Renée Bäcker

Regelmäßige Backups

Für einen Kunden habe ich eine webbasierte Anwendung mit Datenbankanbindung und Uploadmöglichkeiten geschrieben. Und diese Daten sollen regelmäßig gesichert werden - auch dafür existiert ein kleines Skript, das die Daten zusammen mit Meta-Informationen in ein Archiv schreibt. Damit das nicht immer wieder vom Administrator per Hand gemacht werden muss, sollte das ganze automatisch und regelmäßig passieren.

Derjenige, der die Anwendung betreut, kennt sich mit dem Task-System auf Windows nicht aus. Unter Unix findet sich `cron` und Unix-Anwender kennen sich eher mit `cron` aus, als Windows-Anwender mit dem Task-System. Damit ich den Task nicht einrichten muss und nicht bei einem Server-Umzug das ganze wieder eingerichtet werden muss, habe ich ein entsprechendes Perl-Skript geschrieben.

Wie so häufig, gibt es auf CPAN ein Modul, das für diese Aufgabe bestens geeignet ist: `Win32::TaskScheduler`. Allerdings ist die Installation aus den Sourcen auch mit StrawberryPerl zumindest in der Version 2.0.3 des Moduls nicht ohne weiteres möglich. Aber da auch mit StrawberryPerl ein PPM-Client (PPM = Perl Package Manager) mitgeliefert wird und das Modul in einem Repository existiert, ist die Installtion dennoch möglich.

```
my %commands = (
    run       => \&do_backup,
    install   => \&install_task,
    uninstall => \&uninstall_task,
);

$command ||= 'help';
my $sub = $commands{$command} ||
                          $commands{help};
$sub->();
```

Listing 1

Mit diesem Modul kann man Tasks mit beliebiger Konfiguration einrichten. Hier soll der oben beschriebene Task gezeigt werden. Anstelle des Backups wird hier einfach eine Nachricht in einem kleinen Fenster angezeigt.

Das Skript soll für alle Aufgaben bezüglich des Tasks zuständig sein, also Einrichten des Tasks, Deinstallieren des Tasks und Erstellen des Backups. Damit die jeweils richtige Funktion aufgerufen wird, benutze ich einen Hash als Dispatchtabelle (Listing 1).

In diesem Artikel soll auf das Erstellen des Backups nicht näher eingegangen werden. Dort wird einfach mit `Archive::Tar` ein Archiv gepackt und dieses an eine bestimmte Stelle verschoben. Hier soll es nur um die Einrichtung und das Deinstallieren der Aufgabe gehen.

Als erstes muss die Aufgabe eingerichtet werden (Listing 2). Ab der Version 2.0.0 von `Win32::TaskScheduler` muss vor dem Zugriff auf den Aufgabenplaner von Windows ein neues Objekt erzeugt werden. Das war vorher nicht der Fall. Als erstes muss eine neue Aufgabe hinzugefügt werden (`New-WorkItem`). Der Methode muss ein eindeutiger Namen übergeben werden. Ist dieser Name schon in der Verwendung, schlägt das Hinzufügen fehl. Weiterhin muss die Trigger-Konfiguration übergeben werden. Die Konfiguration kann etwas tricky sein. Bei meinen ersten Tests mit dem Modul habe ich ständig Fehler "Null pointer exception" bekommen - ohne genaue Fehlermeldung. Nach ein wenig Ausprobieren bin ich darauf gestoßen, dass es an einem Fehler in der Konfiguration lag (fehlende Angabe von `MinutesDuration`).

In der Konfiguration wird festgelegt, ab wann die Aufgabe ausgeführt werden soll und wann die Aufgabe enden soll. Mit `MinutesInterval` wird bestimmt, in welchen Interval-

```
sub install_task {
    my $scheduler = Win32::TaskScheduler->New;

    my %task_config = (
        BeginYear   => 2010,
        BeginMonth  => 8,
        BeginDay    => 1,
        EndYear     => 2201,
        EndMonth    => 8,
        EndDay      => 1,
        StartHour   => 12,
        StartMinute => 1,
        MinutesDuration => 1440,
        MinutesInterval => 5,
        TriggerType => 1,
        Type => {
            DaysInterval => 1,
        },
    );

    my ($has_set_name, $saved) = ("","");
    my $has_new_item = $scheduler->NewWorkItem( $task_name, \%task_config );
    if ( $has_new_item == 1 ) {
        $has_set_name = $scheduler->SetApplicationName( $task_app );
        $scheduler->SetParameters( $task_param );

        $scheduler->SetAccountInformation( 'Administrator', 'passwort' );

        if ( $has_set_name ) {
            $saved = $scheduler->Save;
            $scheduler->Activate( $task_name ) if $saved;
        }
    }

    print "Aufgabe eingerichtet" if $saved;
}
```

Listing 2

len die Aufgabe ausgeführt werden soll. Vom `TriggerType` hängt dann ab, was in dem Subhash `Type` angegeben werden muss.

Können in der Konfiguration im `Type`-Hash mehrere Werte möglich sein (z.B. bei `Months`), werden diese mit einem "|" verbunden. Beispiele dafür kommen in den nächsten Beispielen.

Um es lesbarer zu halten kann man die Konstanten aus dem Modul benutzen. Es gibt diese Trigger-Typen:

• TASK_TIME_TRIGGER_ONCE
Eine Aufgabe muss nur ein einziges Mal ausgeführt werden. Hier werden keine weiteren Angaben im `Type`-Hash benötigt.

• TASK_TIME_TRIGGER_DAILY
Im `Type`-Hash muss `DaysInterval` angegeben werden, das bestimmt, wie viele Tage zwischen den Ausführungen liegen. Die Konfiguration

```
MinutesInterval => 5,
TriggerType     =>
   $scheduler->TASK_TIME_TRIGGER_DAILY,
Type             => {
    DaysInterval => 10,
},
```

bedeutet, dass die Ausgabe an jedem 10. Tag alle 5 Minuten ausgeführt wird.

• TASK_TIME_TRIGGER_WEEKLY
Für Tasks, die im Wochenrythmus ausgeführt werden sollen, gibt es den Trigger `TASK_TIME_TRIGGER_WEEKLY`. Benutzt man diesen Trigger, müssen im `Type`-Hash zwei Angaben gemacht werden - `WeeksInterval` und `DaysOfTheWeek`.

Möchte man einen Task also alle zwei Wochen dienstags und donnerstags ausgeführt haben, muss man das so konfigurieren:

```
TriggerType =>
        $scheduler->TASK_TIME_TRIGGER_WEEKLY,
Type => {
     WeeksInterval => 2,
     DaysOfTheWeek =>
           $scheduler->TASK_TUESDAY |
                 $scheduler->TASK_THURSDAY,
},
```

• TASK_TIME_TRIGGER_MONTHLYDATE

Dieser Trigger ist dafür da, in bestimmten Monaten zu einem bestimmten Datum eine Aufgabe auszuführen. Soll das Programm am 15. Juni und am 15. Dezember ausgeführt werden, muss die Konfiguration wie folgt aussehen:

```
TriggerType =>
   $scheduler->TASK_TIME_TRIGGER_MONTHLYDATE,
Type        => {
    Months => $scheduler->TASK_JUNE |
                  $scheduler->TASK_DECEMBER,
    Days   => 15,
},
```

Hier existiert auch ein Unterschied zur offiziellen Microsoft-API. Während dort mehrere Tage angegeben werden können, ist bei dem Modul immer nur 1 Tag möglich. Muss das Programm an mehreren Tagen in den Monaten ausgeführt werden, muss man mehrere Tasks anlegen.

• TASK_TIME_TRIGGER_MONTHLYDOW

Mit diesem Trigger-Typ ist es möglich, Aufgabe zu Zeiten wie "in den Monaten Februar, Mai, August und November am Montag der jeweils ersten Woche" auszuführen.

Für dieses Beispiel sieht die Konfiguration so aus:

```
TriggerType =>
   $scheduler->TASK_TIME_TRIGGER_MONTHLYDATE,
Type        => {
    Months =>
      $scheduler->TASK_FEBRUARY |
      $scheduler->TASK_MAY |
      $scheduler->TASK_AUGUST |
      $scheduler->TASK_NOVEMBER,
    DaysOfTheWeek => $scheduler->TASK_MONDAY,
    WhichWeek     => 1,
},
```

Wenn man nicht weiß, wie der Trigger aussehen soll, hilft es, auf einem Testrechner manuell so einen Task in der "Aufgabenplanung" (findet man bei "Zubehör" -> "Systemprogramme") anzulegen und dann den Trigger auszulesen:

```
use Data::Dumper;
use Win32::TaskScheduler;

my $task_name =
             'NameDesManuellAngelegtenTasks';
my $scheduler = Win32::TaskScheduler->New;
$scheduler->Activate( $task_name );

my %trigger;
my $result = $scheduler->GetTrigger(
                                0, \%trigger );
warn Dumper \%trigger;
```

Mit dem Modul kann man auch die anderen Tasks beobachten und die Ergebnisse (Status Code) der letzten Läufe bekommen:

```
$scheduler->GetStatus($status);
if ( $status == 267008 ) { #Ready
    print "\tStatus:ready\n";
}
elsif ( $status == 267009 ) { #Running
    print "\tStatus:RUNNING\n";
}
elsif ( $status == 267010 ) { #Not Scheduled
    print "\tStatus:Not Scheduled\n";
}
else {
    print "\tStatus:UNKNOWN\n";
}
$scheduler->GetExitCode($exitcode);
print "\tExitCode:".$exitcode."\n";
```

Gerade wenn man Software für Windows-Nutzer schreibt, ist dieses Modul sehr nützlich. Da die Dokumentation nicht die allerbeste ist, muss man ein paar Sachen ausprobieren.