

1. Desenvolvendo aplicativos Internet

A **Tecnocoop Sistemas** incorporou às linguagens OPUS e OPUSWin algumas ferramentas de programação Internet, Intranet e Extranet, que facilitam o desenvolvimento de aplicativos em servidores WEB. Todos esses novos comandos e funções constituem uma extensão das linguagens OPUS e OPUSWin e recebem o nome de OPUSWEB.

As plataformas e ambientes operacionais contemplados pela OPUSWEB são os seguintes:

- ◆ **UNIX** (interface CGI).
- ◆ **Windows 9X/2000** (interface ISAPI).

As linguagens **OPUS e OPUSWin** oferecem recursos para a **construção dinâmica** de aplicativos Cliente/Servidor em ambientes Internet e Intranet, utilizando as interfaces **CGI** (Common Gateway Interface) e **ISAPI** (Internet Server Application Program Interface), padrões amplamente utilizados nos servidores WEB.

Mais adiante, estabeleceremos um paralelo entre as interfaces **ISAPI e CGI**, relacionando suas vantagens e desvantagens.

1.1 Conceitos básicos

Ao desenvolver aplicativos **ISAPI** ou **CGI**, é necessário compreender claramente os mecanismos de funcionamento dessas interfaces de programação. Todos os problemas encontrados na utilização da OPUSWEB se relacionam, direta ou indiretamente, com o desconhecimento dos padrões e mecanismos CGI e/ou ISAPI.

Por isso, a compreensão destes assuntos constitui requisito básico para desenvolver **scripts, programas, extensões** ou **filtros** para servidores Web.

Abordamos, inicialmente, algumas idéias básicas a respeito dos seguintes assuntos:

- a) O **modelo Internet** (cliente/servidor).
- a) O **cliente** (“Browser”, “User Agent”)
- b) O **servidor** (Web Server / HTTP Server)
- c) O **protocolo** (HTTP - Hypertext transfer protocol)

1.1.1 O modelo Web cliente/servidor

Os serviços da plataforma Web estão baseados no modelo cliente/servidor, que permite distribuir e compartilhar os seus componentes básicos, ou seja, a **interface** com o usuário, a **lógica** dos programas e os **dados**.

No modelo cliente/servidor, o **cliente**:

- ◆ Pode requerer dados do servidor
- ◆ Pode enviar dados para o servidor
- ◆ Pode solicitar do servidor a execução de processos
- ◆ Pode executar processos

No modelo cliente/servidor, o **servidor**:

- ◆ Pode enviar dados ao cliente
- ◆ Providencia o acesso a bases de dados
- ◆ Executa processos

Nos modelos tradicionais, os clientes e os servidores são classificados de “magros” ou “gordos”. Estes termos indicam mais uma relação funcional do que características físicas. Trata-se de uma divisão do trabalho ditada pelo perfil do próprio aplicativo. Por exemplo, é bastante freqüente que o servidor seja otimizado para fornecer dados a múltiplos clientes e, usualmente, a aplicação cliente é otimizada para, apenas, interagir com o usuário final.

O servidor é “gordo” quando toda a lógica funcional reside nele. Este é, ainda, o modelo mais comum na Web. O cliente “magro” é, usualmente, um Browser, que fornece apenas a interface com o usuário. Ou seja, as aplicações **CGI** ou **ISAPI** fornecem a lógica funcional dentro do servidor HTTP e o cliente **apenas exhibe** os dados. Existem outros modelos **cliente / servidor**, com serviços distribuídos. Neste caso, as atividades são **compartilhadas** entre cliente e servidor.

1.1.2 O cliente (“User Agent”)

Os clientes Web são também chamados “User agents” ou, simplesmente, Browsers. No passado, por serem os Browsers meras interfaces com o usuário, o modelo era “servidor gordo, cliente magro”. Contudo, atualmente, existem tecnologias (por exemplo, Applets, Client-side Scripts, Style Sheets, plug-ins ...) que permitem maior grau de programação e processamento do lado do cliente.

A expressão cliente, ou “User Agent” se refere, usualmente, ao parceiro de uma sessão HTTP que inicia o pedido (“request”) a ser atendido (“response”) pelo servidor Web.

Na medida que a grande rede mundial (World Wide Web) cresce, em recursos e complexidade, novos tipos de “User Agents” são inventados para prover novas funcionalidades junto aos usuários.

1.1.3 O servidor (“Web Server”)

Servidores Web são processos que aceitam conexões (ou seja, sessões HTTP) solicitadas por clientes Web (Browsers) e, em resposta, fornecem informações na forma de mensagens e documentos de variados tipos, por exemplo, textos, imagens, som, vídeo ... etc ...

O desenvolvimento de servidores Web começou em 1990 e, atualmente, existem centenas de milhares de servidores Web na Internet, além de um número desconhecido de servidores utilizados nas Intranets corporativas.

Inicialmente, e durante algum tempo, apenas havia opções de servidores Web para plataformas **UNIX**. Atualmente, existem bons Servidores Web para plataformas Win32 e Mac.

1.1.4 O protocolo HTTP

Os servidores Web utilizam o protocolo HTTP (Hypertext Transfer Protocol) que foi implementado para permitir uma transferência rápida e eficiente de documentos na Internet”.

1.1.4.1 Estrutura das transações HTTP

HTTP é um protocolo fácil de entender, pois é baseado no paradigma **pedido e resposta** (*request e response*). Uma transação HTTP, independente da sua complexidade, possui a seguinte estrutura elementar:

- O cliente abre uma conexão com o servidor, via TCP/IP, na porta 80.

- Estabelecida a conexão, o cliente envia um pedido ao servidor, na forma de mensagem.
- O servidor processa o pedido do cliente e responde, enviando de volta ao cliente a informação solicitada, em forma de documento padrão ou código de erro.
- O servidor fecha a conexão, terminando o processo ou “thread”.

O protocolo HTTP é “**stateless**” por natureza. Isto quer dizer que cada transação (pedido e resposta) é completada de forma independente, não sendo preservadas as informações de status referentes às transações anteriormente completadas. Isto exige um nível maior de controle e complexidade na programação de aplicativos Web.

Existem vários métodos e recursos que permitem administrar essa complexidade das transações HTTP.

1.1.4.2 O pedido do cliente (“request”)

Como já dissemos, o protocolo HTTP é constituído de pedidos (“requests”) e respostas (“responses”). Os “requests” são, usualmente, gerados, num formato padrão, pelo cliente Browser, atendendo a eventos como, por exemplo, o “click” do usuário sobre um hyperlink.

Existem vários métodos para o cliente encaminhar um pedido ao servidor Web. O método GET é adequado para obter informações do servidor. Porém, quando for necessário enviar dados para o servidor, geralmente é utilizado o método POST, que permite utilizar FORMs como entrada de dados.

O tipo de ação a ser executada pelo servidor Web é determinado pelo **método** adotado em relação ao pedido do cliente. A codificação completa dos comandos e a sintaxe correta do pedido, em conformidade com o método adotado e as normas do protocolo HTTP, são fornecidas pelo Browser, de maneira automática. Não é necessário que o programa CGI ou ISAPI especifiquem os cabeçalhos do pedido HTTP, ou seja, os comandos que precedem o corpo do documento transferido.

Os pedidos do cliente são formulados em conformidade com as normas do protocolo HTTP. Cada pedido é acompanhado de um **cabeçalho (“request header”)**, que proporciona informações adicionais a respeito da transação HTTP. Esses cabeçalhos constituem, de fato, um *protocolo de entendimento entre o cliente Browser e o servidor Web*. Veja, a seguir, um exemplo de cabeçalho de pedido tal como seria enviado pelo Browser Internet Explorer 3.0:

```
GET /default.htm HTTP/1.0
Accept: */*
Accept-Language: en
UA-pixels: 1024x768
UA-color: color16
UA-OS: Windows NT
UA-CPU: x86
User-Agent: Mozilla/2.0 (compatible; MSIE 3.0B; Windows NT)
Host: 199.1.154.46
```

Analisando o cabeçalho acima, vemos que:

- ◆ O pedido adota o método **GET**, solicitando o documento “default.htm”, usando a versão 1.0 do protocolo HTTP.
- ◆ O cabeçalho **Accept** comunica ao servidor Web que o cliente Browser está preparado para aceitar todos os tipos e subtipos de documento no formato padrão MIME (Mutipurpose Internet Mail Extensions). É isso que significa o símbolo “*/*”.

- ◆ O cabeçalho **Accept-Language** mostra que a linguagem preferida do cliente é o Inglês (“en”). O servidor deverá tomar isso em consideração na hora de construir a resposta.
- ◆ Os cabeçalhos **UA-pixels** e **UA-color** fornecem informações sobre as características do vídeo do cliente. **UA-OS** e **UA-CPU** mostram qual é o sistema operacional e o processador do cliente.
- ◆ O cabeçalho **User-Agent** diz que o Browser é o Internet Explorer 3.0B

Os aplicativos **CGI** ou **ISAPI** utilizam as informações dos cabeçalhos para montar o corpo da resposta. Desta forma, o cliente (Browser) pode receber o documento solicitado, sendo este apresentado da maneira mais correta e adequada possível.

1.1.4.3 Resposta do servidor (“response”)

A resposta do servidor a um pedido do cliente (caso a transação seja completada com sucesso), é precedida de códigos de status e cabeçalhos que definem as características das informações contidas nessa resposta.

Veja, a seguir, um exemplo de uma resposta do servidor Web:

```

HTTP/1.0 200 OK
Server: Microsoft-IIS/2.0
Date: Fri, 31 Dez 1996 16:53:58 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Fri, 12 Jul 1996 01:30:00 GMT
Content-Length: 4051
[linha em branco terminada com CRLF]
<!doctype html public "-//IETF//DTD HTML//EN">
<html>
<head>
</head>
<body background="/images/backgrnd.gif">
[ corpo da entidade documento ]

```

1.1.4.4 “Request / Response Headers”

Os principais cabeçalhos de pedido e resposta, especificados pelo protocolo HTTP, reconhecidos e utilizados pela maioria dos servidores Web e clientes Browser, são os seguintes:

Request Header	Descrição
Accept	Especifica os tipos de mídia que serão aceitáveis na resposta do servidor. Pode ser utilizado para mostrar que o pedido é limitado a um conjunto de tipos de dado, por exemplo, um pedido solicitando uma imagem.
Accept-Language	Similar ao Accept. Indica um conjunto dos idiomas preferidos na resposta ao pedido.
Authorization	Serve para que o cliente se autentique a si mesmo
Cache-Control	Estabelece diretivas a serem observadas pelos mecanismos de caching, ao longo da cadeia de pedido/resposta. Estas diretivas se sobrepõem ao algoritmo padrão de caching.

Request Header	Descrição
Content-Base	Especifica a URI básica, a ser utilizada para resolver as referências relativas a URLs.
Content-Length	Informa o tamanho do corpo do documento sendo transmitido.
Content-Type	Especifica o tipo de mídia do documento enviado.
Date	Representa a data e hora em que a mensagem foi originada.
Expires	Representa a data/hora após a qual a entidade (documento) perde a validade. Esta informação é importante em relação ao controle de “caching” efetuado pelo Browser e ao “refresh” obrigatório quando as informações não são mais válidas.
Host	Especifica o Host Internet e o número da porta HTTP.
Server	Informa sobre o software utilizado pelo servidor que originou a resposta.
User-Agent	Guarda informações sobre o cliente (user agent) que originou o pedido. Esta informação pode ser utilizada com objetivos estatísticos, trace, log... etc ...
Location	Especifica a exata localização de um recurso, para ser usada, por exemplo em caso de redirecionamento de URLs

1.1.4.5 Status codes do HTTP

A primeira linha, no cabeçalho de resposta, é a “linha de status”, que informa a versão HTTP em uso e o código de status da transação, conforme a seguinte tabela:

Código	Texto da resposta	Descrição
200	OK	O pedido foi processado com sucesso
201	Created	O pedido foi processado com sucesso e um novo recurso foi criado. Este novo recurso pode ser referenciado pela URI(s) retornadas na resposta.
202	Accepted	O pedido foi aceito mas o processamento do mesmo ainda não foi completado.
204	No Content	O servidor processou o pedido mas não foram geradas informações para enviar de volta.
300	Multiple Choices	Este código de resposta não é diretamente usado pelo protocolo HTTP, mas serve para interpretar a classe 3XX de respostas.
301	Moved Permanently	O recurso requisitado foi associado de forma definitiva com uma outra nova URL. Quaisquer referências no futuro usarão esta URL.
302	Moved Temporarily	O recurso solicitado reside temporariamente numa outra URL

Código	Texto da resposta	Descrição
304	Not Modified	O cliente fez um GET condicional e o acesso foi permitido, mas o documento não foi modificado após a data especificada no campo “If-Modified-Since”. O servidor responde com um status code sem enviar ao cliente o corpo da entidade documento.
400	Bad Request	O pedido do cliente não foi reconhecido pelo servidor, devido a uma sintaxe incorreta.
401	Unauthorized	O pedido precisa de autenticação do usuário.
403	Forbidden	O servidor entendeu o pedido mas não vai atendê-lo porque o usuário está proibido de executar o serviço solicitado.
404	Not Found	O servidor não achou a URI solicitada.
500	Internal Server Error	O servidor encontrou uma condição inesperada que impede atender o pedido do cliente.
501	Not Implemented	O servidor não consegue executar o pedido, por exemplo quando ele não reconhece o “request method” e, por tanto, não pode aplicá-lo a nenhum recurso.
502	Bad Gateway	O servidor, quando está atuando como Gateway ou proxy recebe uma resposta inválida de outro servidor.
503	Service Unavailable	O servidor não pode atender o pedido por estar temporariamente sobrecarregado ou em manutenção.

1.1.4.6 Variáveis de ambiente

As **variáveis de ambiente** (como são chamadas no CGI), ou **variáveis do servidor** (como são conhecidas na ISAPI), são usadas, geralmente, para passar informações entre o servidor WEB e o programa. A tabela que segue apresenta algumas das variáveis de ambiente mais utilizadas.

Nome da variável	Descrição da variável
CONTENT_LENGTH	Especifica o tamanho dos dados sendo transferidos, conforme informado pelo cliente.
CONTENT_TYPE	Especifica o tipo da informação quando esta vai anexada (attached) ao pedido, como é o caso dos métodos POST e PUT.
HTTP_USER_AGENT	Informa o software cliente (Browser) utilizado.
PATH_INFO	Especifica uma informação adicional de percurso (“path”). Esta informação pode ser decodificada pelo servidor antes de ser passada para o script CGI ou ISAPI.
QUERY_STRING	Especifica a informação que acompanha o nome da URL, após o sinal de interrogação (“?”).

Nome da variável	Descrição da variável
REMOTE_HOST	Especifica o nome do host que fez o pedido. Se o servidor não possuir esta informação, será fornecido o conteúdo da variável "REMOTE_ADDR".
REMOTE_ADDR	Especifica o endereço IP do host que fez o pedido..
REQUEST_METHOD	Especifica o método com o qual está sendo feito o pedido. Pode ser: GET, HEAD, POST.
SERVER_NAME	Especifica o nome ou endereço IP do servidor.
SERVER_PORT	Número da porta HTTP pela qual o pedido foi atendido no servidor.
HTTP_COOKIE	Conteúdo de Cookies previamente definidas através do comando HTTP "Set Cookie: ...", a ser especificado, dentro de programa OpusWEB, antes do comando OPUSWEB "Html_head()".
SCRIPT_NAME	Nome do script (programa, DLL ...) sendo executado no servidor.

1.2 Interfaces de programação Web

Atualmente, existem várias interfaces disponíveis para desenvolver aplicativos Web. Estas opções incluem, por exemplo: Common Gateway Interface (CGI), Internet Server API (ISAPI), Tecnologias ONE (Netscape) e DNA (Microsoft), Java, JavaScripts, Jscripts, VBScripts, ASP, PHP, Perl .. etc

A Tecnocoop oferece suporte de programação Internet através das seguintes opções:

- A linguagem **OPUS** permite desenvolver **aplicativos Internet para ambientes UNIX**
- A linguagem **OPUSWin** permite desenvolver aplicativos Internet **para ambientes Win32**
- Scripts Java e Visual Basic podem ser embutidos nos aplicativos Web escritos mas linguagens OPUS e OpusWin
- Os Bancos de Dados OpenBASE podem ser acessados como objetos COM/DCOM através de "Server Scripts" em PHP, ASP, Perl ... etc ...

Em suma, utilizando os recursos da **OPUS** e da **OPUSWin** podem ser produzidos aplicativos cliente / servidor para plataformas Internet e Intranet.

Nesses ambientes cliente / servidor:

- o cliente pode ser qualquer Browser Web padrão, residente em qualquer plataforma, seja ela Windows, UNIX, OS/2, Mac ou simples terminais que utilizam o serviço-cliente de um host
- o servidor Web pode ser qualquer um para ambientes **UNIX** e **Win32**

1.2.1 A interface CGI

O termo CGI significa “Common Gateway Interface”. O termo Gateway foi atribuído por se tratar de um programa que atende pedidos de informação e retorna os documentos solicitados, freqüentemente gerados numa dinâmica “on the fly”. A idéia central é que esse programa, também chamado “script”, constitui um Gateway para obter informações e executar funções normalmente não implementadas num servidor Web, por exemplo, acesso a Bancos de Dados, tradução de documentos ... etc ...

Os programas CGI podem ser escritos em várias linguagens, incluindo o C, C++, Java Scripts, VB Scripts, Jscripts, Java, Perl, Pascal, ou mesmo um shell / batch.

Seja qual for o sistema operacional em que reside o servidor Web, uma transação HTTP utilizando a interface CGI apresenta a seguinte estrutura de funcionamento:

- a) O cliente (ou “user agent”) solicita um recurso do servidor HTTP
- b) O servidor HTTP verifica a requisição do cliente e determina se vai ser executado um programa CGI ou (apenas) carregar um documento (página) HTML previamente elaborado
- c) O servidor transmite a esse novo processo (programa CGI) as informações que caracterizam o ambiente
- d) O programa CGI inicia a sua execução, obtendo as variáveis de ambiente (passadas pelo servidor) para determinar que ações deverão ser tomadas
- e) O aplicativo CGI obtém (da “stdin”) as informações “postadas” pelo Browser Web e grava (em “stdout”) os documentos produzidos pelo programa CGI
- f) O servidor Web obtém as informações gravadas pelo aplicativo CGI e as transmite ao cliente Browser em resposta ao pedido originado no cliente
- g) O servidor Web finaliza o processo, liberando todos os recursos associados

1.2.2 A interface ISAPI

ISAPI (Internet Server Application Programming Interface) é uma interface de programação para servidores Web que permite:

- ◆ Incorporar funções do aplicativo dentro do próprio servidor WEB, estendendo, as suas funcionalidades sem criar processos adicionais. Estas “extensions” da ISAPI também recebem o nome de “server applications”.
- ◆ Acrescentar “filtros” nos estágios básicos do processamento das transações HTTP, obtendo, assim, um alto grau de funcionalidade, de acordo com as necessidades específicas do aplicativo.

Existem várias razões que aconselham utilizar a ISAPI:

- ◆ A ISAPI não é apenas um CGI melhor: é diferente e foi desenvolvida para resolver as deficiências do CGI
- ◆ A ISAPI precisa de menos recursos do que a CGI, permitindo atender maior número de usuários de forma simultânea.
- ◆ A ISAPI acrescenta novas funcionalidades ao servidor Web, a nível mais profundo que o CGI.

1.3 CGI e ISAPI: Vantagens e desvantagens

Neste capítulo apresentamos uma breve comparação entre a interface CGI e a ISAPI, ressaltando as vantagens e desvantagens de cada uma.

1.3.1 Vantagens do CGI

A interface é o padrão utilizado na construção de sites Web interativos. Originalmente concebido para o mundo UNIX na Internet, tem por objetivo fornecer um método comum de tratar os pedidos que procuram informações não residentes em formato HTML. No próximos itens, relacionamos algumas das vantagens dos aplicativos CGI.

1.3.1.1 Consistência com as práticas UNIX

A interface CGI nasceu no UNIX, que até recentemente predominava entre as plataformas de servidores Internet. A razão disso é simples:

- ◆ Os aplicativos CGI usam as características do próprio sistema operacional para comunicação entre processos.
- ◆ Sendo o UNIX um sistema operacional multitasking, preemptive, é fácil iniciar e terminar processos.
- ◆ Ler STDIN, gravar STDOUT e acessar as variáveis de ambiente são operações simples e tradicionais em ambientes UNIX.

1.3.1.2 Processos executando separadamente

Os aplicativos CGI são processos “standalone” que podem ser desenvolvidos e testados independentemente, utilizando os conhecimentos de UNIX. Esses processos executam separados do servidor, sendo virtualmente impossível que um aplicativo CGI mal comportado “derrube” o servidor Web.

Isto contrasta diretamente com as extensões e filtros ISAPI, que são DLLs carregadas no mesmo address space do servidor Web, constituindo “threads” do mesmo processo. Neste caso, é virtualmente possível “derrubar” o próprio servidor Web.

1.3.1.3 Facilidade de depuração

Fazer “debug” de aplicações CGI é relativamente fácil, dependendo da complexidade das mesmas, pois se trata de processos separados, sem dependências externas, por exemplo, do servidor. A forma como os aplicativos CGI utilizam a entrada e saída padrão do sistema (*stdin* e *stdout*) e as tradicionais variáveis de ambiente, facilita a obtenção e passagem de dados e parâmetros, quando se pretende fazer debug dos aplicativos.

1.3.2 Vantagens da ISAPI

A interface ISAPI permite desenvolver aplicativos Web através da utilização de suas “extensions” e seus “filters”. Nos próximos itens, relacionamos algumas das vantagens dos aplicativos ISAPI.

1.3.2.1 Maior rapidez na resposta

Quando o cliente faz uma requisição ao servidor, o que se espera é que a resposta chegue o mais rapidamente possível de volta ao cliente. As páginas HTML são geradas muito mais rapidamente através da ISAPI, em comparação com o CGI.

Isto tem uma razão: as extensões ISAPI são DLLs, sendo carregadas apenas quando chamadas pela primeira vez. Desta forma, quando invocadas, recebem o controle rapidamente.

1.3.2.2 Menor sobrecarga no servidor

Uma desvantagem do CGI é que cada aplicativo executa como diferente processo em diferente address space, ou seja, separado do servidor Web. Cada pedido do cliente, por exemplo, um simples ENTER no teclado ou um click do mouse sobre um link, gera novos processos, ou seja, novas instâncias do aplicativo. Como resultado:

- ◆ Muitos recursos são consumidos
- ◆ O servidor Web fica sobrecarregado
- ◆ O tempo de resposta aumenta do lado do usuário

Imagine um servidor com centenas ou milhares de transações HTTP executando de maneira simultânea. Esta situação pode ocasionar séria degradação no desempenho pela quantidade de processos iniciados e terminados simultaneamente.

Em contraste, na interface ISAPI, as DLLs são carregadas apenas uma vez e permanecem residentes no mesmo address space do servidor. Para cada pedido de um cliente, o servidor inicia um “thread”, dentro do mesmo processo, para chamar e processar a “extension” ISAPI.

1.3.2.3 Melhor controle do ambiente

A interface ISAPI proporciona maior controle que o CGI sobre os ambientes Web, pois permite gerenciar todos os eventos antes que qualquer “request” passe para uma “extension” ISAPI. Isto pode ser feito através dos filtros, recursos não existentes no CGI.

Os filtros podem processar os pedidos e passá-los adiante ou rejeitá-los, não permitindo que passe adiante em direção ao seu destino.

1.3.2.4 Maior segurança lógica

A interface ISAPI oferece recursos adicionais, permitindo desenvolver camadas adicionais de segurança. Uma das técnicas mais conhecidas consiste em desenvolver filtros de autenticação dos usuários, dentro do servidor Web. Se o servidor está dentro de uma Intranet corporativa, podem ser utilizados, ainda, os recursos de segurança da API Win32.

1.4 Linguagens de programação Web

São várias as linguagens de programação de aplicativos Internet que utilizam as interfaces CGI e ISAPI, entre elas, as linguagens **OPUS** e **OPUSWin**, da Tecnocoop Sistemas.

Vamos apresentar, a seguir, exemplos para as interfaces **CGI** e **ISAPI**, utilizando as linguagens C/C++, OPUS e OPUSWin.

1.4.1 Aplicativo CGI usando C

O exemplo que segue apresenta ao usuário a mensagem “Bem-vindo ao CGI!”.

```
#include <stdio.h>
void main()
{
printf("Content-type: text/html\n\n");
printf("<html><head><title>CGI C Demo</title></head>");
printf("<body><h1>Bem-vindo ao CGI!</h1>");
printf("</body></html>");
exit(0);
}
```

1.4.2 Aplicativo ISAPI usando C

Este exemplo é idêntico ao anterior, porém, utilizando a interface ISAPI e emitindo a mensagem "Bem-vindo ao ISAPI!".

```
#include <windows.h>
#include <httpext.h>
#include <string.h>
#include <stdio.h>
BOOL WINAPI GetExtensionVersion( HSE_VERSION_INFO *pVer )
{
    pVer->dwExtensionVersion = MAKELONG(HSE_VERSION_MINOR,
    HSE_VERSION_MAJOR);
    lstrcpy( pVer->lpszExtensionDesc, "ISAPI C Demo",
    HSE_MAX_EXT_DLL_NAME_LEN );
    return TRUE;
}
DWORD WINAPI HttpExtensionProc(EXTENSION_CONTROL_BLOCK *pECB)
{
    CHAR buff[2048];
    DWORD dwLen=0;
    pECB->dwHttpStatusCode=0;
    wsprintf(buff,"Content-Type: text/html\r\n\r\n"
    "<head><title>ISAPI C Demo</title></head>\n"
    "<body><h1>Bem-vindo ao ISAPI!</h1>\n");
    dwLen=lstrlen(buff);
    if (!pECB->ServerSupportFunction( pECB->ConnID,
    HSE_REQ_SEND_RESPONSE_HEADER, "200 OK",
    &dwLen,(LPDWORD) buff ))
    {
        return HSE_STATUS_ERROR;
    }
    pECB->dwHttpStatusCode=200;
    return HSE_STATUS_SUCCESS;
}
```

1.4.3 Aplicativo CGI usando OPUS

O exemplo apresentado a seguir é idêntico aos anteriores, utilizando a interface CGI.

```
$web
prog demo
html_init()
html_head()
html_put("<html><head><title>CGI OPUS Demo</title></head>")
html_put("<body><h1>Bem-vindo ao CGI</h1>")
html_put("</body></html>")
html_write("")
return
```

1.4.4 Aplicativo ISAPI usando OPUSWin

O exemplo apresentado a seguir é idêntico aos anteriores, utilizando a interface ISAPI.

```
$dll=demo.dll
```

```

proc HttpExtensionProc
html_init()
html_head()
html_put ("<html><head><title>ISAPI OPUSWin Demo</title></head>")
html_put("<body><h1>Bem-vindo ao ISAPI!</h1>")
html_put ("</body></html>")
html_write("")
return

```

1.5 Utilizando a *OPUSWEB*

As facilidades de programação de aplicativos CGI e ISAPI foram implementadas, na *OPUS* e na *OPUSWin*, através de comandos e funções.

1.5.1 Anatomia de um programa *OPUSWEB*

A estrutura geral de um programa *OPUS/OPUSWin* para gerar programas CGI ou DLLs ISAPI é a seguinte:

Gerar programa CGI “demo”	“Gerar DLL ISAPI “demo.dll”
<pre> \$web prog demo ... var=html_var(“...”) ... html_read() html_get(“...” ,varx) ... html_init() html_head () html_put (“<html><head><title><body>”) html_put(‘<FORM method=“POST” ’) html_put(‘action=“/cgi-bin/demo”>’) ... BeginHtml <INPUT type=“text” name=“xxx”> ... </body></html> ... EndHtml html_write(“”) return </pre>	<pre> \$dll=demo.dll (ou \$dll=demo.isa) proc HttpExtensionProc (ou proc http) ... var=html_var(“...”) ... html_read() html_get(“...” ,varx) html_init() html_head () html_put (“<html><head><title><body>”) ... html_put(‘<FORM method=“POST” ’) html_put(‘action=“/scripts/demo.dll”>’) ... BeginHtml <INPUT type=“text” name=“xxx”> ... </body></html> ... EndHtml html_write(“”) return </pre>

1.5.2 Comandos e funções *OPUSWEB*

Os comandos e funções utilizados para construir programas CGI e DLLs ISAPI são os seguintes:

Comando ou função	Descrição do comando / função	Sintaxe	Obs
-------------------	-------------------------------	---------	-----

Comando ou função	Descrição do comando / função	Sintaxe	Obs
Html_init	Aloca e inicializa uma área, em memória, a ser utilizada na composição do documento HTML que será enviado ao Browser (Cliente). Esta área é alocada de forma eficiente e dinâmica, sendo “esticada” ou “encolhida” conforme a necessidade, ou seja, conforme a quantidade e tamanho dos “ html_put ”.	html_init()	
Html_put	Armazena a cadeia especificada em <varc> numa área temporária, em memória, destinada a construir o documento HTML. Esta área é alocada e inicializada pelo comando html_init() , sendo gerenciada, de forma eficiente, conforme as necessidades.	html_put (<varc>)	
Html_write	Solicita ao servidor para enviar ao cliente (Browser) o documento construído através de sucessivos comandos html_put . Usualmente, o argumento passado é uma cadeia vazia, Porém, pode ser informado, como argumento, o nome de um arquivo, em cujo caso, todos os comandos HTML, especificados no comando html_put , serão gravados nele. Esta opção é útil para debug ou para uso posterior do documento HTML gerado.	html_write (“”) ou html_write (“<nome>”)	
Html_read	<p>Extraí e formata, em áreas temporárias da OPUS ou em vetores definidos dentro do programa, todos os campos de um FORM recebidos no servidor Web através do método POST.</p> <p>Quando este comando for usado sem parâmetros, os campos são armazenados em pares, conforme seu nome e seu valor, para serem disponibilizados, um a um, utilizando sucessivos comandos html_get.</p> <p>Quando este comando for usado com</p>	html_read () ou html_read(<v1>,<v2>)	

Comando ou função	Descrição do comando / função	Sintaxe	Obs
	os parâmetros <v1> e <v2>, os nomes dos campos são armazenados no vetor <v1> e os valores correspondentes são armazenados no vetor <v2>.		
Html_get	Este comando transfere para áreas do programa do usuário o conteúdo de cada um dos campos, previamente extraídos e armazenados pelo comando html_read . Cada comando html_get obtém apenas um campo, aquele identificado em <nome>, sendo que <nome> é o identificador atribuído a cada campo, botão ou figura, na opção INPUT quando da definição do FORM e <varc> é uma variável cadeia onde será colocado o conteúdo do campo identificado por <nome>. Se este campo não existir na área de memória obtida pelo comando html_read (estiver “empty”), ou seja, se não tiver sido digitado pelo usuário (no FORM HTML), o conteúdo de <varc> será uma cadeia vazia.	html_get (<nome>,<varc>)	
Html_getmultiple	Transfere para um vetor de tipo cadeia as seleções múltiplas efetuadas pelo usuário sobre uma lista ComboBox, construída por comandos HTML do tipo: <pre><SELECT MULTIPLE> <OPTION ... > </SELECT></pre> O vetor <vetc> é definido dentro do programa OPUSWeb, conforme as características da ComboBox definida dentro do Formulário. Os valores correspondentes às várias escolhas, dentro da lista ComboBox, são armazenados no vetor <vetc> de maneira seqüencial.	html_getmultiple (<nome>,<vetc>)	
Html_var	Obtém o valor de uma determinada variável de ambiente, identificada por <nome>, colocando-o na variável cadeia varx .	varx=html_var (<nome>)	

Comando ou função	Descrição do comando / função	Sintaxe	Obs
Html_head	<p>Inserir, na resposta do servidor, o cabeçalho especificado em <varc>. Geralmente, o comando html_head serve para especificar o item “Content_Type”, tanto no caso de DLLs ISAPI como programas CGI.</p> <p>Caso não seja especificado o parâmetro <varc> a OPUS constrói o cabeçalho “Content-type” no formato adequado, dependendo da plataforma (Unix ou Windows) em que o programa OPUSWEB está sendo compilado.</p>	html_head (<varc>) ou html_head () ou html_head (“”)	
Html_exec	<p>Solicita ao servidor o redirecionamento do pedido para a URL especificada pelo operando <varc>. Em aplicações CGI, usamos o comando Location para redirecionar uma URL.</p>	html_exec (<varc>)	(01)
Html_log	<p>Solicita ao servidor para gravar a cadeia <cadlog> no log do servidor Web, para ser utilizado de variadas maneiras, por exemplo, para debug e estatísticas.</p>	html_log (<cadlog>)	(01)
BeginHtml	<p>Indica que, a seguir, se inicia a codificação de comandos HTML dentro de um programa OPUS ou OPUSWin.</p> <p>Os blocos de comandos são fechados com o comando EndHtml.</p> <p>Dentro dos blocos BeginHtml até EndHtml são permitidos códigos HTML, CSS, Dynamic HTML, Java Script, VBScript, Jscript, ASP, PHP e outros, conforme a conveniência do aplicativo.</p> <p>Entre um BeginHtml e um EndHtml podem ser especificadas “tags” a serem interpretadas pela OPUSWEB, conforme os seguintes critérios:</p> <ul style="list-style-type: none"> • código que for colocado entre os delimitadores ## será processado pela OPUS e 	BeginHtml	

Comando ou função	Descrição do comando / função	Sintaxe	Obs
	<p>substituído pelo seu resultado, colocado entre aspas (“</p> <ul style="list-style-type: none"> • código que for colocado entre os delimitadores ## será processado pela OPUS e substituído pelo seu resultado • O bloco codificado entre esses delimitadores pode ser uma variável cadeia ou uma chamada de função cadeia 		
EndHtml	<p>Serve para fechar o bloco de comandos HTML aberto pelo comando BeginHtml, previamente declarado num programa OPUSWeb (OPUS ou OPUSWin)</p>	EndHtml	
Html_msg	<p>Serve para emitir uma mensagem na tela do Browser. A mensagem está contida no vetor cadeia <vec>, passado como parâmetro e será mostrada em tantas linhas quantos sejam os elementos do vetor <vec>.</p>	Html_msg(<vec>	
Html_vpost	<p>Serve para emitir uma mensagem na tela do Browser mostrando a lista dos campos de um FORM, que foram submetidos através do método POST. Esta função é útil na depuração de um programa OPUSWeb.</p>	Html_vpost()	
Html_vquery	<p>Serve para emitir uma mensagem na tela do Browser mostrando os dados contidos na variável QUERY_STRING (submetidos, por default, através do método GET). Esta função é útil na depuração de um programa OPUSWeb.</p>	Html_vquery()	
Html_venv	<p>Serve para emitir uma mensagem na tela do Browser mostrando uma lista de todas as variáveis de ambiente, também conhecidas como variáveis do servidor. Esta função é útil na depuração de um programa OPUSWeb.</p>	Html_venv()	

Observações:

(01)→ Os comandos **HtmL_exec** e **HtmL_log** são usados apenas em programas **ISAPI**.

1.5.3 Exemplo OPUSWEB

```
html_init()
html_head()
BeginHtml
<html><head><title>
Teste ISAPI...
</title></head><body>
<h2>Demo ISAPI...</h2>
<hr size=6 noshade>
EndHtml
html_write(“”)
```

1.5.4 Lógica de um programa OPUSWeb

Os componentes CGI e ISAPI constituem extensões do Servidor WEB e, como tais:

- Iniciam recebendo o controle do Servidor Web
- Trabalham como processos(CGI) ou Threads(ISAPI) separados do Servidor Web
- Terminam devolvendo o controle para o Servidor Web

Quando um programa (ou DLL) OPUSWeb recebe o controle, executa os passos informados na tabela a seguir:

Tarefa a executar	Função OPUSWeb
Testar variáveis de ambiente, por exemplo, Content_length e Query_string.	HtmL_var()
Obter os dados do FORM, informados no Browser.	HtmL_read() e HtmL_get()
Processar os dados obtidos	Programa OPUS OPUSWin
Construir documento HTML de resposta	HtmL_put(), BeginHtml e EndHtml
Retornar ao Servidor Web entregando-lhe o documento HTML a ser enviado ao Browser	HtmL_write() Return

1.6 Exemplos OPUSWEB

A seguir apresentamos dois exemplos utilizando a linguagem **OPUSWEB**. O primeiro exemplo gera o programa **teste1** de um aplicativo **CGI**. O segundo exemplo gera a DLL **teste2.dll** de um aplicativo **ISAPI**.

Os dois exemplos apresentam diferenças apenas nas duas primeiras linhas de código e fazem, exatamente, as mesmas coisas, a saber:

- ◆ Apresentar um FORM, solicitando informar o nome da pessoa
- ◆ Pesquisar o nome informado dentro do arquivo PESSOA do Banco de Dados EXEMPLO

- ◆ Apresentar, como resultado da pesquisa, a idade da pessoa. Caso o nome não exista no Banco de Dados, o campo idade é mostrado como “***”

O esquema do banco usado nestes exemplos é o seguinte:

Banco EXEMPLO 1 arqrecup

nome: PESSOA E

nome(0) U20

idade N3

1.6.1 Aplicativo OpusWEB com CGI

```

1 $web,nolib
2 prog teste1
3 gw_nome=rep(“?”,20)
4 gw_cont=””
5 gw_ida=”***”
6 content=html_var("CONTENT_LENGTH")
7 if val(content) > 0
8   html_read()
9   html_get("in_nome",gw_nome)
10  html_get("in_cont",gw_cont)
11  database EXEMPLO 1 a 2
12  use PESSOA
13  find gw_nome
14  if eof()
15    gw_ida=”***”
16  else
17    gw_ida=str(idade,3)
18  endif
19 endif      && se Content_length>0
20 html_init()
21 html_head()
22 BeginHtml
23 <html><head><title>Teste OpusWEB</title></head>
24 <body>
25 <h2>Demonstra&ccedil;&atilde;o OpusWEB com CGI</h2>
26 <form method="post" action="/cgi-bin/teste1">
27 Informe o nome:
28 <input type="text" size=20 name="in_nome" value="##gw_nome##">
29 <input type="submit" name="in_cont" value="Continua ">
30 </form>
31 <br><b>Resultado: Nome=##gw_nome## <==> Idade=##gw_ida##</b>
32 </body></html>
33 EndHtml
34 html_write(“”)
35 return

```

1.6.2 Aplicativo OpusWEB com ISAPI

```

1 $dll=teste2.dll
2 proc Http
3 gw_nome=rep(“?”,20)
4 gw_cont=””

```

```

5  gw_ida="****"
6  content=html_var("CONTENT_LENGTH")
7  if val(content) > 0
8    html_read()
9    html_get("in_nome",gw_nome)
10   html_get("in_cont",gw_cont)
11   database EXEMPLO 1 a 2
12   use PESSOA
13   find gw_nome
14   if eof()
15     gw_ida="****"
16   else
17     gw_ida=str(idade,3)
18   endif
19 endif      && se Content_length>0
20 html_init()
21 html_head()
22 BeginHtml
23 <html><head><title>Teste OpusWEB</title></head>
24 <body>
25 <h2>Demonstra&ccedil;&atilde;o OpusWEB com ISAPI</h2>
26 <form method="post" action="/scripts/teste2.dll">
27 Informe o nome:
28 <input type=text size=20 name="in_nome" value="###gw_nome###">
29 <input type=submit name="in_cont" value="Continua ">
30 </form>
31 <br><b>Resultado: Nome=###gw_nome### <==> Idade=###gw_ida###</b>
32 </body></html>
33 EndHtml
34 html_write("")
35 return

```

1.6.3 Aplicativo OpusWEB: Agenda

O programa OpusWEB a seguir apresenta um aplicativo Internet CGI cujo objetivo é manter uma agenda telefônica, devendo ser compilado com o comando **OPUS** em ambiente UNIX e disponibilizado num Servidor WEB com suporte para CGI, por exemplo, o Apache.

Se você quiser transformá-lo em uma aplicação ISAPI, basta modificar as duas primeiras linhas do programa fonte, conforme a seguir, e compilá-lo com o comando **Opuswin** em ambiente Windows e disponibilizados num Servidor WEB com suporte para ISAPI, por exemplo, PWS (Personal Web Server), IIS (NT/200), Apache for Windows e outros que você pode obter gratuitamente na Internet.

Para construir uma DLL ISAPI, com o nome de **idemow1.dll** (ou **idemow1.isa**), utilizando o mesmo fonte, modifique assim as duas primeiras linhas;

```
$dll=idemow1.dll (ou $dll=idemow1.isa)
```

```
Proc http
```

O Banco de Dados deste aplicativo é o seguinte:

```
banco agenda 1
nome: contatos E
registro:
```

ag_nome(*) U48
ag_fone U72
ag_email U120
ag_obs O4

O código fonte do programa é o seguinte:

```

1  $web
2  prog idemox1
3
4  *private dynamic gw_obs,temp1
5  private gw_obs(361),temp1(361)
6
7  mens1 = ""
8  lmsg = .f.  && mensagem a exibir ?
9  lget = .f.  && faz html_get ?
10 lprim = .f.  && se for a primeira chamada do programa
11 lresult = .f.  && resultado da pesquisa
12
13 Database /home/openserv/db/agenda 1 a 2
14
15 *****
16 ***  inicializa variáveis de trabalho - campos do FORM          ***
17 *****
18
19 gw_nome, gw_fone, gw_email, gw_obs = ""
20 gw_pes, gw_inc, gw_exc, gw_alt, gw_res = ""
21 gw_pro, gw_ant, gw_pri, gw_ult = ""
22
23 *****
24 ***  Obtêm variáveis de ambiente (server variables)          ***
25 *****
26
27 content=html_var("CONTENT_LENGTH")
28 nome_prog=html_var("SCRIPT_NAME")
29 gw_qry=html_var("QUERY_STRING")
30
31 gw_qry=strtran(gw_qry,"%20"," ")  && retira marcas MIME
32 gw_qry=strtran(gw_qry,"+"," ")  && retira marcas MIME
33
34 *****
35 ***  Se houve campos digitados pelo usuário ...          ***
36 ***  ... no FORM apresentado na tela do Browser ...      ***
37 ***  ... então são obtidos e processados esses campos.    ***
38 *****
39
40 cl=val(content)
41 if (cl > 0)
42   lget = .t.

```

```

43  else
44      lprim = .t.
45  endif
46
47  do while (lget)
48      lget = .f.
49
50      html_read()
51
52      *****
53      ***  obtém campos do FORM  ***
54      *****
55
56      html_get("in_nome",gw_nome)
57      html_get("in_fone",gw_fone)
58      html_get("in_email",gw_email)
59      html_get("in_obs",gw_obs)
60
61      html_get("in_pes",gw_pes)
62      html_get("in_alt",gw_alt)
63      html_get("in_inc",gw_inc)
64      html_get("in_exc",gw_exc)
65      html_get("in_res",gw_res)
66      html_get("in_pro",gw_pro)
67      html_get("in_ant",gw_ant)
68      html_get("in_pri",gw_pri)
69      html_get("in_ult",gw_ult)
70
71      *****
72      ***  decide o que deva ser feito ...      **
73      ***  ... de acordo com o botão clicado  **
74      *****
75
76      if !empty(gw_res)
77          gw_nome, gw_fone, gw_email, gw_obs = ""
78          exit
79      endif
80
81      select a
82      use contatos
83
84      If !empty(gw_pes)
85          locate
86          Do While found()
87              lnome, lfone, lemail = .t.
88              lresult = .f.
89              If !empty(gw_nome)
90                  if !(gw_nome $ ag_nome)
91                      lnome = .f.
92                  endif
93              Endif

```

```

94     If !empty(gw_fone)
95         if !(gw_fone $ ag_fone)
96             lfone = .f.
97         endif
98     Endif
99     If !empty(gw_email)
100         If !(gw_email $ ag_email)
101             lemail = .f.
102         endif
103     Endif
104     If (lnome .and. lfone .and. lemail)
105         lresult = .t.    && pesquisa "and": se achou... vai mostrar
106         gw_nome = ag_nome
107         gw_fone = ag_fone
108         gw_email = ag_email
109         if ! memoget (ag_obs,gw_obs)
110             mens1 = "Falhou leitura de registro na agenda ..."
111             lmsg = .t.
112         endif
113         exit
114     endif
115     continue
116 Enddo
117
118     if !lresult
119         mens1 = "Nenhum registro encontrado na agenda ..."
120         lmsg = .t.
121         exit
122     endif
123
124 Endif
125
126 if !empty(gw_pri)
127     locate
128     if eof()
129         mens1 = "Registro n&atilde;o encontrado na agenda! ..."
130         lmsg = .t.
131         exit
132     endif
133     gw_nome = ag_nome
134     gw_fone = ag_fone
135     gw_email = ag_email
136     if ! memoget (ag_obs,gw_obs)
137         mens1 = "Falhou leitura de registro na agenda ..."
138         lmsg = .t.
139     endif
140     exit
141 endif
142
143 if !empty(gw_ult)
144     locate last

```

```

145     if eof()
146         mens1 = "Registro n&atilde;o encontrado na agenda! ..."
147         lmsg = .t.
148         exit
149     endif
150     gw_nome = ag_nome
151     gw_fone = ag_fone
152     gw_email = ag_email
153     if ! memoget (ag_obs,gw_obs)
154         mens1 = "Falhou leitura de registro na agenda ..."
155         lmsg = .t.
156     endif
157     exit
158 endif
159
160 if empty(gw_nome)
161     lmsg = .t.
162     mens1 = "Faltam informa&ccedil;&otilde;es obrigat&oacute;rias ..."
163     exit
164 endif
165
166 if !empty(gw_exc)
167     find gw_nome
168     if !eof( )
169         delete
170         mens1 = "Registro exclu&iacute;do da agenda ..."
171     else
172         mens1 = "Nome n&atilde;o encontrado na agenda ..."
173     endif
174     lmsg = .t.
175     exit
176 endif
177
178 if !empty(gw_pro)
179     locate start gw_qry
180     continue
181     if eof()
182         mens1 = "Chegamos ao fim da agenda! ..."
183         lmsg = .t.
184         exit
185     endif
186     gw_nome = ag_nome
187     gw_fone = ag_fone
188     gw_email = ag_email
189     if ! memoget (ag_obs,gw_obs)
190         mens1 = "Falhou leitura de registro na agenda ..."
191         lmsg = .t.
192     endif
193     exit
194 endif
195

```

```

196  if !empty(gw_ant)
197      locate start gw_qry
198      reverse
199      continue
200  if eof()
201      mens1 = "Chegamos ao início da agenda! ..."
202      lmsg = .t.
203      exit
204  endif
205  gw_nome = ag_nome
206  gw_fone = ag_fone
207  gw_email = ag_email
208  if ! memoget (ag_obs,gw_obs)
209      mens1 = "Falhou leitura de registro na agenda ..."
210      lmsg = .t.
211  endif
212  exit
213  endif
214
215  if len(gw_obs) > 360
216      mens1 = 'Tamanho excedido no campo observa&ccedil;&otilde;es ...'
217      lmsg = .t.
218      exit
219  endif
220
221  if !empty(gw_inc)
222      find gw_nome
223      if !eof()
224          mens1 = "Registro j&aacute; existe na agenda ..."
225      else
226          replace ag_nome with gw_nome,;
227          ag_fone with gw_fone,;
228          ag_email with gw_email
229          insert
230          if ! memoput (ag_obs,gw_obs)
231              mens1 = "Falhou inclus&atilde;o de registro na agenda ..."
232          else
233              mens1 = "Registro inclu&iacute;do na agenda ..."
234          endif
235      endif
236      lmsg = .t.
237      exit
238  endif
239
240  if !empty(gw_alt)
241      find gw_nome
242      if eof()
243          mens1 = "Registro n&atilde;o existe na agenda ..."
244      else
245          replace ag_fone with gw_fone,;
246          ag_email with gw_email

```

```

247     change
248     if ! memoput (ag_obs,gw_obs)
249         mens1 = "Falhou altera&ccedil;&atilde;o de registro na agenda ..."
250     else
251         mens1 = "Registro alterado na agenda ..."
252     endif
253 endif
254 lmsg = .t.
255 exit
256 endif
257
258 enddo    && fim de "do while (lget)"
259
260 if lprim    && se primeira vez, mostra o primeiro registro da agenda
261     select a
262     use contatos
263     locate
264     if !eof()
265         gw_nome = ag_nome
266         gw_fone = ag_fone
267         gw_email = ag_email
268         if ! memoget (ag_obs,gw_obs)
269             mens1 = "Falhou leitura de registro na agenda ..."
270             lmsg = .t.
271         endif
272     endif
273 endif
274
275 *****
276 ***      Build response to web browser      ***
277 *****
278
279 html_init("")
280 html_head()
281
282 if !empty(gw_nome)
283     gx_nome=trim(gw_nome)
284     gx_nome=strtran(gx_nome," ","%20")    && coloca MIME (Netscape!!!)
285     nome_prog = nome_prog + "?" + gx_nome
286 endif
287
288 BeginHtml
289     <html><head>
290     <title>OpenBASE - OpusWEB - Programa para demonstra&ccedil;&atilde;o</title>
291     </head>
292     <body bgcolor="#FFFFDD">
293     <form method="post" action=#+nome_prog#+#>
294 EndHtml
295
296 if lmsg
297     lmsg = .f.

```

```

298     srt_mens = ""
299     str_mens = trim(mens1)
300     html_put('Aviso: <font color="#0000ff"><b>')
301     html_put(str_mens)
302     html_put('</b></font><br>'+chr(10))
303 endif
304
305 BeginHtml
306
307 <div>
308 <h2 align=center color=#FF0000>Agenda telef&ocirc;nica</h2>
309 <hr color=#0000FF size=6 noshade width=100%>
310 </div>
311
312 <div bgcolor="#CCCCFF">
313 <table width="100%" border="0" cellspacing="2" cellpadding="0">
314 <tr>
315 <td align="right" width="30%"><b>Nome:</b></td>
316 <td align="left" width="70%">
317 <input type="text" name="in_nome" size="30" maxlength="48"
318 value=##trim(gw_nome)##></td>
319 </tr>
320 <tr>
321 <td align="right" width="30%"><b>Fone(s):</b></td>
322 <td align="left" width="70%">
323 <input type="text" name="in_fone" size="30" maxlength="72"
324 value=##trim(gw_fone)##></td>
325 </tr>
326 <tr>
327 <td align="right" width="30%"><b>Email(s):</b></td>
328 <td align="left" width="70%">
329 <input type="text" name="in_email" size="60" maxlength="120"
330 value=##trim(gw_email)##></td>
331 </tr>
332 <tr>
333 <td align="right" width="30%"><b>Observa&ccedil;&otilde;es:</b></td>
334 <td align="left" width="70%">
335 <textarea name="in_obs" cols="72" rows="5">@+@trim(gw_obs)@+@
336 </textarea></td>
337 </tr>
338 </table>
339 </div>
340
341 <div>
342 <hr color=#0000FF size=6 noshade width=100%>
343 </div>
344
345 <div>
346 <table width="100%" border="0" cellspacing="2" cellpadding="0">
347 <tr>
348 <td colspan="2" align="center" bgcolor="#ffffca">

```

```

349     <input type="submit" name="in_pes" value=" Pesquisar ">
350     <input type="submit" name="in_alt" value=" Alterar ">
351     <input type="submit" name="in_inc" value=" Incluir ">
352     <input type="submit" name="in_exc" value=" Excluir ">
353     <input type="submit" name="in_res" value=" Limpar ">
354 </tr>
355 <tr>
356     <td colspan="2" align="center" bgcolor="#ffffca">
357         <input type="submit" name="in_pro" value=" Pr&oacute;ximo ">
358         <input type="submit" name="in_ant" value=" Anterior ">
359         <input type="submit" name="in_pri" value=" Primeiro ">
360         <input type="submit" name="in_ult" value=" Último ">
361     </td>
362 </tr>
363 </table>
364 </div>
365
366 <table width="100%" border="0" cellspacing="2" cellpadding="0">
367 <tr>
368     <td colspan="2" align="center">
369         <font color="#ff0000" face="Arial, Helvetica, sans-serif" size="1">
370         &copy; OpenBASE - OpusWEB. Todos os direitos reservados.</font>
371     </td>
372 </tr>
373 </table>
374 </body></html>
375
376 EndHtml
377
378 html_write("")
379 Return

```

2. Configuração de servidores Web

Independente do ambiente operacional escolhido (UNIX, Windows ...), muitos dos problemas encontrados na implantação de aplicativos CGI ou ISAPI provêm de configurações incorretas nos Servidores Web. Por isso, se você, autor de aplicativos Internet, não for o WebMaster, solicite sua ajuda na instalação de programas (scripts) CGI ou “Extensions” e “filters” ISAPI.

Para que um programa escrito em **OPUSWEB** (CGI ou ISAPI) funcione corretamente são necessários os seguintes requisitos:

- Um Servidor Web ativo (ou seja executando) e corretamente configurado.
- Um Browser ativo e apontando para esse Servidor Web
- Uma conexão TCP/IP entre ambos

Durante o processo de desenvolvimento de aplicativos Internet em **OPUSWEB**, recomendamos a instalação e configuração de servidores Web no próprio computador do aplicativo, permanecendo, desta forma, o cliente (Browser) e o servidor (Web Server) localmente, sem dependência de rede nem de outros hosts.

Existe disponível para download uma boa variedade de servidores Web na Internet, tanto para CGI, quanto para ISAPI. Alguns desses servidores são free, por exemplo, o ServerSeven (para ISAPI sob Windows) e o Apache (para CGI sob UNIX).

Cada Servidor Web possui suas próprias características, sendo necessário observar os procedimentos incluídos na documentação específica da cada um.

Entretanto, existem alguns tópicos comuns a qualquer servidor Web que permitem configurar os ambientes particulares de cada instalação. Alguns desses tópicos se referem a especificações do tipo:

- Qual a pasta ou diretório raiz do Servidor
- Qual a pasta ou diretório onde vão residir os scripts, ou programas executáveis
- Qual a pasta ou diretório para os documentos (textos HTML) e figuras
- Número máximo de processos e usuários simultaneamente
- Quais os “alias” para identificar, dentro dos programas, os componentes e recursos do Servidor, sem necessidade de saber o verdadeiro nome deles.
- Qual o nome default da página inicial, ou Home Page
- Quais as opções de Log e segurança
- Quais os filtros incorporados ao servidor
- Qual é a estrutura de permissões e privilégios do Servidor
- etc ...