft Personal Web Server 1.0a ( Inglês ):
   Host Application = C:\Program Files\WebSvr\System\Inetsw95.exe
   Run Parameters   = -w3svc
No Microsoft Personal Web Server 1.0a ( Português ):
   Host Application = C:\Arquivos de Programas\Servidor
Web Particular\Servidor Web\Inetsw95.exe
   Run Parameters   = -w3svc No Microsoft Personal Web Server 4
   Host Application = C:\Windows\System\inetsrv\inetinfo.exe
   Run Parameters   = -e w3svc
```

Desta maneira poderemos depurar nossos aplicativos WEB (ISAPI) como se fossem um outro aplicativo qualquer.

Leituras Complementares

HTML

Como criar sua HomePage (http://www.dsc.ufpb.br/~helder/html-ref/tutorial/)
Tutorial HTML em Português (http://www.dcc.ufal.br/~arfm/HtmlGuide/GuiaHTML.htm)
Guia de Referência HTML (http://www.dsc.ufpb.br/~helder/html-ref/)
Form HTML especificação (http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-forms/overview.html)

ISAPI e CGI

Developing ISAPI Extensions (http://www.iftech.com/oltc/isapi/isapi0.stm)
CGI, ISAPI and Server-side Programming Links (http://www.commpass.co.uk/~commpass/resource/cgi-isapi.html)
The Common Gateway Interface (http://hoohoo.ncsa.uiuc.edu/cgi/)
Microsoft Internet Server API Reference (http://www.microsoft.com/win32dev/apiext/isapiref.htm)
The Netscape Server API (http://www.netscape.com/newsref/std/server_api.html) - NSAPI

Cookies

Netscape - HTTP Cookies (http://home.netscape.com/newsref/std/cookie_spec.html)
CookieCentral (http://www.cookiecentral.com/)

Delphi

Nick Hodges (http://www.icss.net/~nhodges/DL180.htm) - Ótimo site sobre ISAPI com Delphi 3
Delphi and Internet by Charlie Calvert (http://members.aol.com/charliecal/internet.htm) - Delphi 2
Creating CGI scripts with Delphi (http://statline.vb.cbs.nl/erwin/ekcgi/ekcgi.html)
SnapJax Delphi ISAPI Developer's site (http://delphi.snapjax.com/isapi/)
The ISAPI Developer's site (http://www.genusa.com/isapi/)

Fábio Bomfim Nunes - fbn@aliancaseguros.com.br
Salvador - Bahia - Brasil