PERL Win32 Quick Reference

Author: Jialong He
Jialong_he@bigfoot.com
http://www.bigfoot.com/~jialong_he

Win32::Internet

```
use Win32::Internet;
$INET = new Win32::Internet();
$file = $INET->FetchURL("http://www.yahoo.com");
binmode STDOUT;
print $file;
```

Open HTTP session

```
use Win32::Internet;
$INET= new Win32::Internet();
$INET->HTTP($HTTP, "hostname", " username", "password");
#-------------------------------------------------------
#  @ in email address must be escaped e.g.,
#  jialong_he\@bigfoot.com
#-------------------------------------------------------
($statuscode, $headers, $file)=$HTTP->Request("/");
binmode STDOUT;
print $file;
```

Open FTP session

```
use Win32::Internet;
$INET= new Win32::Internet();
$INET->FTP($FTP, "hostname", "username", "password");
$FTP->Cd ("path");
$FTP->Binary ();
$FTP->Get("myfile.zip");   # file saved to myfile.zip
$FTP->Close();
```

FTP Function Reference

$FTP->Ascii();** **$FTP->Binary();	Sets the ASCII (Binary) transfer mode for this FTP session.
$FTP->Cd("/pub");	Changes the current directory on the FTP remote host.
$FTP->Delete("yourfile.zip");	Deletes a file on the FTP remote host.
$FTP->Get("myfile.zip");	Gets the *remote* FTP file and saves it.
$FTP->Put("newfile.zip");	Upload a file to the server.
@files = $FTP->List("*.txt");	Returns a list containing the files in current directory.
$FTP->Mkdir("NextBuild");	Creates a directory on the FTP remote host.
$path = $FTP->Pwd();	Returns the current directory on the FTP server.
$FTP->Rmdir("olddir");	Removes a directory on the FTP remote host.
$FTP->Rename("old.zip", "new.zip");	Renames a file on the FTP remote host.

Win32 API Functions

```
#-------------------------------------
# Some info about window
#-------------------------------------
use Win32;
print "*** Login Name: ", Win32::LoginName(), "\n";
print "*** Domain Name: ", Win32::DomainName(), "\n";
print "*** File System: ", Win32::FsType(), "\n";
print "*** OS version: ", Win32::GetOSVersion(), "\n";
print "*** IsWindowNT: ", Win32::IsWinNT(), "\n";
print "*** TickCount: ", Win32::GetTickCount(), "\n";
```

Win32::SetCwd(NEWDIRECTORY)
Win32::GetCwd()
Sets/Gets current active drive and directory.

Win32::SetLastError(ERROR)
Win32::GetLastError()
Sets/Gets the value of the last error encountered to ERROR.

Win32::InitiateSystemShutdown(MACHINE, MESSAGE, TIMEOUT, FORCECLOSE, REBOOT)
Shutsdown the specified MACHINE, notifying users with the supplied MESSAGE, within the specified TIMEOUT interval. Forces closing of all documents without prompting the user if FORCECLOSE is true, and reboots the machine if REBOOT is true. This function works only on WinNT.

Win32::AbortSystemShutdown(MACHINE)
Aborts a system shutdown (started by the InitiateSystemShutdown function) on the specified MACHINE.

Win32::IsWinNT()
Win32::IsWin95()
Returns non zero if the Win32 subsystem is Windows NT (Windows 95).

Win32::GetTickCount()
Returns the number of milliseconds elapsed since the last system boot.

Win32::GetOSVersion()
Returns the array (STRING, MAJOR, MINOR, BUILD, ID), where the elements are, respectively: An arbitrary descriptive string, the major version number of the operating system, the minor version number, the build number, and a digit indicating the actual operating system. For ID, the values are 0 for Win32s, 1 for Windows 9X and 2 for Windows NT. In scalar context it returns just the ID.

Win32::MsgBox(MESSAGE [, FLAGS [, TITLE]])
Create a dialogbox containing MESSAGE. FLAGS specifies the required icon and buttons according to the following table:

```
0 = OK
1 = OK and Cancel
2 = Abort, Retry, and Ignore
3 = Yes, No and Cancel
4 = Yes and No
5 = Retry and Cancel
MB_ICONSTOP          "X" in a red circle
MB_ICONQUESTION      question mark in a bubble
MB_ICONEXCLAMATION   exclamation mark in a yellow triangle
MB_ICONINFORMATION   "i" in a bubble
```

TITLE specifies an optional window title. The default is ``Perl".

The function returns the menu id of the selected push button:

```
0  Error
1  OK
2  Cancel
3  Abort
4  Retry
5  Ignore
6  Yes
7  No
```

Win32::DomainName()
Returns the name of the Microsoft Network domain name.

Win32::FsType()

Returns the name of the filesystem of the currently active drive (like 'FAT' o 'NTFS'). In list context it returns three values: (FSTYPE, FLAGS, MAXCOMPLEN). FSTYPE is the filesystem type as before. FLAGS is a combination of values of the following table:

```
0x00000001  supports case-sensitive filenames
0x00000002  preserves the case of filenames
0x00000004  supports Unicode in filenames
0x00000008  preserves and enforces ACLs
0x00000010  supports file-based compression
0x00000020  supports disk quotas
0x00000040  supports sparse files
0x00000080  support s reparse points
0x00000100  supports remote storage
0x00008000  is a compressed volume (e.g. DoubleSpace)
0x00010000  supports object identifiers
0x00020000  supports the Encrypted File System (EFS)
```

MAXCOMPLEN is the maximum length of a filename component (the part between two backslashes) on this file system.

Win32::GetChipName()
Returns the processor type: 386, 486 or 586 for Intel processors, 21064 for t Alpha chip.

Win32::Sleep(TIME)
Pauses for TIME milliseconds. The timeslices are made available to other processes and threads.

Win32::Spawn(COMMAND, ARGS, PID)
Spawns a new process using the supplied COMMAND, passing in arguments in the string ARGS. The pid of the new process is stored in PID. This function is deprecated. Please use the Win32::Process module instead.

Win32::GetProcAddress(INSTANCE, PROCNAME)
Returns the address of a function inside a loaded library. The information about what you can do with this address has been lost in the mist of time. Use the Win32::API module instead of this deprecated function.

Win32::GetNextAvailDrive()
Returns a string in the form of ``<d>:" where <d> is the first available drive letter.

Win32::EventLog

```
use Win32::EventLog;
 $handle=Win32::EventLog->new("System", $ENV{ComputerName})
       or die "Can't open Application EventLog\n";
 $handle->GetNumber($recs)
       or die "Can't get number of EventLog records\n";
 $handle->GetOldest($base)
       or die "Can't get number of oldest EventLog record\n";
 while ($x < $recs) {
       $handle-
>Read(EVENTLOG_FORWARDS_READ|EVENTLOG_SEEK_READ,
                          $base+$x,
                          $hashRef)
             or die "Can't read EventLog entry #$x\n";
       if ($hashRef->{Source} eq "EventLog") {
             Win32::EventLog::GetMessageText($hashRef);
             print "Entry $x: $hashRef->{Message}\n";
       }
       $x++;
 }
```

$handle = Win32::EventLog->new(SOURCENAME [,SERVERNAME]);
Th **new()** method creates a new EventLog object and returns a handle to it. This hande is then used to call the methods below.

$handle->Backup(FILENAME);
The Backup() method backs up the EventLog represented by $handle.

$handle->Clear(FILENAME);
Clears the EventLog represented by $handle. If a FILENAME is provided, the EventLog will be backed.

$handle->Report(HASHREF);
generates an EventLog entry.

$handle->Read(FLAGS, OFFSET, HASHREF);
read an EventLog entry from the EventLog.

Win32::NetAdmin

```
 # FILTER_TEMP_DUPLICATE_ACCOUNTS
 #   Enumerates local user account data on a domain controller.
 #
 # FILTER_NORMAL_ACCOUNT
 #   Enumerates global user account data on a computer.
 #
 # FILTER_INTERDOMAIN_TRUST_ACCOUNT
 #   Enumerates domain trust account data on a domain controller.
 #
 # FILTER_WORKSTATION_TRUST_ACCOUNT
 #   Enumerates workstation or member server account data on a domain
 #   controller.
 #
 # FILTER_SERVER_TRUST_ACCOUNT
 #   Enumerates domain controller account data on a domain controller.
 use Win32::NetAdmin qw(GetUsers GroupIsMember
                        UserGetAttributes UserSetAttributes);
 my %hash;
 GetUsers("", FILTER_NORMAL_ACCOUNT , \%hash)
    or die "GetUsers() failed: $^E";
 foreach (keys %hash) {
    my ($password, $passwordAge, $privilege,
       $homeDir, $comment, $flags, $scriptPath);
    if (GroupIsMember("", "Domain Users", $_)) {
       print "Updating $_ ($hash{$_})\n";
       UserGetAttributes("", $_, $password, $passwordAge, $privilege,
                         $homeDir, $comment, $flags, $scriptPath)
          or die "UserGetAttributes() failed: $^E";
       $scriptPath = "dnx_login.bat"; # this is the new login script
       UserSetAttributes("", $_, $password, $passwordAge, $privilege,
                         $homeDir, $comment, $flags, $scriptPath)
          or die "UserSetAttributes() failed: $^E";
    }
 }
```

GetDomainController(server, domain, returnedName)
Returns the name of the domain controller for server.

GetAnyDomainController(server, domain, returnedName)
Returns the name of any domain controller for a domain that is directly trusted by the server.

UserCreate(server, userName, password, passwordAge, privilege, homeDir, comment, flags, scriptPath)
Creates a user on server with password, passwordAge, privilege, homeDir, comment, flags, and scriptPath.

UserDelete(server, user)
Deletes a user from server.

UserGetAttributes(server, userName, password, passwordAge, privilege, homeDir, comment, flags, scriptPath)
Gets password, passwordAge, privilege, homeDir, comment, flags, and scriptPath for user.

UserSetAttributes(server, userName, password, passwordAge, privilege homeDir, comment, flags, scriptPath)
Sets password, passwordAge, privilege, homeDir, comment, flags, and scriptPath for user.

UserChangePassword(domainname, username, oldpassword, newpassword)
Changes a users password. Can be run under any account.

UsersExist(server, userName
Checks if a user exists.

GetUsers(server, filter, userRef)
Fills userRef with user names if it is an array reference and with the user names and the full names if it is a hash reference.

GroupCreate(server, group, comment)
GroupDelete(server, group)
Creates (Deletes) a group.

GroupGetAttributes(server, groupName, comment)
GroupSetAttributes(server, groupName, comment)
Gets (Sets) the comment.

GroupAddUsers(server, groupName, users)
GroupDeleteUsers(server, groupName, users)
Adds (Deletes) a user in a group.

GroupIsMember(server, groupName, user)
Returns TRUE if user is a member of groupName.

GroupGetMembers(server, groupName, userArrayRef)
Fills userArrayRef with the members of groupName.

LocalGroupCreate(server, group, comment)
LocalGroupDelete(server, group)
Creates (Deletes) a local group.

LocalGroupGetAttributes(server, groupName, comment)
LocalGroupSetAttributes(server, groupName, comment)
Gets (Sets) the comment.

LocalGroupAddUsers(server, groupName, users)
LocalGroupDeleteUsers(server, groupName, users)
Adds (Deletes) a user to a group.

GetServers(server, domain, flags, serverRef)
Gets an array of server names or an hash with the server names and the comments as seen in the Network Neighborhood or the server manager. For flags, see SV_TYPE_* constants.

GetTransports(server, transportRef)
Enumerates the network transports of a computer. If transportRef is an array reference, it is filled with the transport names. If transportRef is a hash reference then a hash of hashes is filled with the data for the transp orts.

LoggedOnUsers(server, userRef)
Gets an array or hash with the users logged on at the specified computer. If userRef is a hash reference, the value is a semikolon separated string of username, logon domain and logon server.

GetServerDisks(server, arrayRef)
Returns an array with the disk drives of the specified server. The array contains two-character strings (drive letter followed by a colon).

LocalGroupGetMembers(server, groupName, userArrayRef)
Fills userArrayRef with the members of groupName.

Win32::Service

```
use Win32::Service;
Win32::Service::GetServices("", \%ServiceList);
while (($key,$value) = each %ServiceList) {
  print "$key<----->$value\n";
}
```

StartService(hostName, serviceName)
StopService(hostName, serviceName)
PauseService(hostName, serviceName)
Start, Stop or Pause the service serviceName on the machine hostName.

GetStatus(hostName, serviceName, status)
Get the status of a service. The third argument must be a hash reference that will be populated with entries corresponding to the SERVICE_STATUS structure of the Win32 API. See the Win32 Platform SDK documentation for details of this structure.

GetServices(hostName, hashref)
Enumerates both active and inactive Win32 services at the specified host. The hashref is populated with the descriptive service names as keys and the short names as the values.

Win32::Sound

```
use Win32::Sound;
Win32::Sound::Volume('100%');
Win32::Sound::Play("c:/winnt/media/notify.wav");
Win32::Sound::Stop();
```

Win32::Sound::Play(SOUND, [FLAGS])
Plays the specified sound: SOUND can the be name of a WAV file or one of the following predefined sound names:

SystemDefault
SystemAsterisk
SystemExclamation
SystemExit
SystemHand
SystemQuestion
SystemStart

Additionally, if the named sound could not be found, the function plays the system default sound (unless you specify the SND_NODEFAULT flag). If no parameters are given, this function stops the sound actually playing (see also Win32::Sound::Stop).

FLAGS can be a combination of the following constants:

SND_ASYNC
The sound is played asynchronously and the function returns immediately after beginning the sound (if this flag is not specified, the sound is played synchronously and the function returns when the sound ends).

SND_LOOP
The sound plays repeatedly until it is stopped. You must also specify SND_ASYNC flag.

SND_NODEFAULT
No default sound is used. If the specified sound cannot be found, the function returns without playing anything.

SND_NOSTOP
If a sound is already playing, the function fails. By default, any new call to the function will stop previously playing sounds.

Win32::Sound::Stop()
Stops the sound currently playing.

($L, $R) = Win32::Sound::Volume();
Win32::Sound::Volume(LEFT, [RIGHT])
Get/Sets the wave device volume.

($hz, $bits, $channels) = Win32::Sound::Format(filename)
Returns information about the specified WAV file format; the array contains.

@devices = Win32::Sound::Devices();
Returns all the available sound devices.

%Info = Win32::Sound::DeviceInfo(DEVICE)
Returns an associative array of information about the sound device named DEVICE.

Win32::Process

```
use Win32;
use Win32::Process;
Win32::Process::Create($ProcessObj,
      "c:\\winnt\\system32\\notepad.exe",
      "notepad temp.txt",
      0,
      NORMAL_PRIORITY_CLASS,
      ".");

$ProcessObj->Suspend();
$ProcessObj->Resume();
$ProcessObj->Wait(INFINITE);
```

Win32::Process::Create($obj,$appname,$cmdline,$iflags,$cflags,$curd
Creates a new process.
Args:

$obj	container for process object
$appname	full path name of executable module
$cmdline	command line args
$iflags	flag: inherit calling processes handles or not
$cflags	flags for creation (see export ed vars below)
$curdir	working dir of new process

Win32::Process::KillProcess($pid, $exitcode)
Terminates any process identified by $pid. $exitcode will be set to the exit code of the process.

$ProcessObj->Suspend()
Suspend the process associated with the $ProcessObj.

$ProcessObj->Resume()
Resume a suspended process.

$ProcessObj->Kill($exitcode)
Kill the associated process, have it terminate with exit code $ExitCode.

$ProcessObj->GetPriorityClass($class)
Get the priority class of the process.

$ProcessObj->SetPriorityClass($class)
Set the priority class of the process (see exported values below for options).

$ProcessObj->GetProcessAffinitymask($processAffinityMask, $systemAffinitymask)
Get the process affinity mask. This is a bitvector in which each bit represent the processors that a process is allowed to run on.

$ProcessObj->SetProcessAffinitymask($processAffinityMask)
Set the process affinity mask. Only available on Windows NT.

$ProcessObj->GetExitCode($exitcode)
Retrieve the exitcode of the process.

$ProcessObj->Wait($timeout)
Wait for the process to die. $timeout should be specified in milliseconds. To wait forever, specify the constant INFINITE.

$ProcessObj->GetProcessID()
Returns the Process ID.

Win32::TieRegistry

```
use Win32::TieRegistry( Delimiter=>"#", ArrayValues=>0 );
$pound= $Registry->Delimiter("/");
$diskKey= $Registry->{"LMachine/System/Disk/"}
  or die "Can't read LMachine/System/Disk key: $^E\n";
$data= $key->{"/Information"}
  or die "Can't read LMachine/System/Disk//Information value: $^E\n";
$remoteKey= $Registry->{"//ServerA/LMachine/System/"}
  or die "Can't read //ServerA/LMachine/System/ key: $^E\n";
$remoteData= $remoteKey->{"Disk//Information"}
  or die "Can't read ServerA's System/Disk//Information value: $^E\n";
foreach $entry (  keys(%$diskKey)  ) {
    ...
}
foreach $subKey (  $diskKey->SubKeyNames  ) {
    ...
}
$diskKey->AllowSave( 1 );
$diskKey->RegSaveKey( "C:/TEMP/DiskReg", [] );
```

Opening keys

```
use Win32::TieRegistry ( Delimiter=>"/", ArrayValues=>1 );
$Registry->Delimiter("/");              # Set delimiter to "/".
$swKey= $Registry->{"LMachine/Software/"};
$winKey= $swKey->{"Microsoft/Windows/CurrentVersion/"};
$userKey= $Registry->
  {"CUser/Software/Microsoft/Windows/CurrentVersion/"};
$remoteKey= $Registry->{"//HostName/LMachine/"};
```

Reading values

```
$progDir= $winKey->{"/ProgramFilesDir"};    # "C:\\Program Files"
$tip21= $winKey->{"Explorer/Tips//21"};     # Text of tip #21.
$winKey->ArrayValues(1);
( $devPath, $type )= $winKey->{"/DevicePath"};
# $devPath eq "%SystemRoot%\\inf"
# $type eq "REG_EXPAND_SZ"  [if you have SetDualVar.pm installed]
# $type == REG_EXPAND_SZ()  [if did C<use Win32::TieRegistry
qw(:REG_)>]
```

Setting values

```
$winKey->{"Setup//SourcePath"}= "\\\\SwServer\\SwShare\\Windows";
# Simple.  Assumes data type of REG_SZ.
$winKey->{"Setup//Installation Sources"}=
  [ "D:\x00\\\\SwServer\\SwShare\\Windows\0\0", "REG_MULTI_SZ" ];
# "\x00" and "\0" used to mark ends of each string and end of list.
$winKey->{"Setup//Installation Sources"}=
  [ ["D:","\\\\SwServer\\SwShare\\Windows"], "REG_MULTI_SZ" ];
# Alternate method that is easier to read.
$userKey->{"Explorer/Tips//DisplayInitialTipWindow"}=
  [ pack("L",0), "REG_DWORD" ];
$userKey->{"Explorer/Tips//Next"}= [ pack("S",3), "REG_BINARY" ];
$userKey->{"Explorer/Tips//Show"}= [ pack("L",0), "REG_BINARY" ];
```

Adding keys

```
$swKey->{"FooCorp/"}= {
    "FooWriter/" => {
      "/Version" => "4.032",
      "Startup/" => {
        "/Title" => "Foo Writer Deluxe ][",
        "/WindowSize" => [ pack("LL",$wid,$ht), "REG_BINARY" ],
        "/TaskBarIcon" => [ "0x0001", "REG_DWORD" ],
      },
      "Compatibility/" => {
        "/AutoConvert" => "Always",
        "/Default Palette" => "Windows Colors",
      },
    },
    "/License", => "0123-9C8EF1-09-FC",
```

Listing all subkeys and values

```
@members= keys( %{$swKey} );
@subKeys= grep(  m#^/#,  keys( %{$swKey->{"Classes/batfile/"}} )  );
# @subKeys= ( "/", "/EditFlags" );
@valueNames= grep(  ! m#^/#,  keys( %{$swKey->{"Classes/batfile/"}} )
);
# @valueNames= ( "DefaultIcon/", "shell/", "shellex/" );
```

Deleting values or keys with no subkeys

```
$oldValue= delete $userKey->{"Explorer/Tips//Next"};
$oldValues= delete $userKey->{"Explorer/Tips/"};
# $oldValues will be reference to hash containing deleted keys values.
```

Closing keys

```
undef $swKey;             # Explicit way to close a key.
$winKey= "Anything else";  # Implicitly closes a key.
exit 0;                   # Implicitly closes all keys.
```