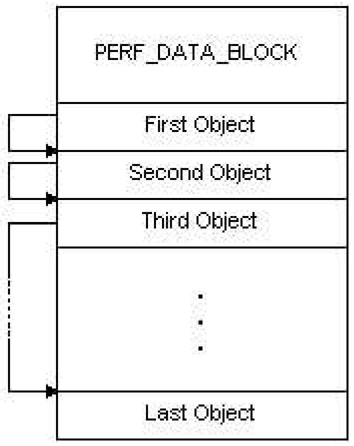
Platform SDK: Performance Monitoring

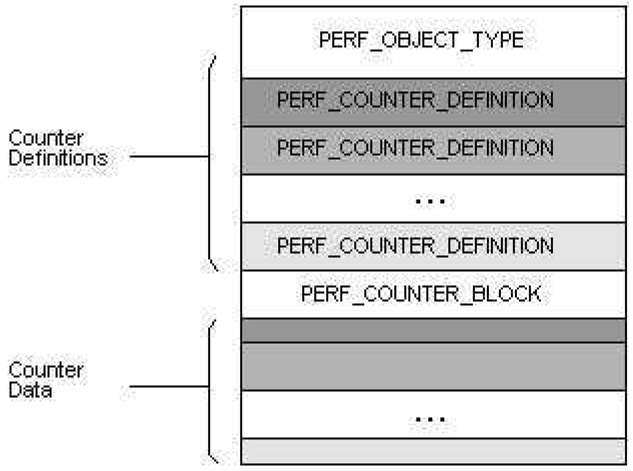
Performance Data Format

The format of the data retrieved by the **RegQueryValueEx** function begins with a fixed-length header structure, **PERF_DATA_BLOCK**. The **PERF_DATA_BLOCK** structure describes the system and the performance data. The **PERF_DATA_BLOCK** structure is followed by variable number of variable-length object data items. The header of the object contains the offset of the next object in the list. The following diagram shows the basic performance data structure.

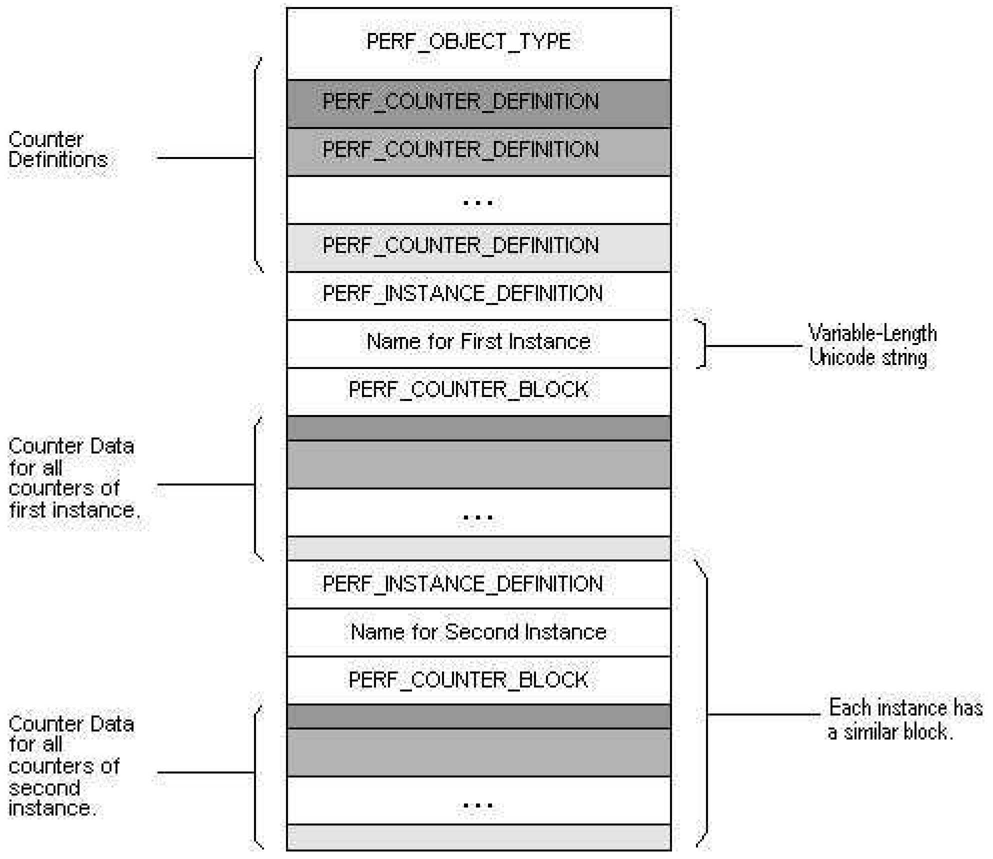


There are two formats for the object data items: one that supports multiple instances, and the other that does not support multiple instances.

Each object information block contains a **PERF_OBJECT_TYPE** structure, which describes the performance data for the object. The **PERF_OBJECT_TYPE** structure is followed by a list of **PERF_COUNTER_DEFINITION** structures, one for each counter defined for the object. For an object with only one instance, the list of **PERF_COUNTER_DEFINITION** structures is followed by a single **PERF_COUNTER_BLOCK** structure, followed by the counter data. Each **PERF_COUNTER_DEFINITION** structure contains the offset from the start of the PERF_COUNTER_BLOCK structure to the corresponding counter data. The following diagram shows the structure of a performance object that does not support multiple instances.



For an object type that supports multiple instances, the list of **PERF_COUNTER_DEFINITION** structures is followed by a list of instance information blocks (one for each instance). Each instance information block contains a **PERF_INSTANCE_DEFINITION** structure, the name of the instance, and a **PERF_COUNTER_BLOCK** structure. The following diagram shows the structure of a performance object that supports two instances.



Last updated: March 2005 | What did you think of this topic? | Order a Platform SDK CD

Platform SDK: Performance Monitoring

PERF_DATA_BLOCK

The **PERF_DATA_BLOCK** structure describes the performance data provided by **RegQueryValueEx** function. The data starts with a **PERF_DATA_BLOCK** structure and is followed by a **PERF_OBJECT_TYPE** structure and other object-specific data for each type of object monitored.

```
typedef struct _PERF_DATA_BLOCK {
  WCHAR Signature[4];
  DWORD LittleEndian;
  DWORD Version;
  DWORD Revision;
  DWORD TotalByteLength;
  DWORD HeaderLength;
  DWORD NumObjectTypes;
  DWORD DefaultObject;
  SYSTEMTIME SystemTime;
  LARGE_INTEGER PerfTime;
  LARGE_INTEGER PerfFreq;
  LARGE_INTEGER PerfTime100nSec;
  DWORD SystemNameLength;
  DWORD SystemNameOffset;
} PERF_DATA_BLOCK;
```

Members

Signature
Pointer to a null-terminated Unicode string, "PERF".

LittleEndian
If the processor is big endian, this member is zero; otherwise it is one.

Version
Version of the performance structures. This member is greater than or equal to one.

Revision
Revision of the performance structures. This member is greater than or equal to zero.

TotalByteLength
Total size of the performance data, in bytes.

HeaderLength
Size of this structure, in bytes.

NumObjectTypes
Number of object types being monitored.

DefaultObject
Object title index of the default object whose performance data is to be displayed. This member can be –1 to indicate that no data is to be displayed.

SystemTime
Time when the system is monitored. This member is in Coordinated Universal Time (UTC) format.

PerfTime
Performance-counter value, in counts, for the system being monitored.

PerfFreq
Performance-counter frequency, in counts per second, for the system being monitored.

PerfTime100nSec
Performance-counter value, in 100 nanosecond units, for the system being monitored.

SystemNameLength
Size of the system name, in bytes.

SystemNameOffset
Offset from the beginning of this structure to the name of the system being monitored.

Requirements

Client	Requires Windows XP, Windows 2000 Professional, or Windows NT Workstation.
Server	Requires Windows Server 2003, Windows 2000 Server, or Windows NT Server.
Header	Declared in Winperf.h; include Windows.h.

See Also

PERF_OBJECT_TYPE, **RegQueryValueEx**

Platform SDK: Performance Monitoring

PERF_OBJECT_TYPE

The **PERF_OBJECT_TYPE** structure describes object-specific performance information. This structure is followed by a list of **PERF_COUNTER_DEFINITION** structures, one for each counter defined for the type of object.

```
typedef struct _PERF_OBJECT_TYPE {
  DWORD TotalByteLength;
  DWORD DefinitionLength;
  DWORD HeaderLength;
  DWORD ObjectNameTitleIndex;
  LPWSTR ObjectNameTitle;
  DWORD ObjectHelpTitleIndex;
  LPWSTR ObjectHelpTitle;
  DWORD DetailLevel;
  DWORD NumCounters;
  DWORD DefaultCounter;
  DWORD NumInstances;
  DWORD CodePage;
  LARGE_INTEGER PerfTime;
  LARGE_INTEGER PerfFreq;
} PERF_OBJECT_TYPE;
```

Members

TotalByteLength
Size of the object-specific data, in bytes. This value includes this structure, the **PERF_COUNTER_DEFINITION** structures, and the **PERF_INSTANCE_DEFINITION** and **PERF_COUNTER_BLOCK** structures for each instance. This member specifies the offset from the beginning of this structure to the next **PERF_OBJECT_TYPE** structure if one exists.

DefinitionLength
Size of the object-specific data, in bytes. This value includes this structure and the **PERF_COUNTER_DEFINITION** structures for this object. This member is the offset from the beginning of the **PERF_OBJECT_TYPE** structure to the first **PERF_INSTANCE_DEFINITION** structure or to the **PERF_COUNTER_DEFINITION** structures if there is no instance data.

HeaderLength
Length, in bytes, of this structure. This member is the offset to the first **PERF_COUNTER_DEFINITION** structure for this object.

ObjectNameTitleIndex
Index to the object's name in the title database.

ObjectNameTitle
Pointer to a null-terminated string that specifies the name of the object. This member initially contains NULL, but it can contain a pointer to the actual string once the string is located.

ObjectHelpTitleIndex
Index to the object's Help title in the title database.

ObjectHelpTitle
Pointer to a null-terminated Unicode string that specifies the title of Help. This member initially contains NULL, but it can contain a pointer to the actual string once the string is located.

DetailLevel
Level of detail. Applications use this value to control display complexity. This value is the minimum detail level of all the counters for a given object. This member can be one of the following values.

Detail level	Meaning
PERF_DETAIL_NOVICE	No technical ability is required to understand the counter data.
PERF_DETAIL_ADVANCED	The counter data is provided for advanced users.
PERF_DETAIL_EXPERT	The counter data is provided for expert users.
PERF_DETAIL_WIZARD	The counter data is provided for system designers.

NumCounters
Number of counters in each counter block. There is one counter block per instance.

DefaultCounter
Default counter whose information is to be displayed when this object is selected. This member is typically greater than or equal to zero. However, this member may be −1 to indicate that there is no default.

NumInstances
Number of object instances for which counters are being provided. If the object can have zero or more instances, but has none at present, this value should be zero. If the object cannot have multiple instances, this value should be PERF_NO_INSTANCES.

CodePage
Code page. This member is zero if the instance strings are in Unicode. Otherwise, this member is the code-page identifier of the instance names.

PerfTime
Current value, in counts, of the high-resolution performance counter.

PerfFreq
Current frequency, in counts per second, of the high-resolution performance counter.

Remarks

If there is only one instance of the object type, the counter definitions are followed by a single **PERF_COUNTER_BLOCK** structure. This structure is followed by data for each counter. (The **PERF_COUNTER_BLOCK** structure contains the total length of the structure and the counter data that follows it.)

If there is more than one instance of the object type, the list of counter definitions is followed by a **PERF_INSTANCE_DEFINITION** structure and a **PERF_COUNTER_BLOCK** structure for each instance. The **PERF_INSTANCE_DEFINITION** structure includes the name, the identifier, and the name of the parent of the instance.

Following the counter data, there is a **PERF_INSTANCE_DEFINITION** structure and a **PERF_COUNTER_BLOCK** structure for each instance specified in the **PERF_DATA_BLOCK** structure that begins the performance-data area.

Requirements

Client	Requires Windows XP, Windows 2000 Professional, or Windows NT Workstation.
Server	Requires Windows Server 2003, Windows 2000 Server, or Windows NT Server.
Header	Declared in Winperf.h; include Windows.h.

See Also

PERF_COUNTER_BLOCK, **PERF_COUNTER_DEFINITION**, **PERF_INSTANCE_DEFINITION**

Platform SDK: Performance Monitoring

PERF_INSTANCE_DEFINITION

The **PERF_INSTANCE_DEFINITION** structure contains the instance-specific information for a block of performance data. There is one **PERF_INSTANCE_DEFINITION** structure for each instance specified in the **PERF_OBJECT_TYPE** structure.

```
typedef struct _PERF_INSTANCE_DEFINITION {
  DWORD ByteLength;
  DWORD ParentObjectTitleIndex;
  DWORD ParentObjectInstance;
  DWORD UniqueID;
  DWORD NameOffset;
  DWORD NameLength;
} PERF_INSTANCE_DEFINITION;
```

Members

ByteLength
Size of this structure, including the subsequent instance name, in bytes.

ParentObjectTitleIndex
Index of the name of the "parent" object in the title database. For example, if the object is a thread, the parent object type is a process, or if the object is a logical drive, the parent is a physical drive.

ParentObjectInstance
Index to an instance of the parent object type that is the parent of this instance. This member may be zero or greater.

UniqueID
Unique identifier used instead of the instance name. This member is PERF_NO_UNIQUE_ID if there is no such identifier.

NameOffset
Offset from the beginning of this structure to the Unicode name of this instance.

NameLength
Size of the instance name, in bytes. This member is zero if the instance does not have a name.

Requirements

Client	Requires Windows XP, Windows 2000 Professional, or Windows NT Workstation.
Server	Requires Windows Server 2003, Windows 2000 Server, or Windows NT Server.
Header	Declared in Winperf.h; include Windows.h.

See Also

PERF_OBJECT_TYPE

Platform SDK: Performance Monitoring

Displaying Object, Instance, and Counter Names

The performance data contains information for a variable number of object types, instances per object, and counters per object type. Therefore, the number and size of blocks in the performance data varies. To ensure that your application correctly receives the performance data, you must use the offsets included in the performance structures to navigate through the data. Every offset is a count of bytes relative to the structure containing it.

Note The reason the system uses offsets instead of pointers is that pointers are not valid across process boundaries. The addresses that the process that installs the counters would store would not be valid for the process that reads the counters.

The following example displays the index and name of each object, along with the indexes and names of its counters.

The object and counter names are stored in the registry, by index. This example creates a function, GetNameStrings, to load the indexes and names of each object and counter from the registry into an array, so that they can be easily accessed. GetNameStrings uses the following standard registry functions to access the data: **RegOpenKey**, **RegCloseKey**, **RegQueryInfoKey**, and **RegQueryValueEx**.

This example creates the following functions for navigating the performance data: FirstObject, FirstInstance, FirstCounter, NextCounter, NextInstance, and NextCounter. These functions navigate the performance data by using the offsets stored in the performance structures.

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>

#define TOTALBYTES    8192
#define BYTEINCREMENT 1024

LPSTR lpNameStrings;
LPSTR *lpNamesArray;

/*****************************************************************
 *                                                               *
 * Functions used to navigate through the performance data.      *
 *                                                               *
 *****************************************************************/

PPERF_OBJECT_TYPE FirstObject( PPERF_DATA_BLOCK PerfData )
{
    return( (PPERF_OBJECT_TYPE)((PBYTE)PerfData +
        PerfData->HeaderLength) );
}

PPERF_OBJECT_TYPE NextObject( PPERF_OBJECT_TYPE PerfObj )
{
    return( (PPERF_OBJECT_TYPE)((PBYTE)PerfObj +
        PerfObj->TotalByteLength) );
}

PPERF_INSTANCE_DEFINITION FirstInstance( PPERF_OBJECT_TYPE PerfObj )
{
    return( (PPERF_INSTANCE_DEFINITION)((PBYTE)PerfObj +
        PerfObj->DefinitionLength) );
}

PPERF_INSTANCE_DEFINITION NextInstance(
    PPERF_INSTANCE_DEFINITION PerfInst )
{
    PPERF_COUNTER_BLOCK PerfCntrBlk;

    PerfCntrBlk = (PPERF_COUNTER_BLOCK)((PBYTE)PerfInst +
        PerfInst->ByteLength);

    return( (PPERF_INSTANCE_DEFINITION)((PBYTE)PerfCntrBlk +
        PerfCntrBlk->ByteLength) );
}

PPERF_COUNTER_DEFINITION FirstCounter( PPERF_OBJECT_TYPE PerfObj )
{
    return( (PPERF_COUNTER_DEFINITION) ((PBYTE)PerfObj +
        PerfObj->HeaderLength) );
}

PPERF_COUNTER_DEFINITION NextCounter(
    PPERF_COUNTER_DEFINITION PerfCntr )
{
    return( (PPERF_COUNTER_DEFINITION)((PBYTE)PerfCntr +
        PerfCntr->ByteLength) );
}

/*****************************************************************
 *                                                               *
 * Load the counter and object names from the registry to the    *
 * global variable lpNamesArray.                                 *
 *                                                               *
 *****************************************************************/

BOOL GetNameStrings( )
{
    HKEY hKeyPerflib;      // handle to registry key
    HKEY hKeyPerflib009;   // handle to registry key
    DWORD dwMaxValueLen;   // maximum size of key values
    DWORD dwBuffer;        // bytes to allocate for buffers
    DWORD dwBufferSize = sizeof(DWORD);  // size of dwBuffer
    LPSTR lpCurrentString; // pointer for enumerating data strings
    DWORD dwCounter;       // current counter index
    LONG lResult;          // return value

// Get the number of Counter items.

    if( RegOpenKeyEx( HKEY_LOCAL_MACHINE,
        "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Perflib",
        0,
        KEY_READ,
        &hKeyPerflib) != ERROR_SUCCESS
      )
        return FALSE;

    lResult = RegQueryValueEx( hKeyPerflib,
        "Last Counter",
        NULL,
        NULL,
        (LPBYTE) &dwBuffer,
        &dwBufferSize );

    RegCloseKey( hKeyPerflib );

    if( lResult != ERROR_SUCCESS )
        return FALSE;

// Allocate memory for the names array.

    lpNamesArray = malloc( (dwBuffer+1) * sizeof(LPSTR) );

    if( lpNamesArray == NULL )
        return FALSE;

// Open the key containing the counter and object names.

    if( RegOpenKeyEx( HKEY_LOCAL_MACHINE,
    "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Perflib\\009",
        0,
        KEY_READ,
        &hKeyPerflib009) != ERROR_SUCCESS
      )
        return FALSE;

// Get the size of the largest value in the key (Counter or Help).

    if( RegQueryInfoKey( hKeyPerflib009,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        &dwMaxValueLen,
        NULL,
        NULL) != ERROR_SUCCESS
      )
        return FALSE;

// Allocate memory for the counter and object names.

    dwBuffer = dwMaxValueLen + 1;

    lpNameStrings = malloc( dwBuffer * sizeof(CHAR) );

    if (lpNameStrings == NULL)
    {
        free( lpNamesArray );
        return FALSE;
    }

// Read the counter value.

    lResult = RegQueryValueEx( hKeyPerflib009,
        "Counters",
        NULL,
        NULL,
        lpNameStrings, &dwBuffer );

    RegCloseKey( hKeyPerflib009 );

    if( lResult != ERROR_SUCCESS )
        return FALSE;

// Load names into an array, by index.

    for( lpCurrentString = lpNameStrings; *lpCurrentString;
         lpCurrentString += (lstrlen(lpCurrentString)+1) )
    {
        dwCounter = atol( lpCurrentString );

        lpCurrentString += (lstrlen(lpCurrentString)+1);

        lpNamesArray[dwCounter] = (LPSTR) lpCurrentString;
    }

   return TRUE;
}

/*****************************************************************
 *                                                               *
 * Display the indexes and/or names for all performance objects, *
 * instances, and counters.                                      *
 *                                                               *
 *****************************************************************/

int main()
{
    PPERF_DATA_BLOCK PerfData = NULL;
    PPERF_OBJECT_TYPE PerfObj;
    PPERF_INSTANCE_DEFINITION PerfInst;
    PPERF_COUNTER_DEFINITION PerfCntr, CurCntr;
    PPERF_COUNTER_BLOCK PtrToCntr;
    DWORD BufferSize = TOTALBYTES;
    DWORD i, j, k;

// Get the name strings through the registry.

    if( !GetNameStrings( ) )
       return FALSE;

// Allocate the buffer for the performance data.
```

```
        PerfData = (PPERF_DATA_BLOCK) malloc( BufferSize );

        if( PerfData == NULL )
           return FALSE;

        while( RegQueryValueEx( HKEY_PERFORMANCE_DATA,
                                  "Global",
                                  NULL,
                                  NULL,
                                  (LPBYTE) PerfData,
                                  &BufferSize ) == ERROR_MORE_DATA )
        {
        // Get a buffer that is big enough.

            BufferSize += BYTEINCREMENT;
            PerfData = (PPERF_DATA_BLOCK) realloc( PerfData, BufferSize );
        }

    // Get the first object type.

        PerfObj = FirstObject( PerfData );

    // Process all objects.

        for( i=0; i < PerfData->NumObjectTypes; i++ )
        {
        // Display the object by index and name.

            printf( "\nObject %ld: %s\n", PerfObj->ObjectNameTitleIndex,
                lpNamesArray[PerfObj->ObjectNameTitleIndex] );

        // Get the first counter.

            PerfCntr = FirstCounter( PerfObj );

            if( PerfObj->NumInstances > 0 )
            {
            // Get the first instance.

                PerfInst = FirstInstance( PerfObj );

            // Retrieve all instances.

                for( k=0; k < PerfObj->NumInstances; k++ )
                {
                // Display the instance by name.

                    printf( "\n\tInstance %S: \n",
                        (char *)((PBYTE)PerfInst + PerfInst->NameOffset));
                    CurCntr = PerfCntr;

                // Retrieve all counters.

                    for( j=0; j < PerfObj->NumCounters; j++ )
                    {
                    // Display the counter by index and name.

                        printf("\t\tCounter %ld: %s\n",
                            CurCntr->CounterNameTitleIndex,
                            lpNamesArray[CurCntr->CounterNameTitleIndex]);

                    // Get the next counter.

                        CurCntr = NextCounter( CurCntr );
                    }

                // Get the next instance.

                    PerfInst = NextInstance( PerfInst );
                }
            }
            else
            {
```

```
            // Get the counter block.

                PtrToCntr = (PPERF_COUNTER_BLOCK) ((PBYTE)PerfObj +
                                PerfObj->DefinitionLength );

            // Retrieve all counters.

                for( j=0; j < PerfObj->NumCounters; j++ )
                {
                // Display the counter by index and name.

                    printf( "\tCounter %ld: %s\n", PerfCntr->CounterNameTitleIndex,
                        lpNamesArray[PerfCntr->CounterNameTitleIndex] );

                // Get the next counter.

                    PerfCntr = NextCounter( PerfCntr );
                }
            }

        // Get the next object type.

            PerfObj = NextObject( PerfObj );
        }

        free( lpNamesArray );
        free( lpNameStrings );
        free( PerfData );

        return TRUE;
    }
```