



EVERSON VOLACO

API do Windows

Programação com Funções Nativas do SO



Microsoft® Windows® xp



EVERSON BORGES VOLACO
(everson@rhealeza.com.br)
é desenvolvedor e instrutor certificado Borland, com experiência em aplicações cliente/servidor, usando Delphi, Interbase e Oracle. Possui três certificações oficiais Borland: Borland Delphi 7.0, Borland CaliberRM 6.0 e Borland StarTeam 6.0.

Neste artigo utilizaremos alguns dos vários métodos disponíveis na API do Windows, para capturar as mais diversas informações do sistema operacional e das aplicações que nele são executadas. Através da API do Windows podemos realizar as mais diversas tarefas, como: acessar informações de hardware, software, interagir com outros programas, criar objetos, alterar configurações e comportamentos do sistema.

Nota: Para o exemplo deste artigo utilizei o Delphi 7 Enterprise e o Windows XP Professional. Dependendo do seu sistema operacional, uma ou outra função do exemplo terá que ser adaptada.

Criando a aplicação de exemplo

Abra o Delphi 7 e inicie uma nova aplicação. Altere o nome do formulário para “FrmPrincipal” e salve a unit como “untFrmPrincipal.pas”. Para o arquivo de projeto dê o nome de “API_Windows.dpr”. Adicione alguns componentes visuais e configure-os de acordo com a **Figura 1**.

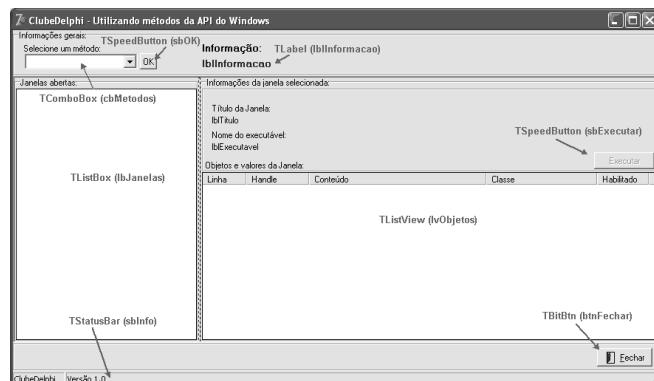


Figura 1. Layout da aplicação de exemplo em tempo de design



A idéia do exemplo é disponibilizar no *cbMetodos* uma lista pré-definida de opções onde cada uma utilizará métodos da API do Windows para retornar a informação solicitada. Concentraremos vários desses métodos em uma *unit* separada do formulário, a fim de facilitar seu uso em outros projetos.

Terminado o desenho da tela, vamos iniciar a implementação do código. Selecione o *cbMetodos* e digite a seguinte lista de opções dentro da sua propriedade *Items*:

```
<Selecione>
Usuário
Máquina
IP
Sistema Operacional
Processador
Clock
Memória
Janelas abertas
```

Ainda com o componente selecionado, altere sua propriedade *ItemIndex* para “0”. Dessa forma o primeiro item da lista ficará selecionado por padrão. No evento *OnChange* do *cbMetodos* digite o seguinte código:

```
lblInformacao.Caption := '';
lblTitulo.Caption := '';
lblExecutavel.Caption := '';
lbJanelas.Items.Clear;
lvObjetos.Items.Clear;
sbExecutar.Enabled := False;
```

Utilizamos o código anterior para “limpar” as informações dos componentes da tela quando o usuário selecionar uma opção da lista. Antes de implementarmos o botão OK, vamos criar as funções que serão chamadas a partir das opções disponíveis na lista. Crie uma nova *unit* (*File>New>Unit*) e salve-a como “untFuncoes.pas”. Logo abaixo da seção *interface* da nova unit, adicione a cláusula *uses* e coloque a referência às seguintes units:

```
uses Windows, Dialogs, Classes, SysUtils, Winsock,
  TLHelp32, ComCtrls, Registry, ComObj, Forms;
```

Utilizaremos classes e métodos dessas units em nossas funções de acesso a API do Windows. Muitos dos métodos da API estão mapeados dentro da unit *Windows*. Antes da seção *implementation* declare as funções que serão implementadas dentro da unit, conforme a **Listagem 1**.

Listagem 1. Funções para a utilização do exemplo

```
{ Retorna o IP da máquina corrente }
function RetornarIP: string;
{ Retorna o nome da máquina corrente }
function RetornoNomeDaMaquina: string;
{ Retorna o login do usuário logado }
function RetornarUsuarioLogado: string;
{ Retorna o nome do arquivo executável de um aplicativo }
function RetornarNameExe(const pHandle: hwnd): string;
{ Retorna o nome do processador da máquina corrente }
function RetornarProcessador: string;
{ Retorna a versão do sistema operacional da máquina corrente }
function RetornarSO: string;
{ Retorna o clock da máquina corrente }
function RetornarClock: string;
{ Retorna o total de memória da máquina corrente }
function RetornarMemoria: string;
```

Antes da declaração de cada função há uma breve explicação (na forma de comentário) sobre sua finalidade. Veremos que algumas dessas funções chegam a utilizar código *assembler* para conseguir capturar a informação que necessita. Não nos preocuparemos muito com o “como funciona” cada linha das funções, pois o objetivo é disponibilizar uma forma de obter tal informação apenas. Veja na **Listagem 2** o código-fonte que implementa as funções recém-declaradas.

Listagem 2. Implementação dos métodos na unit untFuncoes.pas

```
function RetornarIP: string;
var
  WSADATA: TWSAData;
  HostEnt: PHostEnt;
  Name : String;
begin
try
  WSAStartup(2, WSADATA);
  SetLength(Name, 255);
  Gethostname(PChar(Name), 255);
  SetLength(Name, StrLen(PChar(Name)));
  HostEnt := gethostbyname(PChar(Name));
  with HostEnt^ do
    Result := Format('%d.%d.%d.%d',
      [Byte(h_addr^[0]), Byte(h_addr^[1]),
       Byte(h_addr^[2]), Byte(h_addr^[3])]);
  WSACleanup;
except
  on E: Exception do
    MessageDlg('Erro ao tentar capturar o IP' + #13 +
               'Mensagem original: ' + E.Message, mtError, [mbOK], 0);
end;
end;

function RetornoNomeDaMaquina: string;
const
  Buff_Size = MAX_COMPUTERNAME_LENGTH + 1;
var
  lpBuffer : PChar;
  nSize : DWord;
begin
Result := '';
try
  nSize := Buff_Size;
  lpBuffer := StrAlloc(Buff_Size);
  GetComputerName(lpBuffer,nSize);
  Result := AnsiUpperCase(string(lpBuffer));
  StrDispose(lpBuffer);
except
  on E: Exception do
    MessageDlg('Erro ao tentar capturar' + #13 +
               'o nome da máquina' + #13 +
               'Mensagem original: ' + E.Message, mtError, [mbOK], 0);
end;
end;

function RetornarUsuarioLogado: string;
var
  lpUserName : PAnsiChar;
  lpnLength : DWORD;
begin
try
  lpnLength := 0;
  WNetGetUser (nil, nil, lpnLength);
  if lpnLength > 0 then
begin
  GetMem (lpUserName, lpnLength);
  if WNetGetUser (nil, lpUserName,
    lpnLength) = NO_ERROR then
    Result := AnsiUpperCase(lpUserName)
  else
    Result := '';
  FreeMem (lpUserName, lpnLength);
end
else
  Result := '';
except
  on E : Exception do
    MessageDlg('Erro ao tentar capturar' + #13 +
               'o usuário logado' + #13 +
               'Mensagem original: ' + E.Message, mtError, [mbOK], 0);
end;
end;
```



```
function RetornarNomeExe(const pHandle: hwnd): string;
var
  vPID: DWord;
  vASnapShotHandle: THandle;
  vContinueLoop: Boolean;
  vAProcessEntry32: TProcessEntry32;
begin
  try
    GetWindowThreadProcessID(pHandle, @vPID);
    vASnapShotHandle := CreateToolHelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    vAProcessEntry32.dwSize := SizeOf(vAProcessEntry32);
    vContinueLoop := Process32First(vASnapShotHandle,
      { Aponta o ponteiro para o inicio do Task Manager}
      vAProcessEntry32);
    { Varre a lista de tasks do Task Manager }
    while Integer(vContinueLoop) <> 0 do
    begin
      { Caso encontre o pid então }
      if vAProcessEntry32.th32ProcessID = vPID then
      begin
        {Retorna o nome do executável utilizado por este pid}
        result:= vAProcessEntry32.szExeFile;
        break;
      end;
      { Continua o loop (next) }
      vContinueLoop := Process32Next(vASnapShotHandle,
        vAProcessEntry32);
    end;
    CloseHandle(vASnapShotHandle);
  except
    on E : Exception do
      MessageDlg('Erro ao tentar capturar ' + #13 +
                 'o nome do executável' + #13 +
                 'Mensagem original: ' + E.Message, mtError, [mbOK], 0);
  end;
end;

function RetornarProcessador: string;
var
  reg: TRegIniFile;
begin
  try
    reg := TRegIniFile.create;
    reg.RootKey := HKEY_LOCAL_MACHINE;
    reg.OpenKey('HARDWARE\DESCRIPTION\SYSTEM\' +
      'CentralProcessor\', false);
    Result := reg.ReadString('0', 'Identifier', '') +
      ' ' + reg.ReadString('0', 'ProcessorNameString', '');
  finally
    reg.free;
  end;
except
  on E : Exception do
    MessageDlg('Erro ao tentar capturar ' + #13 +
               'o nome do processador' + #13 +
               'Mensagem original: ' + E.Message, mtError, [mbOK], 0);
end;
end;

function RetornarSO: string;
var
  reg: TRegIniFile;
begin
  try
    reg := TRegIniFile.create;
    reg.RootKey := HKEY_LOCAL_MACHINE;
    reg.OpenKey('SOFTWARE\Microsoft\Windows NT\', false);
    if (reg.ReadString('currentversion', 'ProductName', '') = '') then
      Result := 'WindowsNT ' + reg.ReadString(
        'currentversion', 'CurrentVersion', '')
    else
      Result := reg.ReadString('currentversion',
        'ProductName', '');
    reg.CloseKey;
  finally
    reg.free;
  end;
end;
```

```
except
  on E : Exception do
    MessageDlg('Erro ao tentar capturar a ' + #13 +
               'versão do sistema operacional' + #13 +
               'Mensagem original: ' + E.Message, mtError, [mbOK], 0);
end;

function RetornarClock: string;
const
  DelayTime = 500;
var
  TimerHi, TimerLo: DWORD;
  PriorityClass, Priority: Integer;
  cpuspeed: string;
begin
  try
    PriorityClass := GetPriorityClass(GetCurrentProcess);
    Priority := GetThreadPriority(GetCurrentThread);
    SetPriorityClass(GetCurrentProcess, REALTIME_PRIORITY_CLASS);
    SetThreadPriority(GetCurrentThread, THREAD_PRIORITY_TIME_CRITICAL);
    Sleep(10);
  asm
    dw 310Fh // rdtsc
    mov TimerLo, eax
    mov TimerHi, edx
  end;
  Sleep(DelayTime);
  asm
    dw 310Fh // rdtsc
    sub eax, TimerLo
    sbb edx, TimerHi
    mov TimerLo, eax
    mov TimerHi, edx
  end;
  SetThreadPriority(GetCurrentThread, Priority);
  SetPriorityClass(GetCurrentProcess, PriorityClass);
  cpuspeed := FormatFloat('0.00', (
    TimerLo / (1000.0 * DelayTime)));
  Result := cpuspeed;
except
  on E : Exception do
    MessageDlg('Erro ao tentar capturar o clock' + #13 +
               'Mensagem original: ' + E.Message, mtError, [mbOK], 0);
  end;

function RetornarMemoria: string;
var
  MemoryStatus: TMemoryStatus;
begin
  try
    MemoryStatus.dwLength := sizeof(MemoryStatus);
    GlobalMemoryStatus(MemoryStatus);
    Result := FormatFloat('0.00',
      MemoryStatus.dwTotalPhys / 1000000);
  except
    on E : Exception do
      MessageDlg('Erro ao tentar capturar o total de memória ' + #13 +
                 'Mensagem original: ' + E.Message, mtError, [mbOK], 0);
  end;
end;
```

Navegando pelo código fonte dessas funções percebemos que as mesmas utilizam tipos, métodos e comandos que não são comuns no dia a dia do desenvolvedor. A API do Windows possui métodos em C, os quais não possuem um padrão 100% compatível entre as versões do sistema operacional, isso é, esses métodos funcionam no Windows XP Professional, porém alguns podem deixar de funcionar em Windows 2000 ou 98, por exemplo. Alguns métodos sofreram alterações entre as versões do Windows, ganhando novos parâmetros, suporte a novos tipos etc. Terminada a implementação da unit, volte ao formulário e adicione a *unitFuncoes* a seção *uses* através da opção de menu *File>Use Unit*.



Listando as janelas ativas do sistema operacional

Voltando ao *cbMetodos*, repare que adicionamos um item chamado *Janelas abertas* em sua propriedade *Items*. A idéia é que, quando essa opção for selecionada, sejam listadas todas as aplicações abertas no Windows dentro do *lbJanelas* (*ListBox*). Ao selecionar uma janela no *lbJanelas* será carregada a lista de objetos dessa dentro do *lvObjetos* (*ListView*).

Para que possamos implementar essa funcionalidade, vamos adicionar mais alguns métodos da API, porém agora dentro da *unit* do nosso formulário. Antes da implementação dos métodos, declare as seguintes variáveis na seção *var* da *unit*:

```
var
  ValorTexto: Array [1..255] of Char;
  Conteudo: string;
  Linha: Integer;
  handleJanela: HWND;
  PidPrograma: Cardinal;
```

Essas variáveis estão em um escopo global dentro da *unit*, isso é, não faz parte da classe *TFrmPrincipal*. Utilizaremos essas variáveis dentro das funções do Windows que manipularemos a seguir. Para que possamos listar todas as janelas ativas adicione a função da **Listagem 3** dentro da seção *implementation*.

Listagem 3. Função para listar todas as janelas ativas

```
function EnumWindowsProc(
  Wnd: HWND; lb : TListbox): BOOL; stdcall;
var
  caption : Array [0..128] of Char;
begin
  Result := True;
  if IsWindowVisible(Wnd) and ((GetWindowLong(Wnd, GWL_HWNDPARENT) = 0) or
    (HWND(GetWindowLong(Wnd, GWL_HWNDPARENT)) = GetDesktopWindow) and
    ((GetWindowLong(Wnd, GWL_EXSTYLE) and WS_EX_TOOLWINDOW) = 0) then
  begin
    SendMessage(Wnd, WM_GETTEXT, Sizeof(caption),
      integer(@caption));
    lb.Items.AddObject( caption,TObject( Wnd ) );
  end;
end;
```

Utilizaremos a função anterior em conjunto com a função *EnumWindows* (API do Windows) para capturar o título de todas as janelas abertas em nosso sistema operacional, técnica muito utilizada em softwares que precisam monitorar aplicações executadas por usuários em uma máquina. Para que possamos testar as funções já implementadas vamos adicionar o código da **Listagem 4** no evento *OnClick* do botão *sbOK*

Listagem 4. Código do *sbOK*

```
lblInformacao.Caption := '';
lbJanelas.Items.Clear;
case cbMetodos.ItemIndex of
  0: MessageDlg('Selecione um método da lista', mtInformation, [mbOk], 0);
  1: lblInformacao.Caption := RetornarUsuarioLogado;
  2: lblInformacao.Caption := RetornarNomeDaMaquina;
  3: lblInformacao.Caption := RetornarIP;
  4: lblInformacao.Caption := RetornarSO;
  5: lblInformacao.Caption := RetornarProcessador;
  6: lblInformacao.Caption := RetornarClock;
  7: lblInformacao.Caption := RetornarMemoria;
  8: EnumWindows(@EnumWindowsProc, integer(lbJanelas));
end;
```

No código da listagem anterior, verificamos o índice do item selecionado no *cbMetodos* e executamos o método

equivalente à opção. Com exceção do índice 8, todos os demais têm a informação retornada pela função armazenada dentro do *lblInformacao*.

No caso da opção de índice 8 (*Janelas abertas*) utilizamos o método *EnumWindows* para realizar um *loop* em todas as janelas abertas e através do método *EnumWindowsProc* fazemos a captura do texto da barra de título de cada janela e adicionamos ao *lbJanelas*.

Nesse ponto você já pode compilar e rodar a aplicação. Veja a aplicação em execução na **Figura 2**.

Listando os objetos de uma janela ativa

Dando continuidade ao exemplo, vamos implementar agora a funcionalidade para listar todos os objetos de uma janela além de possibilitar ao usuário uma pequena interação com os objetos dessa janela.

Dentro da seção *Informações da janela selecionada* iremos trazer o título da janela e o nome do executável que possui tal janela, além de várias informações sobre os objetos presentes dentro da mesma. No formulário, adicione as funções da **Listagem 5**, dentro da seção *implementation*:

Através dos métodos *EnumProcess* e *EnumChildProc* podemos varrer uma determinada janela a partir do seu *handle* e acessar todos os objetos presentes dentro dela. Para que possamos testar esses métodos selecione o *lbJanelas* e digite o código da **Listagem 6** para o seu evento *OnClick*.

Quando o usuário clicar em um item do *ListBox*, fazemos a captura do *handle* da janela selecionada através do texto da barra de título da mesma e passamos o *handle* como parâmetro para o método *EnumProcess*, que por sua vez chama o método *EnumChildProc*, que varre todos os objetos e adiciona diversas informações sobre eles no *lvObjetos*.

Utilizamos ainda o *RetornarNomeExe* da unit *untFuncoes*, para capturar o nome do executável que contém a janela em questão. Uma vez listados os objetos da janela, vamos adicionar uma funcionalidade para interagirmos com esses objetos, desde que os mesmos sejam do tipo *Button*. Selecione o *lvObjetos* e digite o seguinte código ao seu evento *OnSelectItem*:

```
if (lvObjetos.Selected <> nil) then
  if (AnsiLowerCase(lvObjetos.Selected.SubItems[2]) = 'button') then
    sbExecutar.Enabled := True
  else
    sbExecutar.Enabled := False;
```

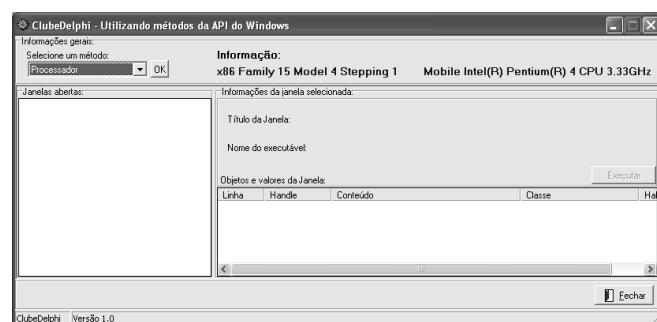


Figura 2. Buscando informações do processador utilizado na máquina



Listagem 5. Métodos para captura de informações de uma janela ativa e de seus objetos

```
function EnumChildProc(hWnd: HWND;
  vX: Integer): Boolean; stdcall;
var
  nomeClasse, Title, caption, habilitado: string;
  passwordChar: Integer;
  flag: Boolean;
begin
  Result := False;
  try
    if (hWnd = NULL) then
      result := false
    else
      begin
        flag := False;
        passwordChar := 0;
        if SendMessage(hWnd, EM_GETPASSWORDCHAR, 0, 0) <> 0 then
          begin
            flag := True;
            passwordChar := SendMessage(hWnd, EM_GETPASSWORDCHAR, 0, 0);
            PostMessage(hWnd, EM_SETPASSWORDCHAR, 0, 0);
            Sleep (500);
            SetForegroundWindow(hWnd);
          end;
        SendMessage( hWnd, WM_GETTEXT, Sizeof(ValorTexto),
          integer(@ValorTexto));
        Conteudo := strpas(@ValorTexto);
        setLength(Conteudo,length(conteudo));

        if flag then
          begin
            PostMessage(hWnd, EM_SETPASSWORDCHAR,
              passwordChar, 0);
            SetForegroundWindow(hWnd);
          end;

        SetLength(nomeClasse, 255);
        SetLength(nomeClasse,GetClassName(hWnd,
          PChar(nomeClasse),Length(nomeClasse)));
        SetLength(title, 255);
        SetLength(title, GetWindowText(hWnd,
          PChar(title), Length(title)));

        Linha := Linha + 1;

        SetLength(caption, 255);
        SetLength(caption, GetWindowText(
          handleJanela, PChar(caption), Length(caption)));
        if (AnsiUpperCase(caption) = AnsiUpperCase(
          FrmPrincipal.lbJanelas.Items[
            FrmPrincipal.lbJanelas.ItemIndex])) then
          begin
            if IsWindowEnabled(hWnd) then
              habilitado := 'SIM'
            else
              habilitado := 'NÃO';
            with FrmPrincipal.lvObjetos.Items.Add do
              begin
                Caption := IntToStr(Linha); //Linha
                SubItems.Add(IntToStr(hWnd)); //Handle
                SubItems.Add(conteudo); //Valor(Conteudo)
                SubItems.Add(nomeClasse);
                SubItems.Add(habilitado);
              end;
          end;
        end;
        Result := true;
      end;
    except
      on E: Exception do
        MessageDlg(E.Message, mtError, [mbOk], 0);
    end;
  end;

function EnumProcess(hWnd: HWND;
  lParam: integer): boolean; stdcall;
var
  pPid, LdCld: DWORD;
  title: string;
begin
  Result := False;
  try
    if (hWnd = NULL) then
      Result := False
    else
      begin
        pPid := 0;
        LdCld := 0;
        GetWindowThreadProcessId(hWnd, pPid);
        SetLength(title, 255);
        SetLength(title, GetWindowText(hWnd,
          PChar(title), Length(title)));
        handleJanela := hWnd;
        PidPrograma := pPid;
        Linha := 0;
        EnumChildWindows(hWnd,@EnumChildProc,
```

Listagem 6. Evento OnClick do lbJanelas

```
procedure TFrmPrincipal.lbJanelasClick(
  Sender: TObject);
var
  handle: HWND;
begin
  if lbJanelas.Items[
    lbJanelas.ItemIndex] <> '' then
    begin
      lvObjetos.Clear;
      handle := FindWindow(nil, PChar(
        lbJanelas.Items[lbJanelas.ItemIndex]));
      if handle = 0 then
        begin
          MessageDlg('Janela "' + lbJanelas.Items[
            lbJanelas.ItemIndex] + '" não encontrada!',',
          mtInformation, [mbOk], 0);
          Exit;
        end;
      Screen.Cursor := crSQLWait;
      EnumProcess(handle, 0);
      lblTitulo.Caption := lbJanelas.Items[
        lbJanelas.ItemIndex];
      lblExecutavel.Caption := RetornarNomeExe(handleJanela);
      Screen.Cursor := crDefault;
    end;
end;
```

No código anterior, verificamos se o usuário selecionou algum item e se a coluna *Classe* possui o valor *Button*, indicando que o objeto é do tipo botão. Dependendo do tipo do objeto selecionado habilitamos ou não o botão *sbExecutar* de nossa aplicação.

A idéia do *sbExecutar* é permitir o envio de um *click* para o botão da janela externa selecionada. Para finalizar nosso exemplo selecione o *sbExecutar* e digite o seguinte código em seu evento *OnClick*:

```
SetForegroundWindow(handleJanela);
SendMessage(handleJanela, WM_COMMAND, MAKEPARAM(GetWindowLong(StrToInt(
  lvObjetos.Selected.SubItems[0]), GWL_ID),
  BN_CLICKED), StrToInt(lvObjetos.Selected.SubItems[0]));
```

No código anterior, utilizamos o *SetForegroundWindow* para colocar a janela selecionada em primeiro plano. Através do método *SendMessage*, enviamos uma simulação do comando *click* para o botão selecionado na lista de objetos da janela através do *handle* da janela e do *handle* do botão. Compile novamente a aplicação e rode a mesma para que possamos testá-la.

Testando a aplicação de exemplo

Para que possamos testar a aplicação, execute a mesma e abra diversos aplicativos no Windows. Entre esses aplicativos abra a Calculadora que acompanha o sistema operacional. Selecione uma opção qualquer no *cbMetodos* e clique sobre o botão OK para capturar a informação desejada.

Caso seja selecionada a última opção, todas as janelas abertas serão listadas no *ListBox*. Selecione o item referente à janela Calculadora. Repare que todos os objetos presentes são listados no *ListView*.

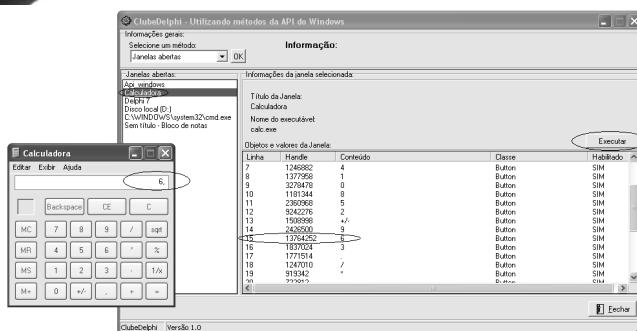


Figura 3. “Manipulando” a calculadora do Windows a partir de uma aplicação Delphi

Vamos utilizar nossa aplicação para acessar os botões da calculadora e efetuar alguns cálculos. Selecione os objetos que possuem sua classe definida como *Button* e clique no botão *Executar* para enviar o comando. Veja a aplicação interagindo com a calculadora do Windows na **Figura 3**.

Conclusão

Vimos neste artigo que através da API do Windows, pode-

mos ter acesso a uma infinidade de opções do sistema operacional, além interagir com aplicativos externos e até mesmo criar nossos próprios objetos e janelas. Meu objetivo foi mostrar que através de aplicativos Delphi, podemos obter e manipular as mais diversas informações do SO, por mais específicas que elas possam ser. Um abraço e até a próxima. ■

Linux? Windows? Unix?
.Net? Java? PHP? ISAM ou SQL?
Alta Performance! Ambiente
Heterogeno! Arquivos 64-Bit!
Thread Safe! Espelhamento ou
notificação de alterações nos
arquivos? VCL ou dbExpress?
Transações com recuperação
automática? Suporte
profissional!

**Você não precisa
ser um gênio**

para descobrir porque o **c-tree** da **FairCom**
é uma ferramenta completa de
desenvolvimento.

Basta fazer o teste!

Baixe hoje mesmo:
www.faircom.com/go/?msdnld

 **FREE
TRIAL**

 **FairCom®**