Windows Utilities Reference Manual

0.1.0 (development)

Generated by Doxygen 1.3.4

Tue Sep 30 11:40:30 2003

Contents

Chapter 1

Windows Utilities Main Page

The Windows Utilities Library contains several functions and classes that encapsulate aspects of the Microsoft 32-bit Windows Application Programming Interface (Win32 API).

This manual contains the following sections:

- Application Log
- Exceptions
- Registry
- Semaphore
- Shutdown and Reboot
- Version information
- Parallel Port Base Addresses
- Support Classes

WindowsUtils is based on [CREGISTRYKEY], [INTERRUPTHOOK], [FONTFILE], [SHUTDOWN], and [WIN32SEMAPHORE]

WindowsUtils License

Chapter 2

Windows Utilities Module Documentation

2.1 Application Log

2.1.1 Detailed Description

This module provides functions to write messages to the Windows Application Log. The following message categories are distinguished :

- error messages
- general messages
- debug messages

Error messages – error messages can be formatted and logged to the Windows Application Log with error(). The programname as set with setprogname() precedes the message.

General messages – function message() is meant to format and log normal messages, for example to trace the program's normal processing. A message also specifies a level and the message is only added to the Windows Application Log, if its level is equal or lower than the current message verbosity level as set with setmsglevel().

Debug messages – function debug() is meant to format and log debug messages, for example to trace the program's data handling. A debug message also specifies a level and the message is only added to the Windows Application Log, if its level is equal or lower than the current debug message verbosity level as set with setdbglevel().

Here is a small example.

```
#include <except>
#include "wapplog.h"

int main( int argc, char *argv[] )
{
   WindowsUtils::setprogname( argv[0] );

   try
   {
      // ... operation that may throw exception
   }
   catch( std::exception& e )
   {
      return WindowsUtils::error( 1, "main(): %s", e.what() );
   }
}
```

The code for this module is inspired by two books by Kernighan & Pike [1984,1999]

Namespaces

- namespace WindowsUtils

Defines

- #define **D**(line)

Enumerations

- enum WindowsUtils::DebugLevel { WindowsUtils::DBG_L0, WindowsUtils::DBG_L1, WindowsUtils::DBG_L2, WindowsUtils::DBG_L3, WindowsUtils::DBG_L4 }

  debug message levels.

Functions

- const char ∗ WindowsUtils::getprogname ()

  return the program's name as set with setprogname(), or "(undefined)".

- void WindowsUtils::setprogname (const char ∗name)

  set the program's name as used with error(), message() and debug().

- int WindowsUtils::getdbglevel ()

  get the current debug message verbosity level.

- void WindowsUtils::setdbglevel (int level)

  set the debug message verbosity level (default it is 0).

- int WindowsUtils::getmsglevel ()

  get the current message verbosity level.

- void WindowsUtils::setmsglevel (int level)

  set the message verbosity level (default it is 0).

- const char ∗ WindowsUtils::error_message ()

  return last error message as C-string.

- int WindowsUtils::error (int status, const char ∗format...)

  format and log error message, return status.

- int WindowsUtils::debug (int level, const char ∗format...)

  format and log debug message, if level permits; return specified debug level.

- int WindowsUtils::message (int level, const char ∗format...)

  format and log message, if level permits; return specified level.

2.1.2 Enumeration Type Documentation

2.1.2.1 enum WindowsUtils::DebugLevel

DebugLevel defines the possible debug messsage levels.

Enumeration values:

DBG_L0	level 0
DBG_L1	level 1
DBG_L2	level 2
DBG_L3	level 3
DBG_L4	level 4

2.1.3 Function Documentation

2.1.3.1 int WindowsUtils::debug (int *level*, const char ∗ *format...*)

debug() formats and logs the error message to the Windows Application Log if the specified level is equal or less than the current debug level. `format` is a printf() format string, optionally followed by arguments.

The message has the following format:

[*progname*:]*message*[: *system-message*]\n

progname is included when a colon starts the format and it has been set via setprogname(); *system-message* is included when the format string ends with a colon (:).

debug() returns the specified level.

Parameters:

level	this message's level
format	a printf format string, optionally followed by arguments

Returns:

level

See also:

setprogname(), getdbglevel(), setdbglevel()

2.1.3.2 int WindowsUtils::error (int *status*, const char ∗ *format*...)

error() formats and logs the error message to the Windows Application Log. `format` is a printf() format string, optionally followed by arguments.

The error message has the following format:

[*progname*:]*message*[: *system-message*]\n

progname is included when a colon starts the format and it has been set via setprogname(); *system-message* is included when the format string ends with a colon (:).

error() returns the specified status.

Parameters:

status the error status to return

format a printf format string, optionally followed by arguments

Returns:

status

See also:

setprogname(), error_message()

2.1.3.3 int WindowsUtils::message (int *level*, const char ∗ *format*...)

message() formats and logs the specified message to the Windows Application Log if its level is equal or less than the current verbosity level as set with setmsglevel(). `format` is a printf() format string, optionally followed by arguments.

The message has the following format:

[*progname*:]*message*[: *system-message*]\n

progname is included when a colon starts the format and it has been set via setprogname(); *system-message* is included when the format string ends with a colon (:).

message() returns the specified level.

Parameters:

level this message's level

format a printf format string, optionally followed by arguments

Returns:

level

See also:

setprogname(), getmsglevel(), setmsglevel()

2.2 Exceptions

2.2.1 Detailed Description

With class Win32Exception you can create standard C++ exceptions for error return values of Win32 API functions. For example:

```
#include "wexception.h"

HKEY openkey( HKEY ahKey, std::string aSubkey, REGSAM aSamDesired )
{
   using WindowsUtils::Win32Exception;

   HKEY newKey;
   LONG result = RegOpenKeyEx( ahKey, aSubkey.c_str(), 0, aSamDesired, &newKey );

   if ( ERROR_SUCCESS != result )
   {
      throw Win32Exception( "openkey", result );
   }

   return newkey;
}
```

This is used in a program as follows.

```
int main()
{
   try
   {
      HKEY hkey = openkey( HKEY_LOCAL_MACHINE, "SYSTEM\\subkey", KEY_READ );
   }
   catch ( const std::exception& e )
   {
      std::cout << e.what() << std::endl;
   }
}
```

Classes

- class Win32Exception

  Exceptions for Windows errors; based on std::exception.

Functions

- void WindowsUtils::noGPErrorBox ()

  disable display of the general-protection-fault message box. This flag should only be set by debugging applications that handle general protection (GP) faults themselves via an appropriate exception handler.

- const char $*$ WindowsUtils::GetLastErrorMessage (DWORD aError)

  return an error message character string for the specified error. The string is owned by this function.

2.3 Registry

2.3.1 Detailed Description

Class RegistryKey provides easy access to keys and values of the Windows registry. It has the following characteristics:

- uniform access to values of various types, like REG_DWORD, REG_BINARY, REG_SZ
- iterator interface to subkeys and values

The original Registry API wrapper was obtained from [CREGISTRYKEY] project at [CODEPROJECT] with this license.

Before

In the discussion below, the following include directives and namespace using declaration are assumed:

```
#include <iostream>          // for std::cout
#include "wregistry.h"       // for RegistryKey

using WindowsUtils::RegistryKey;
```

Keys and Values

To work with the registry, you first open a key, for example:

```
RegistryKey key( HKEY_LOCAL_MACHINE, "HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0" );
```

This opens the specified subkey of the standard open HKEY_LOCAL_MACHINE.

Other ways to open a key are via methods connectRegistry(), openKey(), createKey(), createOrOpenKey() or one of the other constructors.

Note that you don't have to close the key. That is done automatically when `key` goes out of scope. However, you *can* close it by assigning one of the standard open keys to it, for example:

```
key = HKEY_LOCAL_MACHINE;
```

If you want to obtain the contents of a key's value, you may write for example:

```
RegistryKey::Value v = key.queryValue( "~MHz" );
```

This obtains the main processor's clock frequency from value `~MHz`.

Now you have the value's type, name and contents available as `v.type()`, `v.name()` and `static_cast<DWORD>(v)`. The latter means that if `v` is assigned to a DWORD variable, or used as an argument in a function that takes a DWORD parameter (or a parameter of a type that DWORD can be converted to), `v` will be converted to the value's contents.

Thus, if you only need the value's contents, you may also write:

```
DWORD dw = key.queryValue( "~MHz" );
```

The value's contents may also be obtained as a string with `v.toString()`.

These are the three kinds of Values available:

- integral Value (DWORD for REG_DWORD etc.)
- binary Value (std::vector<Byte> for REG_NONE etc.)
- string Value (std::string for REG_SZ, REG_MULTI_SZ)

The following conversions are valid:

- an integral Value may be converted to DWORD
- a binary Value may be converted to RegistryKey::ByteBuffer (std::vector<Byte>)
- any Value may be converted to std::string

Other conversions throw a `RegistryKey::Exception`.

```
DWORD dw = key.queryValue( "SystemBiosVersion" );    // ERROR: string -> DWORD
```

Values and Subkeys may be put on an output stream as follows:

```
std::cout << key.queryValue( "SystemBiosDate" ) << std::endl;
```

Which may give:

```
SystemBiosDate = '02/04/98'
```

Iterators

To iterate over the values of a subkey, you can use a `RegistryKey::ValueIterator`, for example:

```
for ( RegistryKey::ValueIterator pos = key.beginValueIteration();
                                 pos != key.endValueIteration(); ++pos )
{
   std::cout << pos.name() << " = " << pos.toString() << std::endl;
}
```

Instead of *pos.name() << " = " << pos.toString()*, you may also write ∗`pos` to output the value.

To iterate over the subkeys of a key, you can use a `RegistryKey::SubkeyIterator` as follows:

```
for ( RegistryKey::SubkeyIterator pos = key.beginSubkeyIteration();
                                  pos != key.endSubkeyIteration(); ++pos )
{
   std::cout << *pos << std::endl;
}
```

Or, using the std::copy() algorithm:

```
#include <algorithm>         // for std::copy()
#include <iterator>          // for std::ostream_iterator<>

std::copy( key.beginSubkeyIteration(),
             key.endSubkeyIteration(),
               std::ostream_iterator<RegistryKey::Subkey>(std::cout, " ") );
```

Besides the Subkey and Value iterators an InsertIterator exists. It lets you add values and subkeys to the key it represents. It can open the subkey that was last assigned to it with openKey(). This makes it possible to descend a subkey tree while it is being copied. InsertIterator is used by the copy() algorithm.

Algorithms and Function Objects

The C++ Standard Template Library defines many *algorithms* and function-like objects. Other names for function-like object are *function object*, *functor*.

Please read Stroustrup [2000] and Josuttis [1999] for good explanations on this matter.

Class RegistryKey offers a few algorithms to traverse a key's subkeys and values: copy(), for_each(), find_subkey() and find_value().

The following code fragment to sum the numbers of an array, gives an impression of algorithms and function objects.

```
#include <algorithm>         // for std::for_each()
#include <functional>        // for std::unary_function

template <typename T>
class Sum : public std::unary_function<T,void>
{
private:
   T sum;

public:
   Sum( T initial = 0 ) : sum( initial ) { ; }

   void operator()(const T& x) { sum += x; }

   operator T(){ return sum; }
};

void sum()
{
    int a[] = { 3, 9, 27, };

    int sum = std::for_each( a, a + 3, Sum<int>() );
}
```

The same effect may be obtained with the algorithm `std::accumulate(begin, end, initval)`:

```
    int sum = std::accumulate( a, a + 3, 0 );
```

The `for_each()` algorithm takes three parameters and returns the last parameter:

```
operation for_each( begin, end, operation);
```

`for_each()` may have been implemented as follows:

```
namspace std {

   template <typename Iterator, typename UnaryOp>
   UnaryOp for_each( Iterator begin, Iterator end, UnaryOp op)
    {
       for ( ; begin != end; ++begin )
       {
          op( *begin );
       }

       return op;
   }
}
```

`begin` and `end` specify the range [begin,end) of the array that must be taken into account for the operation. begin and end are *iterators*.

`for_each()` returns (a copy of) the operation, the `Sum<int>()` function object in our example. By assigning the returned `Sum` object to an int, its `operation int()` conversion is used to yield the sum.

File-related methods

Several methods exist to save a registry key tree to a file and to load it from a file:

- loadKey() – load specified subkey from file into registry.
- unloadKey() – unload specified key and subkeys from the registry.
- saveKey() – save this key and all of its subkeys and values to a new file.
- restoreKey() – copy the registry information from the specified file over this key.
- replaceKey() – replace the file containing a key and all its subkeys with another file; a restart activates the new values.

Security-related methods

The following methods let you get and set security information of a key:

- getKeySecurity() – return a copy of the security descriptor of this registry key.
- setKeySecurity() – set the security of this registry key.

Event-related methods

Method notifyChangeKeyValue() lets you act – synchronously or asymchronously – on a key-changed notification.

Examples

The Examples section contains four complete programs that use a SubkeyIterator, a ValueIterator, the copy() and for_each() algorithms and the find_value() algorithm respectively.

- print subkeys
- print values
- copy, for_each algorithm
- find_value algorithm

Modules

- Driver Development Kit

  windows driver development kit.

Classes

- class CCountedRegKey

  reference counted registry key type used for the representation in RegistryKey.

- class CRegKeyIterator

  base class for subkey and value iterators.

- class RegistryKey

  provide access to registry subkeys and values.

- class RegistryKey::BinaryValue

  binary values (ByteBuffer).

- class RegistryKey::CountedValue

  reference counted value type used for the representation in Value.

- class RegistryKey::CountedValue::Exception

  exception type used to convert Win32 API call error return values to exceptions ultimately derived from std::exception.

- class RegistryKey::Exception

  exception type used to convert Win32 API call error return values to exceptions ultimately derived from std::exception.

- class RegistryKey::InsertIteratorImpl

  insert iterator implementation class.

- class RegistryKey::IntegralValue

  integral values (DWORD).

- class RegistryKey::StringValue

  string values (std::string).

- class RegistryKey::Subkey

  class to encapsulate a Subkey.

- class RegistryKey::SubkeyIteratorImpl

  subkey iterator implementation class.

- class RegistryKey::Value

  class to encapsulate various registry value types, like REG_DWORD etc.

- class RegistryKey::ValueBuffer

  construct a ByteBuffer from a Value.

- class RegistryKey::ValueInterface

  interface for Values.

- class RegistryKey::ValueIteratorImpl

value iterator implementation class.

- class TRegKeyIterator

  template class for subkey and value iterator.

2.4 Semaphore

2.4.1 Detailed Description

Class Semaphore encapsulates the Win32 semaphore.

A semaphore can be used to guard critical sections from simultanuous access from different threads. For example:

```
#include "wsemaphore.h"

WindowsUtils::Semaphore s;

int shared_count;

void increment()
{
   s.wait();
   ++shared_count;      // critical section
   s.post();
}

void decrement()
{
   s.wait();
   --shared_count;      // critical section
   s.post();
}
```

Class Sempahore is obtained from [WIN32SEMAPHORE] at [BBDSOFT] with this license.

Classes

- class Semaphore

  encapsulate Win32 semaphore.

2.5 Shutdown and Reboot

2.5.1 Detailed Description

The functions in this module enable you to:

- logoff the current user, logoff()
- shutdown the computer, shutdown()
- shutdow and power off the computer, poweroff()
- shutdown and reboot this computer or another computer, remoteShutdown()
- cancel a remote shutdown in progress, abortShutdown()

Here is a small example program that aborts a reboot after 10s.

```
#include "wshutdown.h"
#include "wexception.h"

int main()
{
   try
   {
      WindowsUtils::remoteShutdown(
         TRUE,  // reboot
         NULL,  // this computer
         "reboot will be aborted in 10s",
         60 );  // timeout

      Sleep( 10 * 1000 );

      WindowsUtils::abortShutdown( NULL );
   }
   catch( WindowsUtils::Win32Exception e )
   {
      e.messageBox();
   }
}
```

Functions

- void WindowsUtils::GetPrivilegesNT ()

  if this is a Windows NT platform, obtain the privileges required for shutdown, otherwise do nothing.

- void WindowsUtils::remoteShutdown (bool inReboot, const char ∗inMachine, const char ∗inMessage, int inTimeout)

  shutdown and optionally reboot a (remote) computer.

- void WindowsUtils::abortShutdown (const char ∗inMachine)

  cancel a shutdown initiated by remoteShutdown().

- void WindowsUtils::setWorkingSetSize (int inMinPages, int inMaxPages)

  set the process' resident available memory pages to use without triggering a page fault.

- void WindowsUtils::localShutdown (UINT inShutdownMode)
  shutdown the local computer as specified by inShutdownMode.
- void WindowsUtils::reboot ()
  reboot the local computer
- void WindowsUtils::shutdown ()
  shutdown the local computer
- void WindowsUtils::logoff ()
  logg off current user
- void WindowsUtils::poweroff ()
  shutdown and power off the local computer

2.5.2 Function Documentation

2.5.2.1 void WindowsUtils::localShutdown (UINT *inShutdownMode*)

localShutdown shuts down the local computer in the way specified by `inShutdownMode`. This parameter must be some combination of the following values:

- EWX_FORCE Forces processes to terminate. When this flag is set, Windows does not send the messages WM_QUERYENDSESSION and WM_ENDSESSION to the applications currently running in the system. This can cause the applications to lose data. Therefore, you should only use this flag in an emergency.
- EWX_LOGOFF Shuts down all processes running in the security context of the process that called the ExitWindowsEx function. Then it logs the user off.
- EWX_POWEROFF Shuts down the system and turns off the power. The system must support the power-off feature.Windows NT: The calling process must have the SE_SHUTDOWN_NAME privilege. For more information, see the following Remarks section. Windows 95: Security privileges are not supported or required.
- EWX_REBOOT Shuts down the system and then restarts the system. Windows NT: The calling process must have the SE_SHUTDOWN_NAME privilege. For more information, see the following Remarks section. Windows 95: Security privileges are not supported or required.
- EWX_SHUTDOWN Shuts down the system to a point at which it is safe to turn off the power. All file buffers have been flushed to disk, and all running processes have stopped. Windows NT: The calling process must have the SE_SHUTDOWN_NAME privilege. For more information, see the following Remarks section. Windows 95: Security privileges are not supported or required.

2.5.2.2 void WindowsUtils::remoteShutdown (bool *inReboot*, const char ∗ *inMachine*, const char ∗ *inMessage*, int *inTimeout*)

shutdown() shuts down the computer specified by `inMachine` and optionally reboots it. During the timeout specified by `inTimeout`, a dialog box is shown with the message `inMessage`. While this dialog box is displayed, the shutdown can be stopped with abortShutdown().

Parameters:

inReboot true: reboot after shutdown

inMachine remote or local computer if NULL or ""

inMessage the message for the dialog box, or NULL

inTimeout the time in seconds that the dialog box should be displayed

2.5.2.3 void WindowsUtils::setWorkingSetSize (int *inMinPages*, int *inMaxPages*)

The working set of a process is the set of memory pages currently visible to the process in physical RAM memory. These pages are resident and available for an application to use without triggering a page fault. The size of the working set of a process is specified in bytes. The minimum and maximum working set sizes affect the virtual memory paging behavior of a process.

2.6 Version information

2.6.1 Detailed Description

There are several functions to test for the various flavours of the Windows operating system:

- test for operating system type: isWin95Platform(), isWinNtPlatform()
- test for specific operating system: isWin95() etc.
- obtain an enumeration value for the operating system: OSver

The following table is used to determine the operating system versions.

	dwPlatFormID	dwMajorVersion	dwMinorVersion	dwBuildNumber
95	1	4	0	950
95 SP1	1	4	0	>950 && <=1080
95 OSR2	1	4	<10	>1080
98	1	4	10	1998
98 SP1	1	4	10	>1998 && <2183
98 SE	1	4	10	>=2183
ME	1	4	90	3000
NT 3.51	2	3	51	
NT 4	2	4	0	1381
2000	2	5	0	2195
XP	2	5	1	
CE	3			

This table was obtained from the file `WinVer.cpp` from the [FONTFILE] project at [CODEPROJECT].

Here is a small example program.

```
#include "wversion.h"

int main()
{
   if ( WindowsUtils::isWin95Platform() )
   {
      // do things one way
   }
   else if ( WindowsUtils::isWinNtPlatform() )
   {
      // do things another way
   }
   else
   {
      // do other things
   }
}
```

Enumerations

- enum WindowsUtils::OSver {
  None, **Win95**, **Win98**, **WinME**, **WinNT3**,
  WinNT4, **Win2000**, **WinXP**, **WinCE** }

    Windows version return values for getWindowsVersion().

Functions

- bool WindowsUtils::isWinNtPlatform ()

  true if running on Windows NT type OS (Windows NT/2000/XP).

- bool WindowsUtils::isWin95Platform ()

  true if running on Windows 95 type OS (Windows 95/98/Me).

- OSver WindowsUtils::getWindowsVersion ()

  get Windows version as W95, W98, WNT4 etc; see OSver.

- void WindowsUtils::getWindowsVersionInfo (DWORD ∗pdwPlatformId, DWORD ∗pdwMajorVersion, DWORD ∗pdwMinorVersion, DWORD ∗pdwBuildNumber)

  obtain Windows version information; arguments may be NULL.

- bool WindowsUtils::isWin95 ()

  true if Windows 95

- bool WindowsUtils::isWin98 ()

  true if Windows 98

- bool WindowsUtils::isWinME ()

  true if Windows ME

- bool WindowsUtils::isWinNT3 ()

  true if Windows NT3

- bool WindowsUtils::isWinNT4 ()

  true if Windows NT4

- bool WindowsUtils::isWin2000 ()

  true if Windows 2000

- bool WindowsUtils::isWinXP ()

  true if Windows XP

2.7 Parallel Port Base Addresses

2.7.1 Detailed Description

Class PportAddresses determines the base addresses of the parallel ports that are available on a computer. These base addresses may be obtained from the BIOS data area (Windows 95 type operating systems), or from the registry (Windows NT type operating systems). The locations for the parallel port base addresses on the various versions of Windows where provided by Arno van Amersfoort, known for his firewall script [IPTABLES_FS].

Class PportAddresses provides an iterator interface to loop through the parallel port base addresses:

```
#include "wpportaddr.h"

using WindowsUtils::PportAddresses;

PportAddresses thePportAddresses;

for ( PportAddresses::iterator pos = thePportAddresses.begin();
                               pos != thePportAddresses.end(); ++pos )
{
   std::cout << *pos << ' ';
}
```

Program printpportaddr shows how class PportAddresses may be used to print the base addresses of the parallel ports available on a computer.

Classes

- class PportAddresses

  determine the parallel port base addresses.

- class PportAddresses::CountedBody

  implementation for 'generic' operating system.

- class PportAddresses::Win2000Body

  determine valid parallel port base addresses for Windows 2000.

- class PportAddresses::Win95Body

  determine valid parallel port base addresses for Windows 95.

- class PportAddresses::Win98Body

  determine valid parallel port base addresses for Windows 98.

- class PportAddresses::WinMeBody

  determine valid parallel port base addresses for Windows ME.

- class PportAddresses::WinNtBody

  determine valid parallel port base addresses for Windows NT 4.

- class PportAddresses::WinXpBody

  determine valid parallel port base addresses for Windows XP.

2.8 Support Classes

2.8.1 Detailed Description

Reference Counting

There are two template mix-in classes to provide other classes with reference counting behaviour:

- TReferenceCounted<typename Base> – free the reference counted object when no more references remain to it.
- TReferenceCountedResource<typename Base, typename Resource> – as above *and* release the resource when no more references remain to the reference counted object.

These mix-in classes are usually used for reference-counted body objects in the handle-body idiom.

See program refcount for an example that uses the handle-body idiom in which the body is the reference-counted implementation.

Classes

- class TReferenceCounted

  template mix-in class for reference counting.

- class TReferenceCountedResource

  template mix-in class for reference counting a resource.

2.9 Driver Development Kit

2.9.1 Detailed Description

This section describes the information needed from the Windows Driver Development Kit to interpret the resource list obtained from the registry.

The information is obtained from project [INTERRUPTHOOK] at [CODEPROJECT].

Classes

- struct _CM_FULL_RESOURCE_DESCRIPTOR

  registry full resource descriptor.

- struct _CM_PARTIAL_RESOURCE_DESCRIPTOR

  registry partial resource descriptor.

- struct _CM_PARTIAL_RESOURCE_LIST

  registry partial resource list.

- struct _CM_RESOURCE_LIST

  registry resource list.

Defines

- #define CmResourceTypeNull 0

  ResType_All or ResType_None (0x0000).

- #define CmResourceTypePort 1

  ResType_IO (0x0002).

- #define CmResourceTypeInterrupt 2

  ResType_IRQ (0x0004).

- #define CmResourceTypeMemory 3

  ResType_Mem (0x0001).

- #define CmResourceTypeDma 4

  ResType_DMA (0x0003).

- #define CmResourceTypeDeviceSpecific 5

  ResType_ClassSpecific (0xFFFF).

- #define CmResourceTypeBusNumber 6

  ResType_BusNumber (0x0006).

- #define CmResourceTypeMaximum 7

  ??

- #define CmResourceTypeAssignedResource 8

  BUGBUG–remove.

- #define CmResourceTypeSubAllocateFrom 9

  BUGBUG–remove.

- #define CmResourceTypeNonArbitrated 128

  Not arbitrated if 0x80 bit set.

- #define CmResourceTypeConfigData 128

  ResType_Reserved (0x8000).

- #define CmResourceTypeDevicePrivate 129

  ResType_DevicePrivate (0x8001).

- #define CmResourceTypePcCardConfig 130

  ResType_PcCardConfig (0x8002).

- #define CmResourceTypeMfCardConfig 131

  ResType_MfCardConfig (0x8003).

Typedefs

- typedef enum WindowsUtils::_INTERFACE_TYPE WindowsUtils::INTERFACE_TYPE

  registry interface type.

- typedef enum WindowsUtils::_INTERFACE_TYPE ∗ WindowsUtils::PINTERFACE_TYPE

  registry interface type.

- typedef LARGE_INTEGER WindowsUtils::PHYSICAL_ADDRESS

  registry physical address.

- typedef LARGE_INTEGER ∗ WindowsUtils::PPHYSICAL_ADDRESS

  registry physical address.

- typedef WindowsUtils::_CM_PARTIAL_RESOURCE_DESCRIPTOR WindowsUtils::CM_-PARTIAL_RESOURCE_DESCRIPTOR

  registry partial resource descriptor.

- typedef WindowsUtils::_CM_PARTIAL_RESOURCE_DESCRIPTOR ∗ WindowsUtils::PCM_-PARTIAL_RESOURCE_DESCRIPTOR

  registry partial resource descriptor.

- typedef WindowsUtils::_CM_PARTIAL_RESOURCE_LIST WindowsUtils::CM_PARTIAL_-RESOURCE_LIST

  registry partial resource list.

- typedef WindowsUtils::_CM_PARTIAL_RESOURCE_LIST ∗ WindowsUtils::PCM_PARTIAL_-RESOURCE_LIST *registry partial resource list.*

- typedef WindowsUtils::_CM_FULL_RESOURCE_DESCRIPTOR WindowsUtils::CM_FULL_-RESOURCE_DESCRIPTOR

  registry full resource descriptor.

- typedef WindowsUtils::_CM_FULL_RESOURCE_DESCRIPTOR ∗ WindowsUtils::PCM_FULL_-RESOURCE_DESCRIPTOR

  registry full resource descriptor.

- typedef WindowsUtils::_CM_RESOURCE_LIST WindowsUtils::CM_RESOURCE_LIST

  registry resource list.

- typedef WindowsUtils::_CM_RESOURCE_LIST ∗ WindowsUtils::PCM_RESOURCE_LIST

  registry resource list.

Enumerations

- enum WindowsUtils::_INTERFACE_TYPE {
  InterfaceTypeUndefined = -1, **Internal**, **Isa**, **Eisa**, **MicroChannel**,
  TurboChannel, **PCIBus**, **VMEBus**, **NuBus**, **PCMCIABus**,
  CBus, **MPIBus**, **MPSABus**, **ProcessorInternal**, **InternalPowerBus**,
  PNPISABus, **PNPBus**, **MaximumInterfaceType** }

  registry interface type.

Chapter 3

Windows Utilities Namespace Documentation

3.1 WindowsUtils::applog Namespace Reference

Variables

- char ∗ progname = 0

  progname (not set)

- int dbglevel = 0

  current debug level

- int msglevel = 0

  current messaeg level

- std::string errmsg

  copy of last error message

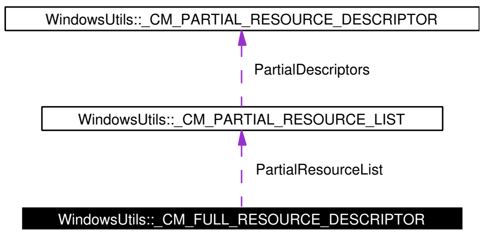
Chapter 4

Windows Utilities Class Documentation

4.1 WindowsUtils::_CM_FULL_RESOURCE_DESCRIPTOR Struct Reference

`#include <wregistry-ddk.h>`

Collaboration diagram for WindowsUtils::_CM_FULL_RESOURCE_DESCRIPTOR:


WindowsUtils::_CM_PARTIAL_RESOURCE_DESCRIPTOR
PartialDescriptors
WindowsUtils::_CM_PARTIAL_RESOURCE_LIST
PartialResourceList
WindowsUtils::_CM_FULL_RESOURCE_DESCRIPTOR

Public Attributes

- INTERFACE_TYPE **InterfaceType**
- ULONG **BusNumber**
- CM_PARTIAL_RESOURCE_LIST **PartialResourceList**

The documentation for this struct was generated from the following file:

- wregistry-ddk.h

4.2 WindowsUtils::_CM_PARTIAL_RESOURCE_DESCRIPTOR Struct Reference

`#include <wregistry-ddk.h>`

Public Attributes

- UCHAR **Type**
- UCHAR **ShareDisposition**
- USHORT **Flags**
- union {
    struct {
        PHYSICAL_ADDRESS **Start**
        ULONG **Length**
    } **Generic**
    struct {
        PHYSICAL_ADDRESS **Start**
        ULONG **Length**
    } **Port**
    struct {
        ULONG **Level**
        ULONG **Vector**
        ULONG **Affinity**
    } **Interrupt**
    struct {
        PHYSICAL_ADDRESS **Start**
        ULONG **Length**
    } **Memory**
    struct {
        ULONG **Channel**
        ULONG **Port**
        ULONG **Reserved1**
    } **Dma**
    struct {
        ULONG **Data** [3]
    } **DevicePrivate**
    struct {
        ULONG **Start**
        ULONG **Length**
        ULONG **Reserved**
    } **BusNumber**
    struct {
        ULONG **DataSize**
        ULONG **Reserved1**
        ULONG **Reserved2**
    } **DeviceSpecificData**
  } **u**

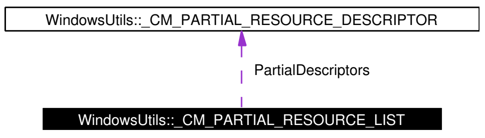
The documentation for this struct was generated from the following file:

- wregistry-ddk.h

4.3 WindowsUtils::_CM_PARTIAL_RESOURCE_LIST Struct Reference

`#include <wregistry-ddk.h>`

Collaboration diagram for WindowsUtils::_CM_PARTIAL_RESOURCE_LIST:


WindowsUtils::_CM_PARTIAL_RESOURCE_DESCRIPTOR
PartialDescriptors
WindowsUtils::_CM_PARTIAL_RESOURCE_LIST

Public Attributes

- USHORT **Version**
- USHORT **Revision**
- ULONG **Count**
- CM_PARTIAL_RESOURCE_DESCRIPTOR **PartialDescriptors** [1]

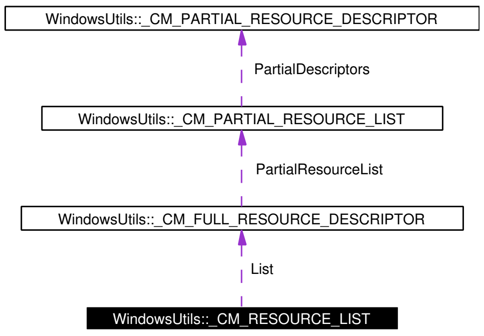
The documentation for this struct was generated from the following file:

- wregistry-ddk.h

4.4 WindowsUtils::_CM_RESOURCE_LIST Struct Reference

`#include <wregistry-ddk.h>`

Collaboration diagram for WindowsUtils::_CM_RESOURCE_LIST:


WindowsUtils::_CM_PARTIAL_RESOURCE_DESCRIPTOR
PartialDescriptors
WindowsUtils::_CM_PARTIAL_RESOURCE_LIST
PartialResourceList
WindowsUtils::_CM_FULL_RESOURCE_DESCRIPTOR
List
WindowsUtils::_CM_RESOURCE_LIST

Public Attributes

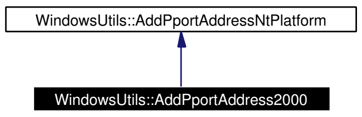
- ULONG **Count**
- CM_FULL_RESOURCE_DESCRIPTOR **List** [1]

The documentation for this struct was generated from the following file:

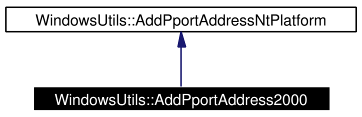
- wregistry-ddk.h

4.5 WindowsUtils::AddPportAddress2000 Class Reference

Inheritance diagram for WindowsUtils::AddPportAddress2000:


WindowsUtils::AddPportAddressNtPlatform
WindowsUtils::AddPportAddress2000

Collaboration diagram for WindowsUtils::AddPportAddress2000:


WindowsUtils::AddPportAddressNtPlatform
WindowsUtils::AddPportAddress2000

4.5.1 Detailed Description

AddPportAddress2000::operator() receives values that may have a numeric name like "0". The values with a numeric name specify a partial key that can be used to obtain further information of the device.

```
"0"="Root\\*PNP0400\\1_0_20_0_0_0"
```

These partial keys are used as follows to get at the "LogConf" key for that port:

```
"HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\" + {partial key} + "\LogConf"
```

The "BootConfig" value of the "LogConf" key provides the resource list with the parallel port address.

```
"BootConfig"=hex(8):01,00,00,00,05,00,00,00,01,00,00,00,01,00,01,00,04,00,00,\
  00,03,00,04,00,00,00,00,d8,00,00,00,00,00,00,00,02,03,00,00,00,00,00,00,d4,\
  00,00,00,00,00,40,00,00,03,00,00,00,00,00,00,d5,00,00,00,00,00,00,80,00,02,\
  03,00,00,09,00,00,00,09,00,00,00,ff,ff,ff,ff
```

Public Member Functions

- AddPportAddress2000 (PportAddressBuffer &aPportAddressBuffer)

  constructor.

- void operator() (RegistryKey::Value v) const

  determine parallel port addresses from partial registry keys with numeric names.

The documentation for this class was generated from the following file:

- wpportaddr.cpp

4.6 WindowsUtils::AddPportAddress95 Class Reference

4.6.1 Detailed Description

AddPportAddress95::operator() receives values that directly represent a parallel port address (DWORD), for example `0x378`. Unless they are zero, operator() inserts these values in the PportAddressBuffer address collection specified with the contructor.

Public Member Functions

- AddPportAddress95 (PportAddressBuffer &aPportAddressBuffer)

  constructor.

- void operator() (PportAddressBuffer::value_type v) const

  add given parallel port address to collection, unless it is zero.

Protected Attributes

- PportAddressBuffer & iPportAddresses

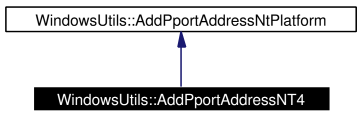
  collection with parallel port addresses

The documentation for this class was generated from the following file:

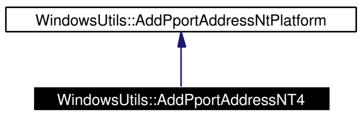
- wpportaddr.cpp

4.7 WindowsUtils::AddPportAddressNT4 Class Reference

Inheritance diagram for WindowsUtils::AddPportAddressNT4:


WindowsUtils::AddPportAddressNtPlatform
WindowsUtils::AddPportAddressNT4

Collaboration diagram for WindowsUtils::AddPportAddressNT4:


WindowsUtils::AddPportAddressNtPlatform
WindowsUtils::AddPportAddressNT4

4.7.1 Detailed Description

AddPportAddressNT4 receives values with a name and a resource list. The example below shows that there are two values for parallel port zero.

```
                                name   resource list

"\\Device\\ParallelPort0.Translated" {01,00,00,00,01,00,00,00,00,00,00,00,
                                      00,00,00,00,01,00,00,00,01,01,01,00,
                                      78,03,00,00,00,00,00,00,03,00,00,00}
       "\\Device\\ParallelPort0.Raw" {01,00,00,00,01,00,00,00,00,00,00,00,
                                      00,00,00,00,01,00,00,00,01,01,01,00,
                                      78,03,00,00,00,00,00,00,03,00,00,00}
```

We use both values and remove duplicate parallel port addresses lateron in the destructor (This is not necesary when a set<> is used as the container.).

Public Member Functions

- ∼AddPportAddressNT4 ()

  destructor; remove duplicates from address list.

- AddPportAddressNT4 (PportAddressBuffer &aPportAddressBuffer)

  constructor.

- void operator() (RegistryKey::Value v) const

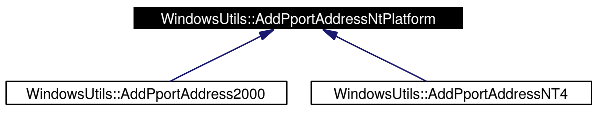
  for Windows NT 4, determine parallel port addresses from registry.

The documentation for this class was generated from the following file:

- wpportaddr.cpp

4.8 WindowsUtils::AddPportAddressNtPlatform Class Reference

Inheritance diagram for WindowsUtils::AddPportAddressNtPlatform:


WindowsUtils::AddPportAddressNtPlatform
WindowsUtils::AddPportAddress2000
WindowsUtils::AddPportAddressNT4

4.8.1 Detailed Description

class AddPportAddressNtPlatform provides function handleResourceList() that is used for Windows NT, 2000 and XP to extract the parallel port addresses from a registry resource list.

Public Member Functions

- virtual ∼AddPportAddressNtPlatform ()

  destructor.

- AddPportAddressNtPlatform (PportAddressBuffer &aPportAddressBuffer)

  constructor.

- virtual void operator() (RegistryKey::Value v) const =0

  determine parallel port address from value and insert it in collection.

- void handleResourceList (RegistryKey::ByteBuffer b) const

  extract parallel port (CmResourceTypePort) addresses from a resource list.

Protected Attributes

- PportAddressBuffer & iPportAddresses

  collection with parallel port addresses

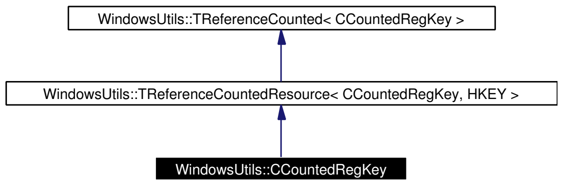
The documentation for this class was generated from the following file:

- wpportaddr.cpp

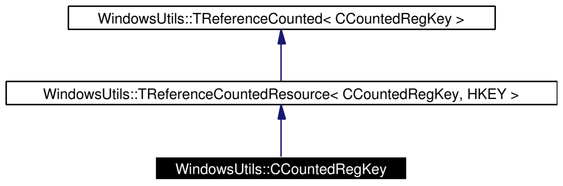
4.9 WindowsUtils::CCountedRegKey Class Reference

`#include <wregistry-crk.h>`

Inheritance diagram for WindowsUtils::CCountedRegKey:


WindowsUtils::TReferenceCounted< CCountedRegKey >
WindowsUtils::TReferenceCountedResource< CCountedRegKey, HKEY >
WindowsUtils::CCountedRegKey

Collaboration diagram for WindowsUtils::CCountedRegKey:


WindowsUtils::TReferenceCounted< CCountedRegKey >
WindowsUtils::TReferenceCountedResource< CCountedRegKey, HKEY >
WindowsUtils::CCountedRegKey

4.9.1 Detailed Description

Class CCountedRegKey provides a reference counted representation of the registry key resource that is used by class RegistryKey.

Public Member Functions

- CCountedRegKey (const HKEY hKey)

  constructor.

Protected Member Functions

- ∼CCountedRegKey ()

  destructor.

- virtual void preRelease ()

  release resource: close registry key.

Private Member Functions

- CCountedRegKey (const CCountedRegKey &rhs)

  prevent copying

- CCountedRegKey & operator= (const CCountedRegKey &rhs)

  prevent copying

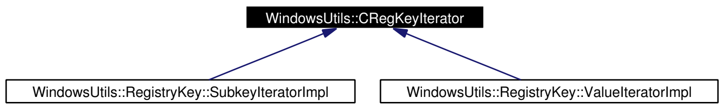
The documentation for this class was generated from the following file:

- wregistry-crk.h

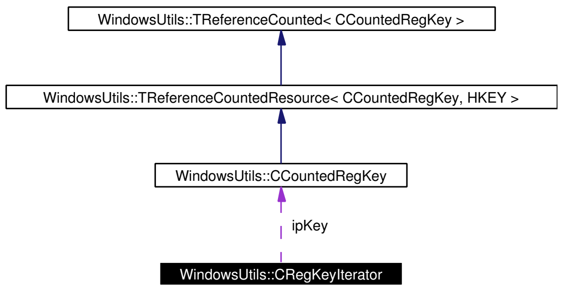
4.10 WindowsUtils::CRegKeyIterator Class Reference

`#include <wregistry-rki.h>`

Inheritance diagram for WindowsUtils::CRegKeyIterator:


WindowsUtils::CRegKeyIterator
WindowsUtils::RegistryKey::SubkeyIteratorImpl
WindowsUtils::RegistryKey::ValueIteratorImpl

Collaboration diagram for WindowsUtils::CRegKeyIterator:


WindowsUtils::TReferenceCounted< CCountedRegKey >
WindowsUtils::TReferenceCountedResource< CCountedRegKey, HKEY >
WindowsUtils::CCountedRegKey
ipKey
WindowsUtils::CRegKeyIterator

Public Member Functions

- virtual ∼CRegKeyIterator ()

  destructor.

Protected Member Functions

- CRegKeyIterator (CCountedRegKey ∗apKey)

  constructor.

- CRegKeyIterator (const CRegKeyIterator &rhs)

  copy constructor.

- CRegKeyIterator & operator= (const CRegKeyIterator &rhs)

  copy assignment.

- bool operator== (const CRegKeyIterator &rhs) const

  test for iterator equality.

Protected Attributes

- CCountedRegKey ∗ ipKey

its counted key

- DWORD iIndex

  its current index for use with the Win32 API functions

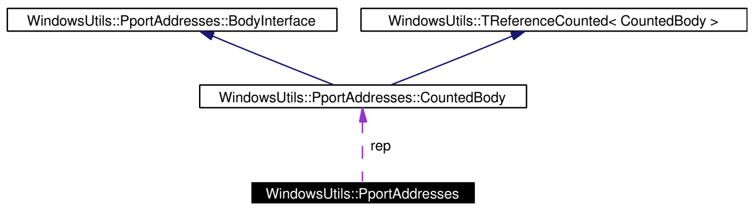
The documentation for this class was generated from the following file:

- wregistry-rki.h

4.11 WindowsUtils::PportAddresses Class Reference

`#include <wpportaddr.h>`

Collaboration diagram for WindowsUtils::PportAddresses:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
rep
WindowsUtils::PportAddresses

4.11.1 Detailed Description

Class PportAddresses determines the base addresses of the available parallel ports. These base addresses may be obtained from the BIOS data area (Windows 95 type operating systems), or from the registry (Windows NT type operating systems).

Class PportAddresses provides an iterator interface to loop through all parallel port base addresses:

```
PportAddresses thePportAddresses;

for ( PportAddresses::iterator pos = thePportAddresses.begin();
                               pos != thePportAddresses.end(); ++pos )
{
   std::cout << *pos << ' ';
}
```

Class PportAddresses uses the handle–body idiom to provide a 'virtual' constructor: the type of body it constructs, depends on the type of operating system and rep can be a Win95Body, Win98Body, WinMe-Body, WinNtBody, Win2000Body, WinXpBody or a generic Body.

Class PportAddresses is a helper class for class EppRulbusInterface of the Rulbus Device Library and of the `setrulbus` program to set the `rulbus` environment variable.

Examples:
printpportaddr.out.

Public Types

- typedef long value_type
  parallel port address type
- typedef std::vector< value_type > PportAddressBuffer
  parallel port address buffer type
- typedef PportAddressBuffer::iterator iterator
  iterator type
- typedef PportAddressBuffer::const_iterator const_iterator
  const iterator type

Public Member Functions

- ∼PportAddresses ()

  destructor.

- PportAddresses ()

  ’virtual’ constructor. construct the appropriate representation for rep.

- PportAddresses (const PportAddresses &rhs)

  copy constructor.

- PportAddresses & operator= (const PportAddresses &rhs)

  assignment.

- bool isValid (value_type base) const

  true if if the specified parallel port base address is valid.

- const char ∗ platform () const

  return a string describing the Windows version.

- iterator begin ()

  iterator for the first element.

- iterator end ()

  iterator past the last element.

- const_iterator begin () const

  iterator for the first element.

- const_iterator end () const

  iterator past the last element.

- long operator[ ] (int index) const

  random access.

Static Public Attributes

- const value_type basePCIMin = 0x0D00

  lowest acceptable base address for PCI EPP cards

- const value_type basePCIMax = 0xFFF8

  highest acceptable base address for PCI EPP cards

- const value_type basePCIStp = 8

  base address delta for PCI EPP cards

Private Attributes

- CountedBody ∗ rep

  representation

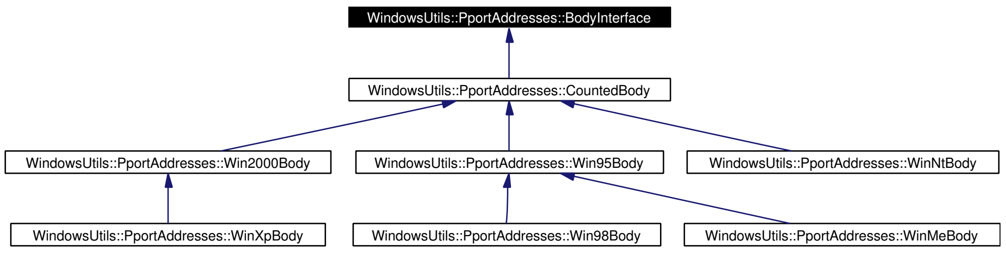
The documentation for this class was generated from the following files:

- wpportaddr.h
- wpportaddr.cpp

4.12 WindowsUtils::PportAddresses::BodyInterface Class Reference

```
#include <wpportaddr.h>
```

Inheritance diagram for WindowsUtils::PportAddresses::BodyInterface:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::Win2000Body
WindowsUtils::PportAddresses::Win95Body
WindowsUtils::PportAddresses::WinNtBody
WindowsUtils::PportAddresses::WinXpBody
WindowsUtils::PportAddresses::Win98Body
WindowsUtils::PportAddresses::WinMeBody

4.12.1 Detailed Description

Class BodyInterface specifies the interface for classes CountedBody, Win95Body, WinNtBody, Win2000Body and WinXpBody.

Public Member Functions

- virtual bool isValid (value_type base) const =0

  true if specified address is valid

- virtual bool empty () const =0

  true if address collection is empty

- virtual long size () const =0

  number of addresses in collection

- virtual iterator begin ()=0

  iterator for the first element

- virtual iterator end ()=0

  iterator past the last element

- virtual const_iterator begin () const =0

  iterator for the first element

- virtual const_iterator end () const =0

  iterator past the last element

- virtual long operator[ ] (int index) const =0

  return element at index

- virtual const char ∗ platform () const =0 *return string describing Windows version*

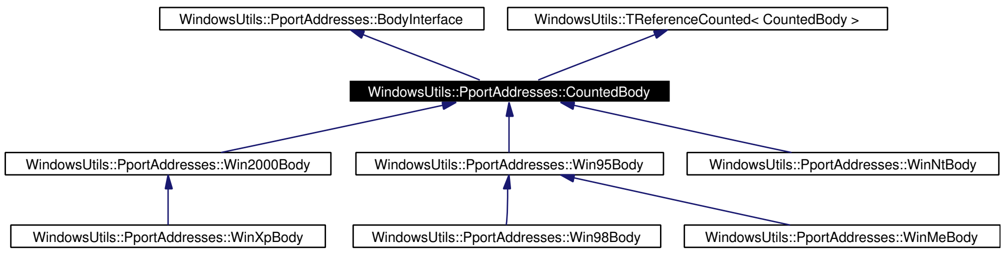
The documentation for this class was generated from the following file:

- wpportaddr.h

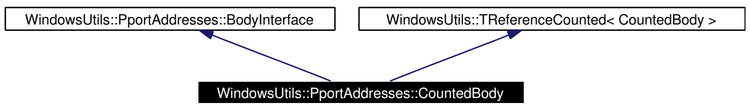
4.13 WindowsUtils::PportAddresses::CountedBody Class Reference

`#include <wpportaddr.h>`

Inheritance diagram for WindowsUtils::PportAddresses::CountedBody:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::Win2000Body
WindowsUtils::PportAddresses::Win95Body
WindowsUtils::PportAddresses::WinNtBody
WindowsUtils::PportAddresses::WinXpBody
WindowsUtils::PportAddresses::Win98Body
WindowsUtils::PportAddresses::WinMeBody

Collaboration diagram for WindowsUtils::PportAddresses::CountedBody:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody

4.13.1 Detailed Description

Class CountedBody and its derivates build a list of valid parallel port base addresses. It provides method isValid() to verify if an address is a valid parallel port base address and it provides an iterator type and methods begin() and end() to iterate through the list.

CountedBody provides the following default valid parallel port addresses.

In addition to the standard parallel port addresses from kPportAddresses, the following address range used by PCI EPP cards is valid: multiples of 8 in the range 0x0D00-0xFFF8.

Public Member Functions

- CountedBody ()

  constructor.

- bool isValid (value_type base) const

  true if specified address is valid

- bool empty () const

  true if collection is empty.

- long size () const

  number of elements.

- iterator begin ()

iterator to the first element.

- iterator end ()

  iterator past the last element.

- const_iterator begin () const

  iterator to the first element.

- const_iterator end () const

  iterator past the last element.

- long operator[ ] (int index) const

  random access to collection.

- const char ∗ platform () const

  return "Unknown" Windows version.

Protected Attributes

- PportAddressBuffer iPportAddresses

  the parallel port base address list

Static Protected Attributes

- value_type kPportAddresses [ ]

  (default) list of valid base addresses.

4.13.2 Member Data Documentation

4.13.2.1 long WindowsUtils::PportAddresses::CountedBody::kPportAddresses `[static, protected]`

Initial value:

```
{
   0x3BC, 0x378, 0x278
}
```

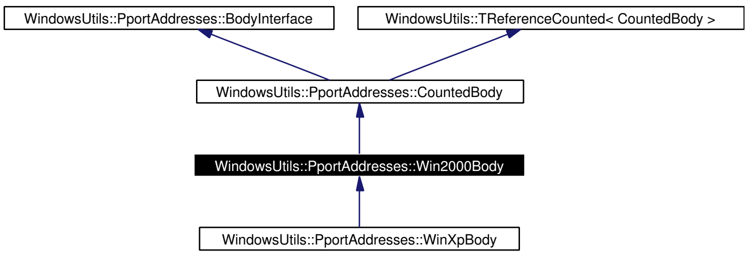
The documentation for this class was generated from the following files:

- wpportaddr.h
- wpportaddr.cpp

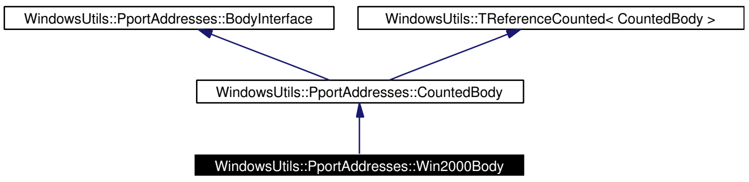
4.14 WindowsUtils::PportAddresses::Win2000Body Class Reference

`#include <wpportaddr.h>`

Inheritance diagram for WindowsUtils::PportAddresses::Win2000Body:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::Win2000Body
WindowsUtils::PportAddresses::WinXpBody

Collaboration diagram for WindowsUtils::PportAddresses::Win2000Body:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::Win2000Body

4.14.1 Detailed Description

Class Win2000Body determines the parallel port addresses from the following registry key:

```
"HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Parport\Enum"
```

The values of the Enum key look like:

```
"0"="Root\\*PNP0400\\1_0_20_0_0_0"
"Count"=dword:00000004
"NextInstance"=dword:00000004
"1"="Root\\*PNP0400\\1_0_20_2_0_0"
"2"="Root\\*PNP0400\\PnPBIOS_14"
"3"="PCI\\VEN_1407&DEV_8000&SUBSYS_00000000&REV_00\\2&ebb567f&0&78"
```

The values with a numeric name yield partial keys. These partial keys are used as follows to get at the "LogConf" key for that port:

```
"HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum\" + {partial key} + "\LogConf"
```

The "BootConfig" value of the "LogConf" key provides the resource list with the parallel port address.

Win2000Body uses class AddPportAddress2000 to obtain the "BootConfig" resource lists, extract the addresses from these lists and insert the addresses in the CountedBody::iPportAddresses collection.

Public Member Functions

- Win2000Body ()

  determine parallel port addresses from registry for Windows XP; see also AddPportAddress2000.

- bool isValid (value_type base) const

  true if specified address is valid

- const char ∗ platform () const

  return "Windows 2000"

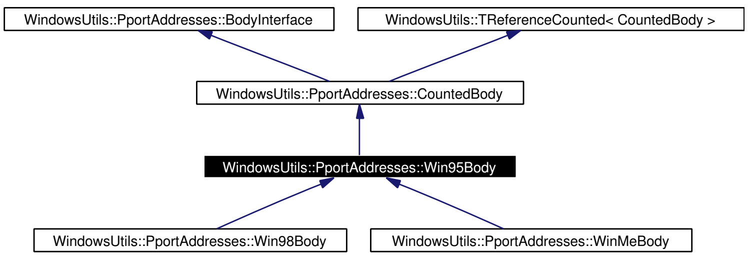
The documentation for this class was generated from the following files:

- wpportaddr.h
- wpportaddr.cpp

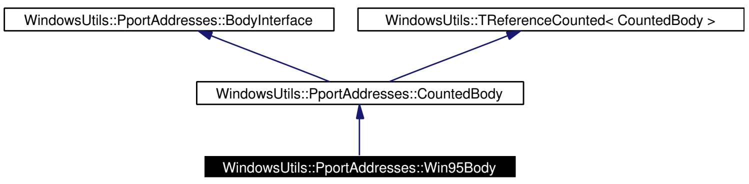
4.15 WindowsUtils::PportAddresses::Win95Body Class Reference

`#include <wpportaddr.h>`

Inheritance diagram for WindowsUtils::PportAddresses::Win95Body:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::Win95Body
WindowsUtils::PportAddresses::Win98Body
WindowsUtils::PportAddresses::WinMeBody

Collaboration diagram for WindowsUtils::PportAddresses::Win95Body:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::Win95Body

4.15.1 Detailed Description

At constuction time, class Win95Body reads the addresses of three parallel ports (printer adapters) from the process' BIOS data area starting at address 0x00000408. Win95Body uses class AddPportAddress95 to insert them in the CountedBody::iPportAddresses collection.

The code looks like:

```
WORD  biosbases[3];
DWORD dwRead;

if ( !ReadProcessMemory(
        GetCurrentProcess(), reinterpret_cast<void *>(0x00000408), biosbases, sizeof(biosbases), &dwRead )
   return;  // failed

iPportAddresses.clear();
std::for_each( biosbases, biosbases + dim(biosbases), AddPportAddress95( iPportAddresses ) );
```

Public Member Functions

- Win95Body ()

  Win95: read the parallel port base addresses from the BIOS data area; see also AddPportAddress95.

- bool isValid (value_type base) const

  true if specified address is valid

- const char ∗ platform () const

return "Windows 95"

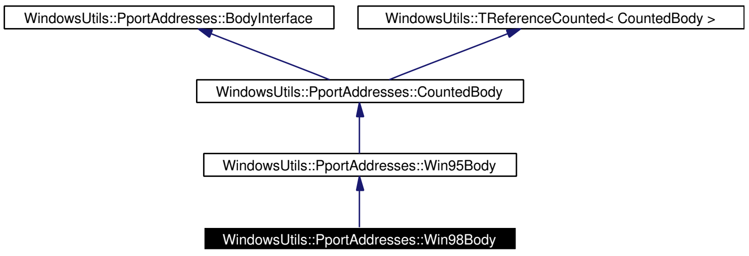
The documentation for this class was generated from the following files:

- wpportaddr.h
- wpportaddr.cpp

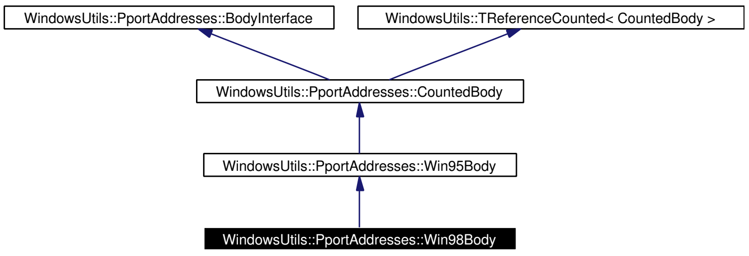
4.16 WindowsUtils::PportAddresses::Win98Body Class Reference

`#include <wpportaddr.h>`

Inheritance diagram for WindowsUtils::PportAddresses::Win98Body:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::Win95Body
WindowsUtils::PportAddresses::Win98Body

Collaboration diagram for WindowsUtils::PportAddresses::Win98Body:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::Win95Body
WindowsUtils::PportAddresses::Win98Body

4.16.1 Detailed Description

Class Win98Body uses Win95Body to determine the parallel port addresses.

Public Member Functions

- const char ∗ platform () const

  return "Windows 95"

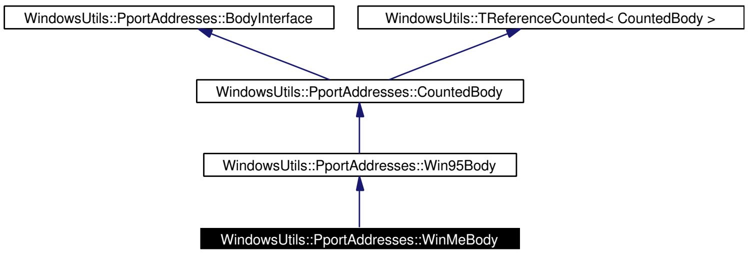
The documentation for this class was generated from the following file:

- wpportaddr.h

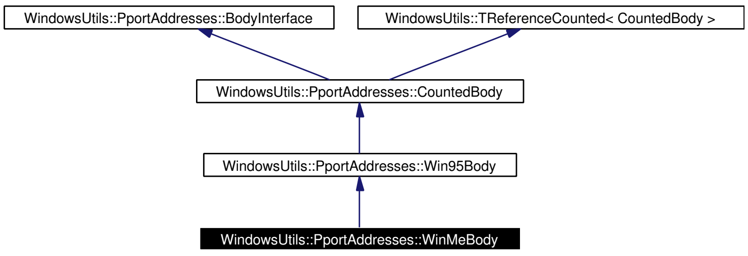
4.17 WindowsUtils::PportAddresses::WinMeBody Class Reference

`#include <wpportaddr.h>`

Inheritance diagram for WindowsUtils::PportAddresses::WinMeBody:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::Win95Body
WindowsUtils::PportAddresses::WinMeBody

Collaboration diagram for WindowsUtils::PportAddresses::WinMeBody:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::Win95Body
WindowsUtils::PportAddresses::WinMeBody

4.17.1 Detailed Description

Class WinMeBody uses Win95Body to determine the parallel port addresses.

Public Member Functions

- const char ∗ platform () const

    return "Windows ME"

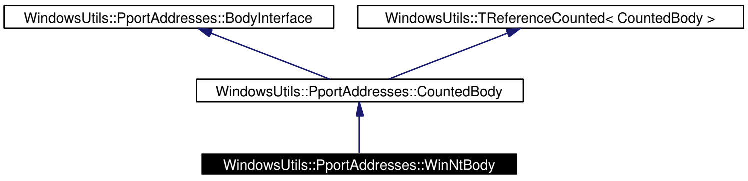
The documentation for this class was generated from the following file:

- wpportaddr.h

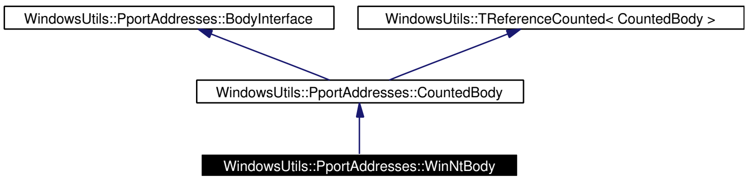
4.18 WindowsUtils::PportAddresses::WinNtBody Class Reference

`#include <wpportaddr.h>`

Inheritance diagram for WindowsUtils::PportAddresses::WinNtBody:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::WinNtBody

Collaboration diagram for WindowsUtils::PportAddresses::WinNtBody:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::WinNtBody

4.18.1 Detailed Description

Class WinNtBody determines the parallel port addresses from the following registry key:

```
"HKEY_LOCAL_MACHINE\HARDWARE\RESOURCEMAP\LOADED PARALLEL DRIVER RESOURCES\Parport"
```

The values for this key are like:

```
"\\Device\\ParallelPort0.Translated" {01,00,00,00,01,00,00,00,00,00,00,00,
                                      00,00,00,00,01,00,00,00,01,01,01,00,
                                      78,03,00,00,00,00,00,00,03,00,00,00}
       "\\Device\\ParallelPort0.Raw" {01,00,00,00,01,00,00,00,00,00,00,00,
                                      00,00,00,00,01,00,00,00,01,01,01,00,
                                      78,03,00,00,00,00,00,00,03,00,00,00}
```

Note that the same port is recorded twice: as "\Device\ParallelPort0.Translated" and as "\Device\ParallelPort0.Raw".

WinNtBody uses class AddPportAddressNT4 to extract the addresses from the resource list and insert them in the CountedBody::iPportAddresses collection.

Public Member Functions

- WinNtBody ()

  determine parallel port addresses from registry for Windows NT 4; see also AddPportAddressNT4.

- bool isValid (value_type base) const

  true if specified address is valid

- const char ∗ platform () const

  return "Windows NT"

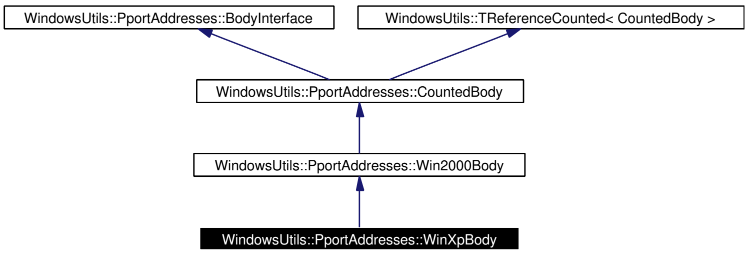
The documentation for this class was generated from the following files:

- wpportaddr.h
- wpportaddr.cpp

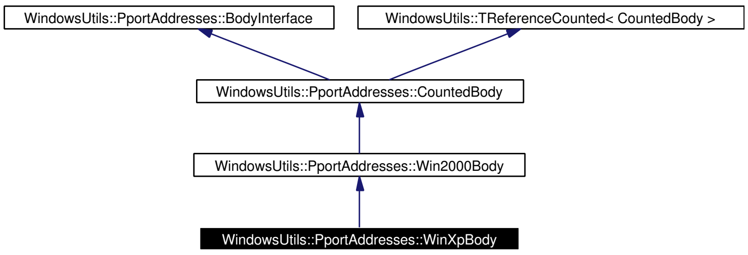
4.19 WindowsUtils::PportAddresses::WinXpBody Class Reference

`#include <wpportaddr.h>`

Inheritance diagram for WindowsUtils::PportAddresses::WinXpBody:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::Win2000Body
WindowsUtils::PportAddresses::WinXpBody

Collaboration diagram for WindowsUtils::PportAddresses::WinXpBody:


WindowsUtils::PportAddresses::BodyInterface
WindowsUtils::TReferenceCounted< CountedBody >
WindowsUtils::PportAddresses::CountedBody
WindowsUtils::PportAddresses::Win2000Body
WindowsUtils::PportAddresses::WinXpBody

4.19.1 Detailed Description

Class WinXpBody uses Win2000Body to determine the parallel port addresses.

Public Member Functions

- const char ∗ platform () const

  return "Windows XP"

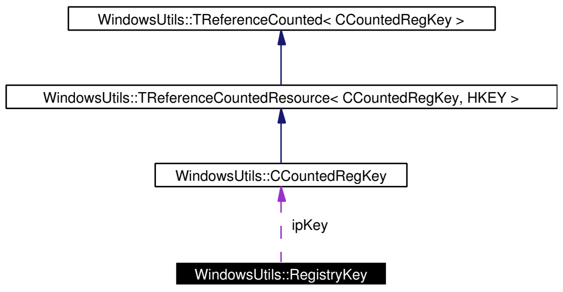
The documentation for this class was generated from the following file:

- wpportaddr.h

4.20 WindowsUtils::RegistryKey Class Reference

`#include <wregistry.h>`

Collaboration diagram for WindowsUtils::RegistryKey:


WindowsUtils::TReferenceCounted< CCountedRegKey >
WindowsUtils::TReferenceCountedResource< CCountedRegKey, HKEY >
WindowsUtils::CCountedRegKey
ipKey
WindowsUtils::RegistryKey

4.20.1 Detailed Description

The original Registry API wrapper was obtained from [CREGISTRYKEY] project at [CODEPROJECT] with this license.

These are the most important changes with respect to that version:

- various Standard Template Library (STL) type are used: string, vector
- using vector<> instead of TExpandableBuffer
- using string instead of char ∗, LPCTSTR
- doing all data access via reference counted class Value
- subclassed class Value: IntegralValue, BinaryValue, StringValue
- added class Subkey
- split class TReferenceCounted into TReferenceCounted<B> and –Resource<B,T>
- swapped class TReferenceCountedResource template arguments <T,B> to <B,T>,
- renamed GetName(), GetClass() to name() and klass()
- added method type() to class ValueIterator
- derived Subkey and Value Iterators from std::iterator<> to support use of STL alorithms
- added InsertIterator
- added (shallow/deep) copy algorithm
- added (shallow/deep) for_each algorithm
- added (shallow/deep) find_subkey algorithm
- added (shallow/deep) find_value algorithm
- added stream output methods and operators for Subkeys and Values
- added Doxygen documentation

Here is an overview of the arcitecture of the RegistryKey, RegistryKey::SubkeyIterator, Registry-Key::ValueIterator and RegistryKey::Value classes.

The following diagram shows the collaboration between the various classes that form the registry wrapper.

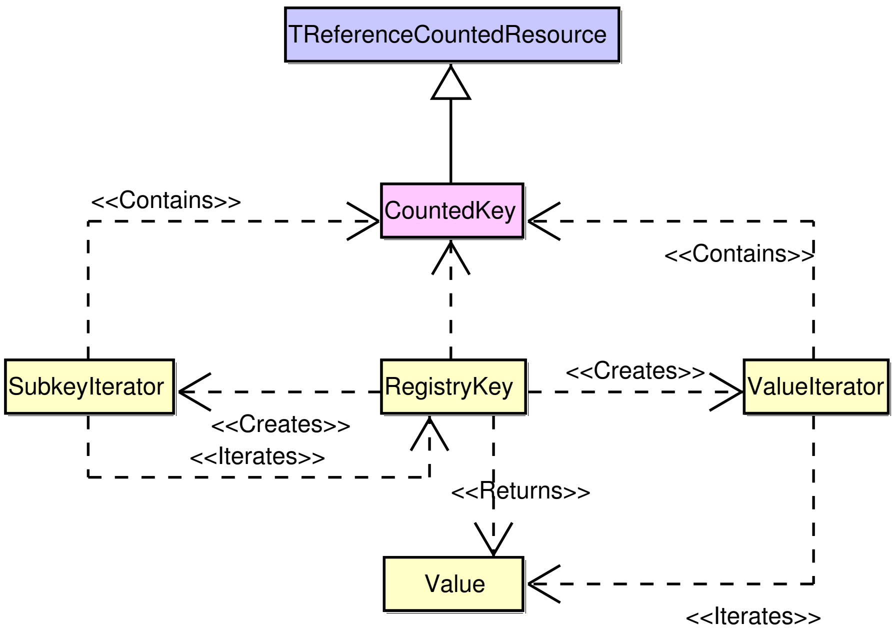


Figure 4.1: RegistryKey

The following diagram shows the architecture of the subkey and value iterators.

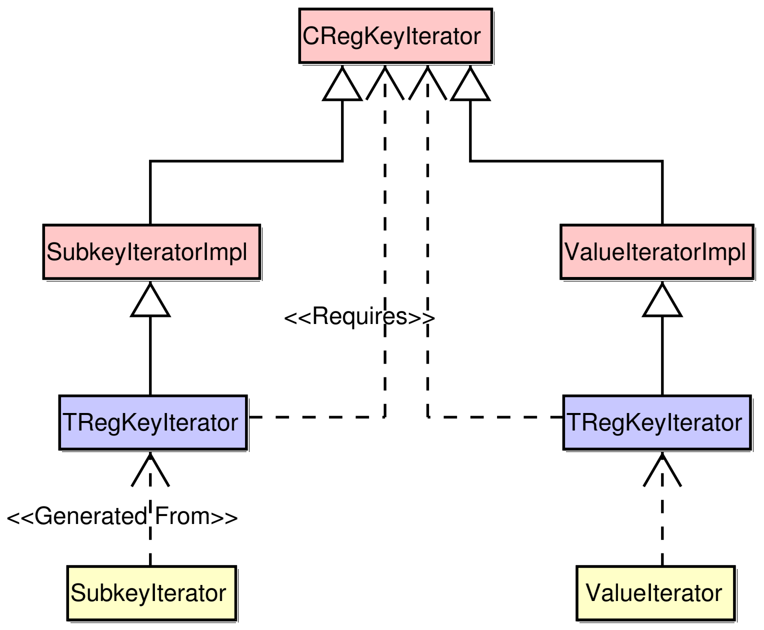


Figure 4.2: Iterator

The last diagram shows the the architecture of the value class.

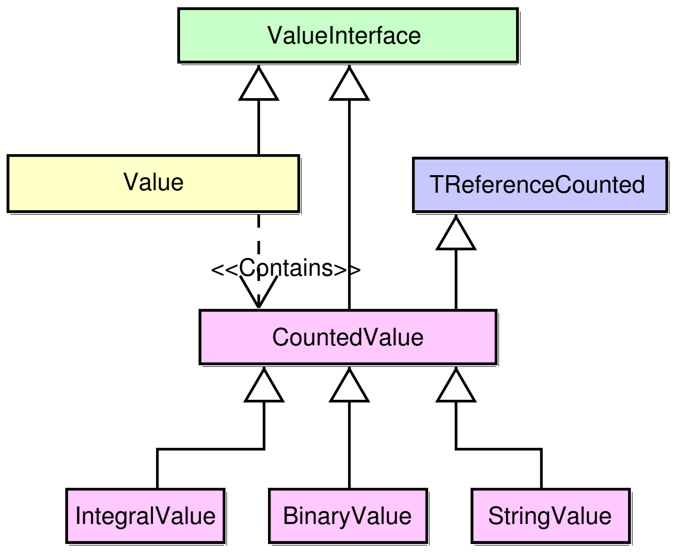


Figure 4.3: Value

Examples:
algorithm.out, algorithm2.out, printsubkeys.out, and printvalues.out.

Public Types

- typedef std::vector< BYTE > ByteBuffer

  type to hold the registry data (Win32 API).

- typedef TRegKeyIterator< SubkeyIteratorImpl > SubkeyIterator

  subkey iterator type.

- typedef TRegKeyIterator< ValueIteratorImpl > ValueIterator

  value iterator type

- typedef InsertIteratorImpl InsertIterator

  insert iterator type

Public Member Functions

- RegistryKey (HKEY hKey)

  construct from a HKEY.

- RegistryKey (const std::string &aComputerName, HKEY ahKey)

  construct from a computername and a HKEY; may throw Exception.

- RegistryKey (HKEY ahKey, const std::string &aSubkey, REGSAM aSamDesired=KEY_READ, const std::string &aComputerName="")

  construct from a HKEY, a subkey string and optionally security and machine name (not implemented); may throw Exception.

- RegistryKey (const RegistryKey &rhs)

  copy constructor.

- ~RegistryKey ()

  destructor.

- RegistryKey & operator= (const RegistryKey &rhs)

  copy assignment.

- RegistryKey & operator= (HKEY ahKey)

  assignment, HKEY.

- RegistryKey & operator= (const Subkey &rhs)

  add a subkey to this key.

- RegistryKey & operator= (const Value &rhs)

  add a value to this key.

- RegistryKey & assign (const RegistryKey &rhs)

  copy assignment.

- RegistryKey & assign (HKEY ahKey)

assignment, HKEY.

- RegistryKey & insert (const Subkey &rhs)

  add a subkey in this key.

- RegistryKey & insert (const Value &rhs)

  add a value to or change a value of this key.

- template<typename OutputIterator> OutputIterator copy (OutputIterator result, bool deep=false) const

  generic (recursive) copy algorithm.

- template<> InsertIterator copy (InsertIterator result, bool deep) const

  copy algorithm specialized for InsertIterator.

- InsertIterator copy (RegistryKey &tokey, bool deep=false) const

  (recursively) copy to another key.

- template<typename UnaryOp> UnaryOp for_each (UnaryOp op, bool deep=false) const

  (recursively) visit this key's values and subkeys.

- RegistryKey connectRegistry (std::string aComputerName) const

  connect to registry on specified computer.

- RegistryKey openKey (const std::string &aSubkey, REGSAM aSamDesired=KEY_READ) const

  open the specified subkey, default with KEY_READ access.

- RegistryKey createKey (const std::string &aSubkey, const std::string &aClass="", DWORD aOptions=REG_OPTION_NON_VOLATILE, REGSAM aSamDesired=KEY_ALL_ACCESS, LPSECURITY_ATTRIBUTES apSecurityAttributes=NULL) const

  create the specified subkey, default with KEY_ALL_ACCESS access.

- RegistryKey createOrOpenKey (const std::string &aSubkey, DWORD ∗apDisposition=NULL, const std::string &aClass="", DWORD aOptions=REG_OPTION_NON_VOLATILE, REGSAM aSamDesired=KEY_ALL_ACCESS, LPSECURITY_ATTRIBUTES apSecurityAttributes=NULL) const

  create or open the specified subkey, default with KEY_ALL_ACCESS access.

- void deleteKey (const std::string &aSubkey) const

  delete the specified key. The behaviour of RegDeleteKey() on Windows 95 differs from Windows NT. -Win95: subkeys are also deleted -WinNT: subkeys are not deleted To improve consistency, deleteKey() will always fail to delete a key with subkeys, whereas DeleteKeyAndSubkey will always work.

- void deleteKeyAndSubkeys (const std::string &aSubkey) const

  delete the specified key with its subkeys.

- bool hasSubkey (const std::string &aSubkey, REGSAM aSamDesired=KEY_READ) const

  true if specified subkey exists.

- void loadKey (std::string aSubkeyName, std::string aFilename) const

  load specified subkey from file into registry.

- void unloadKey (std::string aSubkeyName) const

  unload specified key and subkeys from the registry.

- void saveKey (std::string aFilename, LPSECURITY_ATTRIBUTES aSecurityAttributes=NULL) const

  save this key and all of its subkeys and values to a new file.

- void restoreKey (std::string aFilename, DWORD aFlags=0) const

  copy the registry information from the specified file over this key.

- void replaceKey (std::string aNewFilename, std::string aBupFilename, std::string aSubkeyName="") const

  replace the file containing a key and all its subkeys with another file; a restart activates the new values.

- ByteBuffer getKeySecurity (SECURITY_INFORMATION aSecurityInformation) const

  return a copy of the security descriptor of this registry key.

- void setKeySecurity (SECURITY_INFORMATION aSecurityInformation, const ByteBuffer &a-SecurityDescriptor) const

  set the security of this registry key.

- void notifyChangeKeyValue (HANDLE ahEvent, bool aAndSubkeys=false, DWORD aNotify-Filter=REG_NOTIFY_CHANGE_LAST_SET) const

  asynchronously notify the caller about changes to the attributes or contents of this key.

- void notifyChangeKeyValue (bool aAndSubkeys=false, DWORD aNotifyFilter=REG_NOTIFY_-CHANGE_LAST_SET) const

  synchronously notify the caller (wait) about changes to the attributes or contents of this key.

- void flushKey () const

  writes all the attributes of this key into the registry.

- operator HKEY () const

  get its HKEY.

- template<typename UnaryPredicate> SubkeyIterator find_subkey (UnaryPredicate op, bool deep=false) const

  (recursively) look for subkey matching criterion; return SubkeyIterator.

- SubkeyIterator find_subkey (std::string aSubkeyName, bool deep=false) const

  recursively look for the specified subkey; return SubkeyIterator if found, otherwise return endSubkey-Iteration().

- SubkeyIterator beginSubkeyIteration () const

  return SubkeyIterator for first subkey.

- SubkeyIterator endSubkeyIteration () const

  return SubkeyIterator after last subkey.

- void deleteValue (const std::string &aValueName) const

  delete specified value.

- void setValue (const Value &aValue) const

  set value.

- Value queryValue (const std::string &aValueName) const

  return the Value for the specified value name; may throw Exception.

- template<typename UnaryPredicate> std::pair< RegistryKey, ValueIterator > find_value (UnaryPredicate op, bool deep=false) const

  (recursively) look for value that matches criterion; return RegistryKey,ValueIterator pair.

- std::pair< RegistryKey, ValueIterator > find_value (std::string aValueName, bool deep=false) const

  (recursively) look for a value by its name; return RegistryKey,ValueIterator pair.

- ValueIterator beginValueIteration () const

  return ValueIterator for first value.

- ValueIterator endValueIteration () const

  return ValueIterator after last value.

- InsertIterator beginInsertIteration ()

  return InsertIterator for first value.

Static Private Member Functions

- CCountedRegKey ∗ NewCountedKey (HKEY ahKey, bool abCloseKeyOnFailure=false)

  create a counted key for the representation.

Private Attributes

- CCountedRegKey ∗ ipKey

  its counted key representation

4.20.2 Member Function Documentation

4.20.2.1 RegistryKey::InsertIterator WindowsUtils::RegistryKey::copy (InsertIterator *result*, bool *deep*) const

```
#include <algorithm>
```

Copy this key's contents into the destination `result`

This copy algorithm is a version specialized for InsertIterators.

Parameters:

result destination

deep if true, also traverse subtrees [false]

Returns:

position after the last copied element in the destination range

4.20.2.2 template<typename OutputIterator> OutputIterator WindowsUtils::RegistryKey::copy (OutputIterator *result*, bool *deep* = false) const

`#include <algorithm>`

Copy this key's contents into the destination `result`

You must ensure that the destination range is big enough, or that an insert iterator is used.

Parameters:

result destination

deep if true, also traverse subtrees [false]

Returns:

position after the last copied element in the destination range

4.20.2.3 template<typename UnaryPredicate> RegistryKey::SubkeyIterator WindowsUtils::RegistryKey::find_subkey (UnaryPredicate *op*, bool *deep* = false) const

`#include <algorithm>`

Find subkey that matches criterion `op` and return its position.

Parameters:

op unary operation: bool operator()() required

deep if true, also traverse subtrees [false]

Returns:

position or endSubkeyIteration()

4.20.2.4 template<typename UnaryPredicate> std::pair< RegistryKey, RegistryKey::ValueIterator > WindowsUtils::RegistryKey::find_value (UnaryPredicate *op*, bool *deep* = false) const

`#include <algorithm>`

Find value that matches criterion `op` and return the key containing the value and the value's position.

The result is returned as:

```
std::pair<RegistryKey, RegistryKey::ValueIterator>
```

Parameters:

op unary operation: bool operator()() required

deep if true, also traverse subtrees [false]

Returns:

{key,position} or {key,endValueIteration()}

4.20.2.5 template<typename UnaryOp> UnaryOp WindowsUtils::RegistryKey::for_each (UnaryOp *op*, bool *deep* = false) const

```
#include <algorithm>
```

Apply unary operation `op` to each Subkey and Value.

Besides defining operator()(T), UnaryOp must also define operator++() and operator–() to keep track of the subkey level.

Parameters:

op unary operation: operator()(), operator++() and operator–() required

deep if true, also traverse subtrees [false]

Returns:

op

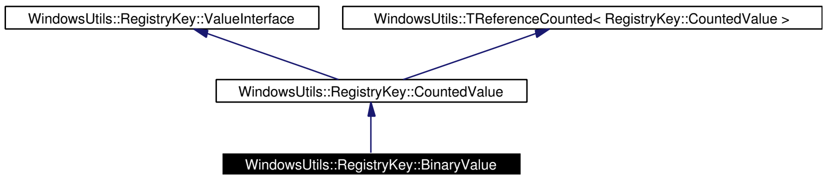
The documentation for this class was generated from the following files:

- wregistry.h
- wregistry-file.cpp
- wregistry-sec.cpp
- wregistry-siter.cpp
- wregistry-subkey.cpp
- wregistry-value.cpp
- wregistry-viter.cpp
- wregistry.cpp

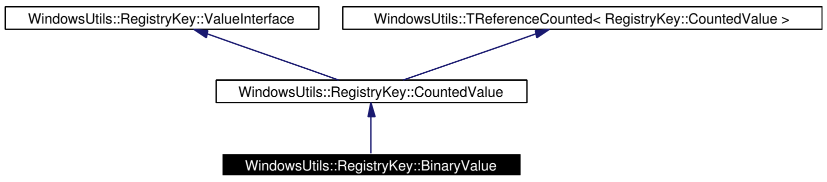
4.21 WindowsUtils::RegistryKey::BinaryValue Class Reference

`#include <wregistry.h>`

Inheritance diagram for WindowsUtils::RegistryKey::BinaryValue:


WindowsUtils::RegistryKey::ValueInterface
WindowsUtils::TReferenceCounted< RegistryKey::CountedValue >
WindowsUtils::RegistryKey::CountedValue
WindowsUtils::RegistryKey::BinaryValue

Collaboration diagram for WindowsUtils::RegistryKey::BinaryValue:


WindowsUtils::RegistryKey::ValueInterface
WindowsUtils::TReferenceCounted< RegistryKey::CountedValue >
WindowsUtils::RegistryKey::CountedValue
WindowsUtils::RegistryKey::BinaryValue

Public Member Functions

- BinaryValue (DWORD adwType, const std::string &aName, const ByteBuffer &aBuffer)

  construct from type, name and data; see Value::Value(DWORD,const std::string&,ByteBuffer).

- operator ByteBuffer () const

  convert to ByteBuffer.

- BYTE operator[ ] (int index) const

  random access into ByteBuffer; may throw std::range_error exception.

- std::string toString () const

  return string representation of value, for example "00 e5 ff".

Private Attributes

- ByteBuffer iBuffer

  its data

- std::string iCache

  its cached string representation

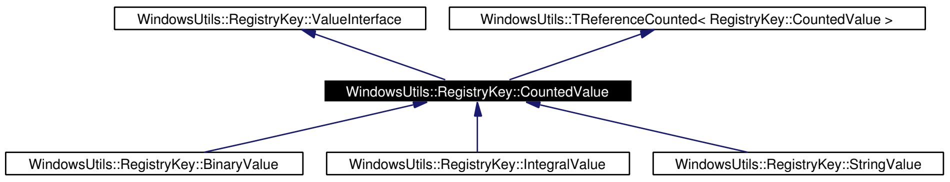
The documentation for this class was generated from the following files:

- wregistry.h
- wregistry-value.cpp

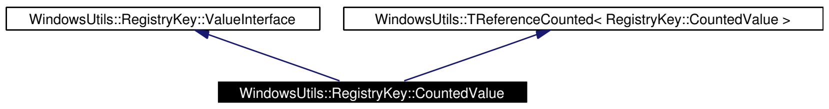
4.22 WindowsUtils::RegistryKey::CountedValue Class Reference

```
#include <wregistry.h>
```

Inheritance diagram for WindowsUtils::RegistryKey::CountedValue:


WindowsUtils::RegistryKey::ValueInterface
WindowsUtils::TReferenceCounted< RegistryKey::CountedValue >
WindowsUtils::RegistryKey::CountedValue
WindowsUtils::RegistryKey::BinaryValue
WindowsUtils::RegistryKey::IntegralValue
WindowsUtils::RegistryKey::StringValue

Collaboration diagram for WindowsUtils::RegistryKey::CountedValue:


WindowsUtils::RegistryKey::ValueInterface
WindowsUtils::TReferenceCounted< RegistryKey::CountedValue >
WindowsUtils::RegistryKey::CountedValue

Public Member Functions

- CountedValue (DWORD adwType, const std::string &aName)

  construct from type and name.

- operator DWORD () const

  default implementation: throw Exception.

- operator ByteBuffer () const

  default implementation: throw Exception.

- operator std::string () const

  default implementation: throw Exception.

- BYTE operator[ ] (int index) const

  default implementation: throw Exception.

- std::string toString () const =0

  return string representation of value

- std::string name () const

  return counted value's name.

- DWORD type () const

  return counted value's type; see Value::Value(DWORD,const std::string&,ByteBuffer).

Private Attributes

- DWORD idwType

    its type

- std::string iName

    its name

The documentation for this class was generated from the following files:

- wregistry.h
- wregistry-value.cpp

4.23 WindowsUtils::RegistryKey::CountedValue::Exception Class Reference

`#include <wregistry.h>`

Public Member Functions

- Exception (const std::string &s)

  constructor.

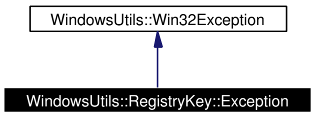
The documentation for this class was generated from the following file:

- wregistry.h

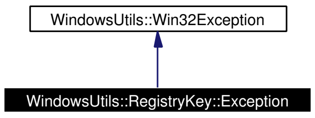
4.24 WindowsUtils::RegistryKey::Exception Class Reference

`#include <wregistry.h>`

Inheritance diagram for WindowsUtils::RegistryKey::Exception:


WindowsUtils::Win32Exception
WindowsUtils::RegistryKey::Exception

Collaboration diagram for WindowsUtils::RegistryKey::Exception:


WindowsUtils::Win32Exception
WindowsUtils::RegistryKey::Exception

Public Member Functions

- Exception (const std::string &aWhere, LONG aError)

  constructor.

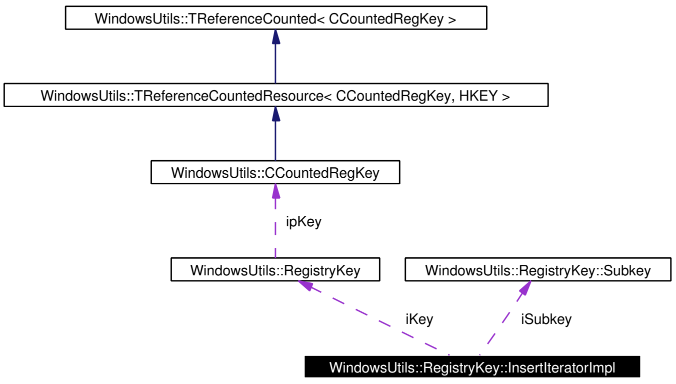
The documentation for this class was generated from the following file:

- wregistry.h

4.25 WindowsUtils::RegistryKey::InsertIteratorImpl Class Reference

```
#include <wregistry.h>
```

Collaboration diagram for WindowsUtils::RegistryKey::InsertIteratorImpl:


WindowsUtils::TReferenceCounted< CCountedRegKey >
WindowsUtils::TReferenceCountedResource< CCountedRegKey, HKEY >
WindowsUtils::CCountedRegKey
ipKey
WindowsUtils::RegistryKey
WindowsUtils::RegistryKey::Subkey
iKey
iSubkey
WindowsUtils::RegistryKey::InsertIteratorImpl

4.25.1 Detailed Description

A Note About the InsertIterator Implementation

The InsertIterator supports descending the subkey tree being created with openKey().

Another idea was to automatically open a subkey when it is assigned to the insert iterator and revert to its parent when `operator++()` is called on the insert iterator.

However, I found it more clear to explicitly use openKey().

Public Member Functions

- InsertIteratorImpl & operator ∗ ()
  object access (no-op).
- InsertIteratorImpl & operator++ ()
  prefix increment (no-op).
- InsertIteratorImpl operator++ (int)
  postfix increment (no-op).
- InsertIteratorImpl & operator= (const Value &rhs)
  insert Value.
- InsertIteratorImpl & operator= (const Subkey &rhs)
  insert Subkey.
- RegistryKey openKey () const
  open subkey last inserted.

Protected Member Functions

- InsertIteratorImpl (RegistryKey &aKey)

  construct from counted key in RegistryKey.

Private Attributes

- RegistryKey iKey

  its registry key

- Subkey iSubkey

  its last assigned subkey

Friends

- class **RegistryKey**

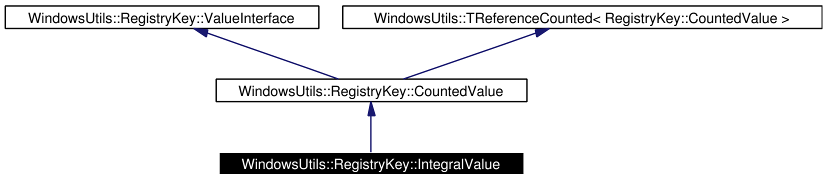
The documentation for this class was generated from the following files:

- wregistry.h
- wregistry-iiter.cpp

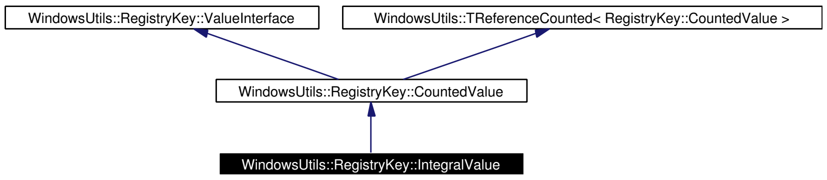
4.26 WindowsUtils::RegistryKey::IntegralValue Class Reference

`#include <wregistry.h>`

Inheritance diagram for WindowsUtils::RegistryKey::IntegralValue:


WindowsUtils::RegistryKey::ValueInterface
WindowsUtils::TReferenceCounted< RegistryKey::CountedValue >
WindowsUtils::RegistryKey::CountedValue
WindowsUtils::RegistryKey::IntegralValue

Collaboration diagram for WindowsUtils::RegistryKey::IntegralValue:


WindowsUtils::RegistryKey::ValueInterface
WindowsUtils::TReferenceCounted< RegistryKey::CountedValue >
WindowsUtils::RegistryKey::CountedValue
WindowsUtils::RegistryKey::IntegralValue

Public Member Functions

- IntegralValue (DWORD adwType, const std::string &aName, DWORD aWord)

  construct from type, name and data; see Value::Value(DWORD,const std::string&,ByteBuffer).

- IntegralValue (DWORD adwType, const std::string &aName, const ByteBuffer &aBuffer)

  construct from type, name and data; see Value::Value(DWORD,const std::string&,ByteBuffer).

- operator DWORD () const

  convert value to DWORD.

- std::string toString () const

  return string representation of value, for example "0x123".

Private Attributes

- DWORD idwValue

  its value

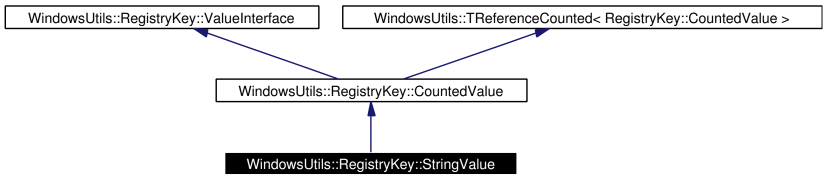
The documentation for this class was generated from the following files:

- wregistry.h
- wregistry-value.cpp

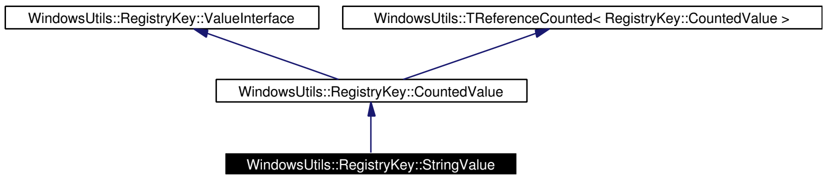
4.27 WindowsUtils::RegistryKey::StringValue Class Reference

`#include <wregistry.h>`

Inheritance diagram for WindowsUtils::RegistryKey::StringValue:


WindowsUtils::RegistryKey::ValueInterface
WindowsUtils::TReferenceCounted< RegistryKey::CountedValue >
WindowsUtils::RegistryKey::CountedValue
WindowsUtils::RegistryKey::StringValue

Collaboration diagram for WindowsUtils::RegistryKey::StringValue:


WindowsUtils::RegistryKey::ValueInterface
WindowsUtils::TReferenceCounted< RegistryKey::CountedValue >
WindowsUtils::RegistryKey::CountedValue
WindowsUtils::RegistryKey::StringValue

Public Member Functions

- StringValue (DWORD adwType, const std::string &aName, const std::string &aString)

  construct from type, name and string.

- StringValue (DWORD adwType, const std::string &aName, const ByteBuffer &aBuffer)

  construct from type, name and data; see Value::Value(DWORD,const std::string&,ByteBuffer).

- std::string toString () const

  return string representation of value, for example "hello world.".

Private Attributes

- std::string iString

  its string

The documentation for this class was generated from the following files:

- wregistry.h
- wregistry-value.cpp

4.28 WindowsUtils::RegistryKey::Subkey Class Reference

`#include <wregistry.h>`

Public Member Functions

- Subkey (const std::string &aName="", const std::string &aClass="")

  (default) constructor.

- Subkey (const Subkey &rhs)

  copy constructor.

- Subkey & operator= (const Subkey &rhs)

  copy assignment.

- std::string name () const

  return subkey's name.

- std::string klass () const

  return subkey's class name.

- std::string operator ∗ () const

  return subkey's name.

- std::ostream & printOn (std::ostream &strm) const

  print the subkey's string representation on the specified stream.

Private Attributes

- std::string iName

  subkey name

- std::string iClass

  subkey class name

Friends

- class **SubkeyIteratorImpl**

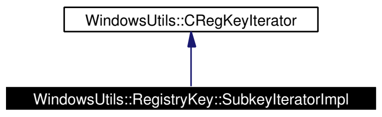
The documentation for this class was generated from the following file:

- wregistry.h

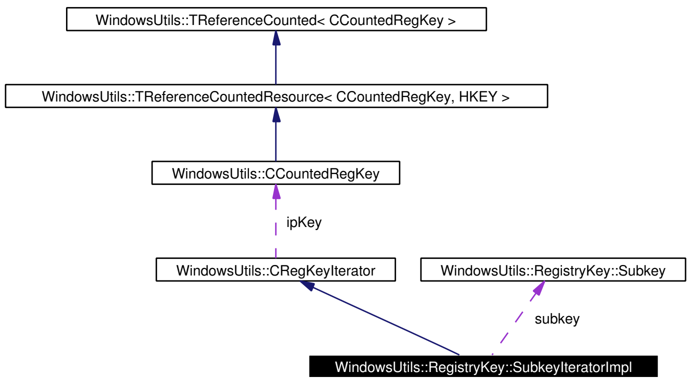
4.29 WindowsUtils::RegistryKey::SubkeyIteratorImpl Class Reference

`#include <wregistry.h>`

Inheritance diagram for WindowsUtils::RegistryKey::SubkeyIteratorImpl:


WindowsUtils::CRegKeyIterator
WindowsUtils::RegistryKey::SubkeyIteratorImpl

Collaboration diagram for WindowsUtils::RegistryKey::SubkeyIteratorImpl:


WindowsUtils::TReferenceCounted< CCountedRegKey >
WindowsUtils::TReferenceCountedResource< CCountedRegKey, HKEY >
WindowsUtils::CCountedRegKey
ipKey
WindowsUtils::CRegKeyIterator
WindowsUtils::RegistryKey::Subkey
subkey
WindowsUtils::RegistryKey::SubkeyIteratorImpl

Public Member Functions

- RegistryKey openKey (REGSAM aSamDesired=KEY_READ) const

  open current subkey, default with read access.

- std::string name () const

  return current subkey's name.

- std::string klass () const

  return subkey's class name.

- reference operator $*$ ()

  return current subkey.

- pointer operator $\rightarrow$ ()

  return pointer to current subkey.

Protected Member Functions

- SubkeyIteratorImpl (CCountedRegKey $*$apKey) *construct from counted key in RegistryKey.*

- bool getItem ()

  read current subkey information from registry; may throw Exception.

- bool operator== (const SubkeyIteratorImpl &rhs) const

  test if subkey iterators are equal.

Private Attributes

- Subkey subkey

  current subkey

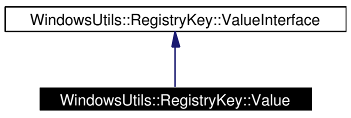
The documentation for this class was generated from the following files:

- wregistry.h
- wregistry-siter.cpp

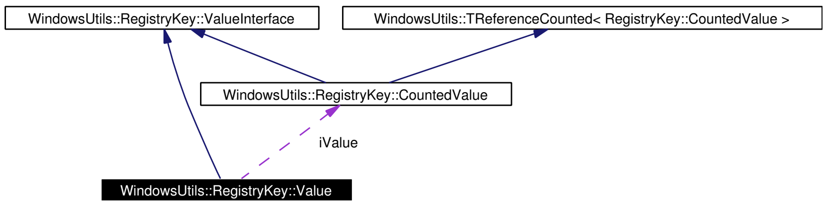
4.30 WindowsUtils::RegistryKey::Value Class Reference

`#include <wregistry.h>`

Inheritance diagram for WindowsUtils::RegistryKey::Value:


WindowsUtils::RegistryKey::ValueInterface
WindowsUtils::RegistryKey::Value

Collaboration diagram for WindowsUtils::RegistryKey::Value:


WindowsUtils::RegistryKey::ValueInterface
WindowsUtils::TReferenceCounted< RegistryKey::CountedValue >
WindowsUtils::RegistryKey::CountedValue
iValue
WindowsUtils::RegistryKey::Value

Public Member Functions

- ∼Value ()

  destructor.

- Value (const std::string &aName, DWORD aWord)

  construct Value from name and number.

- Value (const std::string &aName, const std::string &aString)

  construct Value from name and string.

- Value (const std::string &aName, const ByteBuffer &aBuffer)

  construct Value from name and data.

- Value (DWORD adwType, const std::string &aName, const ByteBuffer &aBuffer)

  construct Value from type, name and data.

- Value (const Value &rhs)

  copy constructor.

- Value & operator= (const Value &rhs)

  copy assignment.

- operator DWORD () const

  convert value to DWORD; may throw Exception.

- operator ByteBuffer () const

  convert value to ByteBuffer; may throw Exception.

- operator std::string () const

  convert value to std::string; may throw Exception.

- BYTE operator[ ] (int index) const

  random access into ByteBuffer; may throw Exception.

- std::string toString () const

  return string representation of value's data.

- std::string name () const

  return value's name.

- DWORD type () const

  return value's type; see Value::Value(DWORD,const std::string&,ByteBuffer).

- std::ostream & printOn (std::ostream &strm) const

  print the value's string representation on the specified stream.

Private Attributes

- CountedValue ∗ iValue

  its counted value representation

The documentation for this class was generated from the following files:

- wregistry.h
- wregistry-value.cpp

4.31 WindowsUtils::RegistryKey::ValueBuffer Class Reference

`#include <wregistry.h>`

Public Member Functions

- ValueBuffer (const Value &rhs)

    constructor.

Public Attributes

- ByteBuffer **buffer**

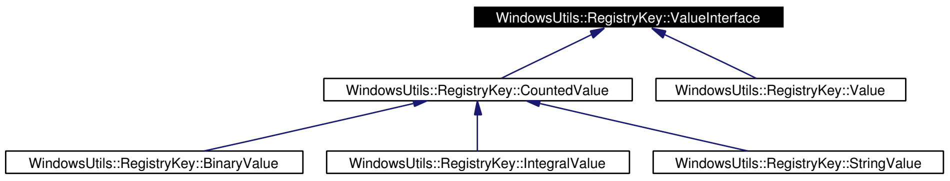
The documentation for this class was generated from the following files:

- wregistry.h
- wregistry-value.cpp

4.32 WindowsUtils::RegistryKey::ValueInterface Class Reference

`#include <wregistry.h>`

Inheritance diagram for WindowsUtils::RegistryKey::ValueInterface:


WindowsUtils::RegistryKey::ValueInterface
WindowsUtils::RegistryKey::CountedValue
WindowsUtils::RegistryKey::Value
WindowsUtils::RegistryKey::BinaryValue
WindowsUtils::RegistryKey::IntegralValue
WindowsUtils::RegistryKey::StringValue

Public Member Functions

- virtual operator DWORD () const =0

  convert value to DWORD

- virtual operator ByteBuffer () const =0

  convert value to ByteBuffer

- virtual operator std::string () const =0

  convert value to std::string

- virtual BYTE operator[ ] (int index) const =0

  random access into ByteBuffer

- virtual std::string toString () const =0

  return string representation of value

- virtual std::string name () const =0

  return value's name

- virtual DWORD type () const =0

  return value's type

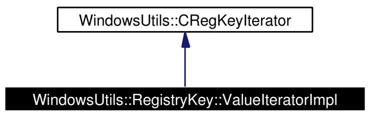
The documentation for this class was generated from the following file:

- wregistry.h

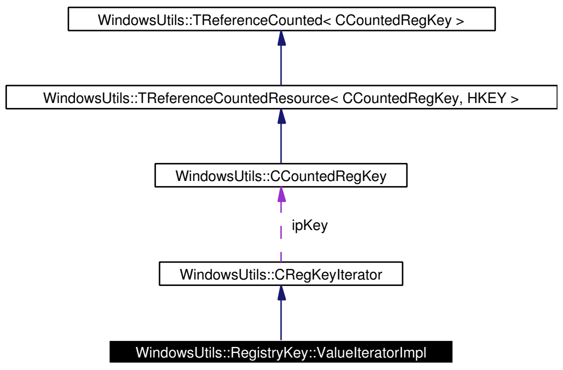
4.33 WindowsUtils::RegistryKey::ValueIteratorImpl Class Reference

```
#include <wregistry.h>
```

Inheritance diagram for WindowsUtils::RegistryKey::ValueIteratorImpl:


WindowsUtils::CRegKeyIterator
WindowsUtils::RegistryKey::ValueIteratorImpl

Collaboration diagram for WindowsUtils::RegistryKey::ValueIteratorImpl:


WindowsUtils::TReferenceCounted< CCountedRegKey >
WindowsUtils::TReferenceCountedResource< CCountedRegKey, HKEY >
WindowsUtils::CCountedRegKey
ipKey
WindowsUtils::CRegKeyIterator
WindowsUtils::RegistryKey::ValueIteratorImpl

Public Member Functions

- std::string name () const

  return iterator's current value's name.

- DWORD type () const

  return iterator's current value's type.

- std::string toString () const

  return a string representation for iterator's current value.

- **operator value_type** () const
- value_type operator ∗ () const

  return iterator's current value.

Protected Member Functions

- ValueIteratorImpl (CCountedRegKey ∗apKey)

  construct from counted key in RegistryKey.

- bool getItem ()

  read current value information from registry; may throw Exception.

- bool operator== (const ValueIteratorImpl &rhs) const

  test if value iterators are equal.

Private Attributes

- std::string iName

  current value name

- ByteBuffer iBuffer

  current value data

- DWORD idwType

  current value type

The documentation for this class was generated from the following files:

- wregistry.h
- wregistry-viter.cpp

4.34 WindowsUtils::Semaphore Class Reference

```
#include <wsemaphore.h>
```

4.34.1 Detailed Description

Class Sempahore is obtained from [WIN32SEMAPHORE] at [BBDSOFT] with this license.

Public Member Functions

- Semaphore (const char ∗const inName=0, const unsigned long inInitialCount=0, const unsigned long inMaximumCount=0)

  constructor.

- ∼Semaphore ()

  destructor.

- Semaphore & post (const unsigned long inPostCount=1)

  release semaphore.

- Semaphore & wait (const unsigned long inTimeout=0)

  acquire semaphore.

- unsigned long maxCount () const

  return theMaxCount.

Private Member Functions

- Semaphore (const Semaphore &)

  disable copy constructor and assignment operator.

- Semaphore & **operator=** (const Semaphore &)

Private Attributes

- unsigned theSemaphoreHandle

  semaphore handle.

- unsigned long theMaxCount

  maximum count.

4.34.2 Constructor & Destructor Documentation

4.34.2.1 WindowsUtils::Semaphore::Semaphore (const char ∗const *inName* = 0, const unsigned long *inInitialCount* = 0, const unsigned long *inMaximumCount* = 0)

Create the semaphore named `inName` with initial count `inInitialCount` and maximum count `inMaximumCount`. If the named semaphore already exists, that semaphore is used. If the semaphore could not be created or opened the following runtime error is generated: "Semaphore: failed to create.".

Exceptions:

Semaphore: failed to create. if the semaphore could not be created or opened this runtime error message is generated.

Parameters:

inName a null-terminated string specifying the name of the semaphore object. The name is limited to MAX_PATH characters, and can contain any character except the backslash path-separator character (\). Name comparison is case sensitive. If lpName matches the name of an existing named semaphore object, this function requests SEMAPHORE_ALL_ACCESS access to the existing object. In this case, the lInitialCount and lMaximumCount parameters are ignored because they have already been set by the creating process. If the lpSemaphoreAttributes parameter is not NULL, it determines whether the handle can be inherited, but its security-descriptor member is ignored.

If lpName is NULL, the semaphore object is created without a name. If lpName matches the name of an existing event, mutex, or file- mapping object, the function fails and the GetLastError function returns ERROR_INVALID_HANDLE. This occurs because event, mutex, semaphore, and file-mapping objects share the same name space.

inInitialCount specifies an initial count for the semaphore object. This value must be greater than or equal to zero and less than or equal to lMaximumCount. The state of a semaphore is signaled when its count is greater than zero and nonsignaled when it is zero. The count is decreased by one whenever a wait function releases a thread that was waiting for the semaphore. The count is increased by a specified amount by calling the ReleaseSemaphore function.

inMaximumCount specifies the maximum count for the semaphore object. This value must be greater than zero.

4.34.2.2 WindowsUtils::Semaphore::∼Semaphore ()

The destructor closes the semaphore.

Exceptions:

Semaphore: failed to close. if the semaphore could not be closed this runtime error message is generated.

4.34.3 Member Function Documentation

4.34.3.1 Semaphore & WindowsUtils::Semaphore::post (const unsigned long *inPostCount* = 1)

post() increments semaphore counter by inPostCount. inPostCount must not be 0.

Parameters:

inPostCount [1]

Returns:

4.34.3.2 Semaphore & WindowsUtils::Semaphore::wait (const unsigned long *inTimeout* = 0)

wait() waits while semaphore counter is greater than 0 and decrements semaphore counter by 1.

Parameters:

inTimeout [0]

Returns:

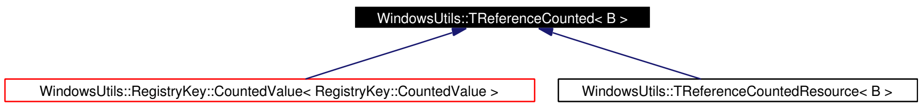
The documentation for this class was generated from the following files:

- wsemaphore.h
- wsemaphore.cpp

4.35 WindowsUtils::TReferenceCounted< B > Class Template Reference

```
#include <wrefcount.h>
```

Inheritance diagram for WindowsUtils::TReferenceCounted< B >:


WindowsUtils::TReferenceCounted< B >
WindowsUtils::RegistryKey::CountedValue< RegistryKey::CountedValue >
WindowsUtils::TReferenceCountedResource< B >

4.35.1 Detailed Description

template<typename B> class WindowsUtils::TReferenceCounted< B >

Class TReferenceCounted is a template mix-in class to give a derived class reference counted behavior. Using addRef() and release(), a handle class for a reference counted implementation object adjusts the object's reference count when copying or destructing the handle. If no more references to the implementation object exist, release() deletes this object.

Examples:
refcount.out.

Public Member Functions

- TReferenceCounted ()
  default constructor.
- virtual B ∗ addRef ()
  add a reference.
- virtual B ∗ release ()
  release a reference; delete this object if no more references exist.
- long numrefs ()
  return reference count.

Static Public Member Functions

- B ∗ copy (B ∗pRefCounted)
  copy: add reference to specified object, unless it is null.

Protected Member Functions

- virtual ∼TReferenceCounted ()
  we destruct in release()

Protected Attributes

- long iCount

  its count

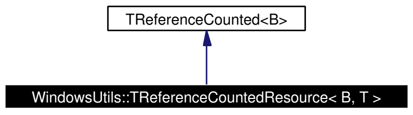
The documentation for this class was generated from the following file:

- wrefcount.h

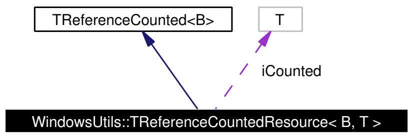
4.36 WindowsUtils::TReferenceCountedResource< B, T > Class Template Reference

`#include <wrefcount.h>`

Inheritance diagram for WindowsUtils::TReferenceCountedResource< B, T >:



Collaboration diagram for WindowsUtils::TReferenceCountedResource< B, T >:



4.36.1 Detailed Description

template<typename B, typename T> class WindowsUtils::TReferenceCountedResource< B, T >

Class TReferenceCountedResource is a template mix-in class to:

- give a derived class reference counted behavior: addRef() and release()
- take care of a resource: preRelease()

A handle class for a reference counted implementation object adjusts the object's reference count when copying or destructing the handle. If no more references to the implementation object exist, release() first calls preRelease() to free the resource and then deletes this object.

Template class TReferenceCountedResource provides the mechanism for reference counting that is used by class CCountedRegKey.

Public Member Functions

- TReferenceCountedResource (T counted)

  constructor.

- const T & getCounted () const

  return reference to resource.

- operator T & () const

  return reference to resource.

- operator T ∗ () const

return pointer to resource.

- B ∗ release ()

  remove a reference to the resource, call preRelease() before destruction when no more references exist.

Protected Member Functions

- virtual ∼TReferenceCountedResource ()

  destructor.

- virtual void preRelease ()=0

  release resource

Protected Attributes

- T iCounted

  its resource

Private Member Functions

- TReferenceCountedResource (const TReferenceCountedResource &rhs)

  prevent copying

- TReferenceCountedResource & operator= (const TReferenceCountedResource &rhs)

  prevent copying

The documentation for this class was generated from the following file:

- wrefcount.h

4.37 WindowsUtils::TRegKeyIterator< B > Class Template Reference

`#include <wregistry-rki.h>`

4.37.1 Detailed Description

template<typename B> class WindowsUtils::TRegKeyIterator< B >

Template class TRegKeyIterator provides the mechanism to create the SubkeyIterator and ValueIterator classes from the SubkeyIteratorImpl and ValueIteratorImpl with common base class CRegKeyIterator.

Public Member Functions

- TRegKeyIterator< B > & operator++ ()

  preincrement.

- TRegKeyIterator< B > operator++ (int)

  postincrement.

- bool operator!= (const TRegKeyIterator< B > &rhs) const

  test for iterator inequality.

- bool operator== (const TRegKeyIterator< B > &rhs) const

  test for iterator inequality.

Protected Member Functions

- TRegKeyIterator (CCountedRegKey ∗apKey)

  constructor.

- void advance ()

  acquire next element; release key if at end of iteration.

Friends

- class **RegistryKey**

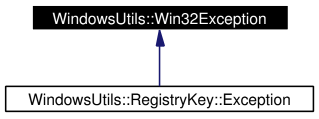
The documentation for this class was generated from the following file:

- wregistry-rki.h

4.38 WindowsUtils::Win32Exception Class Reference

```
#include <wexception.h>
```

Inheritance diagram for WindowsUtils::Win32Exception:


WindowsUtils::Win32Exception
WindowsUtils::RegistryKey::Exception

Public Member Functions

- Win32Exception (const char ∗apWhere, DWORD aError)

  constructor.

- Win32Exception (const std::string &aWhere, DWORD aError)

  constructor.

- const char ∗ what () const throw ()

  return the error message as "where: what"; the character string is owned by Win32Exception.

- const char ∗ where () const

  return the "where" character string.

- DWORD error () const

  return exception's error number.

- virtual const char ∗ message () const

  return the Windows message for the error number.

- void messageBox (HWND hWnd=NULL) const

  display messagebox with message describing the exception using what().

Private Attributes

- std::string iWhere

  the location of the error

- DWORD iError

  the Windows error number

The documentation for this class was generated from the following files:

- wexception.h
- wexception.cpp

Chapter 5

Windows Utilities Example Documentation

5.1 algorithm.out

The following program shows the use of the copy() and for_each() algorithms to copy and print a key subtree.

It also shows the use of a function object, or visitor [Gamma et al., 1995] to print the subkeys and values. This function object is not derived from std::unary_function, because it accepts two different types for its function call operator: Values and Subkeys.

Note that the function objects used with for_each() must define the pre-increment and pre-decrement operators (`operator++()`, `operator-()`). These operators are used to keep track of the subkey level.

```
/*
 * algorithm.cpp - copy, for_each.
 *
 * compile: prompt>bcc32 algorithm.cpp winutils.lib
 */

#include <algorithm>    // for std::copy, for_each
#include <iostream>     // for std::cout, cerr
#include "wregistry.h"  // for WindowsUtils::RegistryKey

using WindowsUtils::RegistryKey;

/*
 * function object to print an indented subkey or value.
 */

class Printer
{
public:
   explicit Printer( std::ostream& astrm = std::cout );

   std::string indent();        // indentation string

   void operator++ ();          // pre-increment
   void operator-- ();          // pre-decrement

   void operator ()( const RegistryKey::Subkey& s );
   void operator ()( const RegistryKey::Value&  v );
private:
   int level;                    // current subkey level
   std::ostream& strm;           // stream to write to
};

/*
 * the program
 */

int main()
{
   try
   {
      bool deep    = true;          // deep copy, visit
      bool shallow = false;         // shallow copy, visit

      RegistryKey fromkey(
         HKEY_LOCAL_MACHINE, "HARDWARE\\DESCRIPTION\\System\\CentralProcessor" );

      RegistryKey swkey(
         HKEY_CURRENT_USER, "Software" );

      RegistryKey tokey = swkey.createOrOpenKey( "MyAlgorithmTest" );

      std::cout << "\nFrom:\n"; fromkey.for_each( Printer(), deep );

      std::cout << "\nTo:\n";     tokey.for_each( Printer(), deep );

      std::cout << "\nCopy:\n"; fromkey.copy    (   tokey  , deep );

      std::cout << "\nTo:\n";     tokey.for_each( Printer(), deep );

      swkey.deleteKeyAndSubkeys( "MyAlgorithmTest" );   // clean-up
   }
   catch ( const std::exception& e )
   {
      std::cerr << "Error: " << e.what() << std::endl;
   }
   catch ( ... )
   {
      std::cerr << "Error: unknown exception" << std::endl;
   }
}

/*
 * class Printer implementation:
 */

Printer::Printer( std::ostream& astrm ) :
  level(), strm(astrm)
{
}

std::string Printer::indent()
{
   return std::string( level * 2, ' ' );
}

void Printer::operator++ ()
{
   ++level;
}

void Printer::operator-- ()
{
   --level;
}
```

```
void Printer::operator ()( const RegistryKey::Subkey& s )
{
   strm << indent() << s << std::endl;
}

void Printer::operator ()( const RegistryKey::Value& v )
{
   strm << indent() << v << std::endl;
}
```

The program may produce the following output.

```
From:
0 (Processor)
  Component Information = ’00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00’
  Identifier = ’x86 Family 6 Model 3 Stepping 4’
  Configuration Data = ’ff ff ff ff ff ff ff ff 00 00 00 00 00 00 00 00’
  VendorIdentifier = ’GenuineIntel’
  FeatureSet = ’0x1ff’
  ~MHz = ’0xe9’
  Update Signature = ’00 00 00 00 37 00 00 00’
  Update Status = ’0x0’
  Previous Update Signature = ’00 00 00 00 33 00 00 00’

To:

Copy:

To:
0 (Processor)
  Component Information = ’00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00’
  Identifier = ’x86 Family 6 Model 3 Stepping 4’
  Configuration Data = ’ff ff ff ff ff ff ff ff 00 00 00 00 00 00 00 00’
  VendorIdentifier = ’GenuineIntel’
  FeatureSet = ’0x1ff’
  ~MHz = ’0xe9’
  Update Signature = ’00 00 00 00 37 00 00 00’
  Update Status = ’0x0’
  Previous Update Signature = ’00 00 00 00 33 00 00 00’
```

5.2 algorithm2.out

The following program shows the use of the find_value(std::string) and find_value(UnaryPredicate) algorithms to find a value.

```
/*
 * algorithm2.cpp - find a value.
 *
 * compile: prompt>bcc32 algorithm2.cpp winutils.lib
 */

#include <algorithm>    // for std::find, copy, for_each
#include <functional>   // for std::unary_function
#include <iostream>     // for std::cout, cerr
#include "wregistry.h"  // for WindowsUtils::RegistryKey

using WindowsUtils::RegistryKey;

/*
 * match Subkey or Value name template:
 */

template <typename T>
class MatchName : public std::unary_function<T, bool>
{
public:
   MatchName( const std::string& aName ) :
      iName( aName ) { ; }

   bool operator()( const T& v )
   {
      return v.name() == iName;
   }

private:
   std::string iName;
};

typedef MatchName<RegistryKey::Value> MatchValueName;

/*
 * the program
 */

int main()
{
   try
   {
      bool deep    = true;          // deep visit
      bool shallow = false;         // shallow visit

      std::string valueName = "~MHz";

      RegistryKey key(
         HKEY_LOCAL_MACHINE, "HARDWARE\\DESCRIPTION\\System" );

      std::pair<RegistryKey,RegistryKey::ValueIterator> pair  =
         key.find_value( valueName, deep );

      std::pair<RegistryKey,RegistryKey::ValueIterator> pair2 =
         key.find_value( MatchValueName( valueName ), deep );

      if ( pair.second != key.endValueIteration() )
         std::cout << "find by string: " << valueName << ": " << *pair.second << std::endl;
      else
         std::cout << "find by string: " << valueName << ": not found" << std::endl;
      if ( pair2.second != key.endValueIteration() )
         std::cout << "find by predicate: " << valueName << ": " << *pair2.second << std::endl;
      else
         std::cout << "find by predicate: " << valueName << ": not found" << std::endl;
   }
   catch ( const std::exception& e )
   {
      std::cerr << "Error: " << e.what() << std::endl;
   }
   catch ( ... )
   {
      std::cerr << "Error: unknown exception" << std::endl;
   }
}
```

The program may produce the following output.

```
find by string: ~MHz: ~MHz = ’0xe9’
find by predicate: ~MHz: ~MHz = ’0xe9’
```

5.3 printpportaddr.out

The following program shows how class PportAddresses may be used to print the base addresses of the parallel ports available on a computer.

```
/*
 * printpportaddr.cpp - print the parallel port addresses found.
 *
 * compile: prompt>bcc32 printpportaddr.cpp winutils.lib
 */

#include <except>       // for std::exception
#include <iostream>     // for std::cout, cerr
#include <algorithm>    // for std::copy()
#include <iterator>     // for std::ostream_iterator<>
#include "wpportaddr.h" // for WindowsUtils::PportAddresses

int main()
{
   try
   {
      using WindowsUtils::PportAddresses;       // shorthand for PportAddresses

      typedef PportAddresses::value_type vt;    // shorthand for value_type

      PportAddresses thePportAddresses;         // determine the parallel port addresses

      std::cout.setf( std::ios::showbase );     // print hexadecimal with leading ’0x’
      std::cout.setf( std::ios::hex, std::ios::basefield );

      std::cout << "\nParallel port addresses found: ";
      std::copy( thePportAddresses.begin(), thePportAddresses.end(),
                                            std::ostream_iterator<vt>(std::cout, " ") );
      std::cout << std::endl;
   }
   catch ( const std::exception& e )            // handle exceptions
   {
      std::cerr << e.what() << std::endl;
   }
   catch ( ... )                                // handle exceptions
   {
         std::cerr << "Error: unknown exception" << std::endl;
   }
}
```

The program may give the following result.

```
Parallel port addresses found: 0x378 0xd068
```

5.4 printsubkeys.out

The following program show the use of a SubkeyIterator: It opens the key "HKEY_LOCAL_-MACHINE\\HARDWARE\\DESCRIPTION\\System" and recursively iterates over its subkeys.

```
/*
 * printsubkeys.cpp - print the subkeys of a registry key.
 *
 * compile: prompt>bcc32 printsubkeys.cpp winutils.lib
 */

#include <except>       // for std::exception
#include <string>       // for std::string
#include <iostream>     // for std::cout, cerr
#include "wregistry.h"  // for WindowsUtils::RegistryKey

using WindowsUtils::RegistryKey;

void printSubkeys( RegistryKey key );

#pragma argsused
int main( int argc, char *argv[] )
{
   try
   {
      std::string computername = argv[1] ? argv[1] : "";

      RegistryKey key( computername, HKEY_LOCAL_MACHINE );

      key = key.openKey( "HARDWARE\\DESCRIPTION\\System" );

      printSubkeys( key );
   }
   catch ( const std::exception& e )
   {
      std::cerr << "Error: " << e.what() << std::endl;
   }
   catch ( ... )
   {
      std::cerr << "Error: unknown exception" << std::endl;
   }
}

void printSubkeys( RegistryKey key )
{
   static int level = 0;

   for ( RegistryKey::SubkeyIterator pos = key.beginSubkeyIteration();
                                     pos != key.endSubkeyIteration(); ++pos )
   {
      std::cout << std::string( level*2, ' ' ) << *pos << std::endl;

      ++level;
      printSubkeys( pos.openKey() );
      --level;
   }
}
```

The program may produce the following output.

```
CentralProcessor (Processor)
  0 (Processor)
FloatingPointProcessor (Processor)
  0 (Processor)
```

```
MultifunctionAdapter (Adapter)
  0 (Adapter)
  1 (Adapter)
  2 (Adapter)
  3 (Adapter)
    DiskController (Controller)
      0 (Controller)
        DiskPeripheral (Peripheral)
          0 (Peripheral)
        FloppyDiskPeripheral (Peripheral)
          0 (Peripheral)
    KeyboardController (Controller)
      0 (Controller)
        KeyboardPeripheral (Peripheral)
          0 (Peripheral)
    ParallelController (Controller)
      0 (Controller)
    PointerController (Controller)
      0 (Controller)
        PointerPeripheral (Peripheral)
          0 (Peripheral)
    SerialController (Controller)
      0 (Controller)
      1 (Controller)
```

5.5 printvalues.out

The following program show the use of a ValueIterator: It opens the key "HARDWARE\\DESCRIPTION\\System" and iterates over its values, printing their name and contents.

```
/*
 * printvalues.cpp - print the value contents of a registry key.
 *
 * compile: prompt>bcc32 printvalues.cpp winutils.lib
 */

#include <except>       // for std::exception
#include <string>       // for std::string
#include <iostream>     // for std::cout, cerr
#include "wregistry.h"  // for WindowsUtils::RegistryKey

std::string typestring( DWORD tp );

#pragma argsused
int main( int argc, char *argv[] )
{
   try
   {
      using WindowsUtils::RegistryKey;

      std::string computername = argv[1] ? argv[1] : "";

      RegistryKey key( computername, HKEY_LOCAL_MACHINE );

      key = key.openKey( "HARDWARE\\DESCRIPTION\\System" );

      for ( RegistryKey::ValueIterator pos = key.beginValueIteration();
                                       pos != key.endValueIteration(); ++pos )
      {
         std::cout << typestring( pos.type() ) << " " << *pos << std::endl;
      }
   }
   catch ( const std::exception& e )
   {
      std::cerr << "Error: " << e.what() << std::endl;
   }
   catch ( ... )
   {
      std::cerr << "Error: unknown exception" << std::endl;
   }
}

#define   e(x) { x, #x }
#define dim(a) ( sizeof(a) / sizeof((a)[0]) )

std::string typestring( DWORD tp )
{
   struct Table
   {
      DWORD tp; const char *ts;
   }
   table[] =
   {
      e( REG_NONE  ),
      e( REG_DWORD ),
      e( REG_DWORD_BIG_ENDIAN ),
      e( REG_DWORD_LITTLE_ENDIAN ),
      e( REG_BINARY ),
      e( REG_RESOURCE_LIST ),
      e( REG_FULL_RESOURCE_DESCRIPTOR ),
      e( REG_LINK ),
      e( REG_SZ ),
      e( REG_MULTI_SZ ),
      e( REG_EXPAND_SZ ),
   };

   for ( Table *p = table; p < table + dim(table); ++p )
      if ( tp == p->tp )
         return p->ts;

   return "[unknown]";
}
```

The program may produce the following output.

Note that there are embedded `'\0's` in the REG_MULTI_SZ values that you must handle yourself.

```
REG_BINARY Component Information = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
REG_SZ Identifier = AT/AT COMPATIBLE
REG_FULL_RESOURCE_DESCRIPTOR Configuration Data = ff ff ff ff ff ff ff ff 00 00 00 ...
REG_SZ SystemBiosDate = 02/04/98
REG_MULTI_SZ SystemBiosVersion = Award Modular BIOS v4.51PG
REG_SZ VideoBiosDate = 03/24/98
REG_MULTI_SZ VideoBiosVersion = Stealth 3D 4000 Vers. 1.04 (c) Diamond Multimedia Systems, Inc. Ver. 1.00
```

5.6 refcount.out

The following program shows how class TReferenceCounted<> may be used to create a reference-counted implementation (body) in a handle-body idiom.

```
/*
 * refcount.cpp - handle with reference counted body.
 *
 * compile: prompt>bcc32 refcount.cpp winutils.lib
 */

#include <iostream>     // for std::cout, cerr
#include "wrefcount.h"  // for TReferenceCounted<>

using WindowsUtils::TReferenceCounted;

/*
 * the handle:
 */

class Handle
{
public:
   class Body;

   ~Handle();
   Handle( int aValue = int() );
   Handle( const Handle& rhs );
   Handle& operator= ( const Handle& rhs );

   int     numrefs() const;
   operator    int() const;

private:
   Body *rep;           // its representation
};

/*
 * the reference-counted implementation:
 */

class Handle::Body : public TReferenceCounted<Handle::Body>
{
private:
   int itsValue;

protected:
   ~Body();

public:
   Body( int aValue = 0 );

   operator int() const;
};

/*
 * print: name: value, N references
 */

void print( const char *name, const Handle& h )
{
   std::cout << name << ": " << h << ", " << h.numrefs() << " reference(s)" << std::endl;
}

/*
 * the program:
```

```
 */

int main()
{
   Handle a(456);       // constructor
   Handle b = a;        // copy constructor

   {                    // scope c
      Handle c;

      print( "a", a );
      print( "b", b );
      print( "c", c );

      c = b;            // copy assignment

      print( "\na", a );
      print(   "b", b );
      print(   "c", c );
   }

   b = 987;             // copy assignment

   print( "\na", a );
   print(   "b", b );
//   print( "c", c );   // c is out of scope
}

/*
 * the handle implementation:
 */

Handle::~Handle()
{
   rep->release();
}

Handle::Handle( int aValue /* = 0 */ ) :
   rep( new Body( aValue ) )
{
}

Handle::Handle( const Handle& rhs ) :
   rep( rhs.rep->addRef() )
{
}

Handle& Handle::operator= ( const Handle& rhs )
{
   if ( this != &rhs )
   {
      Body *newRep = rhs.rep->addRef();
      rep->release();
      rep = newRep;
   }

   return *this;
}

int Handle::numrefs() const
{
   return rep->numrefs();
}

Handle::operator int() const
{
   return rep->operator int();
}

/*
 * the body implementation:
 */

Handle::Body::~Body()
{
}

Handle::Body::Body( int aValue /* = 0 */ ) :
   itsValue( aValue )
{
}

Handle::Body::operator int() const
{
   return itsValue;
}
```

The program gives the following result.

```
a: 456, 2 reference(s)
b: 456, 2 reference(s)
c: 0, 1 reference(s)

a: 456, 3 reference(s)
b: 456, 3 reference(s)
c: 456, 3 reference(s)

a: 456, 1 reference(s)
b: 987, 1 reference(s)
```

Chapter 6

Windows Utilities Page Documentation

6.1 Some Ideas for Further Development

Application Log

The application logging functions error(), debug() and message() do not check for Win32 API errors. Should these functions throw a Win32Exception on failure to open/write/close the event log?

RegistryKey

I implemented several algorithms that work on all an entire key: Is it worthwhile to supply algorithms that take a range as input?

Problems are:

1. there are two different kind of items to iterate over: Values and Subkeys
2. subkeys may be descended into, or not

You can of course, create an iterator that traverses the subkey tree (using a stack to track the parents), but for an output or insert iterator it is not clear when to return to a parent.

6.2 References

URL catalog

[BBDSOFT] Software design, development, installation, test and support services for industrial automation.

http://www.bbdsoft.com/

[CODEPROJECT] The Code Project. Your place for free C++, C# and .NET articles, code snippets, discussions, news and the best bunch of developers on the net.

http://www.codeproject.com/

[CREGISTRYKEY] Len Holgate's Registry API wrapper classes.

http://www.codeproject.com/system/cregistrykey.asp and http://www.jetbyte.com/portfolio-showarticle.asp?articleId=21&cat-Id=1&subcatId=2

[FONTFILE] Hans Dietrich's Finding a Font file from a Font name.

http://www.codeproject.com/gdi/fontfile.asp

[INTERRUPTHOOK] Alexander M.'s Interrupt Hooking and retrieving device information on Windows NT/2000/XP.

http://www.codeproject.com/useritems/interrupthook.asp

[IPTABLES_FS] Arno's IPTABLES Firewall Script.

http://freshmeat.net/projects/iptables-firewall/

[SHUTDOWN] Functions to loggoff current user, shutdown computer.

http://www.bbdsoft.com/downloads/msvc/reboot101.zip

[WIN32SEMAPHORE] Class for thread synchronization using Semaphores.

http://www.bbdsoft.com/downloads/win32/semap.zip

Book references

[Gamma et al., 1995] Erick Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns; Elements of Reusable Object-Oriented Software.* Reading, Massachusetts: Addison-Wesley. ISBN 0-201-63361-2.

[Josuttis, 1999] Nicolai M. Josuttis. 1999. *The C++ Standard Library: A Tutorial and Reference.* Reading, Massachusetts: Addison-Wesley. ISBN 0-201-37926-0.

[Kernighan & Pike, 1984] Brian W. Kernighan and Rob Pike. 1984. *The Unix Programming Environment.*. Prentice Hall, Inc.. ISBN 0-13-937681-X (paperback), 0-13-937699-2 (hardback).

[Kernighan & Pike, 1999] Brian W. Kernighan and Rob Pike. 1999. *The Practice of Programming.* Reading, Massachusetts: Addison-Wesley. ISBN 0-201-61586-X.

[Stroustrup, 2000] Bjarne Stroustrup. 2000. *The C++ Programming Language: Special Edition, 3/E.* Boston: Addison Wesley Professional. ISBN 0-201-70073-5.

6.3 License of CRegistryKey

6.4 License of Semaphore

Index