The Windows Registry

What do we have in this session?

Brief Intro

The registry is a **system-defined database** in which **applications and system components store and retrieve configuration data**. The data stored in the registry varies according to the version of Microsoft Windows. Applications use the registry API to retrieve, modify, or delete registry data.

You should not edit registry data that does not belong to your application unless it is absolutely necessary. If there is an error in the registry, your system may not function properly. If this happens, you can restore the registry to the state it was in when you last started the computer successfully.

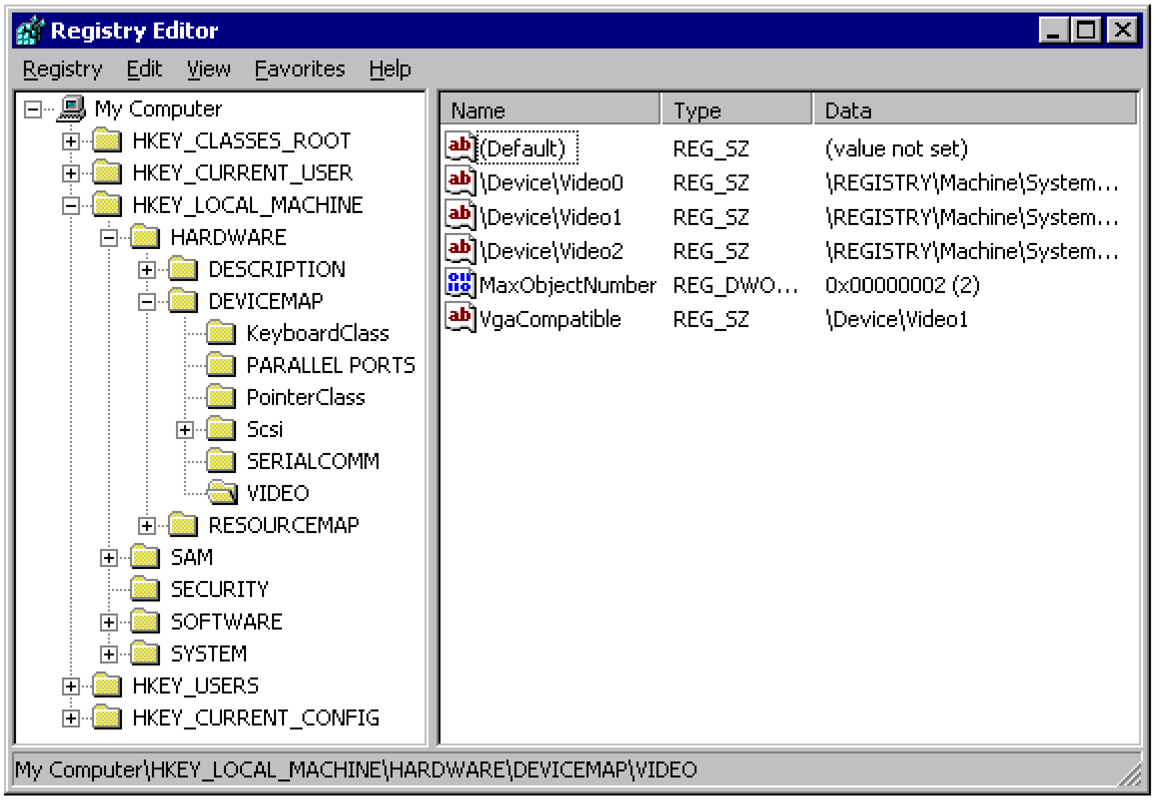
Structure of the Registry

The registry is a hierarchical database that contains **data that is critical for the operation of Windows and the applications and services that run on Windows**. The data is structured in a **tree format**. **Each node in the tree is called a key**. Each key can contain both **subkeys** and data entries called **values**. Sometimes, the presence of a key is all the data that an application requires; other times, an application opens a key and uses the values associated with the key. A key can have any number of values, and the values can be in any form.

Each key has a name consisting of one or more printable characters. Key names are not case sensitive. Key names cannot include the backslash character (\), but any other printable character can be used. Value names and data can include the backslash character.

The name of each subkey is unique with respect to the key that is immediately above it in the hierarchy. Key names are not localized into other languages, although values may be.

The following illustration is an example registry key structure as displayed by the Registry Editor.



```
My Computer
  HKEY_CLASSES_ROOT
  HKEY_CURRENT_USER
  HKEY_LOCAL_MACHINE
    HARDWARE
      DESCRIPTION
      DEVICEMAP
        KeyboardClass
        PARALLEL PORTS
        PointerClass
        Scsi
        SERIALCOMM
        VIDEO
      RESOURCEMAP
    SAM
    SECURITY
    SOFTWARE
    SYSTEM
  HKEY_USERS
  HKEY_CURRENT_CONFIG
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
\Device\Video0	REG_SZ	\REGISTRY\Machine\System...
\Device\Video1	REG_SZ	\REGISTRY\Machine\System...
\Device\Video2	REG_SZ	\REGISTRY\Machine\System...
MaxObjectNumber	REG_DWO...	0x00000002 (2)
VgaCompatible	REG_SZ	\Device\Video1

My Computer\HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\VIDEO

Each of the trees under My Computer is a key. The HKEY_LOCAL_MACHINE key has the following subkeys:

1. HARDWARE,
2. SAM,
3. SECURITY,
4. SOFTWARE, and
5. SYSTEM.

Each of these keys in turn has subkeys. For example, the HARDWARE key has the subkeys DESCRIPTION, DEVICEMAP, and RESOURCEMAP; the DEVICEMAP key has several subkeys including VIDEO.
Each value consists of a value name and its associated data, if any. MaxObjectNumber and VgaCompatible are values that contain data under the VIDEO subkey. A registry tree can be 512 levels deep. You can create up to 32 levels at a time through a single registry API call.

Registry Storage Space

Although there are few technical limits to the type and size of data an application can store in the registry, certain practical guidelines exist to promote system efficiency. **An application should**

store configuration and initialization data in the registry, and **store other kinds of data elsewhere**.

Generally, data consisting of more than one or two kilobytes (K) should be stored as a file and referred to by using a key in the registry rather than being stored as a value. Instead of duplicating large pieces of data in the registry, an application should save the data as a file and refer to the file. Executable binary code should never be stored in the registry.

A value entry uses much less registry space than a key. To save space, an application should group similar data together as a structure and store the structure as a value rather than storing each of the structure members as a separate key. (Storing the data in binary form allows an application to store data in one value that would otherwise be made up of several incompatible types.)

Views of the registry files are mapped in the computer **cache address space**. Therefore, regardless of the size of the registry data, it is not charged more than 4 megabytes (MB). There are no longer any explicit limits on the total amount of space that may be consumed by hives in paged pool memory, and in disk space. The size of the system hive is limited only by physical memory.

Windows 2000

Registry data is stored in the **paged pool**, an area of physical memory used for system data that can be written to disk when not in use. The RegistrySizeLimit value establishes the maximum amount of paged pool that can be consumed by registry data from all applications. This value is located in the following registry key:

```
HKEY_LOCAL_MACHINE
   System
      CurrentControlSet
         Control
```

By default, the registry size limit is 25 percent of the paged pool. (The default size of the paged pool is 32 MB, so this is 8 MB.) The system ensures that the minimum value of RegistrySizeLimit is 4 MB and the maximum is approximately 80 percent of the PagedPoolSize value. If the value of this entry is greater than 80 percent of the size of the paged pool, the system sets the maximum size of the registry to 80 percent of the size of the paged pool. This prevents the registry from consuming space needed by processes. Note that setting this value does not allocate space in the paged pool, nor does it assure that the space will be available if needed.

The paged pool size is determined by the PagedPoolSize value in the following registry key:

```
HKEY_LOCAL_MACHINE
```

```
System
   CurrentControlSet
      Control
         SessionManager
            MemoryManagement
```

The maximum paged pool is approximately 300,470 MB so the registry size limit is 240-376 MB. However, if the /3GB switch is used, the maximum paged pool size is 192 MB, so the registry can be a maximum of 153.6 MB.

Predefined Keys

An application must open a key before it can add data to the registry. To open a key, an application must supply a handle to another key in the registry that is already open. The system defines **predefined keys that are always open**. Predefined keys help an application navigate in the registry and make it possible to develop tools that allow a system administrator to manipulate categories of data. Applications that add data to the registry should always work within the framework of predefined keys, so administrative tools can find and use the new data.
An application can use handles to these keys as entry points to the registry. These handles are valid for all implementations of the registry, although the use of the handles may vary from platform to platform. In addition, other predefined handles have been defined for specific platforms. The following are handles to the predefined keys.

Handle	Description
HKEY_CLASSES_ROOT	Registry entries subordinate to this key define **types (or classes) of documents and the properties associated with those types**. Shell and COM applications use the information stored under this key. This key also provides backward compatibility with the Windows 3.1 registration database by storing information for DDE and OLE support. File viewers and user interface extensions store their OLE class identifiers in HKEY_CLASSES_ROOT, and in-process servers are registered in this key. This handle should not be used in a service or an application that impersonates different users.
HKEY_CURRENT_CONFIG	Contains information about the current **hardware profile of the local computer system**. The information under HKEY_CURRENT_CONFIG describes only the differences between the current hardware configuration and

	the standard configuration. Information about the standard hardware configuration is stored under the Software and System keys of HKEY_LOCAL_MACHINE. HKEY_CURRENT_CONFIG is an alias for HKEY_LOCAL_MACHINE\System\CurrentControlSet\Hardware Profiles\Current.
HKEY_CURRENT_USER	Registry entries subordinate to this key define **the preferences of the current user**. These preferences include the settings of environment variables, data about program groups, colors, printers, network connections, and application preferences. This key makes it easier to establish the current user's settings; the key maps to the current user's branch in HKEY_USERS. In HKEY_CURRENT_USER, software vendors store the current user-specific preferences to be used within their applications. Microsoft, for example, creates the HKEY_CURRENT_USER\Software\Microsoft key for its applications to use, with each application creating its own subkey under the Microsoft key. The mapping between HKEY_CURRENT_USER and HKEY_USERS is per process and is established the first time the process references HKEY_CURRENT_USER. The mapping is based on the security context of the first thread to reference HKEY_CURRENT_USER. If this security context does not have a registry hive loaded in HKEY_USERS, the mapping is established with HKEY_USERS\.Default. After this mapping is established it persists, even if the security context of the thread changes. This handle should not be used in a service or an application that impersonates different users. Instead, call the RegOpenCurrentUser() function.
HKEY_LOCAL_MACHINE	Registry entries subordinate to this key define the **physical state of the computer, including data about the bus type, system memory, and installed hardware and software**. It contains subkeys that hold current configuration data, including Plug and Play information (the Enum branch, which includes a complete list of all hardware that has ever been on the system), network logon preferences, network

	security information, software-related information (such as server names and the location of the server), and other system information.
HKEY_PERFORMANCE_DATA	Registry entries subordinate to this key **allow you to access performance data**. The data is not actually stored in the registry; the registry functions cause the system to collect the data from its source.
HKEY_PERFORMANCE_NLSTEXT	Registry entries subordinate to this key reference the text strings that describe counters in the local language of the area in which the computer system is running. These entries are **not available to Regedit.exe and Regedt32.exe**. Windows 2000: This key is not supported.
HKEY_PERFORMANCE_TEXT	Registry entries subordinate to this key reference the text strings that describe counters in US English. These entries are **not available to Regedit.exe and Regedt32.exe**. Windows 2000: This key is not supported.
HKEY_USERS	Registry entries subordinate to this key define the **default user configuration for new users on the local computer and the user configuration for the current user**.

The RegOverridePredefKey() function enables you to map a predefined registry key to a specified key in the registry. For instance, a software installation program could remap a predefined key before installing a DLL component. This enables the installation program to easily examine the information that the DLL's installation procedure writes to the predefined key.
The RegDisablePredefinedCache() and RegDisablePredefinedCacheEx() functions disable handle caching for predefined registry handles. Services that use impersonation should call RegDisablePredefinedCacheEx() before using predefined registry handles. The predefined handles are not thread safe. Closing a predefined handle in one thread affects any other threads that are using the handle.

Registry Hives

A hive is a logical group of keys, subkeys, and values in the registry that has **a set of supporting files containing backups of its data**.
Each time a new user logs on to a computer, a new hive is created for that user with a separate file for the user profile. This is called the **user profile hive**. A user's hive contains specific registry information pertaining to the user's application settings, desktop, environment, network connections, and printers. User profile hives are located under the HKEY_USERS key.

Registry files have the following two formats: **standard and latest**. The standard format is the only format supported by Windows 2000. It is also supported by later versions of Windows for backward compatibility. The latest format is supported starting with Windows XP. On versions of Windows that support the latest format, the following hives still use the standard format:

1. HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE\SAM
2. KEY_LOCAL_MACHINE\Security, and
3. HKEY_USERS\.DEFAULT;

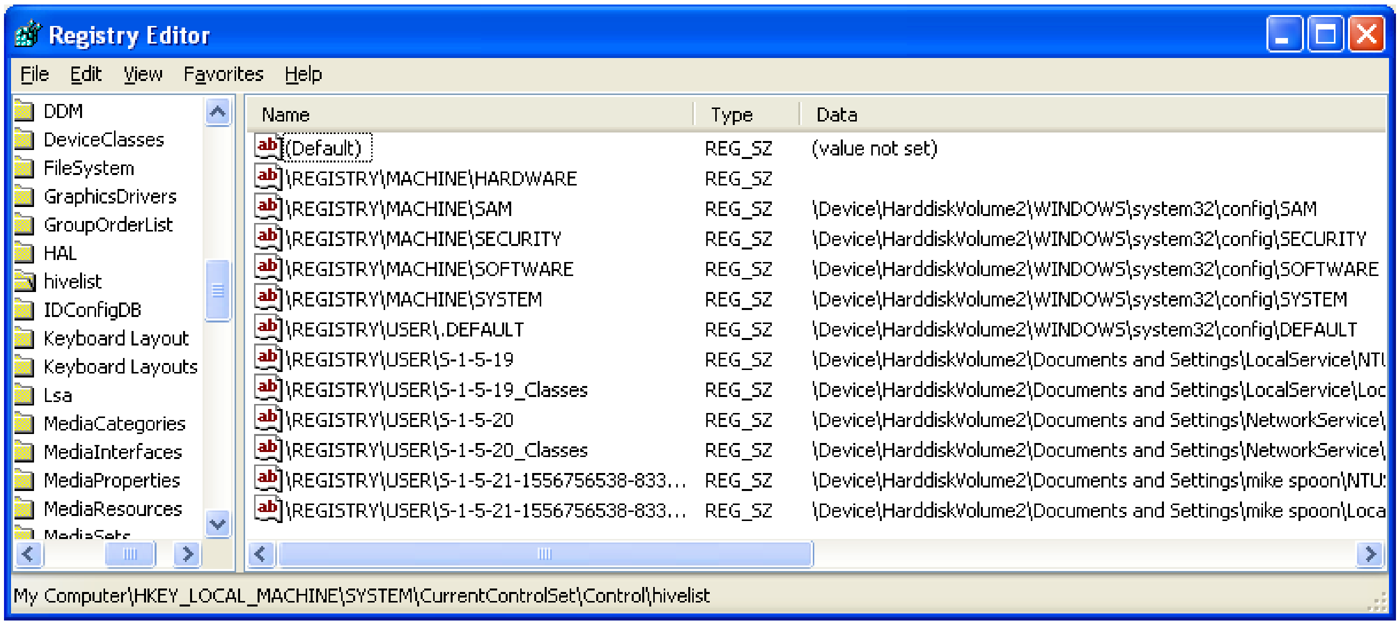
All other hives use the latest format.
Most of the **supporting files for the hives** are in the **%SystemRoot%\System32\Config** directory. For example, it is C:\WINDOWS\System32\Config in Windows XP. These files are **updated each time a user logs on**. The file name extensions of the files in these directories, or in some cases a lack of an extension, indicate the type of data they contain. The following table lists these extensions along with a description of the data in the file.

Extension	Description
none	A complete copy of the hive data.
.alt	A backup copy of the critical HKEY_LOCAL_MACHINE\System hive. Only the System key has an .alt file.
.log	A transaction log of changes to the keys and value entries in the hive.
.sav	Copies of the hive files as they looked at the end of the text-mode stage in Setup. Setup has two stages: text mode and graphics mode. The hive is copied to a .sav file after the text-mode stage of setup to protect it from errors that might occur if the graphics-mode stage of setup fails. If setup fails during the graphics-mode stage, only the graphics-mode stage is repeated when the computer is restarted; the .sav file is used to restore the hive data.

The following table lists the standard hives and their supporting files.

Registry hive	Supporting files
HKEY_CURRENT_CONFIG	System, System.alt, System.log, System.sav
HKEY_CURRENT_USER	Ntuser.dat, Ntuser.dat.log
HKEY_LOCAL_MACHINE\SAM	Sam, Sam.log, Sam.sav
HKEY_LOCAL_MACHINE\Security	Security, Security.log, Security.sav
HKEY_LOCAL_MACHINE\Software	Software, Software.log, Software.sav
HKEY_LOCAL_MACHINE\System	System, System.alt, System.log, System.sav
HKEY_USERS\.DEFAULT	Default, Default.log, Default.sav

A list of all active hives can be found at
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\hivelist`



Name	Type	Data
(Default)	REG_SZ	(value not set)
\REGISTRY\MACHINE\HARDWARE	REG_SZ	
\REGISTRY\MACHINE\SAM	REG_SZ	\Device\HarddiskVolume2\WINDOWS\system32\config\SAM
\REGISTRY\MACHINE\SECURITY	REG_SZ	\Device\HarddiskVolume2\WINDOWS\system32\config\SECURITY
\REGISTRY\MACHINE\SOFTWARE	REG_SZ	\Device\HarddiskVolume2\WINDOWS\system32\config\SOFTWARE
\REGISTRY\MACHINE\SYSTEM	REG_SZ	\Device\HarddiskVolume2\WINDOWS\system32\config\SYSTEM
\REGISTRY\USER\.DEFAULT	REG_SZ	\Device\HarddiskVolume2\WINDOWS\system32\config\DEFAULT
\REGISTRY\USER\S-1-5-19	REG_SZ	\Device\HarddiskVolume2\Documents and Settings\LocalService\NTU
\REGISTRY\USER\S-1-5-19_Classes	REG_SZ	\Device\HarddiskVolume2\Documents and Settings\LocalService\Loc
\REGISTRY\USER\S-1-5-20	REG_SZ	\Device\HarddiskVolume2\Documents and Settings\NetworkService\
\REGISTRY\USER\S-1-5-20_Classes	REG_SZ	\Device\HarddiskVolume2\Documents and Settings\NetworkService\
\REGISTRY\USER\S-1-5-21-1556756538-833...	REG_SZ	\Device\HarddiskVolume2\Documents and Settings\mike spoon\NTU
\REGISTRY\USER\S-1-5-21-1556756538-833...	REG_SZ	\Device\HarddiskVolume2\Documents and Settings\mike spoon\Loca

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\hivelist

HKEY_LOCAL_MACHINE\HARDWARE has no corresponding file because it is a volatile key that is created (and built) by the kernel at system start.

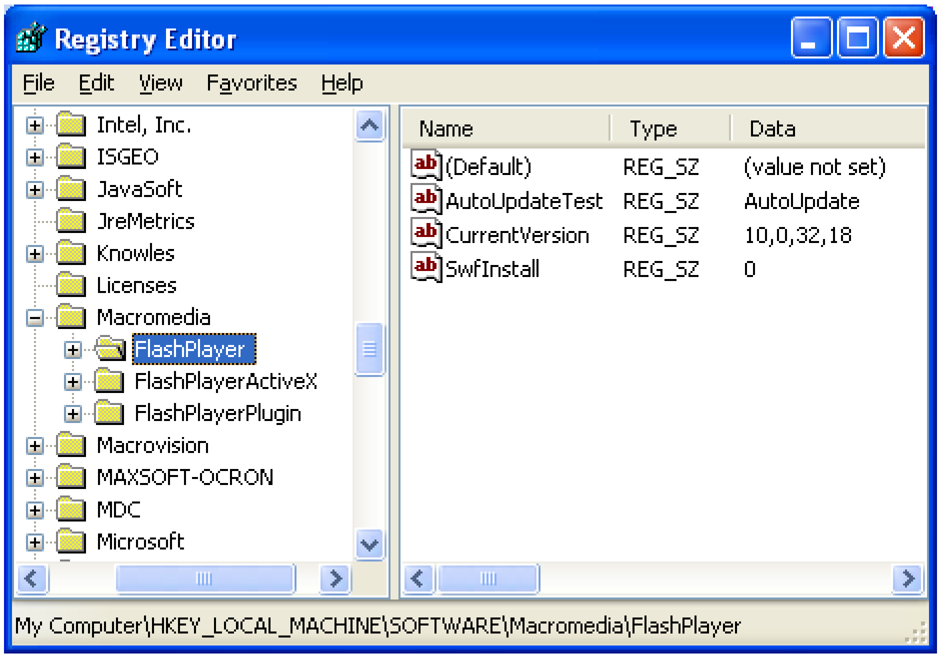
Categories of Data

Before putting data into the registry, an application should divide the data into two categories:

1. Computer-specific data and
2. User-specific data.

By making this distinction, an application can support multiple users, and yet locate user-specific data over a network and use that data in different locations, allowing location-independent user profile data. (A user profile is a set of configuration data saved for every user.)
When the application is installed, it should record the computer-specific data under the HKEY_LOCAL_MACHINE key. In particular, it should create keys for the company name, product name, and version number, as shown in the following example:

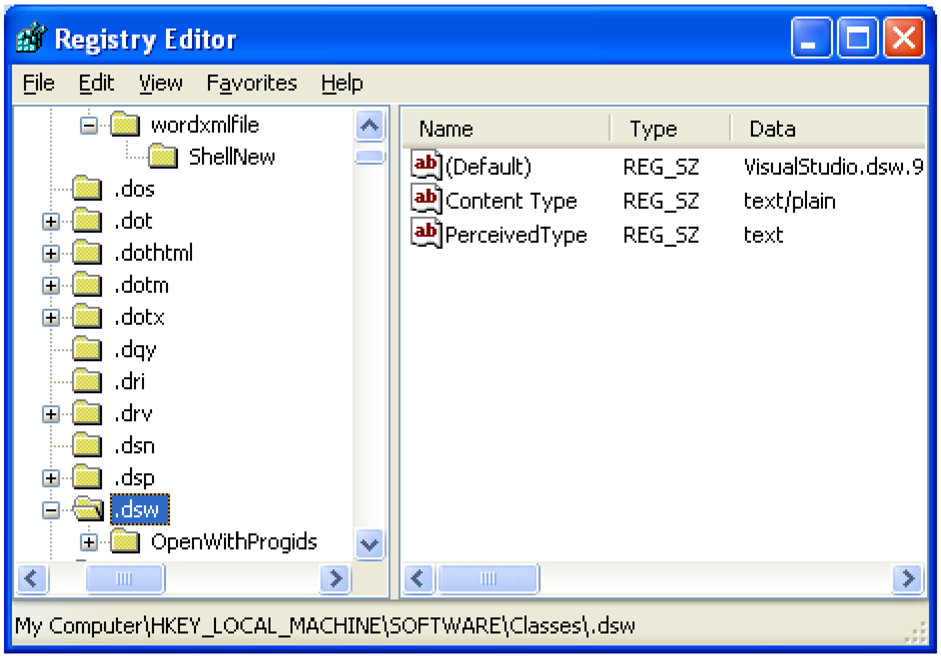
```
HKEY_LOCAL_MACHINE\Software\MyCompany\MyProduct\1.0
```



Name	Type	Data
(Default)	REG_SZ	(value not set)
AutoUpdateTest	REG_SZ	AutoUpdate
CurrentVersion	REG_SZ	10,0,32,18
SwfInstall	REG_SZ	0

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Macromedia\FlashPlayer

If the application supports COM, it should record that data under `HKEY_LOCAL_MACHINE\Software\Classes`.

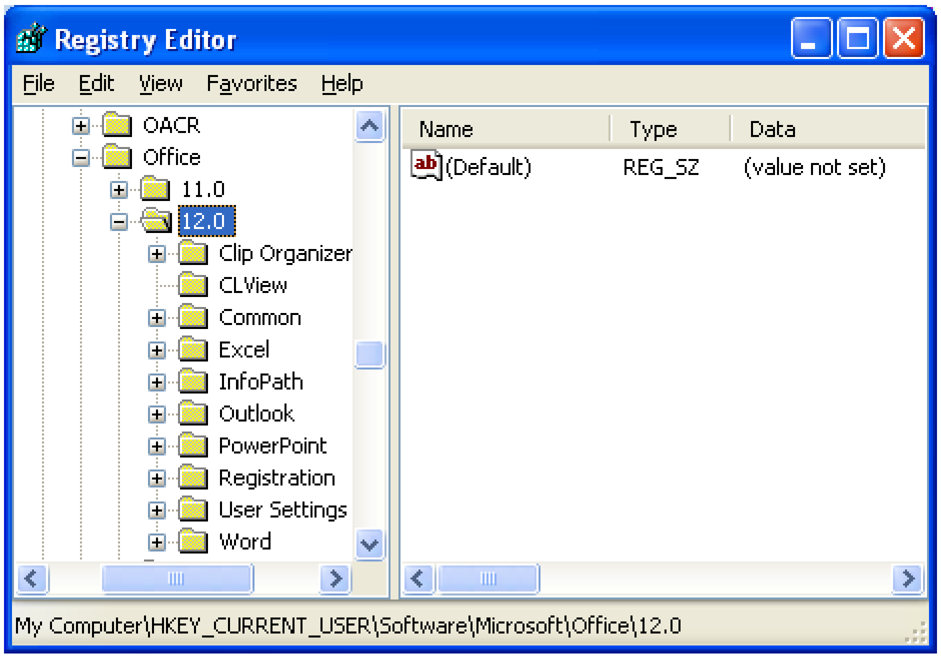


Name	Type	Data
(Default)	REG_SZ	VisualStudio.dsw.9
Content Type	REG_SZ	text/plain
PerceivedType	REG_SZ	text

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Classes\.dsw

An application should record user-specific data under the HKEY_CURRENT_USER key, as shown in the following example:

```
HKEY_CURRENT_USER\Software\MyCompany\MyProduct\1.0
```



Name	Type	Data
(Default)	REG_SZ	(value not set)

My Computer\HKEY_CURRENT_USER\Software\Microsoft\Office\12.0

Opening, Creating, and Closing Keys

Before an application can add data to the registry, it must create or open a key. To create or open a key, an application always refers to the key as a subkey of a currently open key. The following predefined keys are always open:

1. HKEY_LOCAL_MACHINE,
2. HKEY_CLASSES_ROOT,
3. HKEY_USERS, and
4. HKEY_CURRENT_USER

An application uses the RegOpenKeyEx() function to open a key and the RegCreateKeyEx() function to create a key. A registry tree can be 512 levels deep. You can create up to 32 levels at a time through a single registry API call.
An application can use the RegCloseKey() function to close a key and write the data it contains into the registry. RegCloseKey() does not necessarily write the data to the registry before returning; it can take as much as several seconds for the cache to be flushed to the hard disk. If an application must explicitly write registry data to the hard disk, it can use the RegFlushKey() function. RegFlushKey(), however, uses many system resources and should be called only when absolutely necessary.

Writing and Deleting Registry Data

An application can use the RegSetValueEx() function to associate a value and its data with a key. To delete a value from a key, an application can use the RegDeleteValue() function. To delete a key, it can use the RegDeleteKey() function. A deleted key is not removed until the last handle to it has been closed. Subkeys and values cannot be created under a deleted key. It is not possible to lock a registry key during a write operation to synchronize access to the data. However, you can control access to a registry key using security attributes.

Retrieving Data from the Registry

To retrieve data from the registry, an application typically enumerates the subkeys of a key until it finds a particular one and then retrieves data from the value or values associated with it. An application can call the RegEnumKeyEx() function to enumerate the subkeys of a given key.
To retrieve detailed data about a particular subkey, an application can call the RegQueryInfoKey() function. The RegGetKeySecurity() function retrieves a copy of the security descriptor protecting a key. An application can use the RegEnumValue() function to enumerate the values for a given key, and RegQueryValueEx() function to retrieve a particular value for a key. An application typically calls RegEnumValue() to determine the value names and then RegQueryValueEx() to retrieve the data for the names.
The RegQueryMultipleValues() function retrieves the type and data for a list of value names associated with an open registry key. This function is useful for dynamic key providers because it assures consistency of data by retrieving multiple values in an atomic operation.
Because other applications can change the data in a registry value between the time your application can read a value and use it, you may need to ensure your application has the latest data. You can use the RegNotifyChangeKeyValue() function to notify the calling thread when there are changes to the attributes or contents of a registry key, or if the key is deleted. The function signals an event object to notify the caller. If the thread that calls RegNotifyChangeKeyValue() exits, the event is signaled and the monitoring of the registry key is stopped.
You can control or specify what changes should be reported through the use of a notify filter or flag. Usually, changes are reported by signaling an event that you specify to the function. Note that the RegNotifyChangeKeyValue() function does not work with remote handles.

Registry Files

Applications can save part of the registry in a file and then load the contents of the file back into the registry. A registry file is useful when a large amount of data is being manipulated, when many entries are being made in the registry, or when the data is transitory and must be loaded and then unloaded again. Applications that back up and restore parts of the registry are likely to use registry files.

To save a key and its subkeys and values to a registry file, an application can call the RegSaveKey() or RegSaveKeyEx() function.
RegSaveKey() and RegSaveKeyEx() create the file with the archive attribute. The file is created in the current directory of the process for a local key, and in the **%systemroot%\system32** directory for a remote key.
As mentioned before, registry files have the following two formats: standard and latest. The standard format is the only format supported by Windows 2000. It is also supported by later versions of Windows for backward compatibility. RegSaveKey() creates files in the standard format.
The latest format is supported starting with Windows XP. Registry files that are created in this format cannot be loaded on Windows 2000. RegSaveKeyEx() can save registry files in either format by specifying either REG_STANDARD_FORMAT or REG_LATEST_FORMAT. Therefore, it can be used to convert registry files that use the standard format to the latest format.
To write the registry file back to the registry, an application can use the RegLoadKey(), RegReplaceKey(), or RegRestoreKey() functions as follows.

1. RegLoadKey() loads registry data from a specified file into a specified subkey under HKEY_USERS or HKEY_LOCAL_MACHINE on the calling application's computer or on a remote computer. The function creates the specified subkey if it does not already exist. After calling this function, an application can use the RegUnLoadKey() function to restore the registry to its previous state.
2. RegReplaceKey() replaces a key and all its subkeys and values in the registry with the data contained in a specified file. The new data takes effect the next time the system is started.
3. RegRestoreKey() loads registry data from a specified file into a specified key on the calling application's computer or on a remote computer. This function replaces the subkeys and values below the specified key with the subkeys and values that follow the top-level key in the file.

The RegConnectRegistry() function establishes a connection to a predefined registry handle on another computer. An application uses this function primarily to access information from a remote registry on other machines in a network environment, which you can also do by using the Registry Editor. You might want to access a remote registry to back up a registry or regulate network access to it. Note that you must have appropriate permissions to access a remote registry using this function.

Registry Key Security and Access Rights

The Windows security model enables you to control access to registry keys. You can specify a security descriptor for a registry key when you call the RegCreateKeyEx() or

RegSetKeySecurity() function. If you specify NULL, the key gets a default security descriptor. The ACLs in a default security descriptor for a key are inherited from its direct parent key.
To get the security descriptor of a registry key, call the GetNamedSecurityInfo() or GetSecurityInfo() function.
The valid access rights for registry keys include the DELETE, READ_CONTROL, WRITE_DAC, and WRITE_OWNER standard access rights. Registry keys do not support the SYNCHRONIZE standard access right. The following table lists the specific access rights for registry key objects.

Value	Meaning
KEY_ALL_ACCESS (0xF003F)	Combines the STANDARD_RIGHTS_REQUIRED, KEY_QUERY_VALUE, KEY_SET_VALUE, KEY_CREATE_SUB_KEY, KEY_ENUMERATE_SUB_KEYS, KEY_NOTIFY, and KEY_CREATE_LINK access rights.
KEY_CREATE_LINK (0x0020)	Reserved for system use.
KEY_CREATE_SUB_KEY (0x0004)	Required to create a subkey of a registry key.
KEY_ENUMERATE_SUB_KEYS (0x0008)	Required to enumerate the subkeys of a registry key.
KEY_EXECUTE (0x20019)	Equivalent to KEY_READ.
KEY_NOTIFY (0x0010)	Required to request change notifications for a registry key or for subkeys of a registry key.
KEY_QUERY_VALUE (0x0001)	Required to query the values of a registry key.
KEY_READ (0x20019)	Combines the STANDARD_RIGHTS_READ, KEY_QUERY_VALUE, KEY_ENUMERATE_SUB_KEYS, and KEY_NOTIFY values.
KEY_SET_VALUE (0x0002)	Required to create, delete, or set a registry value.
KEY_WOW64_32KEY (0x0200)	Indicates that an application on 64-bit Windows should operate on the 32-bit registry view. This flag must be combined using the OR operator with the other flags in this table that either query or access registry values. Windows 2000: This flag is not supported.
KEY_WOW64_64KEY (0x0100)	Indicates that an application on 64-bit Windows should operate on the 64-bit registry view. This flag must be combined using the OR operator with the other flags in this table that either query or access

	registry values. Windows 2000: This flag is not supported.
KEY_WRITE (0x20006)	Combines the STANDARD_RIGHTS_WRITE, KEY_SET_VALUE, and KEY_CREATE_SUB_KEY access rights.

When you call the RegOpenKeyEx() function, the system checks the requested access rights against the key's security descriptor. If the user does not have the correct access to the registry key, the open operation fails. If an administrator needs access to the key, the solution is to enable the SE_TAKE_OWNERSHIP_NAME privilege and open the registry key with WRITE_OWNER access.
You can request the ACCESS_SYSTEM_SECURITY access right to a registry key if you want to read or write the key's system access control list (SACL). To view the current access rights for a key, including the predefined keys, use the Registry Editor (Regedt32.exe). After navigating to the desired key, go to the Edit menu and select Permissions.

32-bit and 64-bit Application Data in the Registry

On 64-bit Windows, portions of the registry entries are stored separately for 32-bit application and 64-bit applications and mapped into separate logical registry views using the registry redirector and registry reflection, because the 64-bit version of an application may use different registry keys and values than the 32-bit version. There are also shared registry keys that are not redirected or reflected.
The parent of each 64-bit registry node is the **Image-Specific Node** or **ISN**. The registry redirector transparently directs an application's registry access to the appropriate ISN subnode. Redirection subnodes in the registry tree are created automatically by the WOW64 component using the name Wow6432Node. As a result, it is essential not to name any registry key you create Wow6432Node.
The KEY_WOW64_64KEY and KEY_WOW64_32KEY flags enable explicit access to the 64-bit registry view and the 32-bit view, respectively. To disable and enable registry reflection for a particular key, use the RegDisableReflectionKey() and RegEnableReflectionKey() functions. Applications should disable reflection only for the registry keys that they create and not attempt to disable reflection for the predefined keys such as HKEY_LOCAL_MACHINE or HKEY_CURRENT_USER. To determine which keys are on the reflection list, use the RegQueryReflectionKey() function.

Registry Virtualization

Registry virtualization is an **application compatibility technology that enables registry write operations that have global impact to be redirected to per-user locations**. This redirection is

transparent to applications reading from or writing to the registry. It is **supported starting with Windows Vista**.
This form of virtualization is an **interim application compatibility technology**; Microsoft intends to remove it from future versions of the Windows operating system as more applications are made compatible with Windows Vista. Therefore, it is important that your application does not become dependent on the behavior of registry virtualization in the system.

Virtualization Overview

Prior to Windows Vista, applications were **typically run by administrators**. As a result, applications could freely access system files and registry keys. If these applications were run by a standard user, they would fail due to insufficient access rights. Windows Vista improves application compatibility for these applications by **automatically redirecting these operations**. For example, registry operations to the global store (HKEY_LOCAL_MACHINE\Software) are **redirected to a per-user location within the user's profile known as the virtual store** (HKEY_USERS\<User SID>_Classes\VirtualStore\Machine\Software). Registry virtualization can be broadly classified into the following types:

Virtualization Type	Meaning
Open	If the caller does not have write access to a key but opens the key with KEY_ALL_ACCESS, the key is opened with the maximum allowed access for that caller. If the REG_KEY_DONT_SILENT_FAIL flag is disabled for a key, this implicitly disables virtualization for that key.
Write	If the caller does not have write access to a key and attempts to write a value to it or create a subkey, the value is written to the virtual store. For example, if a limited user attempts to write a value to the following key: HKEY_LOCAL_MACHINE\Software\AppKey1, virtualization redirects the write operation to HKEY_USERS\<User SID>_Classes\VirtualStore\Machine\Software\AppKey1.
Read	If the caller reads from a key that is virtualized, the registry presents a merged view of both the virtualized values (from the virtual store) and the non-virtual values (from the global store) to the caller. For example, suppose HKEY_LOCAL_MACHINE\Software\AppKey1 contains two values V1 and V2 and that a limited user writes a value V3 to the key. When the user attempts to read values from this key, the merged view includes values V1 and V2 from the global store and value V3 from the virtual store.

	Note that virtual values take precedence over global values when present. In the example above, even if the global store had value V3 under this key, the value V3 would still be returned to the caller from the virtual store. If V3 were to be deleted from the virtual store, then V3 would be returned from the global store. In other words, if V3 were to be deleted from HKEY_USERS\<User SID>_Classes\VirtualStore\Machine\Software\AppKey1 but HKEY_LOCAL_MACHINE\Software\AppKey1 had a value V3, then that value would be returned from the global store.

Registry Virtualization Scope

Registry virtualization is enabled only for the following:

1. 32-bit interactive processes.
2. Keys in HKEY_LOCAL_MACHINE\Software.
3. Keys that an administrator can write to. (If an administrator cannot write to a key, then the application would have failed on previous versions of Windows even if it was run by an administrator.)

Registry virtualization is disabled for the following:

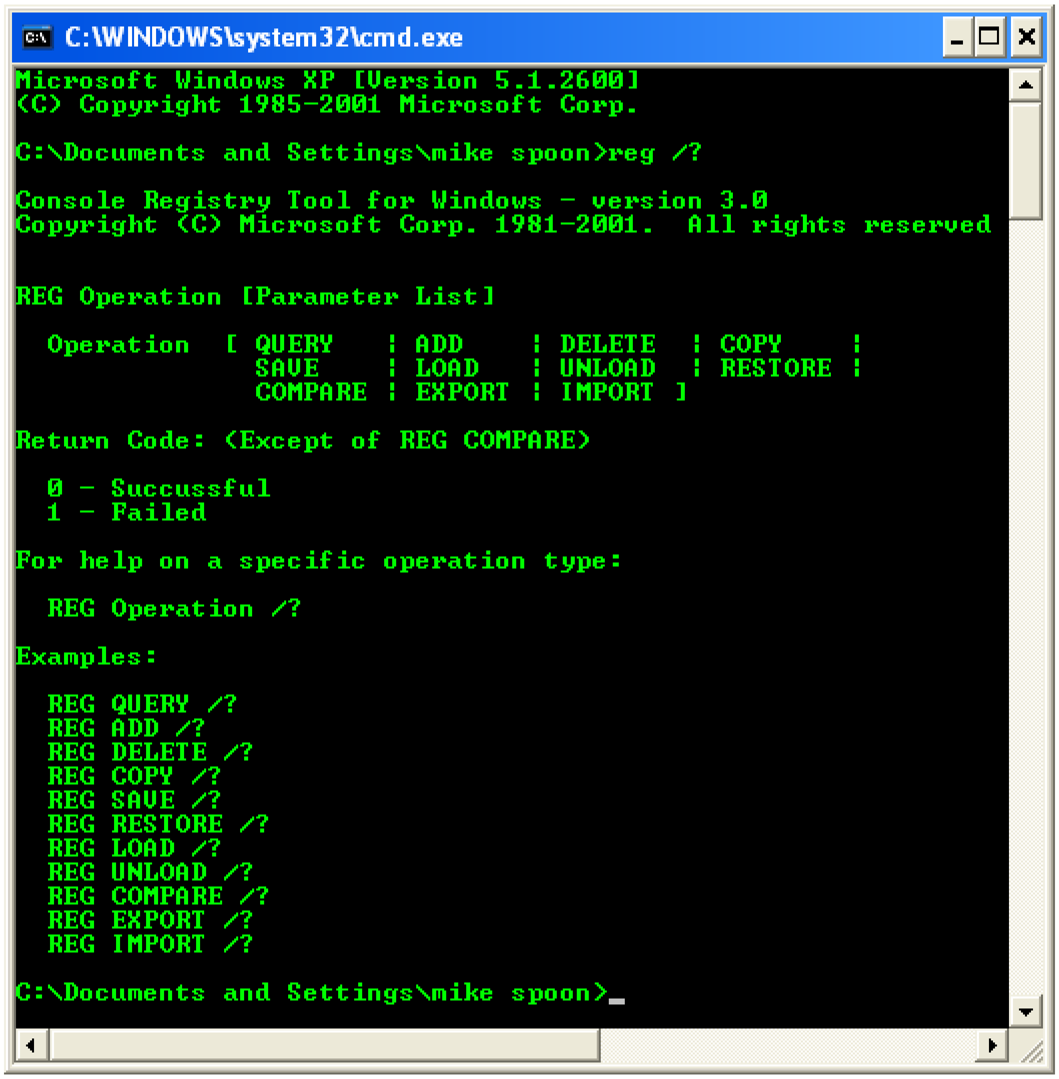
1. 64-bit processes
2. Processes those are not interactive, such as services.
   Note that using the registry as an inter-process communication (IPC) mechanism between a service (or any other process that does not have virtualization enabled) and an application will not work correctly if the key is virtualized. For instance, if an antivirus service updates its signature files based on a value set by an application, the service will never update its signature files because the service reads from the global store but the application writes to the virtual store.
3. Processes that impersonate a user. If a process attempts an operation while impersonating a user, that operation will not be virtualized.
4. Kernel-mode processes such as drivers.
5. Processes that have requestedExecutionLevel specified in their manifests.
6. Keys and subkeys of HKEY_LOCAL_MACHINE\Software\Classes, HKEY_LOCAL_MACHINE\Software\Microsoft\Windows, and HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT.

Controlling Registry Virtualization

In addition to controlling virtualization at an application level by using requestedExecutionLevel in the manifest, you can also enable or disable it on a per-key basis for keys in HKEY_LOCAL_MACHINE\Software. The command line utility **Reg.exe** has a new FLAGS option that administrators can use for this purpose.

```
C:\>reg flags HKLM\Software\AppKey1 QUERY

HKEY_LOCAL_MACHINE\Software\AppKey1

        REG_KEY_DONT_VIRTUALIZE: CLEAR
        REG_KEY_DONT_SILENT_FAIL: CLEAR
        REG_KEY_RECURSE_FLAG: CLEAR
```

The operation completed successfully.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\mike spoon>reg /?

Console Registry Tool for Windows - version 3.0
Copyright (C) Microsoft Corp. 1981-2001.  All rights reserved


REG Operation [Parameter List]

  Operation  [ QUERY   | ADD    | DELETE  | COPY    |
               SAVE    | LOAD   | UNLOAD  | RESTORE |
               COMPARE | EXPORT | IMPORT  ]

Return Code: (Except of REG COMPARE)

  0 - Succussful
  1 - Failed

For help on a specific operation type:

  REG Operation /?

Examples:

  REG QUERY /?
  REG ADD /?
  REG DELETE /?
  REG COPY /?
  REG SAVE /?
  REG RESTORE /?
  REG LOAD /?
  REG UNLOAD /?
  REG COMPARE /?
  REG EXPORT /?
  REG IMPORT /?

C:\Documents and Settings\mike spoon>_
```

Whenever auditing is enabled on a key that is being virtualized, a new virtualization audit event is generated to indicate that the key is being virtualized (addition to the usual audit events). Administrators can use this information to monitor the status of virtualization on their systems. Virtualization is intended only to provide compatibility for existing applications. Applications designed for Windows Vista should not write to sensitive system areas, nor should they rely on virtualization to remedy any problems. When updating existing code to run on Windows Vista, developers should ensure that applications only store data in per-user locations or in computer locations within **%alluserprofile%** that properly use an access control list (ACL).

Registry Element Size Limits

The following table identifies the size limits for the various registry elements.

Registry Element	Size Limit
Key name	255 characters
Value name	16,383 characters Windows 2000: 260 ANSI characters or 16,383 Unicode characters.
Value	Available memory (latest format) 1 MB (standard format)
Tree	A registry tree can be 512 levels deep. You can create up to 32 levels at a time through a single registry API call.

Long values (more than 2,048 bytes) should be stored in a file, and the location of the file should be stored in the registry. This helps the registry perform efficiently.
The file location can be the name of a value or the data of a value. Each backslash in the location string must be preceded by another backslash as an escape character. For example, specify "C:\\mydir\\myfile" to store the string "C:\mydir\myfile". A file location can also be the name of a key if it is within the 255-character limit for key names and does not contain backslashes, which are not allowed in key names.

Registry Value Types

A registry value can store data in various formats. When you store data under a registry value, for instance by calling the RegSetValueEx() function, you can specify one of the following values to indicate the type of data being stored. When you retrieve a registry value, functions such as RegQueryValueEx() use these values to indicate the type of data retrieved. The following registry value types are defined in Winnt.h.

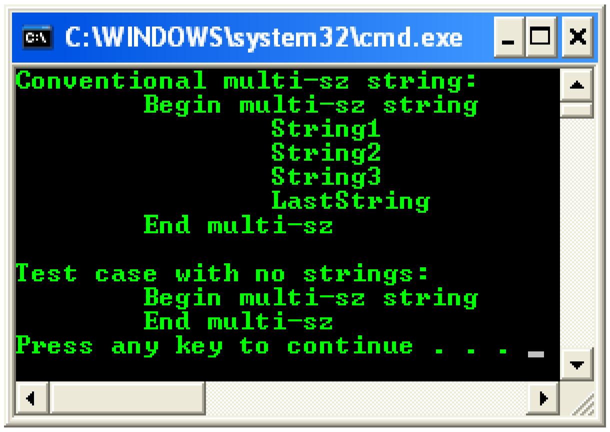
Value	Type
REG_BINARY	Binary data in any form.
REG_DWORD	A 32-bit number.
REG_DWORD_LITTLE_ENDIAN	A 32-bit number in little-endian format. Windows is designed to run on little-endian computer architectures. Therefore, this value is defined as REG_DWORD in the Windows header files.
REG_DWORD_BIG_ENDIAN	A 32-bit number in big-endian format. Some UNIX systems support big-endian architectures.
REG_EXPAND_SZ	A null-terminated string that contains unexpanded references to environment variables (for example, "%PATH%"). It will be a Unicode or ANSI string depending on whether you use the Unicode or ANSI

	functions. To expand the environment variable references, use the ExpandEnvironmentStrings() function.
REG_LINK	A null-terminated Unicode string that contains the target path of a symbolic link that was created by calling the RegCreateKeyEx() function with REG_OPTION_CREATE_LINK.
REG_MULTI_SZ	A sequence of null-terminated strings, terminated by an empty string (\0). The following is an example: String1\0String2\0String3\0LastString\0\0 The first \0 terminates the first string, the second to the last \0 terminates the last string, and the final \0 terminates the sequence. Note that the final terminator must be factored into the length of the string.
REG_NONE	No defined value type.
REG_QWORD	A 64-bit number.
REG_QWORD_LITTLE_ENDIAN	A 64-bit number in little-endian format. Windows is designed to run on little-endian computer architectures. Therefore, this value is defined as REG_QWORD in the Windows header files.
REG_SZ	A null-terminated string. This will be either a Unicode or an ANSI string, depending on whether you use the Unicode or ANSI functions.

String Values

If data has the REG_SZ, REG_MULTI_SZ, or REG_EXPAND_SZ type, the string may not have been stored with the proper terminating null characters. Therefore, when reading a string from the registry, you must ensure that the string is properly terminated before using it; otherwise, it may overwrite a buffer. (Note that REG_MULTI_SZ strings should have two terminating null characters.)
When writing a string to the registry, you must specify the length of the string, including the terminating null character (\0). A common error is to use the strlen function to determine the length of the string, but to forget that strlen returns only the number of characters in the string, not including the terminating null. Therefore, the length of the string should be calculated as follows: `wcslen(string) + 1`
A REG_MULTI_SZ string ends with a string of length 0. Therefore, it is not possible to include a zero-length string in the sequence. An empty sequence would be defined as follows: \0. The following example walks a REG_MULTI_SZ string.

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

void SampleSzz(PTSTR pszz)
{
   wprintf(L"\tBegin multi-sz string\n");

   while (*pszz)
   {
      wprintf(L"\t\t%s\n", pszz);
      pszz = pszz + wcslen(pszz) + 1;
   }
   wprintf(L"\tEnd multi-sz\n");
}

int wmain(int argc, WCHAR **argv)
{
   // Because the compiler adds a \0 at the end of quoted strings,
   // there are two \0 terminators at the end
   wprintf(L"Conventional multi-sz string:\n");
   SampleSzz(L"String1\0String2\0String3\0LastString\0");

   wprintf(L"\nTest case with no strings:\n");
   SampleSzz(L"");

   return 0;
}
```



```
Conventional multi-sz string:
        Begin multi-sz string
                String1
                String2
                String3
                LastString
        End multi-sz

Test case with no strings:
        Begin multi-sz string
        End multi-sz
Press any key to continue . . .
```

Byte Formats

In little-endian format, a multi-byte value is stored in memory from the lowest byte (the "little end") to the highest byte. For example, the value 0x12345678 is stored as (0x78 0x56 0x34 0x12) in little-endian format.

In big-endian format, a multi-byte value is stored in memory from the highest byte (the "big end") to the lowest byte. For example, the value 0x12345678 is stored as (0x12 0x34 0x56 0x78) in big-endian format.

---------------------------------------o0o---

Using the Registry: Program Examples

The following sample code demonstrates how to use the registry functions.

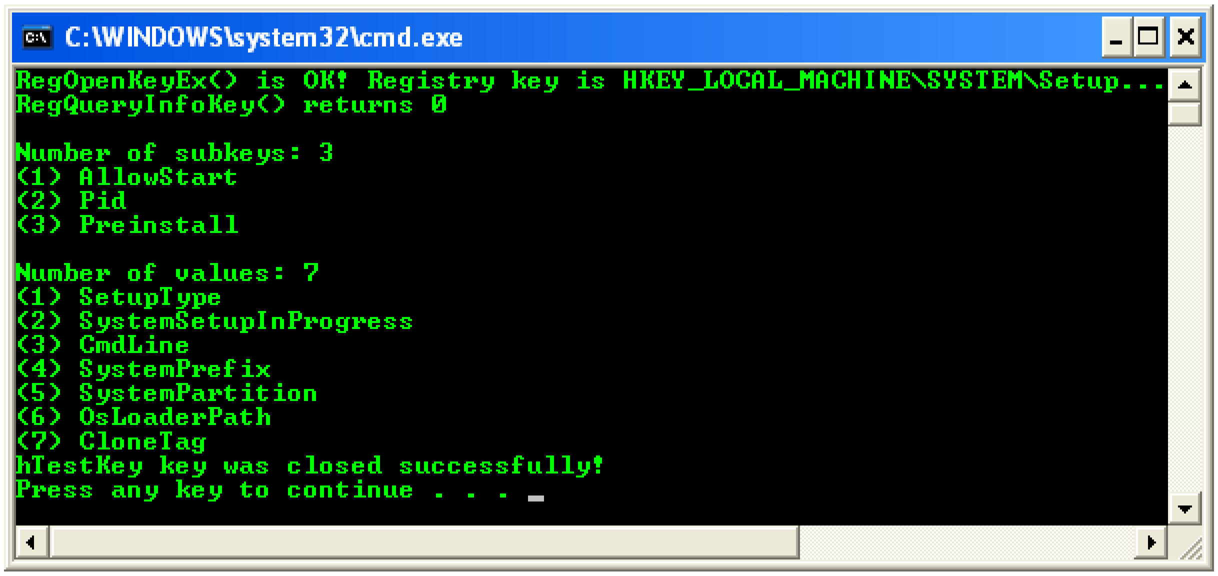
1. Enumerating Registry Subkeys Program Example
2. Creating the Registry Subkey and Value Program Example
3. Deleting Registry Key with Subkeys Program Example
4. Determining the Registry Size Program Example
5. Querying the Registry Value Program Example

Enumerating Registry Subkeys Program Example

The following example uses the RegQueryInfoKey(), RegEnumKeyEx(), and RegEnumValue() functions to enumerate the subkeys of the specified key. The hKey parameter passed to each function is a handle to an open key. This key must be opened before the function call and closed afterward.
Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.
Then, add the source file and give it a suitable name.
Next, add the following source code.

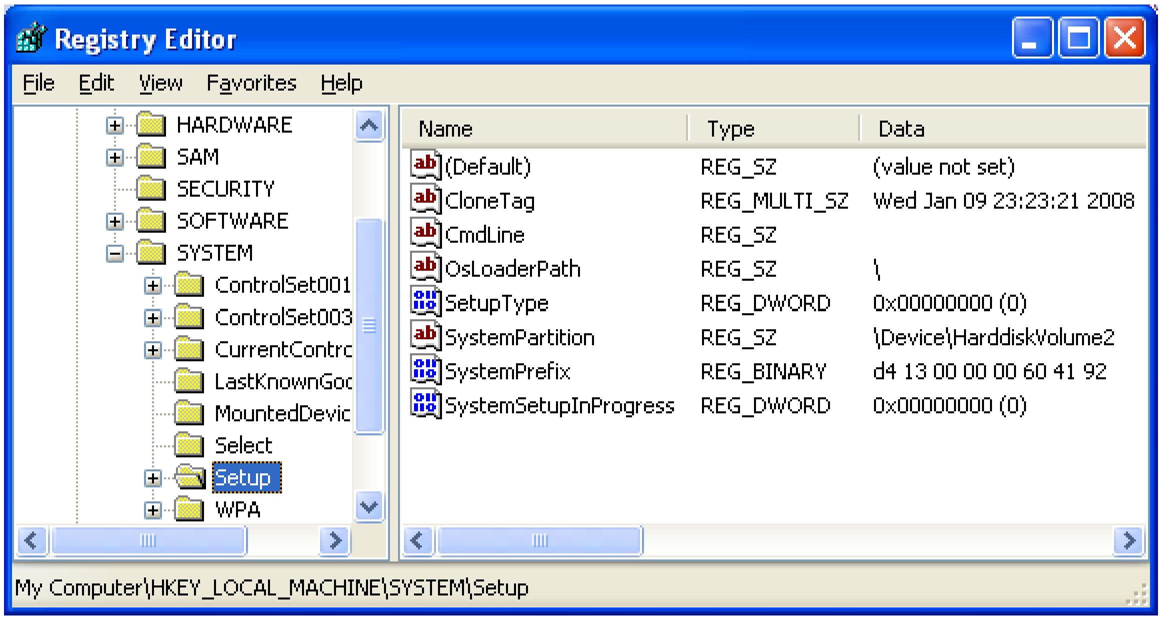
```
// QueryKey - Enumerates the subkeys of key and its associated values.
//     hKey - Key whose subkeys and values are to be enumerated.
#include <windows.h>
#include <stdio.h>

#define MAX_KEY_LENGTH 255
#define MAX_VALUE_NAME 16383

void QueryKey(HKEY hKey)
{
    WCHAR    achKey[MAX_KEY_LENGTH];   // buffer for subkey name
    DWORD    cbName;                   // size of name string
    WCHAR    achClass[MAX_PATH] = TEXT("");  // buffer for class name
    DWORD    cchClassName = MAX_PATH;  // size of class string
    DWORD    cSubKeys=0;               // number of subkeys
    DWORD    cbMaxSubKey;              // longest subkey size
    DWORD    cchMaxClass;              // longest class string
    DWORD    cValues;              // number of values for key
    DWORD    cchMaxValue;          // longest value name
    DWORD    cbMaxValueData;       // longest value data
    DWORD    cbSecurityDescriptor; // size of security descriptor
    FILETIME ftLastWriteTime;      // last write time

    DWORD i, retCode;

    WCHAR  achValue[MAX_VALUE_NAME];
    DWORD cchValue = MAX_VALUE_NAME;

    // Get the class name and the value count.
    retCode = RegQueryInfoKey(
        hKey,                    // key handle
        achClass,                // buffer for class name
        &cchClassName,           // size of class string
        NULL,                    // reserved
        &cSubKeys,               // number of subkeys
        &cbMaxSubKey,            // longest subkey size
        &cchMaxClass,            // longest class string
        &cValues,                // number of values for this key
        &cchMaxValue,            // longest value name
        &cbMaxValueData,         // longest value data
        &cbSecurityDescriptor,   // security descriptor
        &ftLastWriteTime);       // last write time

      wprintf(L"RegQueryInfoKey() returns %u\n", retCode);

    // Enumerate the subkeys, until RegEnumKeyEx() fails
    if (cSubKeys)
    {
        wprintf(L"\nNumber of subkeys: %d\n", cSubKeys);

        for (i=0; i<cSubKeys; i++)
        {
            cbName = MAX_KEY_LENGTH;
            retCode = RegEnumKeyEx(hKey,
i,achKey,&cbName,NULL,NULL,NULL,&ftLastWriteTime);
            if (retCode == ERROR_SUCCESS)
            {
                wprintf(L"(%d) %s\n", i+1, achKey);
            }
        }
    }
      else
            wprintf(L"No subkeys to be enumerated!\n");

    // Enumerate the key values
    if (cValues)
    {
        wprintf(L"\nNumber of values: %d\n", cValues);

        for (i=0, retCode=ERROR_SUCCESS; i<cValues; i++)
        {
            cchValue = MAX_VALUE_NAME;
            achValue[0] = '\0';
            retCode = RegEnumValue(hKey, i, achValue, &cchValue, NULL, NULL, 
NULL, NULL);

            if (retCode == ERROR_SUCCESS)
            {
                wprintf(L"(%d) %s\n", i+1, achValue);
            }
        }
    }
      else
            wprintf(L"No values to be enumerated!\n");
}

int wmain(int argc, WCHAR *argv[])
{
   HKEY hTestKey;

   // Change the key and subkey accordingly...
   // In this case: HKEY_USERS\\S-1-5-18\\...
   if(RegOpenKeyEx(HKEY_LOCAL_MACHINE,  L"SYSTEM\\Setup" /*L"S-1-5-18"*/,  0, 
KEY_READ,  &hTestKey) == ERROR_SUCCESS)
   // if(RegOpenKeyEx(HKEY_LOCAL_MACHINE,  L"SYSTEM\\WPA",  0, KEY_READ, 
&hTestKey) == ERROR_SUCCESS)
   {
         wprintf(L"RegOpenKeyEx() is OK! Registry key is 
HKEY_LOCAL_MACHINE\\SYSTEM\\Setup...\n");
         QueryKey(hTestKey);
   }
   else
         wprintf(L"RegOpenKeyEx() failed!\n");

   if(RegCloseKey(hTestKey) == ERROR_SUCCESS)
         wprintf(L"hTestKey key was closed successfully!\n");
   else
         wprintf(L"Fail to close hTestKey key!\n");

   return 0;
}
```

Build and run the project. The following screenshot is a sample output.



```
RegOpenKeyEx() is OK! Registry key is HKEY_LOCAL_MACHINE\SYSTEM\Setup...
RegQueryInfoKey() returns 0

Number of subkeys: 3
(1) AllowStart
(2) Pid
(3) Preinstall

Number of values: 7
(1) SetupType
(2) SystemSetupInProgress
(3) CmdLine
(4) SystemPrefix
(5) SystemPartition
(6) OsLoaderPath
(7) CloneTag
hTestKey key was closed successfully!
Press any key to continue . . . _
```

Verify the result against the real registry keys.



Name	Type	Data
(Default)	REG_SZ	(value not set)
CloneTag	REG_MULTI_SZ	Wed Jan 09 23:23:21 2008
CmdLine	REG_SZ	
OsLoaderPath	REG_SZ	\
SetupType	REG_DWORD	0x00000000 (0)
SystemPartition	REG_SZ	\Device\HarddiskVolume2
SystemPrefix	REG_BINARY	d4 13 00 00 00 60 41 92
SystemSetupInProgress	REG_DWORD	0x00000000 (0)

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\Setup

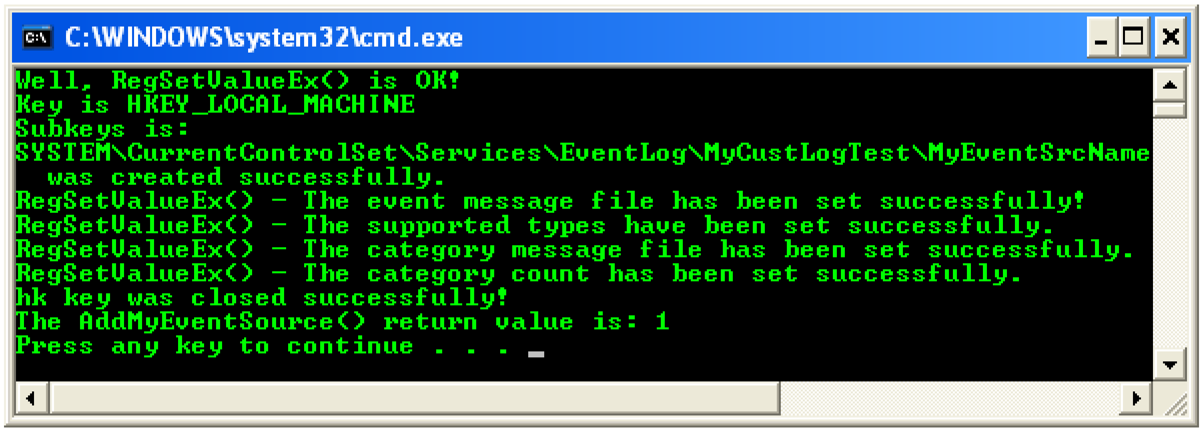
Creating the Registry Subkey and Value Program Example

In the following program example, we try to open the registry key, create subkeys and assign values to the subkeys.
Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.
Then, add the source file and give it a suitable name.

Next, add the following source code.

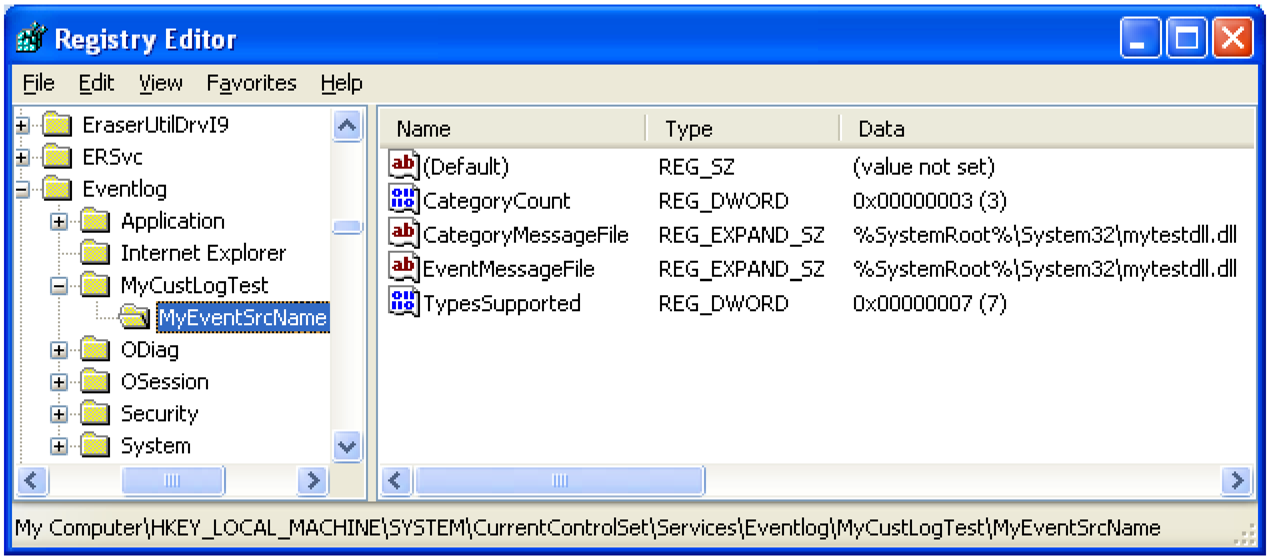
```
// WARNING!!!
// If you don't know what you are doing, please don't try
// this code...and don't forget to delete the key or use
// RegDeleteKey()...

// #define _WIN32_WINNT 0x0502   // Windows Server 2003 family
// Other Win OS, please change accordingly...
#define _WIN32_WINNT 0x0501           // For Win Xp
// #define _WIN32_WINNT 0x0500   // Windows 2000
// #define _WIN32_WINNT 0x0400   // Windows NT 4.0
// #define _WIN32_WINDOWS 0x0500 // Windows ME
// #define _WIN32_WINDOWS 0x0410 // Windows 98
// #define _WIN32_WINDOWS 0x0400 // Windows 95

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

BOOL AddMyEventSource(

   LPTSTR pszLogName, // Application log or a custom log
   LPTSTR pszSrcName, // event source name
   LPTSTR pszMsgDLL,  // path for message DLL
   DWORD  dwNum)      // number of categories
{
   HKEY hk;
   DWORD dwData;
   TCHAR szBuf[MAX_PATH];

   // Create the event source as a subkey of the log.
   wsprintf(szBuf, L"SYSTEM\\CurrentControlSet\\Services\\EventLog\\%s\\%s",
pszLogName, pszSrcName);

   //*********************************************
   // Create the registry key
   if(RegCreateKey(HKEY_LOCAL_MACHINE, szBuf, &hk) != ERROR_SUCCESS)
   {
      wprintf(L"RegCreateKey() - Could not create the registry key.");
      return FALSE;
   }
   else
   {
         wprintf(L"Well, RegSetValueEx() is OK!\n");
         wprintf(L"Key is HKEY_LOCAL_MACHINE\n");
         wprintf(L"Subkeys is: \n");
         wprintf(L"SYSTEM\\CurrentControlSet\\Services\\EventLog\\%s\\%s\n
was created successfully.\n", pszLogName, pszSrcName);
   }

   //*********************************************
   // Set the name of the message file
    if(RegSetValueEx(hk,                          // subkey handle
           L"EventMessageFile",                   // value name
           0,                                // must be zero
           REG_EXPAND_SZ,              // value type
           (LPBYTE) pszMsgDLL,         // pointer to value data
           (DWORD) lstrlen(szBuf)+1)   // length of value data
               != ERROR_SUCCESS)
   {
      wprintf(L"RegSetValueEx() - Could not set the event message file.");
      return FALSE;
   }
      else
      wprintf(L"RegSetValueEx() - The event message file has been set
successfully!\n");

   // Set the supported event types.
   dwData = EVENTLOG_ERROR_TYPE | EVENTLOG_WARNING_TYPE |
EVENTLOG_INFORMATION_TYPE;


   //*********************************************
   if(RegSetValueEx(hk,       // subkey handle
           L"TypesSupported", // value name
           0,                 // must be zero
           REG_DWORD,         // value type
           (LPBYTE) &dwData,  // pointer to value data
           sizeof(DWORD))     // length of value data
               != ERROR_SUCCESS)

   {
      wprintf(L"RegSetValueEx() - Could not set the supported types.");
      return FALSE;
   }
   else
      wprintf(L"RegSetValueEx() - The supported types have been set
successfully.\n");

   //*********************************************************
   // Set the category message file and number of categories.
   if(RegSetValueEx(hk,                // subkey handle
           L"CategoryMessageFile",     // value name
           0,                          // must be zero
           REG_EXPAND_SZ,              // value type
           (LPBYTE) pszMsgDLL,         // pointer to value data here we set
same as "EventMessageFile"
           (DWORD) lstrlen(szBuf)+1) // length of value data
               != ERROR_SUCCESS)
   {
      wprintf(L"RegSetValueEx() - Could not set the category message file.");
      return FALSE;
   }
   else
       wprintf(L"RegSetValueEx() - The category message file has been set
successfully.\n");

   //*********************************************
   if(RegSetValueEx(hk,       // subkey handle
            L"CategoryCount",   // value name
            0,                  // must be zero
            REG_DWORD,          // value type
            (LPBYTE) &dwNum,    // pointer to value data
            sizeof(DWORD))      // length of value data
                == ERROR_SUCCESS)

                wprintf(L"RegSetValueEx() - The category count has been set
successfully.\n");
   else
   {
      wprintf(L"RegSetValueEx() - Could not set the category count.");
      return FALSE;
   }

   // Close the key
   if(RegCloseKey(hk) == ERROR_SUCCESS)
         wprintf(L"hk key was closed successfully!\n");
   else
         wprintf(L"Failed to close hk key!\n");

   return TRUE;
}

int wmain(int argc, WCHAR *argv[])
{
      // Application log or a custom log. Here we put a custom log just for
learning!
      LPTSTR pszLogName = L"MyCustLogTest";

   // The event source name
   LPTSTR pszSrcName = L"MyEventSrcName";

   // The path for message dll, this dll or other executable file must exist
lol!
   // here, mytestdll.dll just a dummy. You will know it when you restart
   // your computer if the created key does not deleted...:o)
   LPTSTR pszMsgDLL = L"%SystemRoot%\\System32\\mytestdll.dll";

   // number of categories.
   DWORD  dwNum = 0x00000003;

   BOOL test = AddMyEventSource(
                    pszLogName, // Application log or a custom log.  Custom
log here...
                    pszSrcName, // event source name.
                    pszMsgDLL,  // path for message DLL.
                    dwNum           // number of categories.
                    );

   // Just to check the return value...
   // 0 - failed, non-zero should be  fine
   wprintf(L"The AddMyEventSource() return value is: %u\n", test);
   return 0;
}
```

Build and run the project. The following screenshot is a sample output.



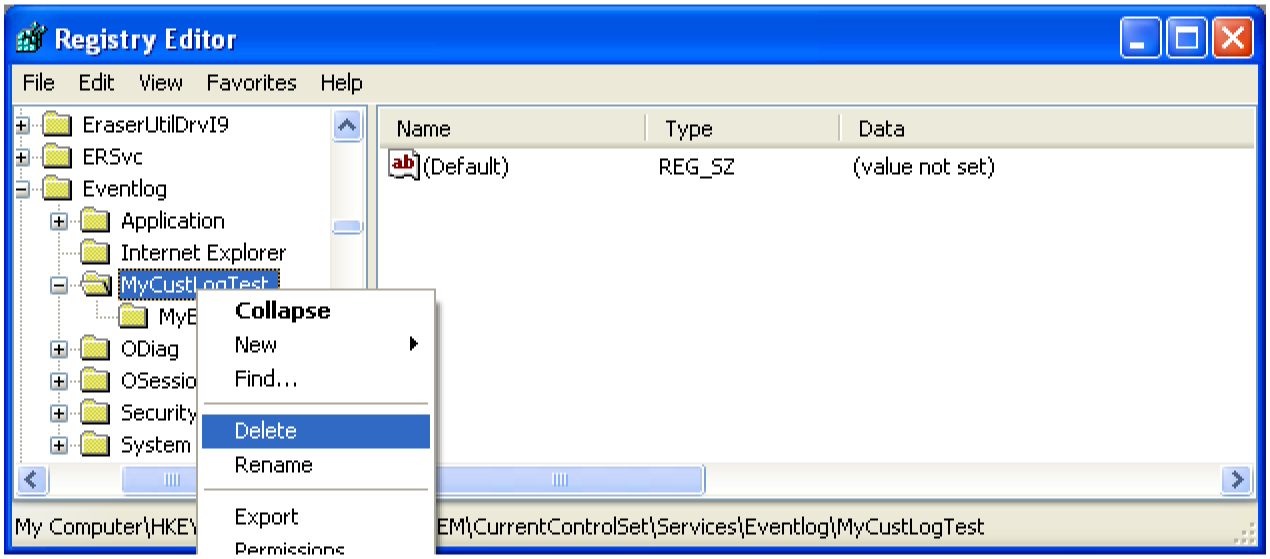
```
Well, RegSetValueEx() is OK!
Key is HKEY_LOCAL_MACHINE
Subkeys is:
SYSTEM\CurrentControlSet\Services\EventLog\MyCustLogTest\MyEventSrcName
  was created successfully.
RegSetValueEx() - The event message file has been set successfully!
RegSetValueEx() - The supported types have been set successfully.
RegSetValueEx() - The category message file has been set successfully.
RegSetValueEx() - The category count has been set successfully.
hk key was closed successfully!
The AddMyEventSource() return value is: 1
Press any key to continue . . . _
```

Do a verification using Registry Editor. Please delete the created registry key and subkeys.



Name	Type	Data
(Default)	REG_SZ	(value not set)
CategoryCount	REG_DWORD	0x00000003 (3)
CategoryMessageFile	REG_EXPAND_SZ	%SystemRoot%\System32\mytestdll.dll
EventMessageFile	REG_EXPAND_SZ	%SystemRoot%\System32\mytestdll.dll
TypesSupported	REG_DWORD	0x00000007 (7)

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\MyCustLogTest\MyEventSrcName

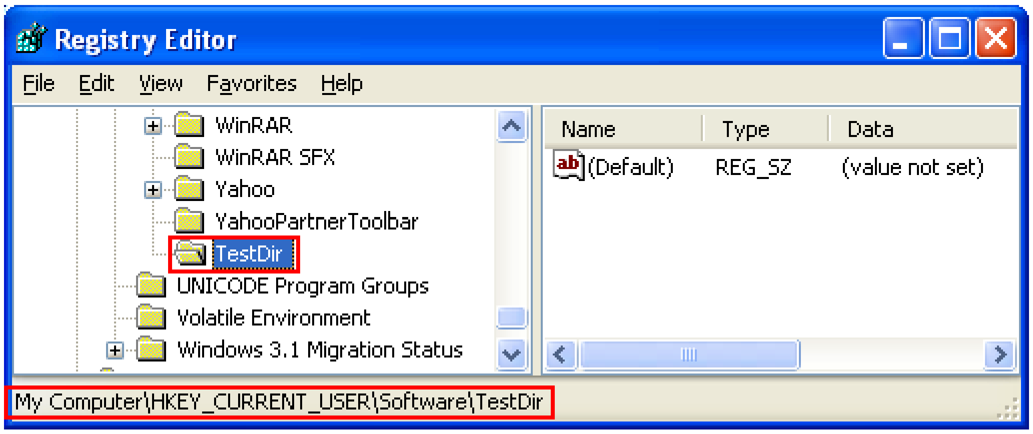


My Computer\HKEY...EM\CurrentControlSet\Services\Eventlog\MyCustLogTest

Deleting Registry Key with Subkeys Program Example

The following program example uses the RegOpenKeyEx(), RegEnumKeyEx(), and RegDeleteKey() functions to delete a registry key with subkeys. To test this example, create the following registry key by using Regedt32.exe manually, and then add a few values and subkeys if you want.

```
HKEY_CURRENT_USER\Software\TestDir
```



My Computer\HKEY_CURRENT_USER\Software\TestDir

Or you can modify the related part of the source code to delete the previously created registry key.
Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.
Then, add the source file and give it a suitable name.

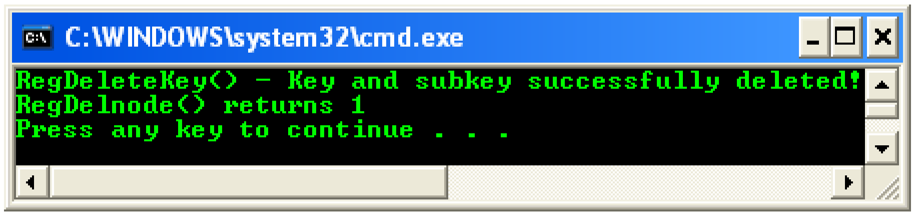
Next, add the following source code.

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

//*************************************************************
//  RegDelnodeRecurse()
//  Purpose:    Deletes a registry key and all it's subkeys / values.
//  Parameters: hKeyRoot    -   Root key
//              lpSubKey    -   SubKey to delete
//  Return:     TRUE if successful.
//              FALSE if an error occurs.
//*************************************************************
BOOL RegDelnodeRecurse(HKEY hKeyRoot, LPTSTR lpSubKey)
{
    LPTSTR lpEnd;
    LONG lResult;
    DWORD dwSize;
    WCHAR szName[MAX_PATH];
    HKEY hKey;
    FILETIME ftWrite;

    // First, see if we can delete the key without having to recurse
    lResult = RegDeleteKey(hKeyRoot, lpSubKey);

    if (lResult == ERROR_SUCCESS)
      {
            wprintf(L"RegDeleteKey() - Key and subkey successfully
deleted!\n");
        return TRUE;
      }
      else
      {
         wprintf(L"RegDeleteKey() - Failed to delete key and subkey! Error
%d.\n", GetLastError());
         wprintf(L"Trying again..\n");
      }

    lResult = RegOpenKeyEx(hKeyRoot, lpSubKey, 0, KEY_READ, &hKey);

    if (lResult != ERROR_SUCCESS)
    {
        if (lResult == ERROR_FILE_NOT_FOUND)
            {
            wprintf(L"RegOpenKeyEx() - Key not found!\n");
            return TRUE;
        }
        else
            {
            wprintf(L"RegOpenKeyEx() - Error opening key, error %d\n",
GetLastError());
            return FALSE;
        }
    }
```

```
      else
            wprintf(L"RegOpenKeyEx() - Key opened successfully!\n");

    // Check for an ending slash and add one if it is missing
    lpEnd = lpSubKey + lstrlen(lpSubKey);

    if (*(lpEnd - 1) != L'\\')
    {
        *lpEnd =  L'\\';
        lpEnd++;
        *lpEnd =  L'\0';
    }

    // Enumerate the keys
    dwSize = MAX_PATH;
    lResult = RegEnumKeyEx(hKey, 0, szName, &dwSize, NULL, NULL, NULL,
&ftWrite);

    if (lResult == ERROR_SUCCESS)
    {
            wprintf(L"RegEnumKeyEx() is pretty fine!\n");
        do {
            StringCchCopy(lpEnd, MAX_PATH*2, szName);

            if (!RegDelnodeRecurse(hKeyRoot, lpSubKey))
                  {
                break;
            }

            dwSize = MAX_PATH;
            lResult = RegEnumKeyEx(hKey, 0, szName, &dwSize, NULL,  NULL,
NULL, &ftWrite);

        } while (lResult == ERROR_SUCCESS);
    }
      else
            wprintf(L"RegEnumKeyEx() failed lol!\n");

    lpEnd--;
    *lpEnd = L'\0';

    if(RegCloseKey(hKey) == ERROR_SUCCESS)
            wprintf(L"hKey key was closed successfully!\n");
      else
            wprintf(L"Failed to close hKey key!\n");

    // Try again to delete the key.
    lResult = RegDeleteKey(hKeyRoot, lpSubKey);

    if (lResult == ERROR_SUCCESS)
      {
            wprintf(L"RegDeleteKey() is OK!\n");
        return TRUE;
      }
      else
```

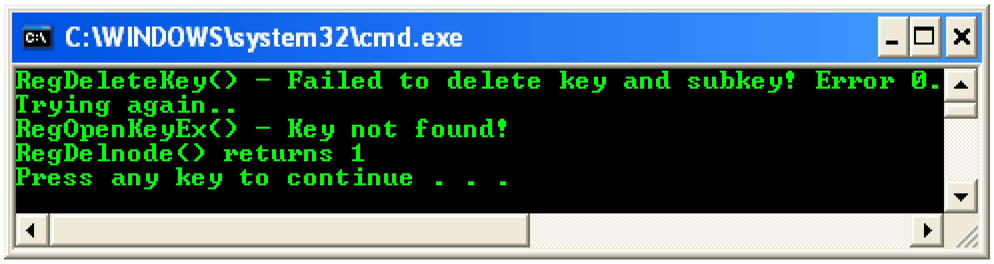
```
            wprintf(L"RegDeleteKey() failed!\n");

    return FALSE;
}

//*************************************************************
//  RegDelnode()
//  Purpose:    Deletes a registry key and all it's subkeys / values.
//  Parameters: hKeyRoot    -   Root key
//              lpSubKey    -   SubKey to delete
//  Return:     TRUE if successful.
//              FALSE if an error occurs.
//*************************************************************
BOOL RegDelnode(HKEY hKeyRoot, LPTSTR lpSubKey)
{
    WCHAR szDelKey[MAX_PATH*2];

    StringCchCopy(szDelKey, MAX_PATH*2, lpSubKey);
      // Recurse starts from root key, HKEY_CLASSES_ROOT
    return RegDelnodeRecurse(hKeyRoot, szDelKey);
}

int wmain(int argc, WCHAR *argv[])
{
      BOOL bRetVal;

      // The first two are the key and subkeys for the previous program
example. Note the variation
      // bRetVal = RegDelnode(HKEY_LOCAL_MACHINE,
L"SYSTEM\\CurrentControlSet\\Services\\Eventlog\\MyCustLogTest\\MyEventSrcNam
e");
      // bRetVal = RegDelnode(HKEY_LOCAL_MACHINE,
L"SYSTEM\\CurrentControlSet\\Services\\Eventlog\\MyCustLogTest");

      // We directly give the desired key to be deleted, shouldn't to recurse
      bRetVal = RegDelnode(HKEY_CURRENT_USER, L"Software\\TestDir");
      // 0 - FALSE, Non-zero - TRUE
      wprintf(L"RegDelnode() returns %d\n", bRetVal);
      return 0;
}
```

Build and run the project. The following screenshot is a sample output.
After running the following code, use the F5 key to refresh (Registry Editor) the registry data, and notice that the TestDir key was deleted.



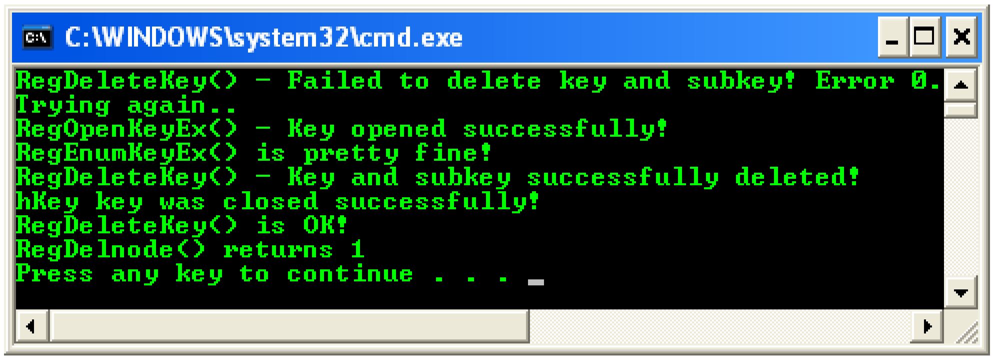
```
RegDeleteKey() - Key and subkey successfully deleted!
RegDelnode() returns 1
Press any key to continue . . .
```

Running the program second time generates the following output.



```
RegDeleteKey() - Failed to delete key and subkey! Error 0.
Trying again..
RegOpenKeyEx() - Key not found!
RegDelnode() returns 1
Press any key to continue . . .
```

The commented lines in the wmain() are the key and subkeys for the previous program example. Please try it.



```
RegDeleteKey() - Failed to delete key and subkey! Error 0.
Trying again..
RegOpenKeyEx() - Key opened successfully!
RegEnumKeyEx() is pretty fine!
RegDeleteKey() - Key and subkey successfully deleted!
hKey key was closed successfully!
RegDeleteKey() is OK!
RegDelnode() returns 1
Press any key to continue . . .
```

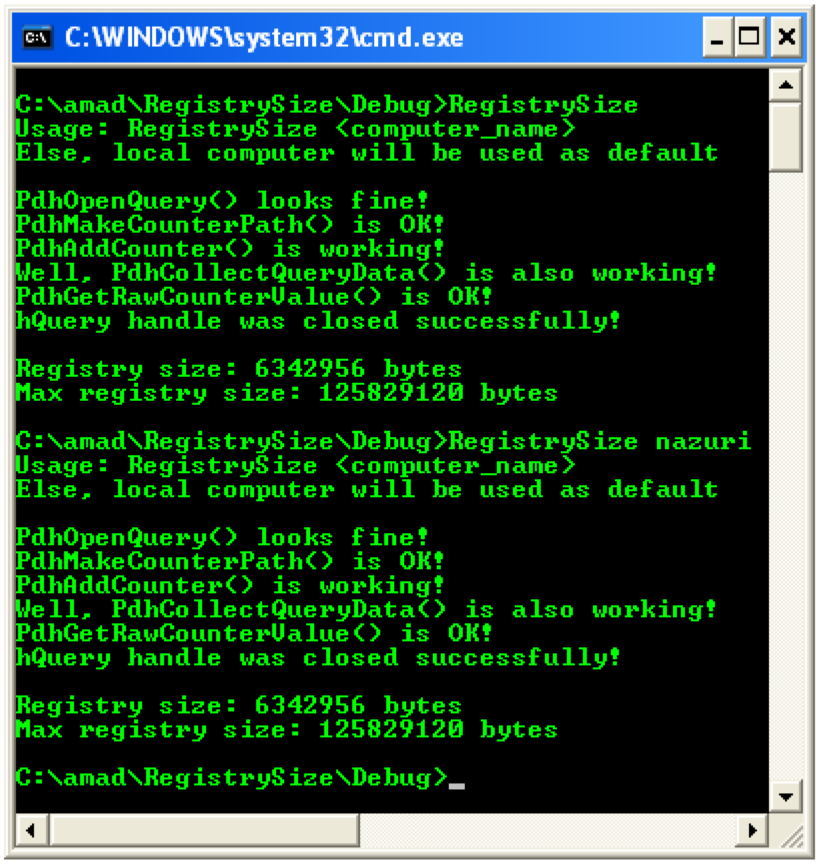
Determining the Registry Size Program Example

On Windows 2000, it is common for an installation utility to check the current and maximum size of the registry to determine whether there is enough available space for the new data it will be adding. The following program example demonstrates how to do this programmatically using the "% Registry Quota In Use" performance counter within the System object.
The following sample uses performance data helper (PDH) to obtain the counter value; it must be linked with Pdh.lib. PDH is a high-level set of APIs used to obtain performance data.
Note that it is not necessary to implement this registry size-check on Windows Server 2003 or Windows XP because they do not have a registry quota limit.
Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.
Then, add the source file and give it a suitable name.
Next, add the following source code.

```
//*****************************************************
//  Determines the current and maximum registry size.
//*****************************************************
#include <windows.h>
#include <stdio.h>
```

```
// Don't forget to link to pdh.lib library
#include <pdh.h>

PDH_STATUS GetRegistrySize(LPTSTR szMachineName, LPDWORD lpdwCurrentSize,
LPDWORD lpdwMaximumSize);

//*************************************************************
//  Entry point for the program. This function demonstrates how to
//  use the GetRegistrySize function implemented below.
//  It will use the first argument, if present, as the name of the
//  computer whose registry you wish to determine. If unspecified,
//  it will use the local computer.
//*************************************************************
int wmain(int argc, WCHAR *argv[])
{
    LPTSTR      szMachineName  = NULL;
    PDH_STATUS  pdhStatus      = 0;
    DWORD       dwCurrent      = 0;
    DWORD       dwMaximum      = 0;
      LPTSTR szMessage;

      wprintf(L"Usage: %s <computer_name>\n", argv[0]);
      wprintf(L"Else, local computer will be used as default\n\n");

    // Allow a computer name to be specified on the command line
      // Else, just query the local machine
    if (argc > 1)
        szMachineName = argv[1];

    // Get the registry size.
    pdhStatus=GetRegistrySize(szMachineName, &dwCurrent, &dwMaximum);

    // Print the results.
    if (pdhStatus == ERROR_SUCCESS)
    {
            wprintf(L"\n");
        wprintf(L"Registry size: %ld bytes\n", dwCurrent);
        wprintf(L"Max registry size: %ld bytes\n", dwMaximum);

    }
    else
    {
        // If the operation failed, print the PDH error message
        szMessage = NULL;

            // A system (GetLastError()) or a PDH error code can be
used/extracted
        FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
            FORMAT_MESSAGE_FROM_HMODULE,
            GetModuleHandle(L"PDH.DLL"), pdhStatus,
            MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
            szMessage, 0, NULL);

        wprintf(L"GetRegistrySize() failed:  %s", szMessage);
        LocalFree(szMessage);
    }

    return 0;
}

//*********************************************************************
//  Retrieves the current and maximum registry size. It gets this
//  information from the raw counter values for the "% Registry Quota
//  In Use" performance counter within the System object.
//  PARAMETERS:
//      szMachineName - Null-terminated string that specifies the
//          name of the computer whose registry you wish to query.
//          If this parameter is NULL, the local computer is used.
//      lpdwCurrentSize - Receives the current registry size.
//      lpdwMaximumSize - Receives the maximum registry size.
//  RETURN VALUE:
//      ERROR_SUCCESS if successful. Otherwise, the function
//      returns a PDH error code. These error codes can be
//      found in PDHMSG.H. A textual error message can be
//      retrieved from PDH.DLL using the FormatMessage function.
//********************************************************************
PDH_STATUS GetRegistrySize(LPTSTR szMachineName, LPDWORD lpdwCurrentSize,
LPDWORD lpdwMaximumSize)
{
    PDH_STATUS  pdhResult   = 0;
    WCHAR       szCounterPath[1024];
    DWORD       dwPathSize  = 1024;
    PDH_COUNTER_PATH_ELEMENTS pe;
    PDH_RAW_COUNTER pdhRawValues;
    HQUERY      hQuery      = NULL;
    HCOUNTER    hCounter    = NULL;
    DWORD       dwType      = 0;

    // Open PDH query
    pdhResult = PdhOpenQuery(NULL, 0, &hQuery);
    if (pdhResult != ERROR_SUCCESS)
        return pdhResult;
      else
          wprintf(L"PdhOpenQuery() looks fine!\n");

      // Exception handling, __try-__finally
    __try
    {
        // Create counter path
        pe.szMachineName     = szMachineName;
        pe.szObjectName      = L"System";
        pe.szInstanceName    = NULL;
        pe.szParentInstance  = NULL;
        pe.dwInstanceIndex   = 1;
        pe.szCounterName     = L"% Registry Quota In Use";

        pdhResult = PdhMakeCounterPath(&pe, szCounterPath, &dwPathSize, 0);
        if (pdhResult != ERROR_SUCCESS)
            __leave;
            else
                  wprintf(L"PdhMakeCounterPath() is OK!\n");

        // Add the counter to the query
        pdhResult=PdhAddCounter(hQuery, szCounterPath, 0, &hCounter);
        if (pdhResult != ERROR_SUCCESS)
            __leave;
            else
                  wprintf(L"PdhAddCounter() is working!\n");

        // Run the query to collect the performance data
        pdhResult = PdhCollectQueryData(hQuery);
        if (pdhResult != ERROR_SUCCESS)
            __leave;
            else
                  wprintf(L"Well, PdhCollectQueryData() is also working!\n");

        // Retrieve the raw counter data:
        //    The dividend (FirstValue) is the current registry size
        //    The divisor (SecondValue) is the maximum registry size
        ZeroMemory(&pdhRawValues, sizeof(pdhRawValues));
        pdhResult = PdhGetRawCounterValue(hCounter, &dwType, &pdhRawValues);
        if (pdhResult != ERROR_SUCCESS)
            __leave;
            else
                  wprintf(L"PdhGetRawCounterValue() is OK!\n");

        // Store the sizes in variables.
        if (lpdwCurrentSize)
            *lpdwCurrentSize = (DWORD)pdhRawValues.FirstValue;

        if (lpdwMaximumSize)
            *lpdwMaximumSize = (DWORD)pdhRawValues.SecondValue;

    }
    __finally
    {
        // Close the query
        if(PdhCloseQuery(hQuery) == ERROR_SUCCESS)
                  wprintf(L"hQuery handle was closed successfully!\n");
    }
    return 0;
}
```

Build and run the project. The following screenshot is a sample output.



```
C:\amad\RegistrySize\Debug>RegistrySize
Usage: RegistrySize <computer_name>
Else, local computer will be used as default

PdhOpenQuery() looks fine!
PdhMakeCounterPath() is OK!
PdhAddCounter() is working!
Well, PdhCollectQueryData() is also working!
PdhGetRawCounterValue() is OK!
hQuery handle was closed successfully!

Registry size: 6342956 bytes
Max registry size: 125829120 bytes

C:\amad\RegistrySize\Debug>RegistrySize nazuri
Usage: RegistrySize <computer_name>
Else, local computer will be used as default

PdhOpenQuery() looks fine!
PdhMakeCounterPath() is OK!
PdhAddCounter() is working!
Well, PdhCollectQueryData() is also working!
PdhGetRawCounterValue() is OK!
hQuery handle was closed successfully!

Registry size: 6342956 bytes
Max registry size: 125829120 bytes

C:\amad\RegistrySize\Debug>_
```

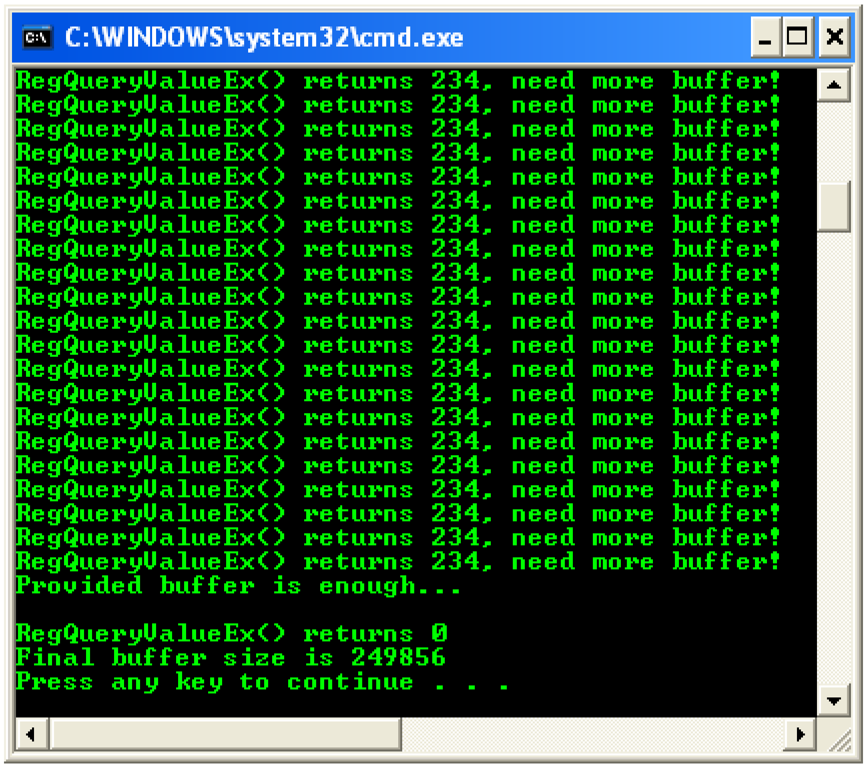
Querying the Registry Value Program Example

An application typically calls RegEnumValue() to determine the value names and then RegQueryValueEx() to retrieve the data for the names.
If the data has the REG_SZ, REG_MULTI_SZ or REG_EXPAND_SZ type, the string may not have been stored with the proper terminating null characters. Therefore, even if the function returns ERROR_SUCCESS, the application should ensure that the string is properly terminated before using it; otherwise, it may overwrite a buffer. (Note that REG_MULTI_SZ strings should have two terminating null characters.) One way an application can ensure that the string is properly terminated is to use RegGetValue(), which adds terminating null characters if needed.
If the data has the REG_SZ, REG_MULTI_SZ or REG_EXPAND_SZ type, and the ANSI version of this function is used (either by explicitly calling RegQueryValueExA() or by not defining UNICODE before including the Windows.h file), this function converts the stored Unicode string to an ANSI string before copying it to the buffer pointed to by lpData.
When calling the RegQueryValueEx() function with hKey set to the HKEY_PERFORMANCE_DATA handle and a value string of a specified object, the returned data structure sometimes has unrequested objects. Do not be surprised; this is normal behavior.
When calling the RegQueryValueEx() function, you should always expect to walk the returned

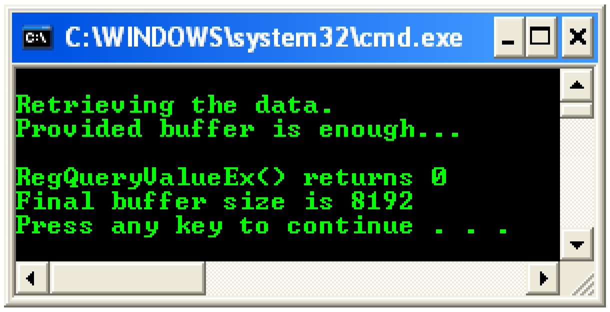
data structure to look for the requested object. Note that operations that access certain registry keys are redirected.
Ensure that you reinitialize the value pointed to by the lpcbData parameter each time you call this function. This is very important when you call this function in a loop, as in the following code example.
Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.
Then, add the source file and give it a suitable name.
Next, add the following source code.

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

#define TOTALBYTES    8192
#define BYTEINCREMENT 4096

int wmain(int argc, WCHAR *argv[])
{
    DWORD BufferSize = TOTALBYTES;
    DWORD cbData;
    DWORD dwRet;
    PPERF_DATA_BLOCK PerfData = (PPERF_DATA_BLOCK) malloc(BufferSize);
    cbData = BufferSize;

    wprintf(L"\nRetrieving the data.  ");

      // Using predefined registry key that always open
      // The name of the registry value is "mike spoon"
      //  If the function succeeds, the return value is ERROR_SUCCESS
      //  If the function fails, the return value is a system error code
      //  If the lpData buffer is too small to receive the data, the function
returns ERROR_MORE_DATA
      //  If the lpValueName registry value does not exist, the function
returns ERROR_FILE_NOT_FOUND.
    dwRet = RegQueryValueEx(HKEY_PERFORMANCE_DATA,
                             L"Global",
                             NULL,
                             NULL,
                             (LPBYTE)PerfData,
                             &cbData);

    while(dwRet == ERROR_MORE_DATA)
    {
            wprintf(L"\nRegQueryValueEx() returns %u, need more buffer!",
dwRet);
        // Get a buffer that is big enough
        BufferSize += BYTEINCREMENT;
        PerfData = (PPERF_DATA_BLOCK)realloc(PerfData, BufferSize);
        cbData = BufferSize;
        dwRet = RegQueryValueEx(HKEY_PERFORMANCE_DATA,
                         L"Global",
                         NULL,
                         NULL,
                         (LPBYTE) PerfData,
                         &cbData);
      }

      wprintf(L"\nProvided buffer is enough...\n");

      if(dwRet == ERROR_FILE_NOT_FOUND)
      {
            // This part failed although providing the non-exist
            // registry value name. It goes to the next 'if'
            wprintf(L"\nRegQueryValueEx() returns %u", dwRet);
            wprintf(L"\nRegistry value does not exist!\n");
      }
    else if(dwRet == ERROR_SUCCESS)
      {
            wprintf(L"\nRegQueryValueEx() returns %u", dwRet);
        wprintf(L"\nFinal buffer size is %d\n", BufferSize);
            // wprintf(L"\nRegistry value found!");
            // wprintf(L"\nPerfData->Version: %u",PerfData->Version);
            // wprintf(L"\nPerfData->TotalByteLength: %u",PerfData-
>TotalByteLength);
            // wprintf(L"\nPerfData->SystemNameLength: %u",PerfData-
>SystemNameLength);
            // wprintf(L"\nPerfData->Revision: %u\n",PerfData->Revision);
      }
    else
            wprintf(L"\nRegQueryValueEx() failed, error code %u\n", dwRet);

      return 0;
}
```

Build and run the project. The following screenshots are sample outputs.



```
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
RegQueryValueEx() returns 234, need more buffer!
Provided buffer is enough...

RegQueryValueEx() returns 0
Final buffer size is 249856
Press any key to continue . . .
```

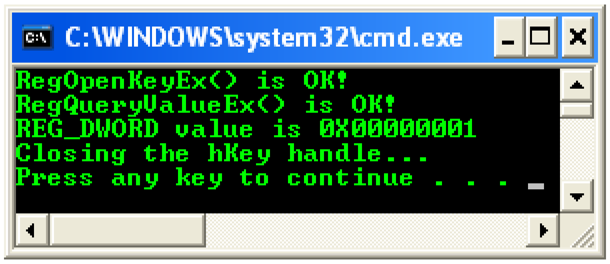


```
Retrieving the data.
Provided buffer is enough...

RegQueryValueEx() returns 0
Final buffer size is 8192
Press any key to continue . . .
```

The following is another loose and simple code example.
Create a new empty Win32 console application project. Give a suitable project name and change the project location if needed.
Then, add the source file and give it a suitable name.
Next, add the following source code.

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

int wmain(int argc, WCHAR **argv)
{
      DWORD dwData;
    HKEY hKey;
    LONG returnStatus;
      DWORD dwType = REG_DWORD;
      DWORD dwSize = sizeof(DWORD)*2;

    returnStatus = RegOpenKeyEx(HKEY_LOCAL_MACHINE,
L"Software\\Microsoft\\ASP.NET", 0L, KEY_ALL_ACCESS, &hKey);

    if (returnStatus  == ERROR_SUCCESS)
    {
            wprintf(L"RegOpenKeyEx() is OK!\n");
        returnStatus = RegQueryValueEx(hKey, L"DefaultDocInstalled", NULL,
&dwType,(LPBYTE)&dwData, &dwSize);

        if (returnStatus == ERROR_SUCCESS)
        {
                  wprintf(L"RegQueryValueEx() is OK!\n");
                  wprintf(L"REG_DWORD value is 0X%.8X\n", dwData);
        }
            else
                  wprintf(L"RegQueryValueEx() failed, error %u\n",
GetLastError());
     }
      else
            wprintf(L"RegOpenKeyEx() failed, error %u\n", GetLastError());

    if(RegCloseKey(hKey) == ERROR_SUCCESS)
            wprintf(L"Closing the hKey handle...\n");

      return 0;
}
```

Build and run the project. The following screenshot is a sample output.



```
RegOpenKeyEx() is OK!
RegQueryValueEx() is OK!
REG_DWORD value is 0X00000001
Closing the hKey handle...
Press any key to continue . . .
```

Registry Reference

The following elements are used with the registry.

1. Registry Functions
2. Registry Structures
3. Registry Element Size Limits
4. Registry Value Types

Registry Functions

The following are the registry functions.

Function	Description
GetSystemRegistryQuota()	Retrieves the current size of the registry and the maximum size that the registry is allowed to attain on the system.
RegCloseKey()	Closes a handle to the specified registry key.
RegConnectRegistry()	Establishes a connection to a predefined registry handle on another computer.
RegCopyTree()	Copies the specified registry key, along with its values and subkeys, to the specified destination key.
RegCreateKeyEx()	Creates the specified registry key.
RegCreateKeyTransacted()	Creates the specified registry key and associates it with a transaction.
RegDeleteKey()	Deletes a subkey and its values.
RegDeleteKeyEx()	Deletes a subkey and its values from the specified platform-specific view of the registry.
RegDeleteKeyTransacted()	Deletes a subkey and its values from the specified platform-specific view of the registry as a transacted operation.
RegDeleteKeyValue()	Removes the specified value from the specified registry key and subkey.
RegDeleteTree()	Deletes the subkeys and values of the specified key recursively.
RegDeleteValue()	Removes a named value from the specified registry key.
RegDisablePredefinedCache()	Disables handle caching for the predefined registry handle for HKEY_CURRENT_USER for the current process.
RegDisablePredefinedCacheEx()	Disables handle caching for all predefined registry handles for the current process.
RegDisableReflectionKey()	Disables registry reflection for the specified key.
RegEnableReflectionKey()	Enables registry reflection for the specified disabled key.
RegEnumKeyEx()	Enumerates the subkeys of the specified open registry key.
RegEnumValue()	Enumerates the values for the specified open registry key.
RegFlushKey()	Writes all attributes of the specified open registry key into the registry.
RegGetKeySecurity()	Retrieves a copy of the security descriptor protecting the specified open registry key.
RegGetValue()	Retrieves the type and data for the specified registry value.
RegLoadKey()	Creates a subkey under HKEY_USERS or

	HKEY_LOCAL_MACHINE and stores registration information from a specified file into that subkey.
RegLoadMUIString()	Loads the specified string from the specified key and subkey.
RegNotifyChangeKeyValue()	Notifies the caller about changes to the attributes or contents of a specified registry key.
RegOpenCurrentUser()	Retrieves a handle to the HKEY_CURRENT_USER key for the user the current thread is impersonating.
RegOpenKeyEx()	Opens the specified registry key.
RegOpenKeyTransacted()	Opens the specified registry key and associates it with a transaction.
RegOpenUserClassesRoot()	Retrieves a handle to the HKEY_CLASSES_ROOT key for the specified user.
RegOverridePredefKey()	Maps a predefined registry key to a specified registry key.
RegQueryInfoKey()	Retrieves information about the specified registry key.
RegQueryMultipleValues()	Retrieves the type and data for a list of value names associated with an open registry key.
RegQueryReflectionKey()	Determines whether reflection has been disabled or enabled for the specified key.
RegQueryValueEx()	Retrieves the type and data for a specified value name associated with an open registry key.
RegReplaceKey()	Replaces the file backing a registry key and all its subkeys with another file.
RegRestoreKey()	Reads the registry information in a specified file and copies it over the specified key.
RegSaveKey()	Saves the specified key and all of its subkeys and values to a new file.
RegSaveKeyEx()	Saves the specified key and all of its subkeys and values to a new file. You can specify the format for the saved key or hive.
RegSetKeyValue()	Sets the data for the specified value in the specified registry key and subkey.
RegSetKeySecurity()	Sets the security of an open registry key.
RegSetValueEx()	Sets the data and type of a specified value under a registry key.
RegUnLoadKey()	Unloads the specified registry key and its subkeys from the registry.

The following shell functions can be used with the registry:

1. AssocCreate()
2. AssocQueryKey()
3. AssocQueryString()
4. AssocQueryStringByKey()
5. SHCopyKey()
6. SHDeleteEmptyKey()
7. SHDeleteKey()
8. SHDeleteValue()
9. SHEnumKeyEx()
10. SHEnumValue()
11. SHGetValue()
12. SHQueryInfoKey()
13. SHQueryValueEx()
14. SHRegCloseUSKey()
15. SHRegCreateUSKey()
16. SHRegDeleteEmptyUSKey()
17. SHRegDeleteUSValue()
18. SHRegDuplicateHKey()
19. SHRegEnumUSKey()
20. SHRegEnumUSValue()
21. SHRegGetBoolUSValue()
22. SHRegGetIntW()
23. SHRegGetPath()
24. SHRegGetUSValue()
25. SHRegOpenUSKey()
26. SHRegQueryInfoUSKey()
27. SHRegQueryUSValue()
28. SHRegSetPath()
29. SHRegSetUSValue()
30. SHRegWriteUSValue()
31. SHSetValue()

The following are the initialization-file functions. They retrieve information from and copy information to a system- or application-defined initialization file. These functions are provided only for compatibility with 16-bit versions of Windows. New applications should use the registry.

Function	Description
GetPrivateProfileInt()	Retrieves an integer associated with a key in the specified

	section of an initialization file.
GetPrivateProfileSection()	Retrieves all the keys and values for the specified section of an initialization file.
GetPrivateProfileSectionNames()	Retrieves the names of all sections in an initialization file.
GetPrivateProfileString()	Retrieves a string from the specified section in an initialization file.
GetPrivateProfileStruct()	Retrieves the data associated with a key in the specified section of an initialization file.
GetProfileInt()	Retrieves an integer from a key in the specified section of the Win.ini file.
GetProfileSection()	Retrieves all the keys and values for the specified section of the Win.ini file.
GetProfileString()	Retrieves the string associated with a key in the specified section of the Win.ini file.
WritePrivateProfileSection()	Replaces the keys and values for the specified section in an initialization file.
WritePrivateProfileString()	Copies a string into the specified section of an initialization file.
WritePrivateProfileStruct()	Copies data into a key in the specified section of an initialization file.
WriteProfileSection()	Replaces the contents of the specified section in the Win.ini file with specified keys and values.
WriteProfileString()	Copies a string into the specified section of the Win.ini file.

Obsolete Functions

The following functions are provided only for compatibility with 16-bit versions of Windows:

1. RegCreateKey()
2. RegEnumKey()
3. RegOpenKey()
4. RegQueryValue()
5. RegSetValue()

Registry Structures

The following structure is used with the registry: VALENT. This structure contains information about a registry value. The RegQueryMultipleValues() function uses this structure. The syntax is:

```
typedef struct value_ent {
  LPTSTR    ve_valuename;
  DWORD     ve_valuelen;
  DWORD_PTR ve_valueptr;
  DWORD     ve_type;
}VALENT, *PVALENT;
```

Members

Member name	Description
ve_valuename	The name of the value to be retrieved. Be sure to set this member before calling RegQueryMultipleValues().
ve_valuelen	The size of the data pointed to by ve_valueptr, in bytes.
ve_valueptr	A pointer to the data for the value entry. This is a pointer to the value's data returned in the lpValueBuf buffer filled in by RegQueryMultipleValues().
ve_type	The type of data pointed to by ve_valueptr.