Capítulo

9

Monitoração de Performance no WNT

“ Não é possível controlar aquilo que não se consegue medir”
(Anônimo)

Autor: Constantino Seixas Filho
UFMG – Departamento de Engenharia Eletrônica

Performance Monitoring

API de Monitoração de performance do WNT proporciona uma monitoração contínua das características de tempo real do sistema. Algumas informações dizem respeito a aspectos gerais do funcionamento do S.O. como o tempo total de *uptime* do sistema, enquanto outras são pertinentes ao seu estado de funcionamento instantâneo como por exemplo o número de threads de um processo em execução em um dado instante e por quanto tempo estão executando. Estas informações são mantidas no *registry* do sistema sob a chave HKEY_PERFORMANCE_DATA, mas são mais facilmente manipuladas através da biblioteca Performance Data Helper (PDH) e está implementada na forma de uma DLL de nome PDH.DLL. Esta DLL e sua documentação acompanham o Win32 SDK.

O comportamento sob o Windows 95 difere muito da operação com o WNT. Até a chave utilizada para acessar a *registry* muda de HKEY_PERFORMANCE_DATA para HKEY_DYN_DATA. As funções estudadas neste capítulo foram desenvolvidas para o Windows NT 5.0 ou superior.

Ao realizar uma *query* solicitando dados de algum aspecto do sistema operacional, muitas vezes dados adicionais são retornados, por exemplo se solicitamos dados sobre threads, informações sobre os processos também são retornadas uma vez que não se pode descrever completamente uma thread sem mencionar o processo que a contém.

Pode-se acessar itens globais cujos valores não variam entre duas leituras. Estes itens incluem system, processor (list), memory, cache, PhysicalDisk (list), LogicalDisk (list), process (list), thread (list), objects, redirector, server, paging file e browser.

As informações dinâmicas são mais caras de se obter e incluem: process address space (list), image (list), thread details (list).

Pode-se obter detalhes de máquina remota de uma rede a partir do nodo corrente bastando endereçar o nome da máquina no caminho de acesso (*path*).

Para os interessados em obter a informação de performance diretamente da *registry* recomendamos os artigos [Pietrek Ma_1996] e [Pietrek Apr_1996]. Mas deixamos a seguinte advertência: é um exercício de tortura. A maior parte dos parâmetros a serem passados na *query* são opcionais e possuem tamanhos variáveis e afinal é para isso que existe a dll.

First things first – definições

Para acessar os dados de interesse é necessário introduzir alguns conceitos:

Contador (Counter)

Um valor de dado a respeito de alguma coisa, por exemplo, o número de threads em um processo, ou o número de troca de contexto por segundo (*task switches/s*). Counters são referenciados por handles denominados HCOUNTERs.

Objeto (*Object*)

O objeto constitui um entidade de nível mais alto à qual contadores foram associados. São exemplos de objetos: Memória(*Memory*), Sistema(*System*), Cache, etc. Alguns objetos possuem diversas instâncias, por exemplo, o objeto Processo (*process*) possui diversas instâncias, uma para cada processo.

Caminho (Path)

Um *path* é um string de texto que define completamente como a PHD.dll irá encontrar um contador no sistema. Por exemplo, o *path* abaixo define o contador de tempo de execução em modo privilegiado (*kernel mode*) de uma thread associada ao processo *MyProgram*, executando na máquina de nome "Paracelso".

```
"\\Paracelso\Thread(MyProgram/0#0)\% Privileged Time"
```

A parte /0#0 indica uma relação de paternidade. 0#0 significa a 0-ésima instância de Myprogram.exe # 0-ésima thread. Se desejássemos nos referir à thread primária da segunda instância do processo MyProgram escreveríamos: /1#0.

O nome da máquina é opcional. Se o nome for omitido será utilizada a máquina local.

A expressão genérica seria:

\\NomeDaMáquina\NomeDoObjeto(NomeDaInstância)\NomeDoContador

Outro exemplo:

```
\\Pavilion\Process(Explorer)\ThreadCount
```

Consulta (Q*uery*)

Queries são referenciadas por *handles* de nome H*QUERY*. Queries são utilizadas para montar um grupo de vários caminhos para uma consulta ao sistema. O mesmo handle de uma *query* pode ser utilizado várias vezes para trazer valores atualizados dos parâmetros.

É responsabilidade do usuário liberar a *query* após o uso através do seu handle. A seqüência de utilização de uma *query* é a seguinte: alocar *query*, associar contadores à *query*, executar a *query*, ler os valores dos contadores e liberar a *query*.

Alguns contadores têm caminhos fixos que independem de sazonalidades, mas a maioria dos contadores só podem ser determinados em tempo real, pois dependem de que processos e que threads estão executando a cada instante.

A referência [Russ Blake 1995] contem um lista de todos os contadores do WNT. Mais tarde novos contadores foram anexados ao Windows 2000.

Exemplos de contadores:

Objeto	Contador	Nível	Explicação
System [Index = 2]	%TotalPrivilegedTime	Advanced	Average percentage of time spent in Privileged mode by all processors.
	%TotalProcessorTime	Novice	Average percentage of time that all the processors on the system are busy executing non-idle threads.
	%TotalUserTime	Advanced	Average percentage pf time spent in User mode by all processors.
	ContextSwitches/s	Advanced	Rate of switches from one thread to another.
	FileDataOperations/s	Novice	Rate that the computer is issuing Read and Write operations to file system devices.
	FileReadOperations/s	Novice	Aggregate of all the file system read operations on the computer.
	FileWriteOperations/s	Novice	Aggregate of all file system write operations on the computer.
	SystemUpTime	Novice	TotalTime, in seconds, that the computer has been operational since it was last started
Memory [Index = 4]	AvailableBytes	Expert	Displays the size of the virtual memory currently on the Zeroed, Free and Standby lists.
	CommitedBytes	Expert	Size of virtual memory (in bytes) that has been committed as opposed to simply reserved.
	CommitLimit	Wizard	Size in bytes of virtual memory that can be committed without having to extend the paging files.
	Cache Bytes	Advanced	Measures the number of bytes currently in use by the system cache.
	CacheFaults/s	Wizard	Occur whenever the cache manager does not find a file’s page in the immediate cache and must ask the memory manager to locate the page elsewhere in memory or on the disk so that it can be loaded into the immediate cache.
	Pages/s	Novice	Number of pages read from the disk or written to the disk to resolve memory references to pages that were not in memory at the time of the reference.
	PageFaults/s	Novice	Number of pages read from disk or written to the disk to resolve memory references to pages that were not in memory at the time of the reference.

Processor [Index = 238]	%PrivilegedTime	Advanced	Percentage of processor time spent in Privileged Mode in Non-Idle threads. The WNT service layer, the executive routines, and the WNT kernel execute in Privileged Mode.
	%ProcessorTime	Novice	Percentage of the elapsed time that a processor is busy executing a non-idle thread. It can be viewed as the fraction of time spent doing useful work.
	%UserTime	Advanced	Percentage of processor time spent in User Mode in non-Idle threads. All application threads execute in User Mode.
	Interrupts/s	Novice	Number of device interrupts the processor is experiencing.
Process [Index = 230]	%PrivilegedTime	Advanced	Percentage of elapsed time that this process'threads have spent executing code in Privileged mode.
	%ProcessorTime	Novice	Percentage of elapsed time that all the threads of this process used the processor to execute instructions.
	%UserTime	Advanced	Percentage of elapsed time that all the threads of this process have spent executing code in User Mode. As do subsystems like the window manager and the graphics engine.
	ElapsedTime	Advanced	The total elapsed time in seconds this process has been running. The default scale is 0.0001.
	HandleCount		
	IDProcess	Novice	ID Process is the unique identifier of this process. ID Process are reused, so they only identify a process for the lifetime of that process.
	PageFaults/s	Novice	Rate of PageFaults by the threads executing in this process. A page fault occurs when a thread refers to a virtual memory page that is not in its working set in main memory.
	PriorityBase	Advanced	The current base priority of this process.
	ThreadCount	Advanced	Number of current threads currently active in this process. Every running process has at least one thread.
	WorkingSet	Novice	Current number of byte in the Working Set of this process. The Working Set is the set of memory pages touched recently by the threads in the process. If free memory in the computer is above a threshold, pages are left in the working set of a process even if they were not in use.
Thread [Index = 232]	%PrivilegedTime	Advanced	Percentage of elapsed time that this thread has spent executing code in Privileged Mode.

	%ProcessorTime	Novice	Percentage of elapsed time that this thread used the processor to execute instructions. An instruction is the basic unit of execution in a processor and a thread is the object that executes instructions.
	%UserTime	Advanced	Percentage of elapsed time that this thread has spent executing code in User Mode.
	ContextSwithes/s	Advanced	Rate of switches from one thread to another. Thread switches can occur either inside of a single process or across processes.
	ElapsedTime	Advanced	The total elapsed time in seconds this thread has been running.
	IDProcess	Wizard	ID Process is the unique identifier of this process. ID Process are reused, so they only identify a process for the lifetime of that process.
	IDThread	Wizard	Unique identifier of this thread. ID Thread numbers are reused so they only identify a thread for the lifetime of that thread.
	PriorityBase	Advanced	The current priority base of this thread. The system may raise the thread's dynamic priority above the base priority if the thread is handling user input, or lower it towards the base priority if the thread becomes compute bound.
	PriorityCurrent	Advanced	The current priority of this thread. The system raise the thread's dynamic priority above the base priority if the thread is handling user input, or lower it towards the base priority if the thread becomes compute bound.
	StartAddress	Wizard	Starting virtual address for this thread.
	ThreadState	Wizard	Current state of the thread. It is 0 for Initialized, 1 for Ready, 2 for Running, 3 for Standby, 4 for Terminated, 5 for Wait, 6 for Transition, 7 for Unknown. A Running thread is using the processor; a Standby thread is about to use one. A Ready thread wants to use a processor, but is waiting for processor because none are free. A thread in Transition is waiting for a resource in order to execute, such as waiting for its execution stack to be paged in from disk. A Waiting thread has no use for the processor because it is waiting for a peripheral operation to complete or a resource to become free.

	ThreadWaitReason	Wizard	ThreadWaitReason is only applicable when the thread is in the Wait state. It is 0 or 7 when the thread is waiting for the Executive, 1 or 8 for a Free Page, 2 or 9 for a Page In, 3 or 10 for a Pool Allocation, 4 or 11 for an Execution Delay, 5 or 12 for a Suspended condition, 6 or 13 for User Request, 14 for an Event Pair High, 15 for an Event Pair Low, 16 for an LPC Receive, 17 for an LPC Replay, 18 for Virtual Memory, 19 for Page Out. Event Pairs are used to communicate with protected subsystems.

Tabela 1 – Exemplos de contadores para vários objetos extraído de [Russ Blake 1995]

PDHCounters – demonstrando o uso da API PDH em C++

Este exemplo foi extraído da referência [Pietrek May_1998].

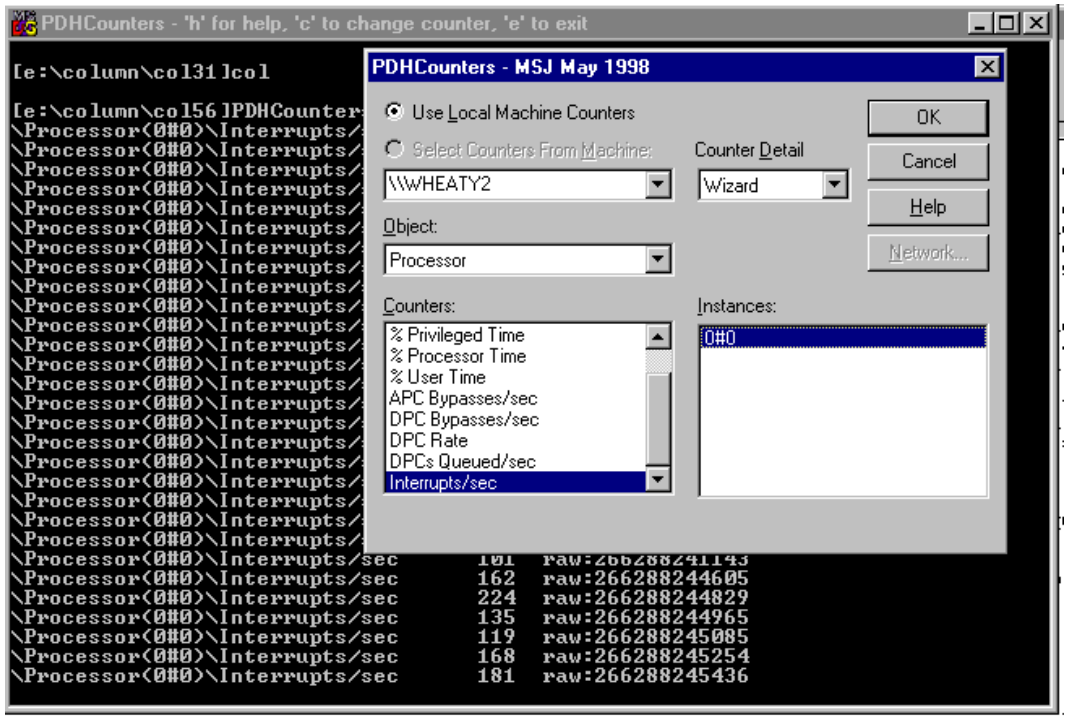


Figura 1- Interface do programa PDHCounters

Este programa permite ao usuário selecionar um dos contadores monitorados pelo WNT em sua própria máquina ou em qualquer máquina da rede. Depois o programa exibe o valor do contador selecionado a cada segundo em uma outra janela em modo console. Apenas um contador pode ser exibido de cada vez, mas o contador pode ser trocado durante a execução do programa. O programa também exibe o *help* que o WNT define para cada contador ao se apertar a tecla < h >. A tecla < c > serve para trocar o contador a ser exibido e a tecla < E > termina o programa.

Código fonte do programa- PDHCounters

```
//================================================
// PDHCounters - Matt Pietrek 1998
// Microsoft Systems Journal, May 1998
// FILE: PDHCounters.CPP
//================================================

#pragma comment(lib,"PDH.lib")

#define DUNICODE        1       // irá usar UNICODE
#define D_UNICODE       1
```

```
#define DWIN32_LEAN_AND_MEAN

#include <windows.h>
#include <stdio.h>
#include <tchar.h>    // Permite uso de qualquer tipo de caracter: ASCII (SBCS), MBCS, ou Unicode
#include <conio.h>
#pragma hdrstop
#include "pdh.h"      // Está no diretório do Visual Studio

//============================ Function prototypes ============================
void ChangeCounter( void );
void ShowCounterHelp( HCOUNTER hCounter );
BOOL CheckForKeystrokes( void );

//============================== Global Variables ============================

HQUERY g_hQuery = 0;
HCOUNTER g_hCounter = 0;
TCHAR g_szCounterPath[256];

//================================ Code ==================================

int main()
{
   // Obtém um handle para uma query
   PdhOpenQuery( NULL, 0, &g_hQuery );

   // Muda o título da janela da aplicação, que opera em modo console, para
   // proporcionar um mínimo de help online
   SetConsoleTitle(  // Converte para UNICODE. A constante _UNICODE deve ser definida
      _T("PDHCounters - 'h' for help, 'c' to change counter, 'e' to exit") );

   // Chama a janela de diálogo para selecionar o contador inicial
   // e retornar um path para ele
   ChangeCounter( );

   // Entra em loop até o usuário clicar 'e' (ou CTRL-C)
   while ( 1 )
   {
      PDH_FMT_COUNTERVALUE fmtValue;
      PDH_RAW_COUNTER rawCtr;

      if ( g_hCounter )
      {
         // Pede atualização dos dados da query
         PdhCollectQueryData( g_hQuery );

         // Busca a versão formatada do valor do contador
         PdhGetFormattedCounterValue(g_hCounter,PDH_FMT_LONG,0, &fmtValue);

         // Busca o valor bruto do contador
         PdhGetRawCounterValue( g_hCounter, 0, &rawCtr );

         // Imprime o path do contador, o valor formatado e o valor bruto
         // Observe o uso do formato %I64u para imprimir um número de 64 bits
         _tprintf( _T("%s %8u  raw:%I64u\n"), g_szCounterPath,
                  fmtValue.longValue, (LONG)rawCtr.FirstValue );
      }

      // Observe se o usuário clicou um comando
```

```
        // Saia do loop se a tecla for < e >
        if ( CheckForKeystrokes() )
            break;

        // Durma 1 second
        Sleep( 1000 );
    }

    return 0;
}

//
// Função que permite ao usuário mudar o contador corrente (g_hCounter)
// Primeiro, a função usa a API PDH para exibir a janela de diálogo para escolha do contador
// Se uma contador for selecionado, ele é adicionado à query )(g_hQuery),
// deslocando o contador anterior.
//
void ChangeCounter( void )
{
    PDH_BROWSE_DLG_CONFIG brwDlgCfg;
    TCHAR szCounterPath[256];  / Buffer local para receber o path do contador selecionado

    // Inicializa todos os campos da estrutura PDH_BROWSE_DLG_CONFIG
    // em preparação para a chamada de PdhBrowseCounters

    brwDlgCfg.bIncludeInstanceIndex = FALSE;
    brwDlgCfg.bSingleCounterPerAdd = FALSE;
    brwDlgCfg.bSingleCounterPerDialog = 1;
    brwDlgCfg.bLocalCountersOnly = 1;
    brwDlgCfg.bWildCardInstances = 0;
    brwDlgCfg.bHideDetailBox = 0;
    brwDlgCfg.bInitializePath = 0;

   / Modificações introduzidas neste trecho devido a alterações em PDH.h
   brwDlgCfg.bDisableMachineSelection = FALSE;        // incluído no WIN 2000
   brwDlgCfg.bIncludeCostlyObjects = FALSE;           // incluído no WIN 2000
   brwDlgCfg.bReserved = 0;
   brwDlgCfg.hWndOwner = 0;
   brwDlgCfg.szDataSource = 0;              //  Windows 2000
   // brwDlgCfg.szReserved = 0;             // Linha original do WNT

    brwDlgCfg.szReturnPathBuffer = szCounterPath;
    brwDlgCfg.cchReturnPathLength = sizeof(szCounterPath);
    brwDlgCfg.pCallBack = 0;          // Parâmetro para função callback associada
    brwDlgCfg.dwCallBackArg = 0;  // Função callback associada
    brwDlgCfg.CallBackStatus = 0;
    brwDlgCfg.dwDefaultDetailLevel = PERF_DETAIL_WIZARD; // Nível de detalhe
    brwDlgCfg.szDialogBoxCaption = _T("PDHCounters - MSJ May 1998");  // Título

    // Chama a janela de diálogo
    if  ( ERROR_SUCCESS != PdhBrowseCounters( &brwDlgCfg ) )
        return;

    // Se ainda houver um contador em uso, remova-o da query.
    if ( g_hCounter )
        PdhRemoveCounter( g_hCounter );

    // Adicione o contador cujo path foi obtido via PdhBrowseCounters
    PdhAddCounter( g_hQuery, szCounterPath, 0, &g_hCounter );
```

```
    // Copia o caminho do contador para uma variável global
    lstrcpy( g_szCounterPath, szCounterPath );
}

//
// Dado um valor de contador, esta função chama a API PDH para buscar o
// texto de auxílio associado a um contador.
// O string resultante é exibido em uma caixa de mensagem
//
void ShowCounterHelp( HCOUNTER hCounter )
{
    // Se texto de explicação for retornado, ele irá vir logo em sequencia à
    // estrutura  PDH_COUNTER_INFO formal.  Constrói um array de bytes longo o suficiente
    // para conter a estrutura e o string ao final
    BYTE counterBuff[ sizeof(PDH_COUNTER_INFO) + sizeof(TCHAR)*2048 ];
    DWORD cbCounterBuff = sizeof(counterBuff);

    if ( ERROR_SUCCESS != PdhGetCounterInfo( hCounter,
                                             TRUE,
                                             &cbCounterBuff,
                                             (PPDH_COUNTER_INFO)counterBuff) )
    {
        MessageBox( 0, _T("PdhGetCounterInfo failed"), 0, MB_OK );
        return;
    }

    // Prepara um apontador do tipo apropriado para acessar os membros da estrutura
    PPDH_COUNTER_INFO pCounterInfo = (PPDH_COUNTER_INFO)counterBuff;

    MessageBox( 0, pCounterInfo->szExplainText, g_szCounterPath, MB_OK );
}

//
// Função que verifica se uma tecla foi apertada. Se foi, a tecla acionada é recuperada,
// e comparada contra uma lista de teclas de funções pre definidas
//
BOOL CheckForKeystrokes( void )
{
    if ( _kbhit() ) // Aposto que você não via este cara há muito tempo...
    {
        char chKeystroke = _getch();    // e este aqui também!

        if ( 'h' == chKeystroke )       // 'h' para ajuda
            ShowCounterHelp( g_hCounter );
        else if ( 'c' == chKeystroke )  // 'c' para "troca contador"
            ChangeCounter();
        else if ( 'e' == chKeystroke )  // 'e' para saída
            return TRUE;
    }

    return FALSE;
}
```

Explicações sobre o programa

O programa se inicia com **PdhOpenQuery**. Esta função retorna no buffer, cujo endereço é passado como último parâmetro, o handle para a *query*, HQUERY.

P d h O p e n Q u e r y

```
PDH_STATUS PdhOpenQuery(

IN LPCTSTR szDataSource,    // Nome do arquivo de log (Windows 2000) de onde os dados de performance serão buscados ou NULL para tempo real.
IN DWORD dwUserData,        // Valor definido pelo usuário a ser associado à query. Você pode chamar PdhGetCounterInfo para recuperar estes dados.
IN HQUERY *phQuery          // Endereço do buffer para receber o handle para a query a ser criada.

);
```

Retorno da função:

Status	**Interpretação**
ERROR_SUCCESS	Sucesso
PDH_MEMORY_ALLOCATION_FAILURE	Não pôde alocar buffer
Código de erro definido em PDHMSG.H	Erro

Agora deve-se associar um contador à *query*. Para executar este passo fazemos uso da API **PdhBrowseCounters** da PDH.DLL que irá exibir uma janela de diálogo permitindo escolher o contador desejado.
A janela de diálogo permite escolher o computador, o objeto e sua instância e o contador. Uma vez realizada a seleção o *path* completo do contador será retornado para o buffer passado na estrutura de configuração do comando.
Observe o campo *counter detail* da janela. Para escolher o contador os usuários são categorizados em quatro níveis: *novice*, *advanced*, *expert* e *wizard*. Quando usando um nível inferior apenas parte dos contadores de mais fácil interpretação são mostrados. Á medida de subimos de classe cada vez mais contadores são mostrados.
PdhBrowseCounters exige a inicialização da estrutura PDH_BROWSE_DLG_CONFIG. O campo mais importante desta estrutura á o campo brwDlgCfg.szReturnPathBuffer que indica o buffer para retorno do *path* do contador escolhido.

Outra função que poderia ser escolhida é a função **PdhVbGetOneCounterPath**, inicialmente concebida para uso com o VB, mas que pode ser utilizada a partir do compilador C/C++, bastando para isso acrescentar um protótipo da função ao arquivo PDH.H.

Agora usamos a função **PdhAddCounter** para adicionar o *path* do *counter* a um HPATH.

P d h A d d C o u n t e r

```
PDH_STATUS PdhAddCounter(
```

```
IN HQUERY hQuery,                  // Handle para a query à qual o contador será adicionado
IN LPCTSTR szFullCounterPath,      // Path qualificado para o contador a ser criado
IN DWORD dwUserData,               // Valor definido pelo usuário, geralmente um índice para a estrutura contador do usuário. Permite associar seu próprio dado ao contador, dado que pode ser recuperado com a instrução PdhGetCounterInfo.
IN HCOUNTER *phCounter             // Endereço do buffer para receber o handle para o counter criado.
);
```

Retorno da função:

Status	**Interpretação**
ERROR_SUCCESS	Sucesso
Código de erro definido em PDHMSG.H	Erro

PhdRemoveCounter é usado para remover um contador antigo do *path* já que apenas um contador é lido de cada vez.

Agora que a *query* está montada em g_hQuery podemos ler e exibir o valor do contador, o que é feito no *loop* principal do programa. **PdhCollectQueryData** atualiza o valor de todos os contadores relacionados na *query* e atualiza o status de cada contador. Esta função é muito poderosa e recupera e realiza o *parsing* dos dados retornados pelo WNT.

C o l l e c t Q u e r y D a t a

```
PDH_STATUS PdhCollectQueryData(
IN HQUERY hQuery,                  // Handle para a query que será atualizada
);
```

Retorno da função:

Status	**Interpretação**
ERROR_SUCCESS	Sucesso
PDH_INVALID_HANDLE	Handle Inválido
PDH_NO_DATA	Nenhum contador definido na *query*

O programa de exemplo lê apenas o conteúdo de um único contador. Afora a função **PdhGetFormattedCounterValue** é chamada. Esta função retorna o valor do contador independente do seu tipo que pode ser: um inteiro de 32-bits (PDH_FMT_LONG), um inteiro de 64 bits (PDH_FMT_LARGE), ou um *double* (PDH_FMT_DOUBLE).
Muitas vezes os valores dos contadores lidos se modificam com o tempo e precisam ser processados convenientemente. Por exemplo, se o contador

selecionado for o número de trocas de contexto, o número lido será um contador cujo valor corresponde ao número de *context switches* desde que o sistema foi incializado. Este valor é incrementado a cada leitura. Para que o resultado faça sentido devemos efetuar duas leituras consecutivas, realizar a subtração dos dois relutados e dividir o valor encontrado pelo tempo entre as duas leituras.

PdhGetRawCounterValue é uma rotina que lê o valor bruto dos contadores, e retorna o valor corrente e o anterior e o *time stamp* da leitura corrente.

Linha do programa:

```
// Busca o valor bruto do contador
PdhGetRawCounterValue( g_hCounter, 0, &rawCtr );
```

PdhGetRawCounterValue

```
PDH_STATUS PdhGetRawCounterValue(

IN HCOUNTER hCounter,          // Handle para contador
IN LPDWORD lpdwType,           // Apontador para buffer que irá receber o tipo do
                                  contador conforme descrito em WINPERF.H,
                                  O parâmetro é opcional.
IN PPDH_RAW_COUNTER pValue     // Apontador para o buffer que irá receber o valor
                                  do contador
);
```

Retorno da função:

Status	**Interpretação**
ERROR_SUCCESS	Sucesso
PDH_INVALID_ARGUMENT	Argumento inválido
PDH_INVALID_HANDLE	Handle para contador inválido

A estrutura _PDH_RAW_COUNTER possui o seguinte formato:

```
typedef struct _PDH_RAW_COUNTER {
  DWORD       CStatus; // status da última operação com contador
  FILETIME    TimeStamp;  // Data e hora da coleta de dados
  LONGLONG    FirstValue; // Primeiro valor bruto lido
  LONGLONG    SecondValue;// Segundo valor bruto lido
  DWORD       MultiCount; // Geralmente 1
} PDH_RAW_COUNTER, *PPDH_RAW_COUNTER;
```

PdhGetFormattedCounterValue já é capaz de identificar se o contador é variável e já retorna o valor trabalhado no caso de dados variantes no tempo o que facilita todo o processamento dos contadores.

Linha do programa:

```
// Busca a versão formatada do valor do contador
PdhGetFormattedCounterValue(g_hCounter,PDH_FMT_LONG,0, &fmtValue);
```

```
PDH_STATUS PdhGetFormattedCounterValue(

IN HCOUNTER hCounter,                  // Handle para contador
IN DWORD dwFormat,                     // Informação de formato
IN LPDWORD lpdwType,                   // Valor de retorno: tipo do
                                          contador
IN PPDH_FMT_COUNTERVALUE pValue        // Buffer para valor do contador
);
```

Comentários sobre os parâmetros:

hCounter	Handle para contador cujo valor será formatado e retornado	
dwFormat	Informação de formatação Retorna dado como:	
	PDH_FMT_DOUBLE	double-precision floating point real
	PDH_FMT_LARGE	64-bit integer
	PDH_FMT_LONG	Long integer
	Esta informação pode ser combinada com a função OR com os seguintes fatores de escala:	
	PDH_FMT_NOSCALE	Não aplica fator de escala
	PDH_FMT_1000	Multiplica valor atual por 1000
lpdwType	Apontador para o DWORD buffer que irá receber o valor do tipo do contador. Pose ser NULL	
pValue	Apontador para o buffer que irá receber o valor do contador	

Retorno da função:

Status	**Interpretação**
ERROR_SUCCESS	Sucesso
PDH_INVALID_ARGUMENT	Argumento inválido
PDH_INVALID_HANDLE	O contador não contém dados válidos
PDH_INVALID_DATA	Handle para contador inválido

O exemplo dado imprime na mesma linha o valor formatado seguido do valor bruto do contador.

A API **PdhGetCounterInfo** retorna informação de auxílio ou explicação associada ao contador

Outra função muito útil, embora não tenha sido usada neste exemplo é *PhdComputeCounterStatistics*. Ela irá calcular o valor mínimo, máximo e médio de um vetor de contadores brutos dado.

P d h C o m p u t e C o u n t e r S t a t i s t i c s

```
PDH_STATUS PdhComputeCounterStatistics(

IN HCOUNTER hCounter,                   // Handle para contador
IN DWORD dwFormat,                      // Informação de formato
IN DWORD dwFirstEntry,                  // Índice começando em 0 do primeiro (mais antigo) contador bruto no buffer.
IN DWORD dwNumEntries,                  // Número de contadores brutos no vetor
IN PPDH_RAW_COUNTER lpRawValueArray,    // Apontador para o vetor de contadores.
IN PPDH_STATISTICS data                 // Buffer para receber a estatística

);
```

```
typedef struct _PDH_STATISTICS {
    DWORD                   dwFormat; // Formato dos contadores
    DWORD                   count;    // Número de contadores
    PDH_FMT_COUNTERVALUE    min;      // Valor mínimo
    PDH_FMT_COUNTERVALUE    max;      // Valor máximo
    PDH_FMT_COUNTERVALUE    mean;     // Valor médio
} PDH_STATISTICS, *PPDH_STATISTICS;
```

A estrutura PDH-STATISTICS conterá os valores máximo, mínimo e médio dos contadores.

Retorno da função:

Status	**Interpretação**
ERROR_SUCCESS	Sucesso
PDH_INVALID_ARGUMENT	Argumento inválido
PDH_INVALID_HANDLE	O contador não contém dados válidos

L i n h a d o p r o g r a m a :

```
if ( ERROR_SUCCESS != PdhGetCounterInfo( hCounter,        // Handle do contador
                                TRUE,                     // TRUE: retorna texto de explicação do contador
                                &cbCounterBuff,       // Tamanho do buffer. Retorna tamanho lido.
                                (PPDH_COUNTER_INFO)counterBuff) ) // Endereço do buffer
....
```

É criado um apontador com o tipo correto para a estrutura:
PPDH_COUNTER_INFO pCounterInfo = (PPDH_COUNTER_INFO)counterBuff;

O campo pCounterInfo->szExplainText contém a informação a ser exibida.

A estrutura PDH_COUNTER_INFO é mostrada abaixo:

```
typedef struct _PDH_COUNTER_INFO {
    DWORD   dwLength;  // Comprimento da estrutura
```

```
    DWORD   dwType;    // Tipo do contador
    DWORD   CVersion;  // Info de versão do contador
    DWORD   CStatus;   // Status corrente do contador
    LONG    lScale;    // Fator de escala corrente
    LONG    lDefaultScale;  // Fator de escala recomendado
    DWORD   dwUserData;     // Valor do campo user data
    DWORD   dwQueryUserData;// Valor do user data para a query
                            // à qual o contador pertence
    LPTSTR  szFullPath;     // Caminho completo do contador
    union   {
        PDH_DATA_ITEM_PATH_ELEMENTS DataItemPath;
        PDH_COUNTER_PATH_ELEMENTS CounterPath;
        struct {
            LPTSTR   szMachineName;
            LPTSTR   szObjectName;
            LPTSTR   szInstanceName;
            LPTSTR   szParentInstance;
            DWORD    dwInstanceIndex;
            LPTSTR   szCounterName;
        };
    };
    LPTSTR  szExplainText;  // Texto de explicação do contador
    DWORD   DataBuffer[1];  // Dado anexado à estrutura pelo user
} PDH_COUNTER_INFO, *PPDH_COUNTER_INFO;
```

Mostrando a hierarquia dos objetos monitorados pelo Performance Monitor – o programa PDHCounters

Este programa também foi extraído da referência [Pietrek May_1998].

O objetivo do programa é exibir uma árvore contendo todos os objetos monitorados, suas instâncias e contadores associados.
Alguns objetos monitorados têm apenas uma instância: memória, sistema, cache. Já processos, threads e processadores podem ter diversas instâncias. Se um objeto não tem instâncias, os contadores estão diretamente associados ao objeto. Se um objeto tem instâncias, os contadores estão associados às instâncias.
A dll PDH possui APIs que facilitam este trabalho.

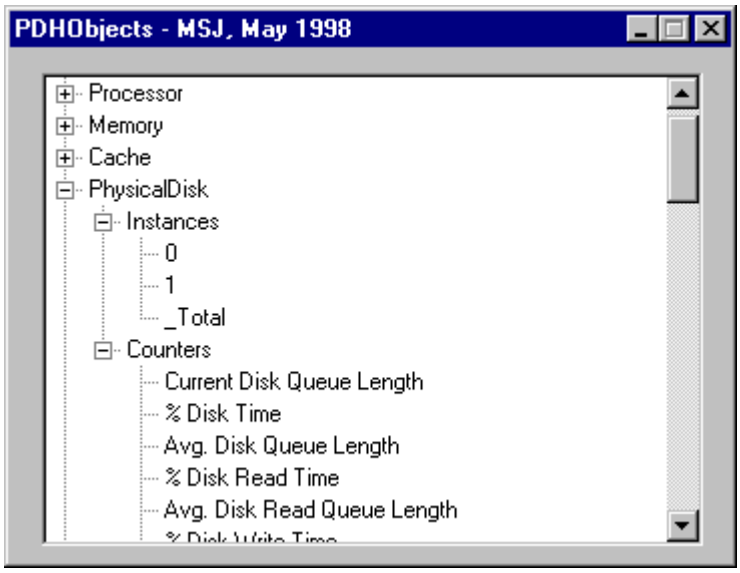


Figura 2 – Interface do programa PDHObjects

Código fonte do programa- PDHCounters

```
//================================================
// PDHObjects - Matt Pietrek 1998
// Microsoft Systems Journal, May 1998
// FILE: PDHOBJECTS.CPP
//================================================

#pragma comment(lib,"PDH.lib")
#pragma comment(lib,"COMCTL32.lib")

#define DUNICODE                1               // irá usar UNICODE
#define D_UNICODE               1
#define DWIN32_LEAN_AND_MEAN

#include <windows.h>
#include <stdio.h>
```

```
#include <COMMCTRL.H>
#include <tchar.h>
#pragma hdrstop
#include "pdh.h"
#include "PDHObjects.h"

// Protótipo das funções
void    Handle_WM_INITDIALOG(HWND hDlg);
void    Handle_WM_CLOSE( HWND hDlg );
BOOL    CALLBACK PDHObjectsDlgProc(HWND,UINT,WPARAM,LPARAM);
void    GetSetPositionInfoFromRegistry( BOOL fSave, POINT *lppt );
void    PopulateTree( HWND hWndTree );

HTREEITEM Add*Treeview*SubItem(   HWND hWndTree, HTREEITEM hTreeItem,
                        LPTSTR pszItemText );

void    AddObjectInstancesAndCounters(  HWND hWndTree, HTREEITEM hTreeItem,
                                LPTSTR pszObject );

// ======================== Strings fixos ==============================
TCHAR gszRegistryKey[]  = _T("Software\\WheatyProductions\\PDHObjects");

// ======================== Início do código ===========================

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                LPSTR lpszCmdLine, int nCmdShow )
{
   InitCommonControls();  // Registra e Incializa as classes de controle de janelas comuns

   // Exibe a interface de operação
   DialogBox(  hInstance, MAKEINTRESOURCE(IDD_PDHOBJECTS),
                0, (DLGPROC)PDHObjectsDlgProc );
   GetLastError();
   return 0;
}

BOOL CALLBACK PDHObjectsDlgProc(
                WND hDlg,UINT msg,WPARAM wParam,LPARAM lParam )
{
   //
   // Procedimentos de diálogo para a janela principal
   //
   switch ( msg )
   {
      case WM_INITDIALOG:
         Handle_WM_INITDIALOG( hDlg ); return TRUE;
      case WM_CLOSE:
         Handle_WM_CLOSE( hDlg ); break;
      // fecha tudo
   }
   return FALSE;
}

//=====================================================================
// Passeia através da lista de objetos, adicionando cada nome de objeto à raiz
// da árvore de visualização
//=====================================================================

void PopulateTree( HWND hWndTree )
{
```

```
   PDH_STATUS status;

   TCHAR mszObjects[2048];
   DWORD sizeObjects = sizeof(mszObjects) / sizeof(TCHAR);

   status = PdhEnumObjects(NULL,
                     NULL,
                     mszObjects,
                     &sizeObjects,
                     PERF_DETAIL_WIZARD,
                     TRUE );

   if ( ERROR_SUCCESS != status )
   {
      _tprintf( _T("PdhEnumObjects failed - code:%u"), status );
      return;
   }

   LPTSTR pszObject = mszObjects;

   while ( *pszObject )  // Para cada objeto faça:
   {
      // Acrescente o objeto à treeview
      HTREEITEM hTreeItem = AddTreeviewSubItem( hWndTree, NULL, pszObject );

     // Acrescente instâncias e contadores relacionados ao objeto à treeview
      AddObjectInstancesAndCounters( hWndTree, hTreeItem, pszObject );

      pszObject += lstrlen(pszObject) + 1;
   }
}

//=====================================================================
// Busca todas as instâncias e contadores de um objeto para um objeto nomeado,
// e adione-os como sub-itens na árvore de controle de visualização
//=====================================================================

void    AddObjectInstancesAndCounters(  HWND hWndTree, HTREEITEM hTreeItem,
                              LPTSTR pszObject )
{
   PDH_STATUS status;

   TCHAR mszCounterList[4096];
   DWORD cchCounterList = sizeof(mszCounterList) / sizeof(TCHAR);
   TCHAR mszInstanceList[4096];
   DWORD cchInstanceList = sizeof(mszInstanceList) / sizeof(TCHAR);

        status = PdhEnumObjectItems(
              NULL,                  // Fonte de dados: NULL = tempo real
              NULL,                  // Máquina: NULL = máquina local
              pszObject,             // Objeto especificado
              mszCounterList,        // @Buffer para receber os contadores
              &cchCounterList,       // Tamanho do buffer em bytes
              mszInstanceList,       // @Buffer para receber lista de instâncias
              &cchInstanceList,      // Tamanho do buffer em bytes
              PERF_DETAIL_WIZARD,   // Nível de detalhe
              0 );


   if ( ERROR_SUCCESS != status )
```

```
      return;

   if ( cchInstanceList > 2 )
   {
      // Introduz o string "Instances" à treeview associada ao item
      // e obtém um handle para o nível Instances
      HTREEITEM hTreeInstances = AddTreeviewSubItem( hWndTree, hTreeItem,
                                          _T("Instances") );
      LPTSTR pszInstance = mszInstanceList;

      while ( *pszInstance ) // Para cada instância:
      {
         // Adicione o nome da instância à treeview debaixo do nível Instances
         AddTreeviewSubItem( hWndTree, hTreeInstances, pszInstance );

         pszInstance += lstrlen(pszInstance) + 1;
      }
   }

   if ( cchCounterList > 2 )
   {
      // Introduz o string "Counters" à treeview associada ao item
      // e obtém um handle para o nivel Counters
      HTREEITEM hTreeCounters = AddTreeviewSubItem( hWndTree, hTreeItem,
                                          _T("Counters") );

      LPTSTR pszCounter = mszCounterList;

      while ( *pszCounter )  // Para cada contador:
      {
         // Adicione o nome do contador à treeview debaixo do nível Counters
         AddTreeviewSubItem(hWndTree, hTreeCounters, pszCounter);

         pszCounter += lstrlen(pszCounter) + 1;
      }
   }
}

void Handle_WM_INITDIALOG(HWND hDlg)
{
   // Busca as coordenadas da janela, onde rodou por último,
   // e move a janela para o local
   POINT pt;

   GetSetPositionInfoFromRegistry( FALSE, &pt ); // Realiza GET
   SetWindowPos(hDlg, 0, pt.x, pt.y, 0, 0,
            SWP_NOSIZE | SWP_NOREDRAW | SWP_NOZORDER | SWP_NOACTIVATE);

   PopulateTree( GetDlgItem(hDlg, IDC_TREE1) );
}

void Handle_WM_CLOSE( HWND hDlg )
{
   // Salva as coordenadas X,Y da janela para a próxima vez
   RECT rect;

   if ( GetWindowRect( hDlg, &rect ) )
      GetSetPositionInfoFromRegistry( TRUE, (LPPOINT)&rect ); // Realiza SET
   EndDialog(hDlg, 0);
}
```

```
void GetSetPositionInfoFromRegistry(BOOL fSave, POINT *lppt )
{
   //
   // Função que salva ou restaura as coordenadas da caixa de diálogo
   // no registry do sistema. Trata o caso em que não há nda lá.
   //
   HKEY hKey;
   DWORD dataSize, err, disposition;
   TCHAR szKeyName[] = _T("DlgCoordinates");

   if ( !fSave )              // No caso de não haver nada lá ainda
      lppt->x = lppt->y = 0;  // retorna 0,0 para as coordenadas

   // Abre a chave para a registry (ou cria uma se for usar pela primeira vez)
   err = RegCreateKeyEx( HKEY_CURRENT_USER, gszRegistryKey, 0, 0,
                  REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS,
                  0, &hKey, &disposition );
   if ( ERROR_SUCCESS != err )
      return;

   if ( fSave )           // Salva as coordenadas
   {
      RegSetValueEx(hKey,szKeyName, 0, REG_BINARY,(PBYTE)lppt,sizeof(*lppt));
   }
   else                   // Le as coordenadas
   {
      dataSize = sizeof(*lppt);
      Reg*Query*ValueEx( hKey, szKeyName, 0, 0, (PBYTE)lppt, &dataSize );
   }

   RegCloseKey( hKey );
}


HTREEITEM Add*Treeview*SubItem(HWND hWndTree, HTREEITEM hTreeItem, LPTSTR
pszItemText)
{
   TVINSERTSTRUCT tvi;  // *Treeview* insert structure

   tvi.hParent = hTreeItem;          // Handle para parent item. Se NULL insere na raiz
   tvi.hInsertAfter = TVI_LAST;      // Posição de inserção na lista: no fim.
                                     // Outros valores: TVI_FIRST, TVI_ROOT, TVI_SORT
   tvi.item.mask = TVIF_TEXT;        // Os membros da estrutura que descrevem texto são válidos
   tvi.item.pszText = pszItemText; // Texto a ser inserido
   tvi.item.cchTextMax = lstrlen( pszItemText );  // Tamanho do texto

   return *Treeview*_InsertItem(hWndTree, &tvi );  // insere novo item no controle *treeview*
}
```

WinMain irá exibir uma janela contendo a interface de operação que foi definida no arquivo PDHObjects.rc.

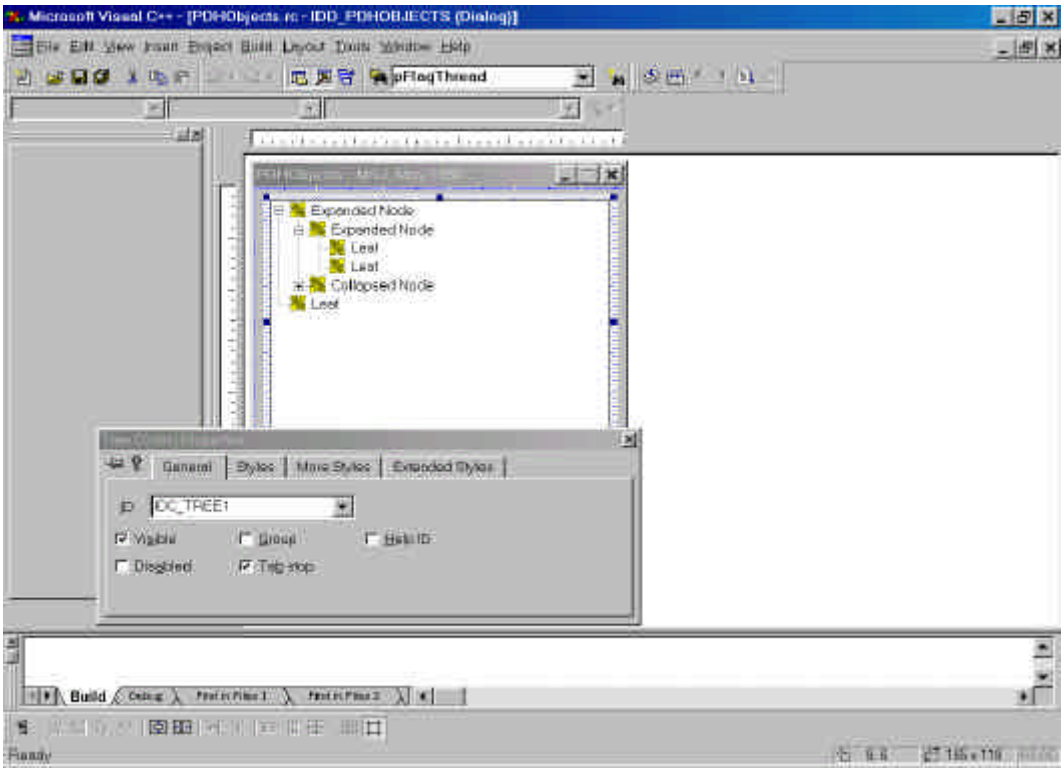


Figura 3 – Arquivo de recursos mostrando janela controle *treeview*

Logo após exibir a janela, a função PopulateTree é chamada para popular o controle *treeview* definido na janela.

A API **PdhEnumObjects** é a primeira que devemos conhecer para entender o código do programa. Ela enumera todos os objetos disponíveis para dados de performance. A função irá encher um buffer com diversos strings terminados em '\0' concatenados. Cada string corresponde ao nome de um objeto. A lista terminada por um string nulo isto é dois caracteres NULL consecutivos.

P d h E n u m O b j e c t s

```
PDH_STATUS PdhEnumObjects(

IN LPCTSTR szDataSource,          // NULL para dados de tempo real
IN LPCTSTR szMachineName,         // Nome da máquina
IN LPTSTR mszObjectList,          // Buffer para receber lista de objetos
IN LPDWORD pcchBufferLength,      // Tamanho do buffer
IN DWORD dwDetailLevel,           // Nível de expertise/detalhe
IN BOOL bRefresh                  // Refresca lista
);
```

Comentários sobre os parâmetros:

szDataSource	WNT: deve ser NULL Win 2000: nome do arquivo de log ou NULL. Se o log file for fornecido, os dados de performance serão retirados deste arquivo. Se NULL os dados serão coletados de uma fonte de tempo real.
szMachineName	Nome da máquina alvo na rede
mszObjectList,	Buffer alocado pela função que chama esta função para receber a lista de objetos MULTI_SZ da máquina específica.
pcchBufferLength,	Apontador para uma DWORD contendo o tamanho em bytes do buffer na chamada da função e a quantidade de dados retornados na saída. O valor de contagem deve incluir o caracter NULL de terminação. Logo 2 significa lista vazia.
dwDetailLevel,	Nível de detalhe dos itens de performance a serem retornados. Todos os itens que estiverem no item de detalhe especificado ou inferior serão retornados. Pode ser: PERF_DETAIL_NOVICE, PERF_DETAIL_ADVANCED, PERF_DETAIL_EXPERT, PERF_DETAIL_WIZARD.
bRefresh	TRUE: Indica se uma nova lista de objetos será obtida para a máquina em questão ou FALSE se será usada a lista amostrada anteriormente.

Retorno da função:

Status	**Interpretação**
ERROR_SUCCESS	Sucesso
Erros relacionados em PDHMSG.H	Erro

Linha do programa:

```
status = PdhEnumObjects(
            NULL,           // Real Time
            NULL,           // Máquina local
            mszObjects,
            &sizeObjects,
            PERF_DETAIL_WIZARD,  // Eu posso tudo …
            TRUE );
```

A função Add*Treeview*SubItem irá adicionar cada nome de objeto à árvore hWndTree e em seguida a função AddObjectInstancesAndCounters irá adicionar todas as instâncias dos objetos e contadores à árvore de visualização.

Trecho do programa:

```
HTREEITEM hTreeItem = Add Treeview SubItem( hWndTree, NULL, pszObject );

LPTSTR pszObject = mszObjects;

AddObjectInstancesAndCounters( hWndTree, hTreeItem, pszObject );
```

Esta última função utiliza **PdhEnumObjectItems** para uma vez especificado um dado objeto, retornar todas as instâncias deste objeto e todos os contadores associados ao objeto ou a suas instâncias conforme apropriado. Tanto a lista de instâncias como a lista de contadores utilizam a mesma formatação explicada para a lista de objetos.

Trecho do programa:

```
TCHAR mszCounterList[4096];
DWORD cchCounterList = sizeof(mszCounterList) / sizeof(TCHAR);
TCHAR mszInstanceList[4096];
DWORD cchInstanceList = sizeof(mszInstanceList) / sizeof(TCHAR);

status = PdhEnumObjectItems(
            NULL,                   // Fonte de dados: NULL = tempo real
            NULL,                   // Máquina: NULL = máquina local
            pszObject,              // Objeto especificado
            mszCounterList,         // @Buffer para receber os contadores
            &cchCounterList,        // Tamanho do buffer em bytes
            mszInstanceList,        // @Buffer para receber lista de instâncias
            &cchInstanceList,       // Tamanho do buffer em bytes
            PERF_DETAIL_WIZARD,   // Nível de detalhe
            0 );
```

Em seguida a palavra “Instances” será acrescentada à *treeview* e depois para cada instância o seu nome será listado debaixo do nodo *instances*.

```
if ( cchInstanceList > 2 )
  {
      // Introduz o string "Instances" à treeview associada ao item
      // e obtém um handle para o nível Instances
      HTREEITEM hTreeInstances = AddTreeviewSubItem( hWndTree, hTreeItem,
                                          _T("Instances") );
      LPTSTR pszInstance = mszInstanceList;

      while ( *pszInstance ) // Para cada instância:
      {
          // Adicione o nome da instância à treeview debaixo do nível Instances
          AddTreeviewSubItem( hWndTree, hTreeInstances, pszInstance );

          pszInstance += lstrlen(pszInstance) + 1;
       }
   }
```

Ao chamar **PdhEnumObjectItems**, passando *Process* como objeto, a lista de processos será obtida. Em seguida use a função **PdhMakeCounterPath** da API para buscar um contador para cada processo, por exemplo o contador IDProcess (identificador do processo) e todas as informações poderão ser obtidas em tempo real.

Desenvolvendo uma aplicação em Delphi

Vamos usar como exemplo a aplicação criada por Mark Smith na referência [Smith 2000].

O autor criou um componente no módulo *PerfDataComp.pas* que deve ser carregado antes de se compilar o programa principal *DemoMain*. No Menu *Component* do Delphi, selecione: *Install Component* e preencha os três campos da janela com os valores pertinentes.

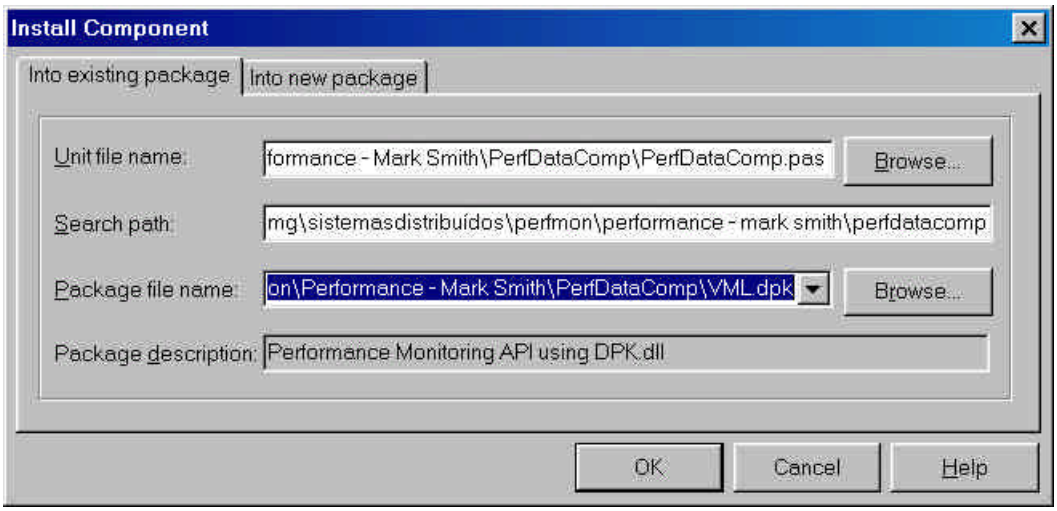


Figura 4 – Janela de instalação do componente

Este componente é compilado e guardado no package *VML.dpk*.

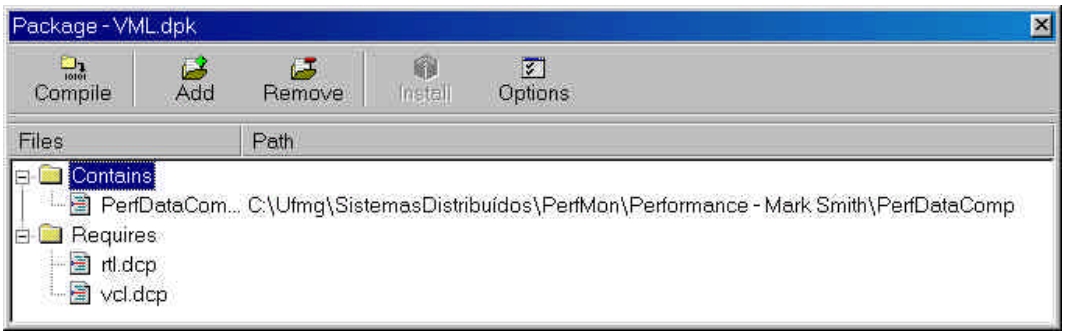


Figura 5 – Janela de compilação do componente

Agora toda vez que você examinar a *component palette* irá ver uma *page tab* correspondente ao componente instalado.

Figura 6 – Palheta de componentes do Delphi com aba *VML*

Se você clicar com o botão da direita vezes sobre a palheta e escolher a opção propriedades, poderá visualizar as propriedades do *package* criado e de seus componentes.

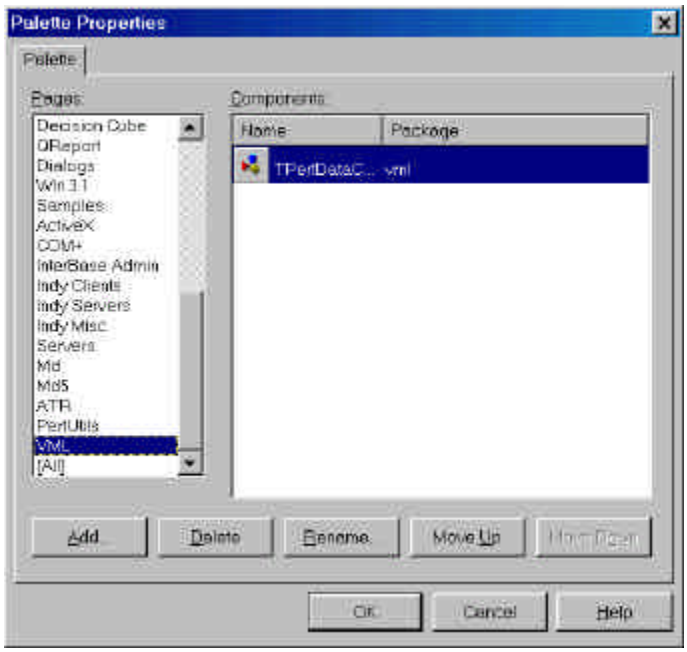


Figura 7 – Janela de propriedades da palheta de componentes

Estudando o código

O arquivo *Pdh.pas* é uma tradução para o Pascal do arquivo pdh.h. Este arquivo também contém o conteúdo de PdhMsg.h que contém as definições e valores de retorno das funções utilizadas.

O programa de exemplo demonstra o uso da library PDH para monitorar sua própria performance.

As principais funções em Pdh.pas são:

P D H C h e c k

```
procedure PDHCheck (ReturnValue: PDH_FUNCTION);
```

Checa o código de retorno de uma função PDH e gera uma exceção em caso de erro.

```
procedure PDHCheck (ReturnValue: PDH_FUNCTION);
begin
  case ReturnValue of
   PDH_CSTATUS_VALID_DATA          :;
   PDH_CSTATUS_NEW_DATA            :;
   PDH_MORE_DATA                   :;
   PDH_DIALOG_CANCELLED            :;
   PDH_CSTATUS_NO_MACHINE,
   PDH_CSTATUS_NO_INSTANCE,
   PDH_CSTATUS_ITEM_NOT_VALIDATED,
   PDH_RETRY,
   PDH_NO_DATA,
   PDH_CALC_NEGATIVE_DENOMINATOR,
   PDH_CALC_NEGATIVE_TIMEBASE,
   PDH_CALC_NEGATIVE_VALUE,
   PDH_CSTATUS_NO_OBJECT,
   PDH_CSTATUS_NO_COUNTER,
   PDH_CSTATUS_INVALID_DATA,
   PDH_MEMORY_ALLOCATION_FAILURE,
   PDH_INVALID_HANDLE,
   PDH_INVALID_ARGUMENT,
   PDH_FUNCTION_NOT_FOUND,
   PDH_CSTATUS_NO_COUNTERNAME,
   PDH_CSTATUS_BAD_COUNTERNAME,
   PDH_INVALID_BUFFER,
   PDH_INSUFFICIENT_BUFFER,
   PDH_CANNOT_CONNECT_MACHINE,
   PDH_INVALID_PATH,
   PDH_INVALID_INSTANCE,
   PDH_INVALID_DATA,
   PDH_NO_DIALOG_DATA,
   PDH_CANNOT_READ_NAME_STRINGS : PDHError(PDHErrorMessage(ReturnValue));
  end;
end;
```

PdhOpenQuery

Abre uma *query*

```
PdhOpenQuery :
        function (pReserved: LPVOID; // Data Source: NULL no NT e tempo real
        dwUserData:DWORD;     // Dados a serem associados à query
        var phQuery: HQUERY):    // Handle da thread
        PDH_FUNCTION cdecl  {$IFDEF WIN32} stdcall {$ENDIF};
```

PdhAddCounter

Adiciona um contador a uma *query*

```
PdhAddCounterA :
        function (hQuery: HQUERY;  // Handle da query
        szFullCounterPath: LPCSTR;  Path para contador a ser criado
        dwUserData: DWORD;    // Definido pelo usuário e recup. Por PdhGetCounterInfo
        var phCounter: HCOUNTER): // Endereço do Buffer para receber handle do contador
        PDH_FUNCTION cdecl  {$IFDEF WIN32} stdcall {$ENDIF};
```

PdhCollectQueryData

Realiza leitura de todos os contadores da *query*

```
PdhCollectQueryData:
        function (hQuery: HQUERY):  // Handle para query
        PDH_FUNCTION cdecl  {$IFDEF WIN32} stdcall {$ENDIF};
```

PdhGetRawCounterValue

Realiza leitura do valor bruto de um contador

```
PdhGetRawCounterValue :
        function (hCounter: HCOUNTER;  // Handle do contador
        lpdwType: LPDWORD; // Apontador para buffer para receber tipo do dado
        pValue: PPDH_RAW_COUNTER): // Pointer para buffer para receber valor do contador
        PDH_FUNCTION cdecl  {$IFDEF WIN32} stdcall {$ENDIF};
```

PdhGetFormattedCounterValue

Realiza leitura do valor formatado de um contador

```
PdhGetFormattedCounterValue:
        function (hCounter: HCOUNTER; // Handle do contador
        dwFormat: DWORD;         // Informação de formato
        lpdwType: LPDWORD;    // Apontador para buffer para receber
         // tipo do dado: PDH_FMT_DOUBLE, PDH_FMT_LARGE, PDH_FMT_LONG
        var pValue: _PDH_FMT_COUNTERVALUE): // Buffer para receber valor do contador
        PDH_FUNCTION cdecl  {$IFDEF WIN32} stdcall {$ENDIF};
```

P d h B r o w s e C o u n t e r s

Configura o *browser* de objetos da dll para escolha de contadores.

```
PdhBrowseCountersA:
        function (pBrowseDlgData: PPDH_BROWSE_DLG_CONFIG_A):
        PDH_FUNCTION cdecl  {$IFDEF WIN32} stdcall {$ENDIF};
```

Existem três classes principais que englobam as principais funcionalidades do componente.

TperfCounter define um contador, como um item de uma Collection.
TperfQuery é uma *query* montada para conter diversos contadores. Cada contador (objeto *TperfCounter*) tem uma propriedade que aponta para a coleção à qual pertence (*TperfQuery*). Cada contador possui um *handle* que o identifica e um formato explicado no comando *PdhGetFormattedCounterValue*.

A classe **TperfQuery** define uma *query* e engloba uma coleção de contadores. O método AddQuery serve para adicionar contadores à *query*.

A classe **TperfDataComp** define um componente que tem associado a ele uma *query*, uma lista de strings de saída e todas as funções para realizar a escolha e leitura de contadores.

```
type

  TPerfCounter = class (TCollectionItem)
  // Define um contador como item de uma collection
  // Possui uma propriedade: collection que aponta para a Collection
  // a qual pertence
  public
    hCounter: Cardinal; // Handle para contador
    Format: DWord; // Formato do contador:  PdhGetFormattedCounterValue
    Name: String;  // Nome do contador: é retornado pelo object browser
  end;


  TPerfQuery = class (TCollection)
  // A query é uma coleção de descrição de contadores
  // Add e Clear são usados para adicionar e retirar itens da coleção
  public
    Query : DWORD;     // Handle para a query retornado por PdhOpenQuery
    UserData : DWORD;  //
    // Adiciona um contador à query. Tipos:
    // inteiro de 32-bits   (PDH_FMT_LONG)
    // inteiro de 64 bits    (PDH_FMT_LARGE)
    // double                (PDH_FMT_DOUBLE).
    function AddCounter(AName:ansiString; AFormat: DWORD) : TPerfCounter;
    destructor Destroy; override;
  end;


  TPerfDataComp = class(TComponent)
  private
    FPerfQuery: TPerfQuery; // Salva handle para a query
    FUserData : Cardinal;
    // Dados do usuário a serem associados à query em PdhOpenQuery
```

```
    FBrowseDlgData: PDH_BROWSE_DLG_CONFIG_A;
    // Dados para configurar o Browser de contadores
    FActive : Boolean;      // Estado: TRUE = ATIVO
    FOutPut : TStrings;     // Lista de strings de saída para exibição
    function GetActive: boolean;// Le estado do componente: ativo ou não
    procedure SetActive(const Value: boolean);
    // Define estado do componente como ativo (TRUE) ou não
  protected
    function NowStr : String;   // Retorna data e hora correntes
    procedure RecordMsg (Msg  :string);
    // Salva mensagem na lista de saída
  public
    procedure Update;         // Atualiza a query
    procedure BrowseCounters; // Exibe diálogo para escolha de contadores
  published
    // Lista de strings de saída
    property OutPut : TStrings read FOutPut write FOutPut;
    // Flag: componente ativo
    property Active : boolean read GetActive write SetActive;
    // Realiza uma query
    property PerfQuery : TPerfQuery read FPerfQuery write FPerfQuery;

    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  end;
```

Programa Fonte Completo

```
unit PerfDataComp;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
     Dialogs,ExtCtrls,PDH;

type

  TPerfCounter = class (TCollectionItem)
  // Define um contador como item de uma collection
  // Possui uma propriedade: collection que aponta para a Collection
  // à qual pertence
  public
    hCounter: Cardinal; // Handle para contador
    Format: DWord; // Formato do contador:  PdhGetFormattedCounterValue
    Name: String;  // Nome do contador: é retornado pelo object browser
  end;


  TPerfQuery = class (TCollection)
  // A query é uma coleção de descrição de contadores
  // Add e Clear são usados para adicionar e retirar itens da coleção
  public
    Query : DWORD;     // Handle para a query retornado por PdhOpenQuery
    UserData : DWORD;  //
    // Adiciona um contador à query. Tipos:
    // inteiro de 32-bits   (PDH_FMT_LONG)
    // inteiro de 64 bits    (PDH_FMT_LARGE)
    // double                (PDH_FMT_DOUBLE).
    function AddCounter(AName:ansiString; AFormat: DWORD) : TPerfCounter;
    destructor Destroy; override;
  end;

  TPerfDataComp = class(TComponent)
```

```
  private
    FPerfQuery: TPerfQuery; // Salva handle para a query
    FUserData : Cardinal;
    // Dados do usuário a serem associados à query em PdhOpenQuery
    FBrowseDlgData: PDH_BROWSE_DLG_CONFIG_A;
    // Dados para configurar o Browser de contadores
    FActive : Boolean;      // Estado: TRUE = ATIVO
    FOutPut : TStrings;     // Lista de strings de saída para exibição
    function GetActive: boolean;// Le estado do componente: ativo ou não
    procedure SetActive(const Value: boolean);
    // Seta estado do componente como ativo (TRUE) ou não
  protected
    function NowStr : String;   // Retorna data e hora correntes
    procedure RecordMsg (Msg  :string);
    // Salva mensagem na lista de saída
  public
    procedure Update;         // Atualiza a query
    procedure BrowseCounters; // Exibe diálogo para escolha de contadores
  published
    // Lista de strings de saída
    property OutPut : TStrings read FOutPut write FOutPut;
    // Flag: componente ativo
    property Active : boolean read GetActive write SetActive;
    // Realiza uma query
    property PerfQuery : TPerfQuery read FPerfQuery write FPerfQuery;

    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  end;


procedure Register;

implementation

// '\\MARK\Process(Project1)\% Processor Time'

procedure Register;   // Registra componente e o associa à aba VML
begin
    RegisterComponents('VML', [TPerfDataComp]);
end;


{ TPerfData ================================================}

constructor TPerfDataComp.Create(AOwner: TComponent);
begin
    inherited;
    // Cria a query como uma coleção de counters (TPerfCounter)
    FPerfQuery := TPerfQuery.Create(TPerfCounter);
    // Abre a query e testa retorno. Seta exceção se há algo errado
    // Retorna handle para a query no último parâmetro
    PDHCheck(PdhOpenQuery (nil, FUserData, FPerfQuery.Query));
end;

destructor TPerfDataComp.Destroy;
begin
    FPerfQuery.Free;  // Por que libera antes de fechar a query ???
    PDHCheck(PdhCloseQuery(FPerfQuery.Query));
    // Encerra a query e testa retorno
    inherited;
end;

// Retorna data e hora correntes no formato desejado
```

```
function TPerfDataComp.NowStr: String;
begin
    Result := FormatDateTime ('yyyy,mm,dd hh:mm:ss:ss', now);
end;

// Adiciona um string ao final da lista de saída no formato:
// NomeDoComponente: Data:Hora Mensagem
// O Nome do componente será definido pelo usuário
// PerfMon: 1999,9,13:12:20:20 Process[DemoPrj%ProcessorTime: 1.00
procedure TPerfDataComp.RecordMsg(Msg: string);
begin
    OutPut.Add(Self.Name + ': ' + NowStr +' ' + Msg);
end;

// Retorna estado do componente se ativo ou não
function TPerfDataComp.GetActive: boolean;
begin
    Result := FActive;
end;

// Define estado do componente conforme valor do parâmetro
// O campo OutPut deve ser definido antes da função ser chamada
procedure TPerfDataComp.SetActive(const Value: boolean);
begin
    // Se componente já estiver carregado e Output indefinido ...
    if (not(csLoading in componentState)) and (FOutPut = nil)
    then raise Exception.Create('Need to set OutPut before activating');
    OutPut.Add (Self.Name + ': Activated at ' + NowStr);
    FActive := Value;
end;

procedure TPerfDataComp.Update;
var
    i : integer;
    RawValue : PDH_RAW_COUNTER;
    // Estrutura para receber valor bruto do contador
    RawVal : PPDH_RAW_COUNTER;  // Apontador para PDH_RAW_COUNTER
    FormattedValue : PDH_FMT_COUNTERVALUE;
    // Estrutura para receber valor formatado do contador
    PerfCounter : TPerfCounter;
begin
    // Realiza leitura dos contadores
    PDHCheck(PdhCollectQueryData(FPerfQuery.Query));
    RawVal := @RawValue; // RawValue contém o endereço de RawValue

    for i := 0 to FPerfQuery.Count -1 do
    // Para todos os contadores da query faça
    begin
        PerfCounter := FPerfQuery.Items[i] as TPerfCounter;
        // Le valor bruto do contador para RawValue
        PDHCheck(PdhGetRawCounterValue(PerfCounter.hCounter, @FUserData, RawVal));
        // Le valor formatado da variável para FormattedValue
        PDHCheck(PdhGetFormattedCounterValue(PerfCounter.hCounter,
        PerfCounter.Format, @FUserData, FormattedValue));
        PDHCheck(FormattedValue.CStatus);
        // Imprime valor formatado
        // Valor bruto foi buscado, mas não foi impresso
        case PerfCounter.Format of
         PDH_FMT_LONG: RecordMsg(PerfCounter.Name + ': ' +
                        IntToStr(FormattedValue.longValue));
```

```
        PDH_FMT_DOUBLE:RecordMsg(PerfCounter.Name + ': ' +
                        floatToStrF(FormattedValue.doubleValue, ffFixed, 9, 2));
    end;
  end;
end;


// Função callback associada ao browser de contadores
// Para cada contador definido, esta função será chamada
function PDHCallBack1 (Arg : DWORD) : PDH_STATUS;  stdcall;
var
  ThePerfDataComp : TPerfDataComp;
begin
     ThePerfDataComp := TPerfDataComp(Pointer(Arg));
     PDHCheck (ThePerfDataComp.FBrowseDlgData.CallBackStatus);
      // testa status de retorno do browser
     // Adiciona um novo contador. Tipo = DOUBLE
     // O string retornado pelo browser é o nome do contador

     ThePerfDataComp.PerfQuery.AddCounter(ThePerfDataComp.FBrowseDlgData.szReturnPath
     Buffer, PDH_FMT_DOUBLE);
     Result := ERROR_SUCCESS;
end;

procedure TPerfDataComp.BrowseCounters;
var
  ReturnPath: shortstring;
begin
     SetLength (ReturnPath, 255);
     // Prepara estrutura PDH_BROWSE_DLG_CONFIG
     FBrowseDlgData.szReturnPathBuffer := @ReturnPath;
     FBrowseDlgData.cchReturnPathLength := 255;
     // Define arg p. função callback: ponteiro para objeto TPerfData
     FBrowseDlgData.dwCallBackArg := Integer(pointer(self));
     // Define endereço da função callback
     FBrowseDlgData.pCallBack := @PDHCallBack1;
     // Desabilita seleção de outra máquina da rede
     FBrowseDlgData.flags := [bHideDetailBox, bDisableMachineSelection];
     // Define nível de detalhe do diálogo:
     FBrowseDlgData.dwDefaultDetailLevel := 3;  // PERF_DETAIL_WIZARD
     FBrowseDlgData.hWndOwner := Application.Handle;
     FBrowseDlgData.szDialogBoxCaption := 'Escolha um contador';
     // Título da janela
     // Chama janela de diálogo
     PDHCheck(PdhBrowseCountersA (@FBrowseDlgData));
end;

{TCounters=====================================================}

function TPerfQuery.AddCounter (AName: AnsiString; AFormat : DWORD): TPerfCounter;
var
  lCounter: HCOUNTER;  // Recebe handle para o contador
begin
     Result := Add as TPerfCounter;
     Result.Name := AName;   // Nome do contador definido no browser
     PDHCheck(PdhAddCounterA(Query,Pchar(Aname),UserData, lCounter));
     Result.hCounter := lCounter;  // Salva handle retornado
     Result.Format := AFormat;
end;
```

```
 // Remove todos os contadores associados à query e termina
destructor TPerfQuery.Destroy;
var
      i : integer;
     Counter : TPerfCounter;
begin
     for i := 0 to Count-1 do
     // Para todos os contadores associados à query faça
     begin
          Counter := Items[i] as TPerfCounter;
          PDHCheck(PdhRemoveCounter (Counter.hCounter));
     end;
     inherited;
end;

end.
```

Assim que o componente é instanciado ele abre uma query através da instrução *PdhOpenQuery*. Esta função retorna um handle para a query que é salva no atributo *query* no objeto FPerfQuery da classe *TperfQuery* pertencente ao componente *TPerfDataComp*. Cada item do objeto *TperfQuery* corresponde a um objeto da classe contador *(TperfCounter).*
Em seguida a função *BrowseCounters* deve ser chamada para exibir a janela de seleção de objetos e contadores, já discutida nesta apostila. Toda vez que o usuário escolher um contador, a função *callback* PDHCallBack1 será chamada e acrescentará um contador à query. Isto é feito pela função *PdhAddCounter*.
O usuário do componente também pode definir ele próprio os contadores que deseja ler através do método *TPerfQuery.AddCounter*. É desta forma que será utilizado no programa principal.

A função *Update* é o coração do componente. Ela inicialmente usa a função PdhCollectQueryData para ler os valores de todos os contadores. Depois lê os valores brutos dos contadores através da função *PdhGetRawCounterValue* e depois os valores formatados através da função *PdhGetFormattedCounterValue.*
Este comando irá encher uma estrutura do tipo PDH_FMT_COUNTERVALUE com os valores retornados. O foramto de dados PDH_FMT_DOUBLE foi o escolhido por ser o mais compatível com qualquer tipo de dados.
Finalmente um string contendo a data e hora da leitura e o valor formatado do registro é gravado numa lista de strings da variável Foutput do próprio componente.

Programa Aplicativo

A aplicação é muito simples.

Inicialmente é criado um formulário contendo um objeto do tipo *Tmemo* que corresponde a um conjunto de strings para visualização de resultados, um *timer* que irá gerar um evento a cada 1s, um botão para chamar o menu do *browser* que permitirá escolher contadores dos objetos do sistema e um *check box* para ativar ou desativar o timer.

A janela resultante é mostrada na Figura 8.

Figura 8 – **Janela do aplicativo durante configuração**

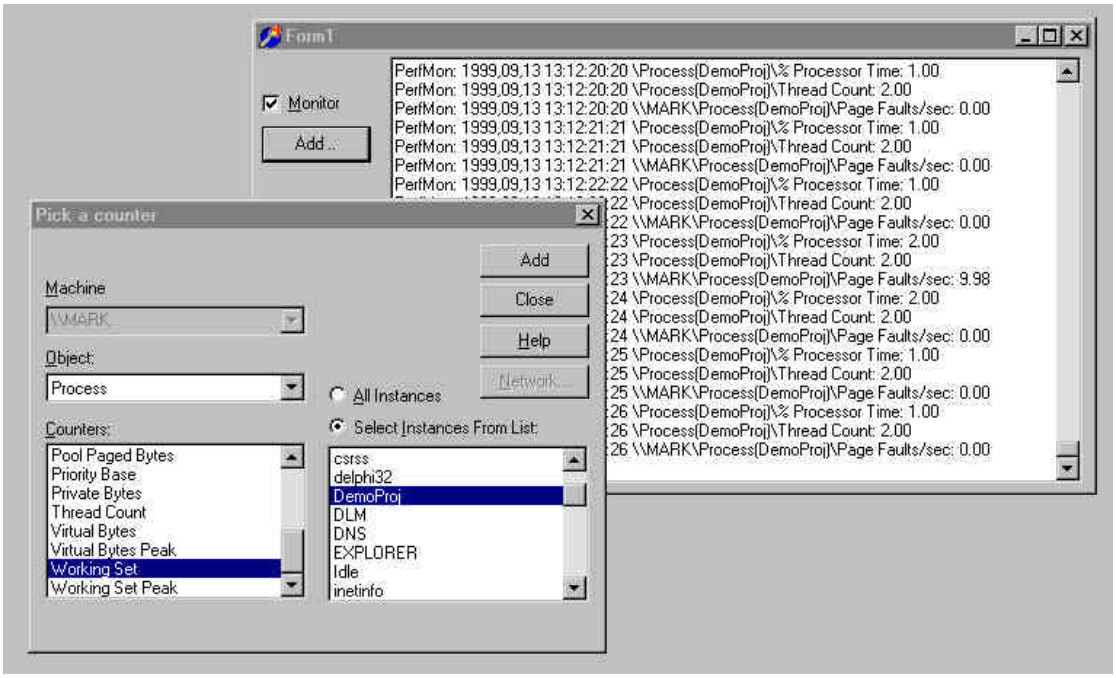


Figura 9 – Janela da aplicação, mostrando janela do object browser

Programa fonte

```
unit DemoMain;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  PerfDataComp, StdCtrls, ExtCtrls, CheckLst, ComCtrls;

type
  TForm1 = class(TForm)
    Memo1: TMemo;           // Exibe resultados
    Timer1: TTimer;         // Gera intervalo de refresh de tela
    CheckBox1: TCheckBox;   // Habilita e desabilita o timer
    Button1: TButton;       // Chama menu para exibir objetos e contadores
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure CheckBox1Click(Sender: TObject);
  private
    FPerfMonitor: TPerfDataComp;  // Componente para ler valores de Performance
    { Private declarations }
  public
    { Public declarations }
    property PerfMonitor : TPerfDataComp read FPerfMonitor write FPerfMonitor;
  end;

var
  Form1: TForm1;

implementation

uses
  PDH;

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var
  PerfCounter : TPerfCounter; // Define uma estrutura contador
begin
     FPerfMonitor := TPerfDataComp.Create (Self);
     // Cria instância do "componente"
     FPerfMonitor.Name := 'PerfMon';
     // Define nome para o "componente"

     // Define uma estrutura do tipo Tstrings para receber saída
     FPerfMonitor.OutPut := Memo1.Lines;

     Memo1.Lines.Add(Application.Name);// Linha 1: Nome do aplicativo
     // Seleciona os contadores a serem lidos através de seu path
     PerfCounter := FPerfMonitor.PerfQuery.AddCounter
                 ('\Process(DemoProj)\% Processor Time', PDH_FMT_DOUBLE);
     PerfCounter := FPerfMonitor.PerfQuery.AddCounter
                 ('\Process(DemoProj)\Thread Count', PDH_FMT_DOUBLE);
     PerfCounter := FPerfMonitor.PerfQuery.AddCounter
                 ('\\MARK\Process(DemoProj)\Page Faults/sec', PDH_FMT_DOUBLE);
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
// Permite selecionar novos contadores
begin
    FPerfMonitor.BrowseCounters;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
// Update a cada Timer1.Interval (1000 ms)
begin
    // Chama a query a cada 1s. Os resultados serão impressos na janela pelo componente
    FPerfMonitor.Update;
end;

procedure TForm1.CheckBox1Click(Sender: TObject);
// Habilita ou desabilita o timer
begin
    Timer1.Enabled := CheckBox1.Checked;
end;


end.
```

São escolhidos três componentes através de seus *pathes* completos, formado pelo nome do objeto seguido do nome do contador:

'\Process(DemoProj)\% Processor Time',
'\Process(DemoProj)\Thread Count',
'\\MARK\Process(DemoProj)\Page Faults/sec'

Obs: // MARK é o nome da máquina

Estes contadores correspondem ao Processor Time, Thread Count e Page Faults/sec.

Outros componentes em Delphi

Alexey Dynnikov construiu uma série de aplicações utilizando um componente denominado PerfUtils. Nesta seção vamos apresentar quatro exemplos construídos à partir de componentes da suite PerfUtils.

São fornecidos sete componentes:

Componente	Descrição
TPerfTitles	Coleta títulos de todos os objetos de performance e contadores do sistema.
TPerfHelps	Componente análogo a TperfTitles, mas que lê strings de help.
TPerfObjects	Componente projetado para coletar informação sobre objetos de performance disponíveis para monitoração.
TPerfCounters	Coleta informação sobre contadores de performance disponíveis para monitoração.
TPerfInstances	Coleta informação sobre todas as instâncias de objetos de performance especificado em seus parâmetros. É a melhor maneira de obter uma lista completa de processos, threads, discos e outros objetos do sistema.
TPerfFilter	Componente facilmente configurável que coleta informação de performance em valor bruto.
TPerfMonitor	Retorna valores de contadores pré processados para objetos de performance de múltiplas instâncias. Na verdade trata-se de um kernel de monitoração de performance em um único componente.

Figura 10 – Componentes de acesso a objetos de performance da suite PerfUtils

O problema é que ao invés de utilizar a pdh.dll, o autor proferiu rrealizar as queries diretamente, o que trouxe maior complexidade aos seus componentes. Apenas o módulo WinPerf.h foi traduzido do c para pascal. Devido ao tamanho e complexidade deste código, o seu estudo não será feito neste texto, mas é deixado como exemplo para o estudante interessado em programar diretamente a monitoração de performance sem o uso da Pdh.dll.

Estes programas de exemplo devem ser compoilados pois são de grande utilidade para o estudo da monitoração de performance no WNT.

Módulo WinPerf

O programa WinPerf.pas corresponde ao arquivo WinPerf.h traduzido para a linguagem pascal por Alexey Dynnikov.

Instalação dos componentes PerfUtils

Para instalar os componentes para Delphi 6 siga os seguintes passos:

1. No Menu *Component* do Delphi, selecione: *Install Component*.

2. Preencha a janela *Install Component* com os seguintes dados:

Unit file name:	*C:\PerfMon\PerfUtilsD6Src - Alexey Dynnikov\PerfUtils D6\RunTime\PerfCounters.pas*
Search path:	*$(DELPHI)\Lib; $(DELPHI)\Bin; $(DELPHI)\Imports; $(DELPHI)\Projects\Bpl; c:\ufmg\sistemasdistribuídos\perfmon\perfutilsd6src - alexey dynnikov\perfutils d6\runtime; c:\ufmg\sistemasdistribuídos\perfmon\perfutilsd6src - alexey dynnikov\perfutils d6\designtime*
Package file name:	*C:\Ufmg\SistemasDistribuídos\PerfMon\PerfUtilsD6Src - Alexey Dynnikov\PerfUtils D6\DesignTime\dclPerfUtils6.dpk*

A janela ficará assim:

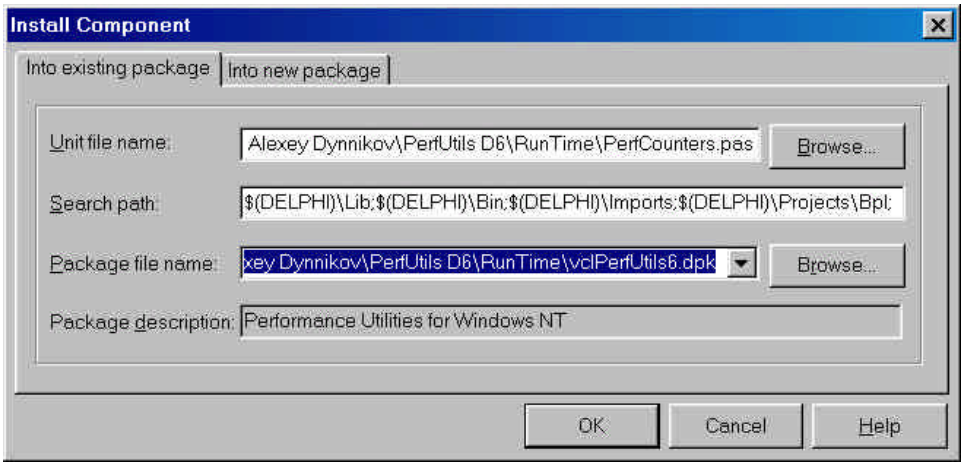


Figura 11 – Instalando o package PerfUtils

3. Clique ok. Será exibida uma janela relacionando todos os arquivos do package com sua lista de requerimentos.

4. Acione Compile e o package será instalado. Tanto o package de RunTime como o de DesignTime serão compilados.

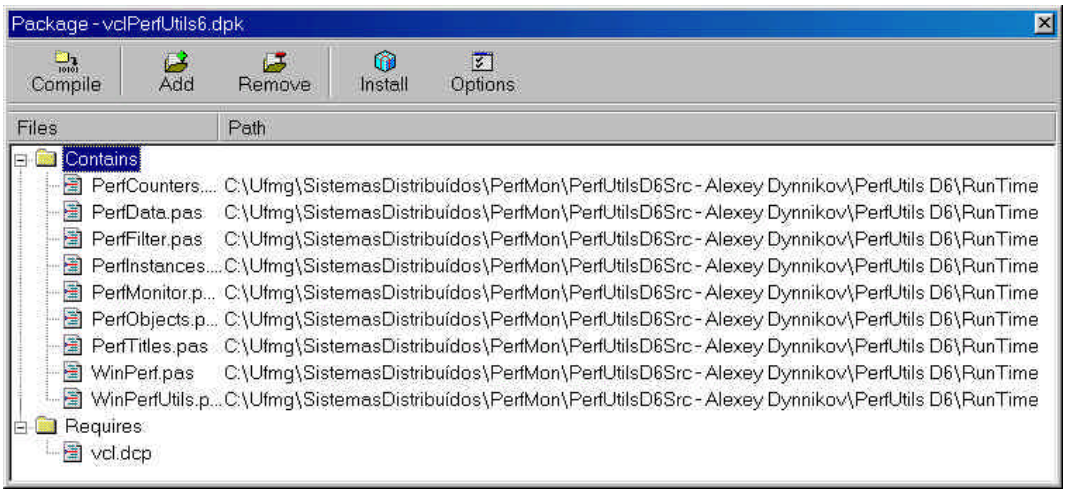


Figura 12 - Janela de compilação do package

5. Para conferir se os componentes da suite PerfUtils estão disponíveis clique na aba PerfUtils que aparecerá como a última aba à direita. Todos os ícones correspondentes aos componentes aparecerão no topo da página correspondente.

Figura 13 – Aba de utilidades PerfUtils da janela do Delphi 6

6. A partir de agora basta clicar duas vezes sobre qualquer um dos componentes que ele será inserido no programa correspondente.

CPU Usage

CPU usage é um aplicativo que exibe um gráfico de barras mostrando o percentual de utilização da CPU. Para criar esta aplicação o componente TperfMonitor foi utilizado. Todas as informações foram extraídas da referência [Dynnikov 1998].

Construção do aplicativo

Os passos para a construção do aplicativo foram os seguintes:

1. Um novo projeto foi criado e os seguintes componentes foram incluídos no *form*, clicando-se duas vezes no seu ícone:
   - TperfMonitor do menu PerfUtils que é o componente responsável pela coleta de dados de performance.
   - Ttimer do menu *System* que irá temporizar a atividade de refresh da janela.
   - TprogressBar do menu *Win32* que exibe o resultado.
   -

   O formulário ficará com o aspecto mostrado na Figura 14:

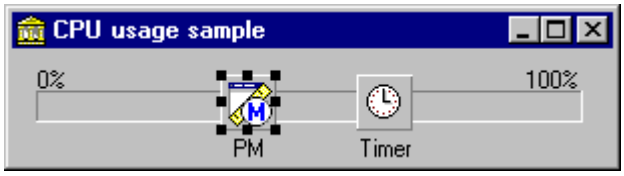


Figura 14 – Janela da aplicação CPU Usage

2. Em seguida o componente PerfMonitor deve ser configurado. A primeira operação consiste em se definir a propriedade LocaleId. Esta propriedade define a linguagem do título do objeto de performance e contadores. O valor 009 seleciona o inglês como idioma.

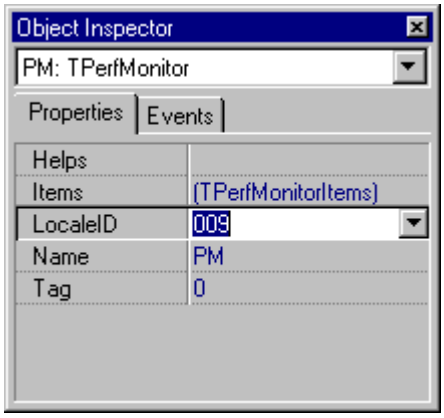


Figura 15 – Configuração do componente PerfMonitor

3. Agora ajustamos a propriedade *Items* deste componente. Esta propriedade contém uma coleção de itens de monitoramento. Adicione um item a ele e defina suas propriedades.

   Inicialmente defina a quarta propriedade: o *ObjectName* para *Processo*. A lista drop-down contém os nomes de todos os objetos de performance existentes em seu sistema e podemos escolher o nome do objeto desta lista.

   Nós devemos escolher a propriedade *CounterName* selecionando: %ProcessorTime da lista de todos os contadores do objeto especificado por ObjectName.

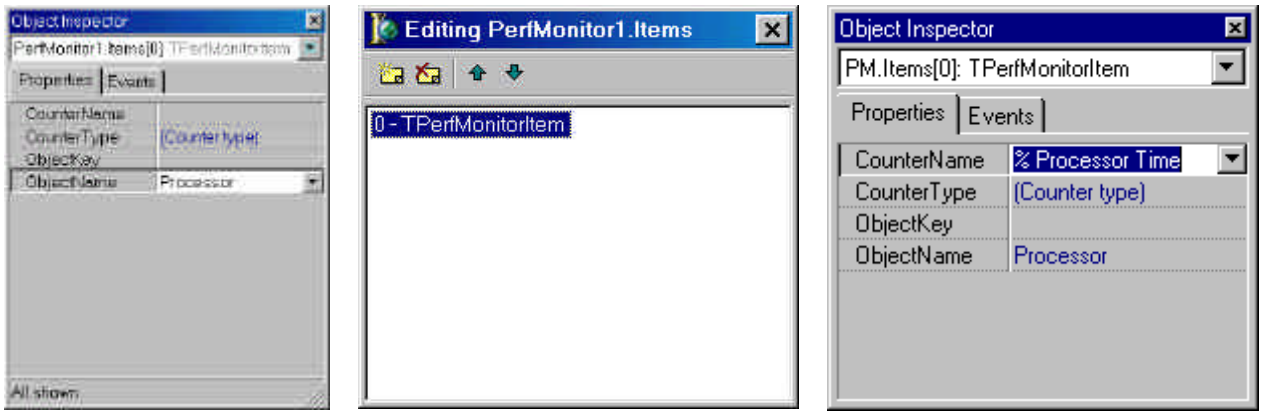


Figura 16 – Definindo a propriedade CounterName

4. O próximo passo consiste em adicionar código ao evento Timer.OnTimer correspondendo ao handler do evento de tempo:

```
procedure TMainForm.TimerTimer(Sender: TObject);
begin
    if _Busy then exit;
       _Busy := True; // Não permite reentrância neste loop
    try
       // Coleta dados de performance
       PM.Collect;

       // ...e mostra-os
       PB.Position:=Round(PM.Items[0].InstanceValues[0].AsFloat);

    finally
       _Busy := False;
    end;
end;
```

   Como o objeto de performance *Processador* possui múltiplas instâncias, já que podemos ter mais de um processador no sistema, a propriedade indexada *InstanceValues* deve ser utilizada para obter o valor do contador.
   A variável _Busy é usada para prevenir reentrância no *loop* de refresh de tela. Esta variável deve ser declarada na seção privada da declaração do formulário.

Programa fonte completo:

```
unit fmMain;

//---------------------------------------------------------------------------
// CPU Usage Example (Windows NT/2000)
//
// This example shows how to use PerfUtils components for CPU usage monitoring
// You will need to install the PerfUtils to compile this example.
// PerfUtils home page is http://www.aldyn.ru/products/perfutils/
//
// This example with step by step instructions is published at
// http://www.aldyn.ru/demos/0017/
//
// (c) 2000 Alexey Dynnikov <aldyn@chat.ru>
//---------------------------------------------------------------------------

interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls, ComCtrls, PerfMonitor;

type
  TMainForm = class(TForm)
    PM: TPerfMonitor;
    PB: TProgressBar;
    Label1: TLabel;
    Label2: TLabel;
    Timer: TTimer;
    procedure TimerTimer(Sender: TObject);
  private
    { Private declarations }
    _Busy : Boolean;  // Variável adicionada para prevenir reentrância no loop do timer
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.TimerTimer(Sender: TObject);
begin
    if _Busy then exit;
    _Busy := True; // Não permite reentrância neste loop
    try
        // Coleta dados de performance
        PM.Collect;

        // ...e mostra-os…
        PB.Position:=Round(PM.Items[0].InstanceValues[0].AsFloat);

    finally
        _Busy := False;
    end;
end;
end.
```

PerInfo

PerInfo lista todos os objetos do sistema operacional e uma vez selecionado um tipo de objeto lista todos os contadores com seus valores atuais e fornece uma descrição do objeto.
Acionado a tecla *More Info...*, ou clicando duas vezes em um objeto, uma nova janela é exibida. Esta janela possui duas páginas denominadas Counters e Instances. A página counters lista todos os contadores, suas propriedades com valores e a descrição do contador. A segunda página lista as instâncias do objeto.

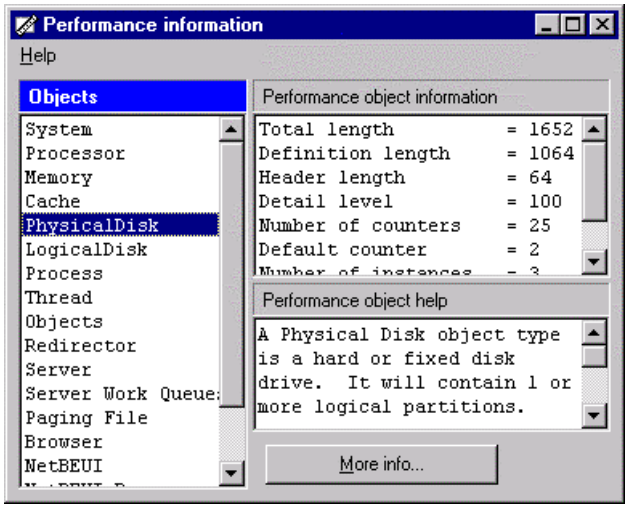


Figura 17 – Janela da aplicação PefInfo de AlexyDynnikov

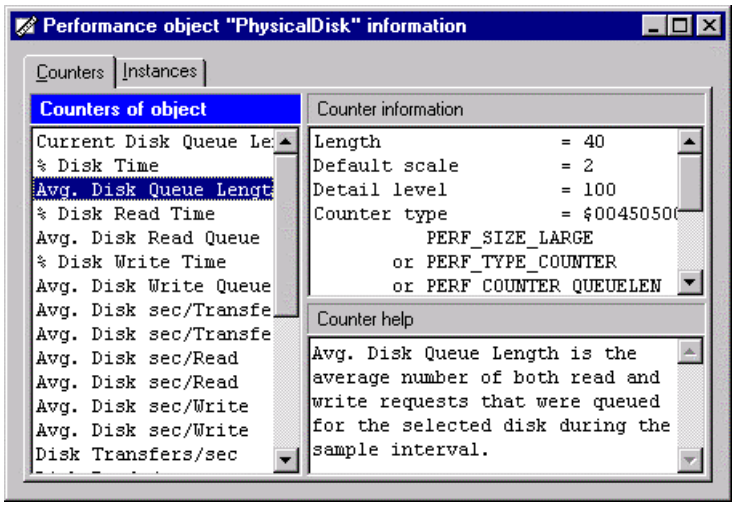
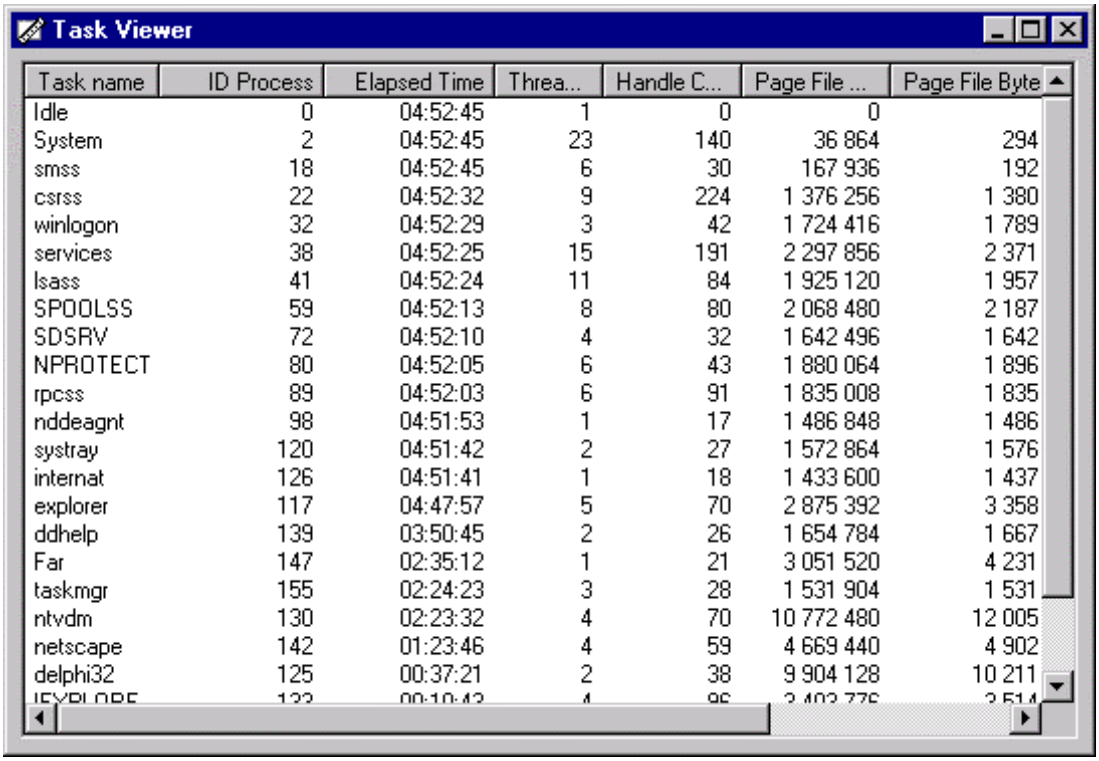


Figura 18 – Janela de More Info...

TaskViewer

TaskViewer é uma aplicação que usa os componentes PerfUtils para exibir informações sobre todos os processos de um sistema. A janela é semelhante à do TaskManager do WNT. Os contadores mostrados são os do objeto processo.



Task name	ID Process	Elapsed Time	Threa...	Handle C...	Page File ...	Page File Byte
Idle	0	04:52:45	1	0	0	
System	2	04:52:45	23	140	36 864	294
smss	18	04:52:45	6	30	167 936	192
csrss	22	04:52:32	9	224	1 376 256	1 380
winlogon	32	04:52:29	3	42	1 724 416	1 789
services	38	04:52:25	15	191	2 297 856	2 371
lsass	41	04:52:24	11	84	1 925 120	1 957
SPOOLSS	59	04:52:13	8	80	2 068 480	2 187
SDSRV	72	04:52:10	4	32	1 642 496	1 642
NPROTECT	80	04:52:05	6	43	1 880 064	1 896
rpcss	89	04:52:03	6	91	1 835 008	1 835
nddeagnt	98	04:51:53	1	17	1 486 848	1 486
systray	120	04:51:42	2	27	1 572 864	1 576
internat	126	04:51:41	1	18	1 433 600	1 437
explorer	117	04:47:57	5	70	2 875 392	3 358
ddhelp	139	03:50:45	2	26	1 654 784	1 667
Far	147	02:35:12	1	21	3 051 520	4 231
taskmgr	155	02:24:23	3	28	1 531 904	1 531
ntvdm	130	02:23:32	4	70	10 772 480	12 005
netscape	142	01:23:46	4	59	4 669 440	4 902
delphi32	125	00:37:21	2	38	9 904 128	10 211
IEXPLORE	133	00:10:43	4	96	2 403 776	2 514

Figura 19 - Janela da aplicação TaskView de Alexy Dynnikov

Performance Explorer

Este aplicativo exibe uma árvore hierárquica dos objetos do sistema. Para cada objeto são exibidos os contadores de performance associados. O painel à direita exibe informação detalhada sobre o nodo selecionado.

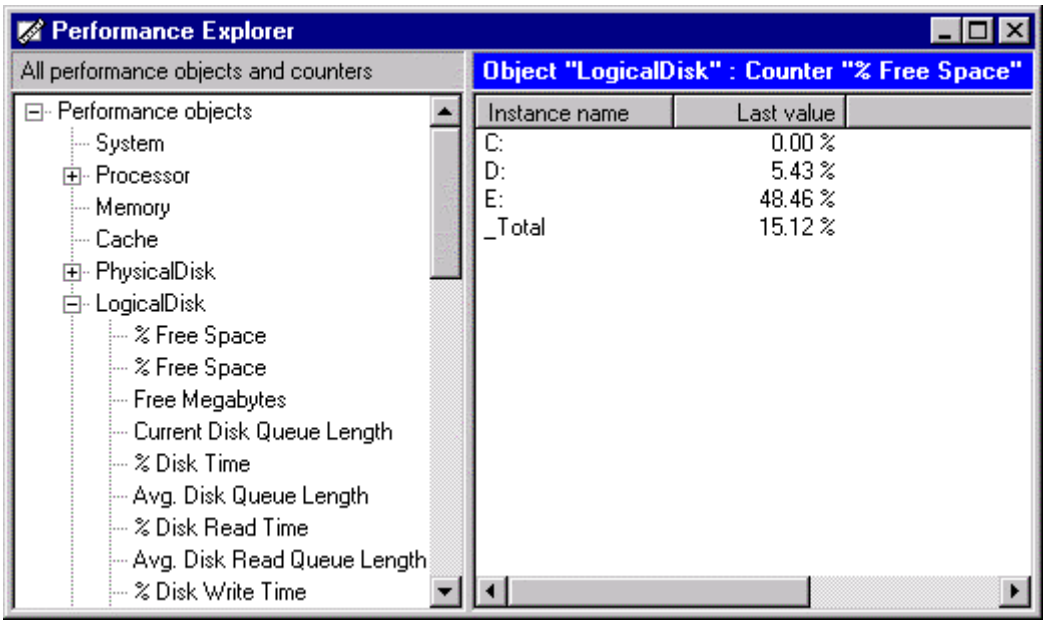


Figura 20 – Janela da aplicação PerfExplorer de AlexyDynnikov

Definindo contadores para uma aplicação própria

É muito útil que se defina contadores de performance para serviços criados pelo próprio usuário.
Por exemplo, softwares como gerenciadores de bancos de dados como Oracle e SQL Server e gerenciadores de serviços de rede geralmente definem uma série de contadores que devem ser monitorados pelos usuários para ajudar a obter a melhor sintonia para o seu sistema. Para saber como usar contadores em sua própria aplicação a referência [Russ Blake 1995] é a fonte seminal. Um artigo bastante interessante e de fácil leitura, abordando o mesmo assunto é [Järvinen 1999].
Eu recomendo esta última referência por ser mais imediata, sintetizando todo o assunto em apenas 12 páginas.

Exercícios

1) Faça um programa em C++ que liste todos os processos em execução com suas respectivas threads e seus principais contadores com atualização dinâmica a cada 1 segundo.

2) Faça um programa para gerenciar os principais recursos da máquina tais como: ocupação de CPU, utilização de memória, etc. exibindo os resultados a cada 2 segundos e plotando um gráfico de tendência histórica. Neste gráfico deve ficar marcado o valor máximo alcançado por cada contador.

3) Liste os atributos conforme a tabela 1 do objeto TCP.

4) Altere a aplicação em Delphi, dada como exemplo, DmoProj, de modo a buscar outros contadores do seu interesse.

5) Construa uma aplicação em Delphi para exibir em um gráfico de barras a taxa de ccupação da CPU e de percentual de utilização de memória.

Bibliografia

[Pietrek Mar_1996] Matt Pietrek, Under the Hood, Microsoft Systems Journal, March 1996

[Pietrek Apr_1996] Matt Pietrek, Under the Hood, Microsoft Systems Journal, April 1996 – foca na leitura dos contadores diretamente da registry

[Pietrek Mar_1998] Matt Pietrek, Under the Hood, Microsoft Systems Journal, March 1998 – foca o uso de PDH com VB

[Pietrek May_1998] Matt Pietrek, Under the Hood, Microsoft Systems Journal, May 1998 – foca o uso da PDH com C/C++

[Russ Blake 1995] Russ Blake, Optimizing Windows NT, Microsoft Windows NT Resource Toolkit, Microsoft Press, 1995

[Dynnikov 1998] Alexey Dynnikov, Alexey Dynnikov Delphi Pages, PerInfo sample application, 1998, http://www.aldyn.ru/demos/0011/index.html.

[Järvinen 1999] Jani Järvinen, Performance Monitoring and Building an HTTP Gateway Server, March 1999, Hardcore Delphi, http://www.pinnaclepublishing.com/dd/DDmain.nsf/getauthor?open &Jani ~Jarvinen

[Smith 2000] Mark Smith, Performance, Dec 2000, http://www.burn-rubber.demon.co.uk/articles/perf.doc