[Perl / WSH] Windows-Registry bearbeiten

Inhalt:

1. Einleitung

Die Möglichkeiten, die Registry-Datenbank von Windows zu bearbeiten, sind vielfältig. Am bekanntesten dürfte das Programm **“regedit”** sein. Damit lassen sich unter einer bequemen grafischen Oberfläche Schlüssel lesen und deren Werte verändern. Manchmal kann es auch ganz praktisch sein, die Bearbeitung der Registry-Datenbank zu automatisieren. Am schnellsten geht das mit Skripts: so reichen wenige Zeilen Code, um zum Ziel zu kommen. Wir wollen uns im Folgenden Funktionen und Methoden zum Zugriff auf die Windows-Registry in zwei verschiedenen, populären Skriptsprachen ansehen: zum einen in **Perl**, zum anderen mit **Windows-Scripts** (VBScript bzw. JScript).

Hinweis:
Microsoft selbst weist darauf hin, dass immer ein *Backup der Schlüssel* angelegt werden soll und man ohnehin nur auf die Registry zugreifen soll, wenn man genau weiß, was man tut. Das sollte man durchaus ernst nehmen!

2. Registry bearbeiten mit Perl

In Perl stehen grundsätzlich drei Packages zur Registry-Bearbeitung zur Verfügung: **Win32::Registry**, **Win32::TieRegistry** und **Win32Api::Registry**. Während Win32::Registry nicht mehr aktiv gepflegt wird und Win32Api::Registry einen recht generischen Zugriff bietet, bietet Win32::TieRegistry den wohl einfachsten und bequemsten Weg. Wir lenken unser Augenmerk deshalb allein auf dieses Package. Es sollte außerdem in allen neueren Versionen von ActivePerl schon enthalten sein.
Das Prinzip ist denkbar einfach: wird das Package eingebunden, steht im globalen Namensraum automatisch eine Variable **$Registry** zur Verfügung. Diese stellt eine Referenz auf ein Hash dar, welches den gesamten Schlüsselbaum der Windows-Registry abbildet. Die Registry wird in gewisser Weise an den Hash “gebunden” (“Tie”), d.h.

Änderungen am Hash werden als Änderung an der Registry wirksam. Wie das aussieht, wird sicher in den folgenden Beispielen klar.

2.1. Wert lesen

Wir wollen als Erstes einen einfachen Wert aus der Registry auslesen, und zwar die derzeit gesetzte Zeitzone samt Sommerzeit/Winterzeit-Einstellung. Der Perl-Code (Datei **regread.pl**):

```
use Win32::TieRegistry;
$Registry->Delimiter("/");
$key =$Registry->{"LMachine/System/CurrentControlSet/Control/TimeZoneInformation/"};
print $key->{"/DaylightName"};
undef %key;
```

Zuerst binden wir mit dem Statement **use** ganz normal das Package **Win32::TieRegistry** ein. Damit steht uns die Hash-Referenz **$Registry** zur Verfügung. Als Erstes setzen wir das Trennzeichen (sog. Delimiter) zwischen den Bestandteilen des Pfades auf / (Standard ist \\).
Den gewünschten Schlüssel erhalten wir durch **Dereferenzierung** von $Registry ("LMachine" steht für HKEY_LOCAL_MACHINE, "CUser" für HKEY_CURRENT_USER usw.). Die Rückgabe (hier gespeichert in der Variable $key) ist wiederum eine Referenz auf einen Hash, der die entsprechende Untermenge speichert. Der eigentliche Wert wird ebenso mittels Dereferenzierung abgefragt und kann in einer anderen Variable gespeichert oder sofort auf den Bildschirm ausgegeben werden (wie in unserem Beispiel). Zuletzt geben wir mit **undef** den Hash %key wieder frei.

Beachte:
Zwischen dem Schlüssel und einem konkreten Wert muss der Delimiter doppelt stehen. Sehen wir genau hin: einen abschließenden Slash finden wir hinter "TimeZoneInformation" (enthält mehrere Informationen zu gesetzten Zeitzonen), einen weiteren vor "DaylightName", also der konkrete Wert. Macht insgesamt 2 statt 1!

Wenn wir ohnehin nur einen Wert aus dem Schlüssel "TimeZoneInformation" abfragen möchten, lohnt es sich möglicherweise nicht, diesen Schlüssel in einer eigenen Hash-Referenz zwischen zu speichern. Stattdessen greifen wir auf den Wert von "DaylightName" direkt über $Registry zu. Der Code wird dann noch kürzer:

```
use Win32::TieRegistry;
$Registry->Delimiter("/");
print $Registry-
>{"LMachine/System/CurrentControlSet/Control/TimeZoneInformation//DaylightName"};
```

2.2. Wert ändern

Als nächstes wollen wir einen Wert in der Registry ändern. Mich hat beispielsweise sehr gestört, dass Notepad standardmäßig keinen automatischen Zeilenumbruch bietet. Man kann fast endlos in einer Zeile weiter schreiben und ist dann gezwungen, horizontal zu scrollen. Dieses Verhalten kann aber geändert werden. Es ist definiert im Schlüssel "HKEY_CURRENT_USER\\Software\\Microsoft\\Notepad" im Wert "fWrap". Diesen Wert

setzen wir auf 1.
Unsere Datei **regalter.pl** sieht so aus:

```
use Win32::TieRegistry;

$Registry->Delimiter("/");
$key = $Registry->{"CUser/Software/Microsoft/Notepad/"};
$key->{"/fWrap"} = [ pack("L",1), "REG_DWORD" ];
undef $key;
print "Wrap bei Notepad aktiviert!";
```

Das Ändern von Werten ist mit Perl denkbar einfach: es genügt, dem entsprechenden, dereferenzierten Hash-Element einen benutzerdefinierten Wert zuzuweisen. Dies muss ein zwei-elementiges Array sein (die eckigen Klammern sind ein Array-Konstruktor und erzeugen eine Referenz auf das enthaltene Array): einerseits der neue Wert des Schlüssels (den wir hier mit der Funktion **pack** in das notwendige Format umwandeln, beim Lesen von Zahlen- oder Binärwerten wäre entsprechend **unpack** notwendig), andererseits den Datentyp. Zur Verfügung stehen **REG_SZ** (eine Zeichenkette), **REG_EXPAND_SZ** (eine Zeichenkette bzw. Pfadangabe mit Umgebungsvariable, die deshalb geparsed werden muss), **REG_DWORD** (eine 4-Byte lange Integer-Zahl), **REG_BINARY** (Binärdaten).

2.3. Neuen Schlüssel anlegen

Das Anlegen eines neuen Schlüssels erfolgt analog zu obigem Beispiel. Wir wollen jetzt unter “HKEY_CURRENT_USER” einen neuen Schlüssel “MadTeaParty” anlegen: er soll zwei Werte enthalten, “VendorName” (Zeichenkette) und “URI” (Pfadangabe mit evtl. Umgebungsvariable), außerdem einen weiteren Schlüssel namens “RegWriter”, der die Werte “VersionNumber” (Zeichenkette) und “DefaultValue” (Integer-Zahl) umfasst. Werfen wir nun einen Blick auf die Datei **regwrite.pl**:

```
use Win32::TieRegistry;

$Registry->Delimiter("/");
$key = $Registry->{"CUser/"};
$key->{"MadTeaParty/"} = {
  "/VendorName" => "Chrstph",
  "/URI" => [ "http://www.mad-teaparty.com", "REG_EXPAND_SZ" ],
  "RegWriter/" => {
    "/VersionNumber" => [ "1.0", "REG_SZ" ],
    "/DefaultValue" => [ pack("L",0), "REG_DWORD" ]
  }
};
print "Registrierungs-Datenbank aktualisiert!";
```

Dem Schlüssel “MadTeaParty” (der noch nicht existiert und damit automatisch angelegt wird) wird ein Hash übergeben, das alle Werte, die wir setzen wollen, in Form von Schlüssel-Wert-Paaren enthält. Der Hash-Schlüssel “RegWriter” hat als Hash-Wert wiederum einen Hash: auf diese Weise lassen sich in einem Aufwasch Unterschlüssel anlegen, und das in beliebiger Tiefe! “RegWriter” wird somit nicht zu einem Wert des Schlüssels “MadTeaParty”, sondern stellt einen eigenen Schlüssel unter

“HKEY_CURRENT_USER\\MadTeaParty” dar. *Beachte:* Da “RegWriter” ein Schlüssel ist und kein Wert, fehlt davor ein Slash. Denn nur Werte sollen durch einen doppelten Delimiter vom Pfad abgetrennt werden.
Wenn wir jetzt noch einmal einen Blick auf die Werte werfen, die wir übergeben, so stellen wir fest, dass “VendorName” nur eine Zeichenkette zugewiesen wird, die Bezeichnung des Typs (REG_SZ) allerdings gänzlich fehlt. Das ist legitim, da REG_SZ automatisch gesetzt wird. Nur wenn Werte anderer Typen gesetzt werden, *muss* der Typ explizit angegeben werden. Die “URI” haben wir übrigens als REG_EXPAND_SZ angegeben. Die zugewiesene URI enthält zwar keine Umgebungsvariable, könnte aber genauso gut nach einer anderen Installation auf ein Spezialverzeichnis des Systems verweisen, bspw. %windir% für das Windows-Verzeichnis.

2.4. Schlüssel löschen

Weil wir unsere Registry sauber halten wollen und einen Schlüssel “MadTeaParty” nicht brauchen, werden wir ihn nun löschen. Datei **regdelete.pl**:

```
use Data::Dumper;
use Win32::TieRegistry;

$Registry->Delimiter("/");
$key = $Registry->{"CUser/"};
$dump = delete $key->{"MadTeaParty/"};
print "Schluessel geloescht!\n";
print Dumper($dump);
```

Unser vorher neu angelegter Schlüssel wird mit **delete** gelöscht. Wir bekommen übrigens einen Rückgabewert: er enthält den soeben gelöschten Schlüssel als Hash-Referenz, gewissermaßen ein Backup des Schlüssels. Wenn wir anschließend den Registry-Editor regedit öffnen, sehen wir, dass der Schlüssel komplett gelöscht wurde, samt aller Werte und Unterschlüssel. Um das in $dump gespeicherte “Backup” des gelöschten Schlüssels in eine druck-/speicherbare Form zu bringen, verwenden wir die Funktion **Dumper** aus dem Paket **Data::Dumper**.
Während obiges Beispiel unter Windows XP ohne Probleme funktionierte, kann es bei älteren Plattformen vorkommen, dass der Unterschlüssel “RegWriter” nicht mehr angezeigt werden kann. Um auch ihn zu visualisieren, verbessern wir unser Skript so:

```
use Data::Dumper;
use Win32::TieRegistry;

$Registry->Delimiter("/");
$key = $Registry->{"CUser/"};
$dump2 = delete $key->{"MadTeaParty/RegWriter/"};
$dump = delete $key->{"MadTeaParty/"};
print "Schluessel geloescht!\n";
print Dumper($dump2),Dumper($dump);
```

Hier muss der Baum von „oben“ abgearbeitet werden, sprich, wir löschen zuerst die Unterschlüssel und fangen ihren Wert ab. Die unterste Pfadebene wird zuletzt gelöscht.

3. Registry bearbeiten mit WSH

Wer nicht mit Perl arbeiten will, statt dessen aber Kenntnisse In VBScript oder JScript besitzt, kann auch mit Windows-Scripts arbeiten. Die Arbeitsweise ist ebenso eingängig wie mit Perl, die Scripte kaum länger oder kürzer. Es genügt, ein **Shell-Objekt** zu erzeugen, welches die Methoden **RegRead** (Lesen von Werten), **RegWrite** (Schreiben von Werten) und **RegDelete** (Löschen eines Schlüssels und aller Werte) bereitstellt.
Im Folgenden werde ich exakt dieselben Beispiele wieder aufgreifen, die ich schon mit Perl dargestellt habe. Die Skripte liegen im WSF-Format vor, jeweils in den Sprachen VBScript, JScript und PerlScript. Während VBScript und JScript hinreichend bekannt sein sollten, stellt PerlScript eine Möglichkeit für den Perl-Liebhaber dar, auch ein Perl-Programm unter dem Windows Scripting Host laufen zu lassen (mit installiertem ActivePerl).
Weitere Informationen und Beispiele können in der WSH-Dokumentation (in Englisch) von Microsoft gefunden werden. Diese kann als Windows-Help-File herunter geladen werden unter: http://msdn.microsoft.com/scripting

3.1. Wert lesen

Wir lesen im Folgenden wieder die Zeitzone aus der Registry Die Datei **regread.wsf** in **VBScript**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<package>
  <job id="VBSRegRead">
    <script language="VBScript">
    <![CDATA[

      Option Explicit
      Const key =
"HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\TimeZoneInformation"
      Dim Sh
      Set Sh = WScript.CreateObject("WScript.Shell")
      Wscript.Echo Sh.RegRead(key & "\DaylightName")

    ]]>
    </script>
  </job>
</package>
```

In **JScript**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<package>
  <job id="JSRegRead">
    <script language="JScript">
    <![CDATA[

      var key =
"HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\Control\\TimeZoneInformation";
      var Sh = WScript.CreateObject("WScript.Shell");
      WScript.Echo(Sh.RegRead(key + "\\DaylightName"));
```

```
   ]]>
   </script>
  </job>
</package>
```

In **PerlScript**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<package>
  <job id="PSRegRead">
    <script language="PerlScript">
    <![CDATA[

      my $key =
"HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\Control\\TimeZoneInformation";
      my $Sh = $WScript->CreateObject("WScript.Shell");
      $WScript->Echo($Sh->RegRead("$key\\DaylightName"));

    ]]>
    </script>
  </job>
</package>
```

Der Code dürfte keiner großen Erklärung bedürfen. Nachdem ein Objekt vom Typ **WScript.Shell** erzeugt wurde (gespeichert in der Variable “Sh”), lässt sich mit seiner Methode **RegRead** der Wert unseres Schlüssels ganz einfach abfragen. Da RegRead eine Funktion (mit Rückgabewert) und keine Prozedur ist, sind in VBScript Klammern um die Argumente nötig. Beachte, dass zwischen Schlüssel (“TimeZoneInformation”) und Wert (“DaylightName”) anders als in Perl nur ein Delimiter notwendig ist. In JScript und in PerlScript ist zudem ein doppelter Backslash notwendig, da der einfache Rückstrich unter JScript und PerlScript ein besonderer Character ist.

3.2. Wert ändern

Und wieder wollen wir bei Notepad die Funktion “Wrap” aktivieren. Die Datei **regalter.wsf** in **VBScript**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<package>
  <job id="VBSRegAlter">
    <script language="VBScript">
    <![CDATA[

      Option Explicit
      Const key = "HKEY_CURRENT_USER\Software\Microsoft\Notepad"
      Dim Sh
      Set Sh = CreateObject("WScript.Shell")
      Sh.RegWrite key & "\fWrap", 1, "REG_DWORD"
      WScript.Echo "Wrap bei Notepad aktiviert!"

    ]]>
```

```
  </script>
 </job>
</package>
```

In **JScript**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<package>
 <job id="JSRegAlter">
  <script language="JScript">
  <![CDATA[

    var key = "HKEY_CURRENT_USER\\Software\\Microsoft\\Notepad";
    var Sh = new ActiveXObject("WScript.Shell");
    Sh.RegWrite(key + "\\fWrap", 1, "REG_DWORD");
    WScript.Echo("Wrap bei Notepad aktiviert!");

  ]]>
  </script>
 </job>
</package>
```

In **PerlScript**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<package>
 <job id="PSRegAlter">
  <script language="PerlScript">
  <![CDATA[

    my $key = "HKEY_CURRENT_USER\\Software\\Microsoft\\Notepad";
    my $Sh = $WScript->CreateObject("WScript.Shell");
    $Sh->RegWrite("$key\\fWrap", 1, "REG_DWORD");
    $WScript->Echo("Wrap bei Notepad aktiviert!");

  ]]>
  </script>
 </job>
</package>
```

Wieder beschaffen wir zuerst ein Shell-Objekt (diesmal zu Demonstrations-zwecken als ActiveX-Objekt). Dann wenden wir die Methode **RegWrite** an, um den Wert zu verändern. Dies ist eine Prozedur (in VBScript braucht es keine runden Klammern um die Argumente), die 3 Argumente erhält: den kompletten Pfad zu Schlüssel und Wert, der Wert, der an diese Stelle geschrieben werden soll und schließlich der Datentyp. Wir schreiben hier eine Ganzzahl, deshalb muss der Typ REG_DWORD sein. Die hier erlaubten Angaben entsprechen denen von Perl, da sie nicht von der Programmiersprache, sondern von der Windows-Registry selbst vorgegeben werden.

3.3. Neuen Schlüssel anlegen

Wie im Perl-Beispiel, legen wir wieder den Schlüssel “MadTeaParty” unter

“HKEY_CURRENT_USER” an. Dies erledigen wir wieder über die Methode **RegWrite** des Shell-Objekts. Zu beachten ist hier Folgendes: für jeden Wert, den wir in die Registry schreiben wollen, muss RegWrite einzeln aufgerufen werden. Existiert der Schlüssel zu dem angegeben Pfad bis dato noch nicht, wird er automatisch angelegt. Und für Daten vom Typ REG_SZ gibt es, anders als in Perl, keine Sonderbehandlung. Ihr Typ muss immer als 3. Parameter übergeben werden.
Die Datei **regwrite.wsf** in **VBScript**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<package>
  <job id="VBSRegWrite">
    <script language="VBScript">
    <![CDATA[

      Option Explicit
      Const key = "HKEY_CURRENT_USER\"
      Dim Sh
      Set Sh = CreateObject("WScript.Shell")

      Sh.RegWrite key & "MadTeaParty\VendorName", "Chrstph", "REG_SZ"
      Sh.RegWrite key & "MadTeaParty\URI", "http://www.mad-teaparty.com",
"REG_EXPAND_SZ"
      Sh.RegWrite key & "MadTeaParty\RegWriter\VersionNumber", "1.0", "REG_SZ"
      Sh.RegWrite key & "MadTeaParty\RegWriter\DefaultValue", 0, "REG_DWORD"
      WScript.Echo "Registrierungs-Datenbank aktualisiert!"

    ]]>
    </script>
  </job>
</package>
```

In **JScript**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<package>
  <job id="JSRegWrite">
    <script language="JScript">
    <![CDATA[

      var key = "HKEY_CURRENT_USER\\";
      var Sh = new ActiveXObject("WScript.Shell");
      Sh.RegWrite(key + "MadTeaParty\\VendorName", "Chrstph", "REG_SZ");
      Sh.RegWrite(key + "MadTeaParty\\URI", "http://www.mad-teaparty.com",
"REG_EXPAND_SZ");
      Sh.RegWrite(key + "MadTeaParty\\RegWriter\\VersionNumber", "1.0", "REG_SZ");
      Sh.RegWrite(key + "MadTeaParty\\RegWriter\\DefaultValue", 0, "REG_DWORD");
      WScript.Echo("Registrierungs-Datenbank aktualisiert!");

    ]]>
    </script>
  </job>
</package>
```

In **PerlScript**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<package>
  <job id="PSRegWrite">
    <script language="PerlScript">
    <![CDATA[

      my $key = "HKEY_CURRENT_USER";
      $Sh = $WScript->CreateObject("WScript.Shell");
      $Sh->RegWrite("$key\\MadTeaParty\\VendorName", "Chrstph", "REG_SZ");
      $Sh->RegWrite("$key\\MadTeaParty\\URI", "http://www.mad-teaparty.com",
"REG_EXPAND_SZ");
      $Sh->RegWrite("$key\\MadTeaParty\\RegWriter\\VersionNumber", "1.0", "REG_SZ");
      $Sh->RegWrite("$key\\MadTeaParty\\RegWriter\\DefaultValue", 0, "REG_DWORD");
      $WScript->Echo("Registrierungs-Datenbank aktualisiert!");

    ]]>
    </script>
  </job>
</package>
```

3.4. Schlüssel löschen

Als Abschluss unseres Tutorials löschen wir wieder den Schlüssel “HKEY_CURRENT_USER\\MadTeaParty”. Wie immer wird dazu ein Shell-Objekt beschafft und deren Methode **RegDelete** aufgerufen. Diese Prozedur erwartet nur ein Argument: den Pfad des zu löschenden Schlüssels. Alle darin enthaltenen Werte und Unterschlüssel werden automatisch gelöscht. Und an dieser Stelle noch einmal der Hinweis: *was gelöscht wurde, ist unwiederbringlich weg*!
Die Datei **regdelete.wsf** in **VBScript**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<package>
  <job id="VBSRegWrite">
    <script language="VBScript">
    <![CDATA[

      Option Explicit
      Const key = "HKEY_CURRENT_USER\"
      Dim Sh
      Set Sh = CreateObject("WScript.Shell")
      Sh.RegDelete key & "MadTeaParty\\"
      WScript.Echo "Schlüssel gelöscht!"

    ]]>
    </script>
  </job>
</package>
```

In **JScript**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<package>
  <job id="JSRegWrite">
    <script language="JScript">
    <![CDATA[

      var key = "HKEY_CURRENT_USER\\";
      var Sh = new ActiveXObject("WScript.Shell");
      Sh.RegDelete(key + "MadTeaParty\\");
      WScript.Echo("Schlüssel gelöscht!");

    ]]>
    </script>
  </job>
</package>
```

In **PerlScript**:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<package>
  <job id="PSRegWrite">
    <script language="PerlScript">
    <![CDATA[

      my $key = "HKEY_CURRENT_USER";
      $Sh = $WScript->CreateObject("WScript.Shell");
      $Sh->RegDelete("$key\\MadTeaParty\\");
      $WScript->Echo("Schlüssel gelöscht!");

    ]]>
    </script>
  </job>
</package>
```

4. Fazit

Mit Perl und WSH stehen uns zwei sehr einfache und schnelle Möglichkeiten zur Verfügung, die Windows-Registry zu bearbeiten. Das hier vorgestellte Wissen sollte ausreichen, um eigene Vorhaben zu verwirklichen. Beispielsweise können so bequem Wallpaper und Screensaver ausgewechselt werden, das Aussehen der Konsole verändert werden, Informationen zu Prozessor und registrierter Hard- und Software ausgelesen werden und und und. Ich möchte an dieser Stelle nochmals betonen, dass beim Umgang mit der Windows-Registry *äußerste Vorsicht* geboten ist. *Jeder ist selbst dafür verantwortlich, dass sein Rechner unbeschädigt bleibt!*

Christoph Bichlmeier
E-Mail: chris 'at' bichlmeier 'dot' info
Website: http://www.bichlmeier.info
erstellt am: 17.10.2004