Ausgabe 3/2007 – www.foo-magazin.de

$foo

PERL MAGAZIN

Perl 6
Der Himmel für Programmierer

ApplePerl
MAC Programme fernsteuern mit Perl

LDAP
Workshop

Nr 03

Sichern Sie Ihren nächsten Schritt in die Zukunft

Astaro steht für benutzerfreundliche und kosteneffiziente Netzwerksicherheitslösungen. Heute sind wir eines der führenden Unternehmen im Bereich der **Internet Security** mit einem weltweiten Partnernetzwerk und Büros in Karlsruhe, Boston und Hongkong. Eine Schlüsselrolle im Hinblick auf unseren Erfolg spielen unsere Mitarbeiter und hoffentlich demnächst auch Sie! Astaro bietet Ihnen mit einer unkomplizierten, kreativen Arbeitsumgebung und einem dynamischen Team beste Voraussetzungen für Ihre berufliche Karriere in einem interessanten, internationalen Umfeld.

Zur Verstärkung unseres Teams in Karlsruhe suchen wir zum nächstmöglichen Eintritt:

Perl Backend Developer (m/w)

Ihre Aufgaben sind:

- Entwicklung und Pflege von Software-Applikationen
- Durchführung eigenständiger Programmieraufgaben
- Optimierung unserer Entwicklungs-, Test- und Produktsysteme
- Tatkräftige Unterstützung beim Aufbau und der Pflege des internen technischen Know-hows

Unsere Anforderungen an Sie sind:

- Fundierte Kenntnisse in der Programmiersprache Perl, weitere Kenntnisse in anderen Programmier- oder Script-Sprachen wären von Vorteil
- Selbstständiges Planen, Arbeiten und Reporten
- Fließende Deutsch- und Englischkenntnisse

Software Developer (m/w)

Ihre Aufgaben sind:

- Entwicklung und Pflege von Software-Applikationen
- Durchführung eigenständiger Programmieraufgaben
- Optimierung unserer Entwicklungs-, Test- und Produktsysteme
- Tatkräftige Unterstützung beim Aufbau und der Pflege des internen technischen Know-hows

Unsere Anforderungen an Sie sind:

- Kenntnisse in den Programmiersprachen Perl, C und/oder C++ unter Linux, weitere Kenntnisse in anderen Programmier- oder Script-Sprachen wären von Vorteil
- Kompetenz in den Bereichen von Internet Core Protokollen wie SMTP, FTP, POP3 und HTTP
- Selbstständiges Planen, Arbeiten und Reporten
- Fließende Deutsch- und Englischkenntnisse

Astaro befindet sich in starkem Wachstum und ist gut positioniert um in den Märkten für IT-Sicherheit und Linux-Adaption auch langfristig ein führendes Unternehmen zu sein. In einer unkomplizierten und kreativen Arbeitsumgebung finden Sie bei uns sehr gute Entwicklungsmöglichkeiten und spannende Herausforderungen. Wir bieten Ihnen ein leistungsorientiertes Gehalt, freundliche Büroräume und subventionierte Sportangebote. Und nicht zuletzt offeriert der Standort Karlsruhe eine hohe Lebensqualität mit vielen Möglichkeiten zur Freizeitgestaltung in einer der sonnigsten Gegenden Deutschlands.

Interessiert?
Dann schicken Sie bitte Ihre vollständigen Unterlagen mit Angabe Ihrer Gehaltsvorstellung an careers@astaro.com. Detaillierte Informationen zu den hier beschriebenen Stellenangeboten und **weitere interessante Positionen** finden Sie unter www.astaro.de. Wir freuen uns darauf, Sie kennen zu lernen!

Astaro AG
Amalienbadstr. 36 • D-76227 Karlsruhe
Monika Heidrich • Tel.: 0721 25516 0

www.astaro.de

astaro
internet security

e-mail | web | net | security

VORWORT

Persönliche Treffen sind toll -

... und in den nächsten Wochen gibt es einige Möglichkeiten dazu: Am Wochenende vom 25. bis 26. August findet in Sankt Augustin die *FrOSCon* statt. Dort geht es nicht nur hauptsächlich um Perl, sondern auch der Austausch mit anderen Entwicklern ist sehr wichtig und schön. Und wir von $foo werden dort auch mit einem kleinen Stand vertreten sein.

Direkt im Anschluss an die FrOSCon geht es weiter nach Wien. Dort findet vom 28. bis zum 30. August die diesjährige *YAPC::Europe* statt. Neben den vielen Perl-Entwicklern aus Deutschland, Österreich und den anderen europäischen Ländern werden auch vier echte "Hochkaräter" anwesend sein: Audrey Tang, die uns mit Pugs einen ersten Einblick in Perl 6 gewährt. Der Autor von "Higher Order Perl" - Marc Jason Dominus - wird ebenfalls an der Konferenz teilnehmen.
Richtig tolle Vorträge wird es von Damian Conway geben und auch den Erfinder von Perl - Larry Wall - kann man in Wien kennenlernen.

Persönliche Treffen waren auch in letzter Zeit nötig um mit den Designern von *//SEIBERT/MEDIA* das Layout für $foo zu besprechen. Und ich denke, dass sich das Ergebnis sehen lassen kann. In Zukunft werden die Ausgaben immer in diesem Stil erscheinen.

Bei dieser Gelegenheit möchte ich mich bei O'Reilly dafür bedanken, dass wir die Kamele für das Layout verwenden dürfen.

Das Layout ist das Einzige, das sich in dieser Ausgabe augenscheinlich geändert hat. Ansonsten haben wir nur bei den Autoren Zuwachs bekommen. Ohne die freiwilligen Autoren wäre so eine Ausgabe nur schwer zu stemmen. Ich bin schon mehrere Male gefragt worden, wie viel Stunden ich denn für so eine Ausgabe aufwenden würde - ich habe es ehrlich gesagt noch nicht verfolgt, aber ein Betriebswirt würde das im Moment wohl noch als "ökonomisch sinnlos" bezeichnen.

Ansonsten bleibt mir nur noch, den Download-Code für die Beispielprogramme zu nennen. Für diese Ausgabe lautet der Code

RTL5+D

Viel Spaß beim Lesen
Renée Bäcker

IMPRESSUM

Herausgeber:	Smart Websolutions Windolph und Bäcker GbR Maria-Montessori-Str. 13 D-64584 Biebesheim
Redaktion:	Renée Bäcker, André Windolph
Anzeigen:	Katrin Blechschmidt
Layout:	//SEIBERT/MEDIA
Auflage:	500 Exemplare
Druck:	powerdruck Druck- & VerlagsgesmbH Wienerstraße 116 A-2483 Ebreichsdorf
ISSN Print:	1864-7537
ISSN Online:	1864-7545

INHALTSVERZEICHNIS

Hier werden kurz die Autoren vorgestellt, die zu dieser Ausgabe beigetragen haben.

Renée Bäcker

Seit 2002 begeisterter Perl-Programmierer und seit 2003 selbständig. Auf der Suche nach einem Perl-Magazin ist er nicht fündig geworden und hat so diese Zeitschrift herausgebracht. In der Perl-Community ist Renée recht aktiv - als Moderator bei Perl-Community.de, Organisator des kleinen Frankfurt Perl-Community Workshops und Mitglied im Orga-Team des deutschen Perl-Workshops.

Herbert Breunung

Ein perlbegeisteter Programmierer aus dem wilden Osten, der eine Zeit lang auch Computervisualistik studiert hat, aber schon vorher ganz passabel programmieren konnte. Er ist vor allem geistigem Wissen, den schönen Künsten und elektronischer sowie handgemachter Tanzmusik zugetan. Seit einigen Jahren schreibt er einen Texteditor in Perl für den noch dringend ein Name gesucht wird und betätigt sich auch aktiv auf Perl-Community.de und der Wikipedia, wo er fast nur noch die Kategorie Programmiersprache Perl betreut.

Martin Fabiani

Martin Fabiani http://www.fabiani.net/ (33 Jahre alt) kommt aus Nordtirol, lebt und arbeitet aber seit 1998 in Deutschland. Seit 1999 ist er freiberuflich als Perl-Entwickler und -Trainer tätig, und hat im Jahr 2000 über Perl das spannende Thema Metadirectories und Identity Management entdeckt, wo man Perl hervorragend für Datensynchronisationen verwenden kann, vor allem bei größeren Datenmengen und komplexeren Synchronisationsalgorithmen, und somit auch für die teilautomatisierte Administration verbundener Systeme.

In seiner Freizeit leitet er das Forum auf http://www.perl-community.de unter dem Pseudonym Strat und versucht, ein Mega-Synchronisationsframework für Perl mit Schnittstellen zu einer Menge verschiedener Datenhaltungssystemen zu entwickeln.

Wolfgang Kinkeldei

Wolfgang Kinkeldei arbeitet als Software-Entwickler bei einem mittelständischen Mediendienstleister in Nürnberg. Zu seinen Hauptaufgaben zählen die Automatisierung von Arbeitsabläufen in der Druckvorstufe sowie die Erstellung von Web-basierten Lösungen. Die meisten seiner Projekte werden mit Perl gelöst.

Max Maischein

Max Maischein ist Baujahr 1973 und studierter Mathematiker. Seit 2001 ist er für die DZ BANK in Frankfurt tätig und betreut dort den Fachbereich Operations und Services im Prozess- und Informationsmanagement.

Mike Schilli

Michael Schilli kümmert sich bei Yahoo in Sunnyvale/Kalifornien um die Perl-Belange des Unternehmens. Er schreibt seit nunmehr fast 10 Jahren die monatliche Perl-Kolumne “Perl-Snapshot“ im Linux-Magazin.

Steffen Schwigon

Steffen Schwigon wurde 1972 geboren und hat kurz danach, ok, 1987, angefangen zu programmieren. Atari Basic, Pascal, C, C++, Java, Perl. In der Reihenfolge.
Debian GNU/Linux ist sein Betriebssystem, XEmacs sein Editor. In Perl programmiert er seit 2002.
Er ist aktives Mitglied der Dresden Perl Mongers und die Rosamunde Pilcher unter den Perl-Workshop-Orgas.

Mac Programme fernsteuern mit Perl

Häufig sind Artikel zu lesen und Vorträge zu hören, die sich damit befassen, wie cool es ist, mittels `Win32::OLE` oder vergleichbaren Modulen GUI Applikationen unter Windows fernzusteuern. Dabei werden in der Regel Dokumente oder Dateien in Formaten erstellt, die mit Bordmitteln von Perl alleine nicht generierbar sind.

Schnell entsteht der Eindruck, dass Windows einzigartig mit diesem Feature ist. Jedoch ist die Technologie, die Apple einsetzt, nicht nur universeller, sondern auch generischer angelegt, selbst dokumentierend, bereits seit Oktober 1993 fester Bestandteil des Betriebssystems und integriert in fast alle Applikationen. Grund genug, auch hier Perl in Aktion treten zu lassen, um diese Technologie geschickt zu erweitern.

Warum fernsteuern?

Gerade auf der Apple Macintosh Plattform gibt es zahlreiche Programme, die vor allem in der professionellen Druckvorstufe sowie der Multimedia und Videobearbeitung eingesetzt werden, um alle Facetten der Grafikbearbeitung, professionellem Schriftsatz oder Video Schnittaufgaben zu erledigen. Sehr häufig jedoch wird die Erstellung von Dokumenten zur lästigen, sich wiederholenden Aufgabe oder muss aus vorliegenden Daten angefertigt werden. Automatisierung anstelle von lästigem Copy&Paste oder unangenehmer Wiederholung von Einzelschritten ist hier also wünschenswert.

Alternative Methoden zur Erzeugung spezieller Dokumente gibt es kaum, da die meisten sonstigen Werkzeuge Limitationen hinsichtlich des notwendigen Workflows oder im Umgang mit Dateiformaten mit sich bringen und damit ausscheiden. Selbstverständlich lassen sich auch beliebige andere Aufgaben damit automatisieren.

AppleScript Grundlagen

Apple setzt sogenannte Apple Events als Methodik der Interprozess-Kommunikation (IPC) ein. Ein Apple Event besteht linguistisch gesehen aus einem Verb (z.B. “setze“, “lösche“ oder “öffne“) meist gefolgt von einem Objekt, auf das der verbale Befehl anzuwenden ist sowie weiteren optionalen Parametern. Das Objekt wird wahlweise durch eine bekannte ID oder durch einen Suchpfad identifiziert (z.B. “zweiter Buchstabe des dritten Wortes im ersten Absatz von Dokument 1“ oder “Textblock ID 4711“), wobei verschiedene Objektklassen unterschiedliche Methoden besitzen um auf deren Unterobjekte zuzugreifen.

Apple Events selbst sind streng typisiert und unterstützen neben mehreren skalaren auch strukturierte Datentypen, die das Perl-Äquivalent zu Arrays und Hashes darstellen. Jeder Apple Event wird in Form eines sogenannten `Apple Event Record` direkt an die Ziel-Applikation gerichtet und liefert einen dem Aufruf entsprechenden Apple Event Record als Antwort zurück, der wiederum beliebige (strukturierte oder skalare) Datentypen enthalten kann.

Soll ein Apple Event versandt werden, so ist zunächst entsprechend der gewünschten Aktion eine entsprechende Datenstruktur zu erstellen, die dann über den IPC Mechanismus des Betriebssystems direkt an die zu steuernde Applikation gesandt wird. Die Applikation wiederum weiß mit der Anforderung umzugehen, verarbeitet die

Anweisungen und liefert eine dem Befehl entsprechende Antwort-Struktur zurück. Sofern der Aufruf synchron war, steht die Antwort dem aufrufenden Programm zur Weiterverarbeitung zur Verfügung.

Viele Entwickler nutzen auf der Macintosh Plattform die Sprache `AppleScript` zur Steuerung von Applikationen. Leider jedoch ist `AppleScript` bei weitem nicht so mächtig wie Perl, wenn es um Dateioperationen, Stringmanipulation, Textkonvertierung oder Datenbankanbindung und Netzwerkfunktionen geht, weshalb man hier gerne auf Perl zurückgreifen möchte.

Jedes Programm, das per `AppleScript` gesteuert werden kann, verfügt automatisch über ein Verzeichnis seines Vokabulars sowie eine Übersicht über die (Klassen- und Assoziations-) Hierarchie seiner Objekte, deren Zusammenhang, Eigenschaften, Methoden und Vererbung. Per GUI kann man schnell über das im Lieferumfang von OS-X enthaltene Programm `Script Editor` über die `Bibliothek Funktion` einen Überblick über die jeweiligen Eigenschaften eines Programmes erhalten. Siehe Abb 1.

Unsere Beispielaufgabe

Das für diesen Artikel gewählte Beispiel benutzt das Programm “Text Edit“, einen einfachen Texteditor, der mit OS-X zusammen ausgeliefert wird. Ein vorhandenes Musterdokument soll um einen entsprechenden Zusatztext erweitert und anschließend als RTF formatierter Text abgespeichert werden.

Dies ist sicher kein typisches Automatisierungsbeispiel, verdeutlicht aber dennoch die typischen Verfahren und ist ohne teure Spezialprogramme und die damit verbundene Einarbeitung schnell nachvollziehbar. Richtig interessannt wird eine solche Automatisierung sicher erst mit besonderen Programmen, die Datenformate erstellen, für die es eben keine Perl Bordmittel gibt. Ein grundlegendes Verständnis für das Vorgehen werden wir mit unserem einfachen Beispiel aber auch erreichen.

In purem AppleScript würde eine einfache Umsetzung wie in Listing 1 aussehen.

Unser einfaches Beispiel-Template vor der Änderung zeigt Abb 2.

AppleScript Programme wie das obige lassen sich wahlweise mit dem Script Editor erstellen oder mit einem beliebigen Texteditor schreiben und dann über die Kommandozeile per /usr/bin/osascript ausführen. Hierbei erfolgt implizit eine Compilierung des Quelltextes sowie die transparente Generierung der zugrundeliegenden Apple Events und deren Versand sowie das Handling der zurück gelieferten Antworten.

Erste Schritte in Perl

Für viele Anwendungen völlig ausreichend ist der Ansatz, jegliche Datenvorbereitung (z.B. Datenbank-Abfragen, Dateikonvertierung etc.) in Perl durchzuführen, aus den

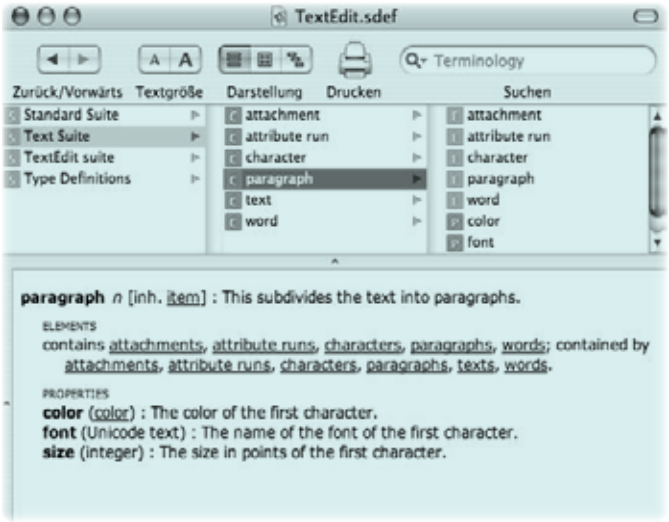

Abb 1: GUI Script Editor

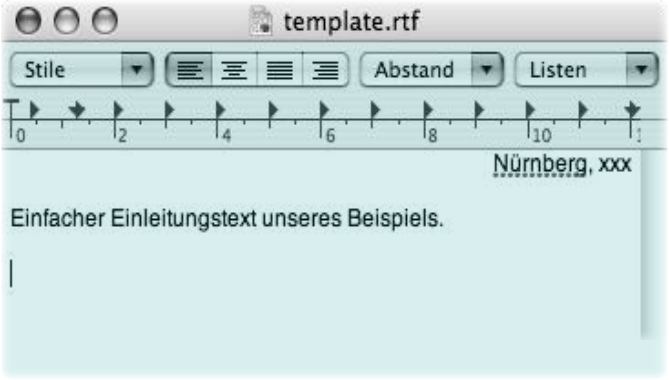

Abb 2: Beispiel Template

gewonnenen Daten ein AppleScript Programm zu erstellen und dieses dann auszuführen. Hierbei kann Perl seine Stärken ausspielen, und dennoch ist eine GUI Fernsteuerung der gewünschten Zielprogramme sehr leicht möglich.

Natürlich gäbe es auch die "Kneifzangenmethode", das erstellte AppleScript als Datei abzulegen und dann per `system()` Aufruf an das Programm `/usr/bin/osascript` zu übergeben, doch es gibt bereits Perl-Module, das die selbe Funktion effizienter mittels passender Systemaufrufe erfüllen: `MacPerl`, `Mac::AppleScript` oder `Mac::OSA::Simple`. Nachfolgend wird `MacPerl` verwendet. Selbstverständlich könnten die Inhalte der diversen "Strings" auch dynamisch generiert werden (siehe Listing 2).

Diese Methode, `AppleScript` mittels `MacPerl` auszuführen ist relativ praktisch. Einerseits sind die `AppleScript` Anweisungen leicht zu lesen und dank Perl's Stringmanipulations-Befehle auch sehr leicht zu erweitern. Die Benutzung von Interpolationen erlaubt das schnelle Anpassen ganzer Template-Bausteine.

Leider hat diese Methode auch seine Nachteile. Die Performanz lässt stark zu wünschen übrig, da jedes einzelne Script zum Zeitpunkt des Aufrufs compiliert und dann erst ausgeführt werden kann. Viele kleine Scripts können schnell unangenehme Zusatzlaufzeiten mit sich bringen. Mein Rechner benötigt ca. 1,2 Sekunden für diese Aufgabe.

Eine Perl-nahe Methode

Das Perl-Modul `Mac::Glue` von Chris Nandor verwendet einen komplett anderen Ansatz, Mac Programme fernzusteuern. In einem vorbereitenden Schritt wird das Voka-

```
1  tell application "TextEdit"
2  activate
3  close documents saving no
4
5  open POSIX file "/Users/wolfgang/Documents/artikel/template.rtf"
6
7  set second word of first paragraph of document 1 to "31.01.2007"
8  set character -1 of document 1 to "Weiterer Text" & return
9
10 close document 1 saving in POSIX file "/Users/wolfgang/xxx.rtf"
11 end tell
```

Listing 1

```
1  #!/usr/bin/perl
2  use strict;
3  use warnings;
4
5  use MacPerl;
6
7  my $script = << _ EOSCRIPT _ ;
8  tell application "TextEdit"
9  activate
10 close documents saving no
11
12 open POSIX file "/Users/wolfgang/Documents/artikel/template.rtf"
13
14 set second word of first paragraph of document 1 to "31.01.2007"
15 set character -1 of document 1 to "Weiterer Text" & return
16
17 close document 1 saving in POSIX file "/Users/wolfgang/xxx.rtf"
18 end tell
19 _ EOSCRIPT _
20
21 my $result = MacPerl::DoAppleScript($script);
22
23 print $result;
```

Listing 2

bular eines zu steuernden Programmes abgerufen und in später wieder verarbeitbarer Form abgespeichert.

```
$ gluemac /Applications/TextEdit.app
```

als Benutzer root aufgerufen extrahiert das Vokabular.

```
$ gluedoc 'TextEdit'
```

erstellt aus dem Extrakt eine man-page, deren Lesbarkeit allerdings einige Eingewöhnung erfordert. Hat man diese Hürde allerdings erst einmal erklommen, kann es losgehen.
Obiges Beispiel-Programm würde dann so aussehen, wie in Listing 3 dargestellt.

Abgesehen von der deutlich besseren Laufzeit mit nur 0,6 Sekunden muss man bei der Benutzung von `Mac::Glue` allerdings einige Typ-Umwandlungen vornehmen, da AppleScript streng typisiert ist, Perl jedoch nicht. Diese "Eigenschaft" von `Mac::Glue` hemmt anfangs etwas die Programmierung, ist aber schnell in Griff zu bekommen.

(abschreckende) Alternative

Theoretisch gäbe es auch die Möglichkeit, direkt Apple Events zu generieren und zu versenden. Dieses Verfahren hätte zwar den Vorteil, dass alle Datentypen, die in einem Apple Event verwendet werden können, direkt spezifiziert werden können. Allerdings muss mann dann die in einem Apple Event verwendeten Codes (historisch gewachsen als 4 Buchstaben dargestellt, aber intern als 32-Bit Zahl repräsentiert), die stellvertretend für die Worte des jeweiligen Vokabulars verwendet werden, selbst einsetzen.

Aus einer einfachen Zeile in AppleScript wie:

```
get document 1 of application "TextEdit"
```

bzw. in Mac::Glue:

```
my $doc = $textedit->obj(document => 1);
```

müsste dann werden:

```
my $doc = do_event(qw(core getd ttxt),
qq['----':obj {want:type(docu),
   form:indx, seld:long(1), ].
qq[from:'null'()}]);
```

Unleserlicher Code, die Verwendung zahlreicher 4-buchstabiger Kürzel und die Konstruktion seltsamer Objekt-Notationen wäre die direkte Folge, wenngleich bei optimaler Performance. Wer sich mit dieser Methode anfreunden möchte, kann gerne die Dokumentation zu

```
1  #!/usr/bin/perl
2  use strict;
3  use warnings;
4
5  use Mac::Glue ':all';
6
7  my $textedit = new Mac::Glue 'TextEdit';
8
9  $textedit->activate();
10 $_->close(saving => enum('no')) for ($textedit->obj('documents'));
11
12 $textedit->open("/Users/wolfgang/Documents/artikel/template.rtf");
13 my $doc = $textedit->obj(document => 1);
14
15 my $date_word = $doc->obj(word => 2, paragraph => 1);
16 $date_word->set(to => '31.01.2007');
17
18 my $last_pos = $doc->obj(character => -1);
19 $last_pos->set(to => 'Und noch ein weiterer Text zum Schluss');
20
21 $doc->close(saving_in => param_type(typeChar,
22 Mac::Files::_Unix2Mac('/Users/wolfgang/xxx.rtf')) );
```

Listing 3

`Mac::AppleEvents` oder `MacAppleEvents::Simple` konsultieren. Ich rate allerdings aufgrund schlechter Wartbarkeit solcher Codes stark davon ab. Der Laufzeitgewinn gegenüber `Mac::Glue` ist auch marginal.

Spezielle Anwendungen

Für eine ganze Reihe von Programmen existieren bereits Module, die meist nach der zu erst beschriebenen Methode per generiertem AppleScript eine Fernsteuerfunktion ausführen.

`Mac::iPhoto`
`Mac::iPhoto::Shell`
`Mac::iTunes`
`Mac::iTunes::AppleScript`
`Mac::EyeTV`
`Mac::Growl`

Fazit

Der Einsatz von Spezialprogrammen zur Erzeugung von Dokumenten beliebiger Art ist auch durch Perl ohne neu zu entwickelnde Bibliotheken möglich. Da fast alle unter der Apple Macintosh Plattform laufenden Programme AppleScript-fähig sind, lassen sich alle manuell ausführbaren Tätigkeiten damit leicht automatisieren.

Gutes Laufzeitverhalten ist mittels `Mac::Glue` zu erreichen, leichtere und schnellere Entwicklung kann durch die Konstruktion einzelner AppleScript Fragmente und der Benutzung von `MacPerl` erreicht werden. Hinzu kommt, dass in einschlägigen Foren fast ausschließlich Programmcode in Form von AppleScript ausgetauscht wird, was die Lösung typischer Probleme stark erleichtert.

Wolfgang Kinkeldei

TPF-Ticker

Perl6 und Parrot

Die Wikis zu Perl6 und Parrot sind im Juni zur Perl-Foundation gezogen. Damit bekommen die beiden Wikis vielleicht auch etwas mehr Aufmerksamkeit.

Desweiteren sind die “Essentials“ (aus dem Buch “Perl 6 and Parrot Essentials“) als Projektdokumentation in das SVN-Repository auf perl.org aufgenommen worden. Da das Buch schon etwas älter ist, sind manche Sachen schon etwas veraltet, aber die Essentials bilden eine solide Basis, auf der die Dokumente aktualisiert werden können.

Parrot ist im Juni in der Version 0.4.13 - dem “Clifton-Release“ veröffentlicht worden. Mit dem neuen Release wurden die Sprachunterstützung für Perl, PHP, Python und einige andere verbessert. Im Kern wurde der Code aufgeräumt und ein paar Speicherleaks beseitigt. Es gibt noch einige weitere Verbesserungen, die auf der Parrot-Seite nachzulesen sind.

TPF-Ticker

Perl 5.9.5

Bis zu Perl 5.10 kann e nicht mehr lange dauern. Am 07.07.07 wurde Perl 5.9.5 auf CPAN hochgeladen. Diese Version gilt als Beta von Perl 5.10, das heißt es werden keine Features mehr hinzugefügt; es werden nur noch kleinere Änderungen und Bugfixes akzeptiert.

INTERVIEW

Joergen Lang über Perldoc2

FM: Hallo Joergen, wer im Usenet unterwegs ist, kennt Deinen Namen wahrscheinlich schon, aber bitte stelle Dich kurz für den Rest der Welt vor.

JWL: Mein Name ist Joergen Lang. Ich arbeite seit 1996 als Webdesigner und -programmierer und übersetze seit 1998 Bücher für den O Reilly-Verlag. Das kam eigentlich recht überraschend. Ich hatte mich im Usenet ein wenig in den Perl-Gruppen engagiert und Nathan Torkingtons mini-FAQ für die deutschsprachige Perl-Community übersetzt und erweitert. Außerdem habe ich eine Liste deutschsprachiger Bücher zu Perl zusammengestellt (hier werden übrigens noch Maintainer gesucht!) und veröffentlicht. Eines Tages kam dann eine Mail von Michael Gerth, der damals noch als Lektor für die O Reillys tätig war. Er fragte, ob ich eigentlich Rezensionsexemplare bräuchte - klar brauchte ich. ;)

Aber eigentlich brauchte ich einen Job und so fragte ich, ob noch Übersetzer gesucht würden - es wurden, und so kam ich zu meinem ersten Übersetzerjob (CGI-Programmierung mit Perl). Im Laufe der Zeit ist die Zahl der von mir übersetzten Bücher auf über 14 gewachsen. Darunter das Lama-Buch, Perl Hacks, die CSS-Reihe, um nur ein paar zu nennen.

FM: Im September/Oktober 2006 hast Du ein ambitioniertes Projekt angefangen: perldoc2. Kannst Du in wenigen Worten erklären, was es mit diesem Projekt auf sich hat?

JWL: Letztes Jahr wurde ich von Tim O Reilly eingeladen, als Gast an der in Brüssel stattfindenden Euro OSCON und dem Euro Foo-Camp teilzunehmen. Also setzte ich mich in mein Auto und fuhr hin. Es war spannend und sehr bereichernd, soviele interessante und inspirierende Persönlichkeiten an einem Ort versammelt zu sehen. Noch besser war allerdings die Möglichkeit, diese Leute einfach direkt ansprechen zu können.

So unterhielt ich mich beim Mittagessen zum Beispiel mit Damian Conway. Ich erzählte ihm von meiner Idee, eine vollständige deutsche Übersetzung der Perl-Dokumentation anzufertigen. Er befürwortete die Idee, meinte jedoch, es sei sinnvoll, hierfür eine Plattform zu haben, auf der Übersetzungen auch für andere natürliche Sprachen angefertigt werden könnten. Und das war die Geburtsstunde von perldoc2. Ich hatte auf dem Rückweg reichlich Zeit (eigentlich waren es drei Rückfahrten aber das ist eine andere Geschichte), mir ein erstes Konzept zu überlegen.

Bei perldoc2 geht es also darum, Leuten die Übersetzung der Perl-Docs zu erleichtern und dafür zu sorgen, dass ein Dokument wirklich nur einmal übersetzt werden muss. Ein Übersetzer soll nicht wissen, was der Befehl $ svn log -r 8 -v macht, um zu übersetzen. Für ihn ist es wichtiger, dass Hash nur noch sehr selten als assoziatives Array bezeichnet wird.

Um diese Hürde zu beseitigen und gleichzeitig eine Art Zentrallager für die Übersetzungen zu schaffen, brauchen wir ein robustes System, das auch in der Lage ist, große Daten- und Dokumentenmengen zu verwalten und zu verarbeiten.

Das momentane Konzept sieht daher eine Kombination aus SVN und Datenbank (Postgre SQL) vor, die über eine einfach zu benutzende Web-Schnittstelle zugänglich sind. Die einzelnen Absätze der Dokumente werden ins gettext-Format übertragen (in Zukunft möglicherweise auch XLIFF). Diese .po-Strings können dann entweder

direkt übers Web oder lokal mit Tools wie poEdit übersetzt werden.

FM: Für was wird diese Webbasierte Plattform benötigt? Welche Funktionalitäten bietet sie?

JWL: In der endgültigen Version soll die Plattform ein ausgewachsenes System zur Verwaltung von Übersetzungen (TMS) sein. Benutzer können Dokumente online oder lokal übersetzen. Durch ein Peer-Review-System soll sichergestellt werden, dass die Richtigkeit und Qualität der Dokumente gewährleistet ist.

Im Hintergrund verwaltet das System den Status die einzelnen Dokumente, sowie deren Status (übersetzt, ledig, ...) und kann über verschiedene statistische Funktionen darüber Auskunft geben.

Zu Beginn habe ich überlegt, einfach das Ubuntu-Rosetta-Projekt zu nutzen. Das war aber aus verschiedenen Gründen nicht möglich: Ubuntu ist nicht FOSS; über eine einmal eingecheckte Übersetzung hat man erst wieder wirkliche redaktionelle Kontrolle, wenn sie fertig gestellt wird. Das wirklich große Problem mit Rosetta ist aber die Datenmenge. Sämtliche Dokumente der Kern-Dokumentation würden allein die Hälfte des gesamten Rosetta-Projekts ausmachen.

Eine weitere Möglichkeit wäre, das schon relativ weit entwickelte Pootle-Projekt einzusetzen. Dem fehlt allerdings die Datenbankunterstützung, was es gerade bei großen Datenmengen ziemlich langsam macht. Dazu kommt, dass es in Python geschrieben ist, was eine Erweiterung für Perl-Programmierer eher knifflig macht.
Einfach gesagt geht es also um die Schaffung einer webgestützten Plattform für die Übersetzung der Perl-Dokumentation (und möglicherweise auch anderer). Mit Hilfe der Hamburger Perl Mongers konnte eine Rohversion der Plattform erstellt. Mit Hilfe von Catalyst hatten wir schnell einen Rohbau fertig.

FM: In letzter Zeit ist es ein wenig ruhiger um das Projekt geworden. Woran liegt es?

JWL: Im November habe ich mich bei der Perl Foundation URL um einen Grant beworben, um mich für einige Zeit ganz der Arbeit am Projekt widmen zu können. Leider wurde er nicht bewilligt. So musste ich zwangsläufig erst einmal etwas Geld verdienen und kurz mal umziehen. Außerdem arbeite ich ja auch noch als Musiker und im Sommer ist da einiges los.

Leider ist darüber das Projekt ein wenig eingeschlafen. Das muss aber nicht so bleiben.

FM: Wie kann der Otto-Normal-Perl-Programmierer helfen, damit die Ziele des Projekts erreicht werden?

JWL: Es braucht eigentlich keine besonderen Perl-Kenntnisse, obwohl das Wissen um Catalyst sicher eine große Hilfe ist. Der Quellcode kann bei Source Forge heruntergeladen werden URL. Jeder Programmierer ist herzlich willkommen, sich zu beteiligen. Wer ernsthaft interessiert ist, sollte sich möglichst vorher mit den Specs auseinander setzen URL und sich dann mit seiner Source Forge-Kennung an mich wenden.

Für die Kommunikation gibt es eine Mailingliste URL und einen eigenen IRC-Kanal URL.

Interessanterweise haben sich viele Leute gemeldet, die übersetzen möchten. Dafür sollte aber erst einmal die Plattform stehen. Wer dennoch gleich übersetzen möchten, kann sich ebenfalls von Source Forge die nötigen Dateien herunterladen.

FM: Es ist ein ambitioniertes Ziel, die Perl-Dokumentation in verschiedene Sprachen zu übersetzen und es kommen ja auch neue Perl-Versionen heraus. Wie willst Du erreichen, dass die Dokumente in perldoc2 immer aktuell sind?

JWL: Im Moment unterstützen wir drei Perl-Versionen, von denen die aktuelle 5.8.8 ist. Daneben gibt es einen Zweig für die aktuelle Entwicklerversion und einen für 5.6.*, um auch bereits vor längerer Zeit übersetzte Dokumente nicht komplett neu übersetzen, sondern nur anpassen zu müssen.

Als Referenzversion dient uns die jeweils unter perldoc.perl.org bereitgestellte Version der Dokumentation. Ziel ist es, die verschiedenen Sprachen schließlich in Form von Subdomains einzubinden.

Während der Vorbereitungen habe ich festgestellt, dass es eine Rangfolge der Dokumente gibt. Einige werden wesentlich häufiger angefordert als andere. Als Quelle diente mir hier das Serverlog von perldoc.perl.org (Details finden sich auf meiner Projektseite.). Über die Hälfte aller Anfragen bezog sich auf perlfunc.pod, gefolgt von perl.pod, perlintro.pod, perlvar.pod und perlre.pod.

Ich habe ausgerechnet, dass die Übersetzung der gesamten Kern-Dokumentation (exklusive der Core-Module) ca. 1-2 Mannjahre dauern würde. Je mehr Leute mithelfen, desto schneller kann eine komplette Übersetzung vorliegen. Diese müsste dann bei neuen Perl-Versionen nur noch angepasst und erweitert werden. Parallel dazu können durch die Unterstützung des Entwickler-Zweiges aber auch schon Übersetzungen kommender Versionen bearbeitet werden.

Vermutlich wird es keine tagesaktuelle Version geben - sie wird der englischen Version immer ein wenig hinterher hinken. Aber mit ein wenig Glück können wir, nachdem die Kernübersetzung einmal steht, recht zeitnah auch eine fremdsprachige Übersetzung anbieten.

FM: In welche Sprachen wurden denn schon Dokumente übersetzt?

JWL: Ich habe insgesamt zehn Projekte gefunden, die sich mit der Übersetzung der Dokumentation in andere Sprachen befassen. Am weitesten gediehen sind interessanterweise die französische und die brasilianisch-portugische Fassung.

Obwohl es im deutschsprachigen Raum (nach den englischsprachigen Ländern) die zweithöchste Dichte an Perl-Nutzergruppen gibt (Perl Mongers), ist nur ein geringer Teil der Dokumentation tatsächlich ins Deutsche übersetzt. Dies mag daran liegen, dass viele Leute die Docs einfach im englischen Original lesen. Die Diskussionen und Fragen in den Perl-Nutzergruppen lassen jedoch durchaus darauf schließen, dass es Bedarf für eine deutsche Übersetzung gibt.

FM: Danke Joergen für die ausführlichen Antworten. Auch für die deutsche Perl-Community wäre es zu wünschen, dass perldoc2 ein Erfolg wird.

JWL: An dieser Stelle möchte ich mich noch einmal bei all denen bedanken, die sich bisher für das Projekt eingesetzt haben. Insbesondere sind dies Damian Conway, der den Stein des Anstosses gab, nekral, der sich speziell um die französiche Übersetzung gekümmert hat und das wunderbare Werkzeug po4a mit entwickelt hat, außerdem die Hamburger Perl Mongers, die bei der Programmierung der Basisversion mehr als nur geholfen haben. Danke auch alle anderen, die sich beteiligt haben Ihr wisst wer Ihr seid!

Hinweis:
Wer sich am Projekt perldoc2 beteiligen möchte, oder Fragen dazu hat, kann sich unter der Adresse `perldoc2@joergen-lang.com` mit Joergen in Verbindung setzen.

Net::LDAP

Ziele dieses Workshops

LDAP wird immer häufiger verwendet - angefangen von E-Mail-Adressbüchern über Benutzerverwaltungen und -authentifizierungen bis hin zu Datensynchronisationen für Metadirectories und Identity Management. Ich habe diese kurze Einführung in LDAP und *Net::LDAP* geschrieben, weil LDAP immer wichtiger wird und in die Arbeitstasche jedes Perl-Programmierers gehören sollte.

Für die Einführung verwende ich den freien Directory Server OpenLDAP, der zum Lernen sehr gut geeignet ist. Er ist auch für Windows verfügbar (z.B. bei cygwin), und die Konfigurationsanleitung bezieht sich auf die cygwin-Version (kann aber mit angepassten Pfaden auch unter Linux usw. verwendet werden). Die Codes laufen in der Regel unter allen Directory Servern, wobei jedoch nicht jeder Directory Server alle Features implementiert.

OpenLDAP konfigurieren

slapd.conf editieren

Für die Beispielcodes reicht es, wenn man in der Konfigurationsdatei /etc/openldap/slapd.conf nur die folgenden Werte verändert/hinzufügt. Dies ist jedoch keine sichere oder vollständige Konfiguration, sondern nur etwas zum Spielen. Wenn OpenLDAP als Benutzerrepository für das Betriebssystem dient (z.B. manchmal unter Linux), bitte darauf achten, dass man sich selbst nicht vom Betriebssystem "aussperrt", wenn man die Konfiguration ändert oder Objekte löscht.

```
include /etc/openldap/schema/core.schema
include /etc/openldap/schema/cosine.schema
include \
    /etc/openldap/schema/inetorgperson.schema

suffix c=de
rootdn cn=root,c=de
rootpw secret
```

Ich verwende in den folgenden Beispielen als Objektklasse für Personenobjekte die Objektklasse inetOrgPerson. Da diese Objektklasse unter Umständen nicht standardmäßig aktiviert ist, müssen eventuell die beiden zusätzlichen Schemadateien cosine.schema und inetorgperson.schema eingebunden werden (sind bei OpenLDAP dabei).

Wurzelelemente erstellen und importieren

Die folgenden Zeilen in eine Datei schreiben (ich habe den Namen ./01_ldap_root.ldif gewählt):

```
dn: c=de
c: de
objectClass: country
description: LDAP-Root
```

```
dn: o=Meine Firma,c=de
o: Meine Firma
objectClass: organization
description: Root meiner Firma
```

und dann mit dem folgenden Befehl importieren:

```
$ /usr/sbin/slapadd.exe -v \
      -l ./01_ldap_root.ldif
added: "c=de" (00000001)
added: "o=Meine Firma,c=de" (00000002)
```

Konfiguration testen und OpenLDAP starten

Dann die Konfiguration folgendermaßen testen:

```
./slaptest
onfig file testing succeeded
```

Und OpenLDAP starten:

```
/usr/sbin/slapd
```

(vielleicht mit ps überprüfen, ob es einen Prozess namens slapd gibt).

LDAP-Browser

Damit man einfach überprüfen kann, ob alles so funktioniert wie geplant, empfehle ich die Installation eines LDAP-Browsers. Vielen Linux-Distributionen bieten schon LDAP-Browser an. Es gibt auch freie LDAP-Browser, z.B.:

Der *LDAP-Browser/Editor* benötigt eine installierte Java Runtime und bietet (mit Ausnahme des Anlegens neuer Objekte und Zugriff auf das Schema) so ziemlich alles an Funktionen, die man so benötigt. Und für den Rest hat man ja Perl...

Der *Softerra LDAP Browser* ist in der kommerziellen Version ein sehr mächtiger LDAP-Browser, in der freien Version jedoch leider nur als Anzeigetool verwendbar.

Begriffsdefinitionen

Ein Directory ist eine Art objektorientierte Datenbank, in der Daten in baumartigen Strukturen gespeichert werden. Häufig wird ein Directory verwendet, um Benutzer in Unternehmenshierarchien zu speichern (z.B. ActiveDirectory, NDS, ...) oder Adress- oder Telefonbuchdaten zu speichern. Als Abfragesprache dient LDAP. Ein Datensatz wird Objekt genannt. Was an Informationen in einem Objekt gespeichert werden kann, wird von der Objektklasse bestimmt. Wo dieses Objekt gespeichert wird, wird vom distinguishedName (kurz: DN) bestimmt, der aus den "Relative DistinguishedNames" (kurz: RDN) zusammengesetzt wird.

```
dn: o=Meine Firma,c=de
o: Meine Firma
objectClass: organization
description: Root meiner Firma
```

In diesem Beispiel ist `o=Meine Firma` der RDN. Dieses Objekt befindet sich unterhalb des Objektes mit dem RDN `c=de`, also lautet der DN `o=Meine Firma,c=de`. Dieses Objekt ist von der Objektklasse organization, die aussagt, dass für eine Organization das Attribut o als Pflichtattribut nötig ist, und einige weitere Attribute vorkommen können (z.B. description, telephoneNumber, ... Siehe auch /etc/openldap/schema/core.schema).

LDAP ist die Sprache und das Protokoll, mit der Objekte abgefragt oder verändert werden, und spielt dieselbe Rolle wie SQL bei Datenbanken oder ein Editor im Dateisystem.

Object ist ein Datensatz (Entspricht einem Datensatz einer Datenbank oder einer Datei)

Attribute ist ein Feld, das einen Wert (Singlevalued) oder mehrere Werte (Multivalued) enthalten kann (entspricht einem Feld in einer Datenbank oder einer Zeile einer Datei). Dabei gibt es Mandatory Attributes (=Pflichtattribute) und Optional Attributes(=können vorkommen oder fehlen) (in der Datenbank: NULL oder NOT NULL). Weiterhin gibt es noch Operational Attributes, die automatische Statusinformationen enthalten (entspricht z.B. Erstellungs-/Änderungsdatum oder Eigentümer einer Datei).

Objectclass ist die Definition, welche Werte ein Object enthalten darf bzw. muss (entspricht in etwa einer Datenbank-Tabelle). Eine ObjectClass kann von anderen, übergeordneten ObjectClasses Informationen vererben (z.B. welche Attribute vorkommen dürfen/müssen).

Relative DistinguishedName (kurz: RDN) ist ein in einer Ebene eindeutiges Attribut (entspricht dem Primärschlüssel oder dem Dateinamen ohne Pfadangabe).

DistinguishedName (kurz: DN) ist ein Attribut, das bestimmt, wo im Baum ein Objekt gespeichert ist (entpricht Datenbank.Primärschlüssel oder einem Dateinamen mit absoluter Pfadangabe). Der DN wird aus dem RDN des Objektes und den RDNs aller übergeordneten Objekte zusammengesetzt (durch Komma getrennt).

Schema enthält die Information, welche ObjectClasses und Attributes (und weiteres) verfügbar sind, und wie sie kombiniert werden dürfen (entspricht der Datenbank- und Tabellendefinition).

Abstract ObjectClass ist die ObjektClass, von der alle anderen ObjectClasses erben (wie UNIVERSAL in Perl-OOP). Sie hat den Namen `top`.

Structural ObjectClass ist die Haupt-ObjectClass eines Objects. Jedes Object muss genau eine structural ObjectClass haben (entspricht einer *normalen Tabelle*, in der ein Datensatz gespeichert wird).

Auxiliary ObjectClass ist eine Hilfsobjektklasse, die meist zusätzliche Attribute bereitstellt (entspricht einem LEFT JOIN auf eine externe Tabelle) oder auch gelegentlich für die Rechtevergabe verwendet wird.

Suffix ist das oberste Objekt, die Wurzel des Baums (entspricht einer Datenbank, einem Laufwerksbuchstaben). In den Beispielen habe ich als Suffix `c=de` gewählt. Häufig wird das Suffix jedoch zweistufig gewählt, z.B. `o=Firma Fabiani,c=de` oder `dc=fabiani,dc=net`.

Einführung in Net::LDAP

Für die folgenden Codes muss OpenLDAP laufen, Perl installiert sein und die Modulgruppe *Net::LDAP* installiert sein (wird auf CPAN oder Activestate auch *perl-ldap* genannt). Es gibt auch noch die anscheinend seltener verwendete Modulgruppe Mozilla::LDAP (auf CPAN **perLDAP** genannt). Es gibt noch weitere Module, die jedoch häufig nur mit Einschränkungen verwendet werden können oder einen geringeren Funktionsumfang bieten.

Connect und Bind (Verbindung herstellen)

Dass zu Beginn des Codes (in Listing 1) noch Shebang, `use strict;` und `use warnings;` gehört, versteht sich hoffentlich von selbst ;-)

Fehlerbehandlung: bei fast jeder Aktion wird ein Objekt der Klasse Net::LDAP::Message zurückgegeben, dessen Methode `code` einen wahren Wert zurückgibt, wenn ein Fehler aufgetreten ist. Um die Codes für diesen Workshop kurz halten zu können, habe ich die Ausgabe einer Fehlermeldung in die Subroutine `PrintLdapError` verbannt.

Sowohl beim `new` als auch beim `bind` kann man noch weitere hilfreiche Optionen mitgeben, siehe perldoc Net::LDAP.

```
1  use Net::LDAP;
2
3  # einige Hilfsfunktionen für die Fehlerbehandlung importieren
4  use Net::LDAP::Util qw( ldap_error_text ldap_error_name ldap_error_desc );
5
6  my( $server, $bindDn, $password ) = ( '127.0.0.1', 'cn=root,c=de', 'secret' );
7
8  # Verbindung aufbauen: connect
9  my $ldap = Net::LDAP->new( $server )
10     or die "Error in connecting to '$server': $@\n";
11
12 # Anmelden: simple bind
13 my $rc = $ldap->bind( $bindDn, password => $password );
14 $rc->code and PrintLdapError( $ldap, $rc, "Error in bind as '$bindDn'" ); # error handling
15
16 # ... weitere Aktionen hier
17
18 $ldap->unbind; # Verbindung trennen: unbind
19
20 sub PrintLdapError { # Fehlerbehandlung
21     my( $ldap, $rc, $msg ) = @_;
22
23     my $name  = ldap_error_name( $rc );
24     my $text  = ldap_error_text( $rc );
25     my $descr = ldap_error_desc( $rc );
26
27     die join "\n", $msg, "Name: $name", "Descr: $descr", "Text: $text", '';
28 } # PrintLdapError
```

Listing 1

```
1  use Net::LDAP::Entry;
2
3  # Objekt lokal anlegen
4  my $entry = Net::LDAP::Entry->new();
5  $entry->dn( "cn=1932,o=Meine Firma,c=de" );
6  $entry->add( cn              => '1932',          # commonName
7               objectClass     => 'inetOrgPerson',
8               sn              => 'Fabiani',       # surname
9               givenName       => 'Martin',
10              l               => 'Frankfurt',     # location
11              st              => 'Hessen',        # state
12              employeeNumber  => '1932',
13              displayName     => 'Fabiani, Martin',
14            );
15
16 $entry->dump; # debug output
17
18 # und dieses Objekt im Directory speichern:
19 my $success = $entry->update( $ldap );
20 $success->code and &PrintLdapError( $ldap, $success, 'Error in adding object' );
```

Listing 2

```
1  perl -e 'for (1..2000) { print "dn: cn=cn $ _ ,o=Meine Firma,c=de\ncn:
2  cn $ _ \nobjectClass: inetOrgPerson\nsn: Surname $ _ \ngivenName:
3  givenName $ _ \nuid: uid$ _ \n\n"}' > bulkUsers.ldif
```

Listing 3

Neue Objekte erstellen

LDAP-Objects werden durch die Klasse `Net::LDAP::Entry` repräsentiert. Wenn wir einen neuen Benutzer anlegen wollen, so könnte das wie in Listing 2 dargestellt geschehen.

Für die folgenden Beispiele ist es hilfreich, wenn man mehrere Objects im Directory findet. Dafür könnte man sich mit dem folgenden Perl-Einzeiler (alles in eine Zeile) eine Datei namens `bulkUsers.ldif` erstellen (siehe Listing 3) und diese Datei dann mit `slapadd -v -l ./bulkUsers.ldif` importieren.

Nach LDAP-Objects suchen (Beispiel siehe Listing 4)

base: ab welchem Knoten soll gesucht werden (Suchbasis)

scope: wie weit soll nach unten gesucht werden? ? (sub => alles, one => eine Ebene, base => nur das Wurzelobjekt selbst)

filter: welche Objekte interessieren uns; für diese Suche habe ich alle Objekte mit Objektklasse `inetOrgPerson` ausgewählt. Diese Filter können jedoch beliebig komplex werden, indem man mehrere Kriterien über eine polnische Notation mit & (=und) oder | (=oder) oder ! (=nicht) verknüpft und mit Wildcards arbeitet (siehe Listing 5) und weitere (siehe perldoc Net::LDAP::Search)

Ich verwende aus Effizienzgründen bei dieser Art der Suche meist shift_entry, habe aber noch nicht gebenchmarkt, weil bei der LDAP-Programmierung meist der Directory Server oder das Netzwerk das Nadelöhr ist.

Wenn man ein solches Net::LDAP::Entry – Objekt hat (heißt bei uns $entry), kann man verschiedene Sachen damit anstellen, siehe perldoc Net::LDAP::Entry. Hier ein paar der wichtigsten Methoden:

dump: gibt das Objekt auf STDOUT aus (gut zum Debuggen)

dn: gibt den distinguishedName des Objektes zurück oder setzt ihn.

attributes: gibt alle Attributnamen des Objekts als Liste zurück

get_value($attr): gibt den oder die Werte für $attr zurück, je nachdem, in welchem Kontext man es aufruft (String/Array-Ref für skalaren Kontext, sonst Array).

```
1  my $searchResult = $ldap->search(
2              base   => 'o=Meine Firma,c=de',
3              filter => '(objectClass=inetOrgPerson)',
4              scope  => 'sub',
5                                );
6  &PrintLdapError( $ldap, $searchResult, "Error in search" )
7     if $searchResult->code;
8  print $searchResult->count, " entries found\n";
```

Listing 4

Auf diese Art (siehe Listing 5) kann man auch sehr große Datenmengen komfortabel ausgeben lassen.

Achtung: wenn man für search den Parameter attrs => \@arrayRef verwendet, werden nur die Attribute zurückgegeben, die in diesem @arrayRef stehen und keine weiteren (der dn wird hingegen immer zurückgegeben).

Achtung: bei dieser Art der Suche kann es Probleme geben; siehe das Kapitel Paging.

Suche mit Callback

Bei der Suche kann man auch eine Callback-Funktion (siehe Listing 7) mitgeben, die für jedes gefundene Objekt ausgeführt wird. Die Callback-Suche (siehe Listing 6) ist recht gut geeignet, wenn man mit einigen der gefundenen Objekte irgendwelche Operationen ausführen will. Wenn man jedoch irgendwas in Variablen speichern will, finde ich sie nicht so schön, weil man dann auf "globale" Variablen oder Closures ausweichen muss. Dann bevorzuge ich die Verarbeitung in einer Schleife, z.B. mit `while( my $entry = $searchResult->shift_entry )` oder `pop_entry`.
Diese Callback-Funktion, die dann für jeden gefundenen Entry ausgeführt wird, bekommt zwei Parameter: Das Message-Objekt und das gefundene Net::LDAP::Entry-Objekt.

Mit dem Entry-Objekt kann man beliebige Funktionen ausführen.

Achtung: bei dieser Art der Suche kann es Probleme geben; siehe das Kapitel Paging.

Paging

Die meisten Server verwenden zum Suchen ein SizeLimit, d.h. sie geben pro Suchvorgang nur eine maximale Anzahl von gefundenen Objects zurück, meist irgendwas zwischen 500 und 2000. Manchmal ist dieses SizeLimit sogar für anonyme, authentifizierte Benutzer oder Administratoren unterschiedlich. Häufig kann man dieses SizeLimit in der Serverkonfiguration vergrößern, manchmal ist dies aber auch nicht möglich (z.B. weil man nicht darf). Wenn ein Programm sich darauf verlässt, dass von einer Suche immer alle Einträge zurückgeliefert werden, die es gibt, und dann - weil sich die Datenmenge im Laufe der Zeit vergrößert - plötzlich auf ein solches SizeLimit stößt, kann man sich ganz schön Probleme einhandeln.

Dabei unterstützen fast alle Directory Server einen Mechanismus, den man Paging nennt. Man sucht zunächst nach einer bestimmten Anzahl von Objects kleiner oder gleich dem SizeLimit, wertet sie aus und feuert dann die nächste Suche

```
1 while( my $entry = $searchResult->shift_entry ) {
2    printf "dn: %s\n", $entry->dn; # dn ist kein normales Attribut
3    foreach my $attr ( $entry->attributes ) { # Liste der Attribute
4        foreach my $value ( $entry->get_value( $attr ) ) { # der Werte
5            print "$attr: $value\n";
6        } # foreach
7    } # foreach
8    print "-" x 60, "\n";
9 } # while
```

Listing 5

```
1 my $searchResult = $ldap->search(
2     base     => 'o=Meine Firma,c=de',
3     filter   => '(objectClass=inetOrgPerson)',
4     scope    => 'sub',
5     callback => \&PrintEntry,
6 );
7 $searchResult->code and
8   &PrintLdapError( $ldap, $searchResult, "Error in search" );
```

Listing 6

```
1 sub PrintEntry {
2   my( $rc, $entry ) = @_;
3
4   $rc->code and &PrintLdapError( undef, $rc, "PrintEntry" );
5   return unless $entry;
6
7   $entry->dump;
8 } # PrintEntry
```

Listing 7

ab, so lange, bis alle Ergebnisse gelesen wurden. Dies funktioniert ähnlich wie ein Pagerprogramm für Dateien (z.B. more, pg, less), wo auch nur immer eine Seite von Ergebnissen der Datei zurückgeliefert wird. Der englische Begriff für Seite lautet *page*, weshalb man diese Technik **Paging** nennt. Eine Suche, die Paging verwendet, könnte in etwa aussehen, wie in Listing 8 dargestellt.

Paging kann man auch mit der Callback-Suche verwenden, siehe perldoc `Net::LDAP::Control::Paged`.

Achtung: Ich empfehle, Paging für alle Suchen zu verwenden, die mehr als eine Handvoll Objects zurückliefern sollen.

```
1  # weitere Module und Konstanten laden
2  use Net::LDAP::Control::Paged;
3  use Net::LDAP::Constant qw( LDAP_CONTROL_PAGED );
4
5  # … Hierher den Code des connect und bind
6
7  # Paged Control erstellen, und das Client-Sizelimit auf 10 setzen.
8  # Für Produktion bevorzuge ich Werte zwischen 100 und 500.
9  my $pagedControl = Net::LDAP::Control::Paged->new( size => 10 );
10
11 # da die Suche eventuell mehrmals abgefeuert werden muss, habe ich
12 # die Suchargumente herausgezogen. Und damit der Server weiß, dass
13 # der Client Paging erwartet, wird bei der Suche die LDAP-Control
14 # Paged mitgegeben.
15 my @searchArgs = ( control => [ $pagedControl ],
16                    base    => 'o=Meine Firma,c=de',
17                    filter  => '(objectClass=inetOrgPerson)',
18                  );
19
20 my $cookie;
21 while(1) {
22
23     # damit man für die Entwicklung sieht, dass gepaged wird:
24     print "=" x 60, "\nNext page\n", "=" x 60, "\n";
25
26     # Suche abfeuern
27     my $searchResult = $ldap->search( @searchArgs );
28
29     $searchResult->code and last; # nur bei LDAP_SUCCESS weitersuchen
30
31     # Mach was mit den Suchergebnissen, z.B.
32     while( my $entry = $searchResult->shift_entry ) { $entry->dump; }
33
34     # Cookie aus der Paged Control herauslesen. Den Cookie benötigt der
35     # Server bei der nächsten Suche, damit er weiß, welche Ergebnisse
36     # er als nächstes zurückliefern soll. Wenn der Cookie leer ist,
37     # bedeutet dies, dass der Server alle Ergebnisse zurückgeliefert
38     # hat und der Suchvorgang beendet werden kann:
39     my( $response ) = $searchResult->control( LDAP_CONTROL_PAGED )
40         or last;
41     my $cookie = $response->cookie or last;
42
43     # Den cookie für die nächste Suche setzen
44     $pagedControl->cookie( $cookie );
45
46 } # while 1
47
48 # wenn der Cookie noch befüllt ist, gab es einen fehlerhaften Abbruch.
49 # Da zunächst dem Server mitteilen, dass wir mit der Suche fertig sind:
50 if( $cookie ) {
51     $pagedControl->cookie( $cookie );
52     $pagedControl->size( 0 );
53     $ldap->search( @searchArgs ); # cookie senden
54 } # if
```

Listing 8

Sortierte Suche ausführen

Bisher kamen die Suchergebnisse immer unsortiert oder in der Reihenfolge, wie wir sie eingefügt haben. Die meisten Directory Server unterstützen Suchergebnisse, die nach bestimmten Kriterien sortiert sind. Dies ist vor allem dann wichtig, wenn man nach großen Datenmengen sucht und dabei *Paging* verwendet.

Auf die Sortierung kann man (ähnlich wie beim Paging) über `Net::LDAP::Control::SortResult` zugreifen und dieser Control als String mitgeben, wonach man sortiert haben will. Z.B. bedeutet `"sn -employeeNumber"`, dass zuerst aufsteigend nach `sn` sortiert wird und danach absteigend (wegen dem vorangestellten Minus) nach `employeeNumber` (siehe Listing 9).

```
1  # weitere Module und Konstanten laden
2  use Net::LDAP::Control::Sort;
3  use Net::LDAP::Constant qw( LDAP_CONTROL_SORTRESULT );
4
5  # … Hierher den Code des connect und bind
6
7  # Sort-Control erzeugen
8  my $sortControl = Net::LDAP::Control::Sort->new( order => '-uid sn' );
9
10 # Suche abfeuern, und als Parameter control => [ $sortControl ] mitgeben:
11 my $searchResult = $ldap->search( control => [ $sortControl ],
12                                   base    => 'o=Meine Firma,c=de',
13                                   filter  => '(objectClass=inetOrgPerson)',
14                                   scope   => 'sub',
15                                 );
16 &PrintLdapError( $ldap, $searchResult, "Error in search" )
17    if $searchResult->code;
18
19 # ueberpruefen, ob die Sortierung funktionierte
20 my( $response ) = $searchResult->control( LDAP_CONTROL_SORTRESULT );
21 $response or die "Error: Server doesn't support sorting\n";
22
23 # versuche, festzustellen, bei welchem Attribut ein Fehler aufgetreten ist
24 if( $response->result ) {
25     my $attr = $response->attr;
26     my $mesage = 'Problem in sorting';
27     $mesage .= " by attribute $attr" if $attr;
28     &PrintLdapError( $ldap, $response->result, "Problem sorting" );
29 } # if
30
31 # und danach über $searchResult die Ergebnisse abholen…
```

Listing 9

Siehe auch perldoc Net::LDAP::Control::Sort und Net::LDAP::Control::SortResult..

Man kann dies auch einfach mit Paging kombinieren, indem man als Suchparameter einfach beide Controls setzt:

```
control => [ $sortControl, $pagedControl ],
```

Objekte löschen

Beim Löschen von Objekten gibt es zwei Wege: direkt über den DN (über Net::LDAP), oder über ein Net::LDAP::Entry-Objekt (z.B. als Suchergebnis).

Löschen über den DN

```
my $dn = "cn=cn 2000,o=Meine Firma,c=de";
 my $rc = $ldap->delete( $dn );
 $rc->code and &PrintLdapError( $ldap,
      $rc, "Error in delete: $dn" );
```

Anstelle der Zeichenkette `$dn` kann hier auch ein `Net::LDAP::Entry`–Objekt angegeben werden, siehe die Methode `delete` in `perldoc Net::LDAP`.

Löschen über ein Net::LDAP::Entry-Objekt

Hole ein `Net::LDAP::Entry`-Objekt in die Variable `$entry`, z.B. über eine Suche

```
$entry->delete;
 # Änderungen in den Server schreiben
 my $rc = $ldap->update( $entry );
 $rc->code and
     &PrintLdapError( $ldap, $rc,
     "Error in delete: " . $entry->dn );
```

Objekte verändern

Um ein Objekt zu ändern, holt man sich ein `Net::LDAP::Entry`–Objekt (z.B. durch eine Suche). Diese Art der Änderungen sind (genauso wie beim Löschen über das `Net::LDAP::Entry`-Objekt) vorerst lokal und werden erst beim nächsten `$entry->update($ldap)` an den Server gesendet. Leider kann man Fehler erst beim nächsten `update` sinnvoll abfragen.

```
# send the updates to the server
my $rcUpdate = $entry->update( $ldap );
$rcUpdate->code and
   &PrintLdapError( $ldap, $rcUpdate,
   "Error in  add attributes" )
```

Achtung: Man kann den RDN und den DN nicht sinnvoll mit diesen folgenden Methoden ändern. Siehe dafür das Kapitel "Objekte verschieben".

Man kann das Objekt auch, wenn man seinen DN kennt, direkt über das `Net::LDAP`–Objekt über die Methode `modify` ändern. Da der DN jedoch auch alle beliebigen UTF8-Zeichen enthalten kann (d.h. wenn das Directory für UTF8 konfiguriert ist), empfehle ich jedoch den aufgezeigten Weg über das `Net::LDAP::Entry`-Objekt.

Attribute hinzufügen

Dies hat die Form: `$entry->add(attribut=>[Wert]);` Der Wert kann bei multivalued Attributen auch eine Liste von Werten sein.

```
$entry->add(telephoneNumber=>
   ["+49 (123) 445-1111"]);
```

Attribute löschen

Dies hat die Form: `$entry-E->delete( attribut );`

```
$entry->delete( telephoneNumber );
```

Wenn man bei multivalued Attributen nicht alle Werte löschen will, kann man dies in Hash-Form angeben:

```
$entry->delete( telephoneNumber
    => [ "+49 (123) 445 - 1111" ] );
```

Achtung: $entry->delete() ohne Parameter will das Objekt löschen, nicht nur ein Attribut!

Attribute ersetzen

Dies hat die Form: `$entry->replace( attribut => neuer Wert );`

```
$entry->replace(telephoneNumber=>
    ["+49   (123) 999 - 1111"]);
```

Object umbenennen oder im Baum verschieben

Den RDN kann man meist nicht mit `replace` verändern, sondern über die Methode `newdn`:

```
my $rc = $ldap->moddn( $entry,
              newrdn    => "cn=neuerRDN",
              deleteoldrdn => 1,
             );
$rc->code and &PrintLdapError( $ldap, $rc,
   "Error in modRDN for " . $entry->dn );
```

Achtung: Das `deleteoldrdn => 1` nicht vergessen, sonst wird einfach ein zweiter RDN hinzugefügt (wenn möglich) und es findet keine Umbenennung statt.

Mit `moddn` kann man auch ein Object im Baum verschieben (den DN ändern):

```
my $rc = $ldap->moddn( $entry,
         newsuperior => "ou=Abteilung1,".
                        "o=Meine Firma,c=de",
                     );
$rc->code and &PrintLdapError( $ldap, $rc,
    "Error in modRDN for " . $entry->dn );
```

(funktioniert nur, wenn es die ou=Abteilung1 gibt (objectClass: organizationalUnit)

`moddn` führt die Änderung direkt aus, ein `$entry->update` kann man nicht verwenden.

Weitere Informationen

Dieser Workshop ist nur ein kleiner Ausschnitt aus Net::LDAP. Weiterführende Informationen findet man in den folgenden perldoc's:

Net::LDAP, Net::LDAP::Filter, Net::LDAP::Search, Net::LDAP::Entry, Net::LDAP::Examples, Net::LDAP::FAQ, Net::LDAP::Util, Net::LDAP::Constant, Net::LDAP::Message, Net::LDAP::Reference, Net::LDAP::Control, Net::LDAP::ControlPaged, Net::LDAP::ControlSort, Net::LDAP::ControlSortResult, Net::LDAP::Schema, Net::LDAP::LDIF, ...

Martin Fabiani

Heise Zeitschriften Verlag

Der Heise Zeitschriften Verlag steht für qualitativ hochwertigen Journalismus. Wir verlegen mit c't und iX zwei auflagenstarke Computertitel, das Technologiemagazin Technology Review sowie das Online-Magazin Telepolis. Unsere Website heise online für Computer- und Internet-Interessierte zählt zu den meistbesuchten deutschen Special-Interest-Angeboten.

Für die Weiterentwicklung von heise online suchen wir ab sofort einen

Senior Web-Entwickler (m/w)

vornehmlich für die Programmierung von datenbankbasierten Web-Anwendungen in Perl.

Ihre Aufgaben:

- Design, Implementierung, Test und Dokumentation neu zu erstellender Applikationen
- Wartung und kontinuierliche Verfeinerung bestehender Anwendungen
- Mitgestaltung der Software-Architektur von heise online sowie weiterer Web-Auftritte
- Direkte Unterstützung des Team-Leiters der Web-Programmierung

Ihre Qualifikation:

- Abgeschlossenes Studium der Informatik oder eines vergleichbaren Studiengangs bzw. vergleichbare Kenntnisse und Qualifikationen
- Mehrjährige Erfahrung in der Projektarbeit
- Ausgeprägte Teamfähigkeit
- Sehr gute Kenntnisse in der Programmiersprache Perl sowie Erfahrungen im Umgang mit dem Repository CPAN
- Sensibilität für die Sicherheitsaspekte von Web-Anwendungen (SQL-Injection, XSS)
- Grundlegende Kenntnisse von CSS und XHTML
- Know-how rund um Template-basierte Entwicklung
- Fundierte Linux-Kenntnisse, die über den Umgang mit der Kommandozeile hinausgehen
- Gute Englischkenntnisse (Wort und Schrift) sind von Vorteil

Wir bieten Ihnen ein inspirierendes Arbeitsumfeld und ein ehrgeiziges Team. Wenn es Sie reizt, am Erfolg von heise online mitzuwirken, freuen wir uns auf Ihre aussagekräftigen Bewerbungsunterlagen unter Angabe Ihrer Gehaltsvorstellung und des frühesten Eintrittstermins.

Für weitere Auskünfte steht Ihnen Herr Wolfgang Schemmel unter E-Mail ws@heise.de zur Verfügung.

Bewerbungen richten Sie bitte an:

Ein Unternehmen der Heise Medien Gruppe

Heise Zeitschriften Verlag GmbH & Co. KG
– Personalservice –
Frau Aurelia Quignon

Helstorfer Straße 7
30625 Hannover
Telefon: 05 11/53 52-263
Aurelia.Quignon@heise-medien.de

Perl parsen

Seit vielen Jahren hält sich der Spruch “Nothing can parse Perl but perl“. Damit soll ausgedrückt werden, dass nur der Interpreter tatsächlich weiß, wie Perl korrekt geparst wird. Perl-Skripte sind in der Tat kompliziert, da es auf den Kontext ankommt und die Syntax ziemlich frei ist - man denke nur daran, dass es möglich ist, Bedingungen nachgestellt zu formulieren oder einen Block zu machen.

Adam Kennedy ist angetreten, den Spruch zu widerlegen und hat das Modul `PPI` geschaffen. `PPI` steht für “Perl::Parse::Isolated“. Man muss jedoch beachten, was mit “parsen“ gemeint ist. Der Interpreter perl versteht unter “parsen“ etwas anderes als Menschen. perl geht das Perl-Skript durch, bei einem `use` springt er in das eingebundene Modul (oder Skript) und parst dieses. Wenn das Modul abgearbeitet ist, macht perl im eigentlichen Skript weiter. Das menschliche Verständnis von “parsen“ sieht in der Regel etwas anders aus: Das Dokument (Perl-Skript) soll in seine Einzelteile zerlegt werden - ohne dass zwischendurch andere Dokumente zerlegt werden.

`PPI` parst Perl-Programme nach dem menschlichen Verständnis von “parsen“. Es zerlegt die Programme in einzelne Tokens. Es versteht auch nur Perl-Code, der für Perl-Programmierer lesbar ist. perl kann Skripte, die mal durch `Acme::Buffy` oder andere Code-verändernde Module gelaufen sind, auch weiterhin starten. Selbst für sehr erfahrene Perl-Programmierer dürfte die Aneinanderreihung von “Buffy“s nicht sehr aussagekräftig sein. Und auch `PPI` kann damit nicht umgehen - auch wenn es im Inneren immer noch Perl-Code ist.

`PPI` ist eine Sammlung von Modulen, die jeweils ein Token darstellen und das Perl Document Object Model (PDOM) bilden. Zwei weitere wichtige Module sind der Tokenizer und der Lexer.

Der Tokenizer

Der Tokenizer wandelt einen String in eine Reihe von Tokens. Dazu wird weder ein komplexer Regulärer Ausdruck noch irgendein System verwendet. Der Tokenizer arbeitet den String Zeichenweise ab (mit einigen wenigen Ausnahmen). Dadurch wird das Parsen relativ langsam.

Der Lexer

Der Lexer bildet aus den Tokens einen Baum, bei dem auch alle Whitespaces erhalten bleiben. Dadurch bleibt immer das Perl-Dokument erhalten oder eine exakte Kopie kann aus dem Baum generiert werden.

PDOM

Für die Darstellung des PDOM existieren über 60 Perl-Klassen, die die einzelnen Tokens darstellen. In der Dokumentation von `PPI` ist eine Klassenliste zu finden. Eine einfache Demonstration des PDOM ist in Listing 1 zu finden.

Ein ganz nützliches Werkzeug ist `PPI::Dumper`, mit dem solche PDOM-Bäume ausgegeben werden können. Das ist sehr hilfreich, wenn man nicht genau weiß, wie ein Token benannt wird. Welche Art von Statement ist es denn? Eine Möglichkeit ist es, sich die Dokumentation anzuschauen oder einen schnellen Blick in den Dump zu werfen.

```
1 #!/usr/bin/perl
2
3 print( `$foo-Magazin` );
4
5    PPI::Document
6      PPI::Token::Comment                `#!/usr/bin/perl\n`
7      PPI::Token::Whitespace             `\n`
8      PPI::Statement
9        PPI::Token::Bareword             `print`
10       PPI::Structure::List             ( ... )
11         PPI::Token::Whitespace         ` `
12         PPI::Statement::Expression
13           PPI::Token::Quote::Single    ``$foo-Magazin``
14         PPI::Token::Whitespace         ` `
15       PPI::Token::Structure            `;`
```

Listing 1

```
1 sub _find_super{
2     my ($self,$doc) = @_;
3     my $ppi    = PPI::Document->new($doc) or die $!;
4
5     my $varref = $ppi->find('PPI::Statement::Variable');
6     my @vars   = ();
7     if($varref){
8         @vars    = $self->_get_isa_values($varref);
9     }
10
11    my $baseref = $ppi->find('PPI::Statement::Include');
12    my @base    = ();
13    if($baseref){
14        @base = $self->_get_base_values([grep{$_->module eq 'base'}@$baseref]);
15    }
16     return [@vars,@base];
17 } # _find_super
```

Listing 2

PPI im Einsatz

In den kommenden Abschnitten wird gezeigt, wie das Modul `Class::Superclasses` die Superklassen von Perl-Klassen herausfindet - mit Hilfe von `PPI`. Als erstes sollte klar sein, dass die Definition von Superklassen grundsätzlich auf zwei verschiedene Wege gemacht werden kann. Zum Einen über das `@ISA`-Array (zum Beispiel `our @ISA = qw(Superclass)`) und zum anderen über das Pragma base (use base qw(Superclass)). Diese beiden Wege sollen abgedeckt werden.

Im Quellcode des Moduls ist die Subroutine `_find_super` (Listing 2) die Initialmethode für die Suche nach den Superklassen.

In Zeile 3 wird ein neues Objekt von `PPI::Document` erzeugt. Als Parameter wird der Pfad der zu analysierenden Perl-Klasse übergeben. Es wird `PPI::Document` genommen, weil vom direkten Umgang mit dem Lexer abgeraten wird. `PPI::Document` startet intern den Lexer.

In Zeile 5 werden alle Tokens gesucht, die vom Typ `PPI::Statement::Variable` sind. Damit werden alle Variablendeklarationen in der Perl-Klasse gefunden. Der `find`-Methode kann man entweder einen Klassennamen übergeben - wie in diesem Fall - oder ein Referenz auf eine eigene Subroutine. Wie die Zeile 11 mit einer Subroutine aussehen würde, ist in Listing 3 dargestellt.

```
1 my $varref = $ppi->find( sub{
2   $_[0] == $_[1]->parent
3   and
4   $_[1]->isa( 'PPI::Statement::Variable' );
5 });
```

Listing 3

Als Parameter wird das Top-`PPI::Node`-Element und das Element, das überprüft werden soll, übergeben.

Wenn eine solche Variablen-Deklaration gefunden wurde, werden die Elemente darauf überprüft, ob es die Deklaration des `@ISA`-Arrays ist.

In den Zeilen 11-15 wird die zweite Möglichkeit, Superklassen zu bestimmen - mit `base` - abgearbeitet. Das `use base` ist ein "Include", so dass in Zeile 11 erstmal alle `use`-State-

ments gesucht werden.
Wenn die Variablendeklarationen herausgesucht wurden, muss noch überprüft werden, ob es die Deklaration des @ISA-Arrays ist. Danach müssen noch die Elemente des Arrays herausgeparst werden. Hier gibt es zwei Wege, wie die Deklaration aussehen kann. Es kann als Ausdruck da stehen (`our @ISA = ("Klasse1","Klasse2")`) oder mit einem Quote-Like-Operater (`our @ISA = qw/Klasse1/`). Die Methode, die das übernimmt, ist in Listing 4 dargestellt.

Bevor die Werte herausgeparst werden können, muss überprüft werden, wie der gesamte Ausdruck aufgebaut ist. Wenn man sich den PDOM-Baum für `our @ISA = qw/Klasse1/` anschaut (Listing 5), dann bekommt man eine Idee, wie man vorgehen muss. Unterhalb des Ausdrucks sind die Informationen enthalten. Deshalb werden alle Unterelemente des Baums mit der Methode `children` geholt (Zeile 5).

Danach wird überprüft, ob eines der Elemente `@ISA` ist. So wird nach dem Variablennamen gesucht.

Wenn es das `@ISA`-Array ist, wird nach einem Element vom Typ `PPI::Statement::Expression` gesucht. Wenn das vorhanden ist, ist der gesamte Ausdruck nach der ersten Möglichkeit aufgebaut. Wenn das der Fall ist, wird der Unterausdruck (zum Beispiel `("Klasse1","Klasse2")`) geparst. Wird ein Element vom Typ `PPI::Token::QuoteLike::Words` - was dem `qw`-Operator entspricht - gefunden, wird der Ausdruck auf eine andere Art geparst. Wenn die Superklassen mittels base festgelegt werden, sieht die Methode ziemlich ähnlich aus.

Mit `_parse_expression` werden Ausdrücke wie `our @ISA = ('FooMagazin')` geparst. Dazu braucht man erstmal alle Ausdrücke. Dies funktioniert mit der Methode `find`. Ausdrücke sind bei `PPI` ein Objekt vom Typ `PPI::Statement::Expression`. Für jeden Ausdruck werden die Unterobjekte (beziehungsweise die Unteräste des PDOM-Baums) geholt. Wenn dieses Unterobjekt ein Objekt der Klasse `PPI::Token::Quote::*` ist, wird das Quote-Zeichen über das `$element->{seperator}` geholt und aus dem Inhalt entfernt.

Die Dokumentation zu `PPI` ist sehr umfangreich. Soll das Modul eingesetzt werden, lohnt es sich, die Dokumentation sehr genau zu lesen und erst etwas herumzuspielen. Nach kurzer Zeit findet man sich dann ganz gut zurecht.

```
1  sub _get_isa_values{
2      my ($self,$varref) = @_;
3      my @parents;
4      for my $variable(@$varref){
5          my @children = $variable->children();
6
7          if(grep{$_->content() eq '@ISA'}@children){
8              if($variable->find_any('PPI::Statement::Expression')){
9                  push(@parents,$self->_parse_expression($variable));
10             }
11             elsif($variable->find_any('PPI::Token::QuoteLike::Words')){
12                 push(@parents,$self->_parse_quotelike($variable));
13             }
14         }
15     }
16     return @parents;
17 }# _get_values
```

Listing 4

```
1 sub _parse_expression{
2     my ($self,$variable) = @_;
3     my $ref = $variable->find('PPI::Statement::Expression');
4     my @parents;
5     for my $element($ref->[0]->children()){
6         if($element->class() =~ /^PPI::Token::Quote::/){
7             my $separator = $element->{seperator};
8             (my $value = $element->content()) =~ s~\Q$separator\E(.*)\Q$separator\E~$1~;
9             push(@parents,$value);
10        }
11    }
12    return @parents;
13 }# _parse_expression
```

Listing 5

```
1 sub _parse_quotelike{
2     my ($self,$variable) = @_;
3     my $words         = ($variable->find('PPI::Token::QuoteLike::Words'))[0]->[0];
4     my $operator      = $words->{operator};
5     my $section_type  = $words->{sections}->[0]->{type};
6     my ($left,$right) = split(//,$section_type);
7     $right = $left unless defined $right;
8     (my $value = $words->content()) =~ s~$operator\Q$left\E(.*)\Q$right\E~$1~;
9     my @parents = split(/\s+/,$value);
10     return @parents;
11 }# _parse_quotelike
```

Listing 7

Fazit

PPI ist sehr nützlich, um Perl-Dokumente zu analysieren. Die Anwendungsbeispiele sind sehr vielfältig. Das Modul ist die Basis für Perl::Critic, und um eigene Policies zu schreiben (siehe Seite 29), sind Kenntnisse von PPI notwendig.

Einige Module zum Erzeugen von UML-Diagrammen basieren auf PPI, und auch der UML-Editor "UPC" verwendet es zum Parsen von Perl-Klassen. Das Parsen ist relativ langsam, allerdings hat man die Möglichkeit, Perl-Programme zu analysieren, ohne sie auszuführen. So wäre es denkbar, erst eine Sicherheitsanalyse zu starten und die Programme erst nach Überprüfung zu starten.

Es gibt noch viele weitere Ideen, was man mit dem Modul machen kann. So wird es wohl möglich sein, mit PPI Perl-IDEs effektiv zu programmieren.

Renée Bäcker

TPF-Ticker

"Help wanted"

Jim Brandt hat im Blog der Perl-Foundation eine neue Kategorie eingeführt: "Help wanted". Jim wird in unregelmäßigen Abständen Sachen vorstellen, die etwas Hilfe brauchen. Dabei geht es nicht darum, irgendwelche Bugs aus dem RT zu bearbeiten, sondern Sachen zu machen, die Perl weiterhelfen können.

Der erste Eintrag geht über SOAP::Lite. Jim schreibt, dass man das Modul schon ganz gut einsetzen kann, dass es aber verbessert werden sollte.

Mit der ersten Aussage kann ich nicht ganz so gut leben, da sich mir die Verwendung des Moduls nicht wirklich erschlossen hat. SOAP ist sicherlich eine gute Sache, aber in Perl ist die Unterstützung dafür noch nicht so wirklich gut - jedenfalls das was ich so gesehen habe.

TPF-Ticker

Neuer "Chef"-Perlmonger

Die Perlmongers haben einen neuen "Chef"-Perlmonger. Nach vier Jahren hat Dave Cross sein Amt an José Castro übergeben, der mit einigen Projekten die Bildung von neuen Perlmonger-Gruppen fördern will.

Ich mache hier die Regeln - Perl::Critic::Policies

Jedes Unternehmen hat eigene Programmierrichtlinien und jeder Programmierer hat einen eigenen Stil. Soll der Code auf den eigenen Stil hin überprüft werden, sind die bestehenden Policies von Perl::Critic häufig ungeeignet. Da müssen also eigene Policies geschrieben werden.

Dafür ist es notwendig, dass man sich einigermaßen mit PPI auskennt und einige Regeln von Perl::Critic beachtet. In diesem Artikel soll das Schreiben eigener Policies am Beispiel "öffnende Klammer einer Schleife muss in der Zeile des Schleifenkopfes stehen" gezeigt werden.

Es soll in Zukunft also so etwas erzwungen werden wie

```
for my $var ( ... ) {
```

und nicht

```
for my $var ( ... )
{
```

Big Perl is watching your Code

Wie schon erwähnt, steht hinter Perl::Critic das Modul PPI. PPI zerlegt den Perl-Code in Tokens, erstellt daraus "Nodes" und baut daraus eine Baumstruktur. Perl::Critic ruft bei jedem "Node" die entsprechende Perl::Critic::Policy-Subklasse auf und wenn ein Code vorhanden ist, der nicht den Regeln entspricht, wird ein Perl::Critic::Violation-Objekt zurückgeliefert.

Das Gerüst muss stehen

Die Policies sind Subklassen von Perl::Critic::Policy, und die Klassen müssen dementsprechend auch in diesem Namensraum liegen. Perl::Critic arbeitet mit Module::Pluggable, um die Policies automatisch zu finden. Die ersten Zeilen einer Policy (nach dem Package-Namen) sollten so aussehen:

```
use strict;
use warnings;
use Perl::Critic::Util;
use Perl::Critic::Violation;

use base qw/Perl::Critic::Policy/;

our $VERSION = '0.01';
```

Die Policy muss einem Nutzer auch sagen, was falsch ist. Dafür gibt es zwei Variablen in jeder Policy: Zum Einen `$desc` und zum Anderen `$expl`. `$desc` ist ein String, der aussagen sollte, was falsch gelaufen ist. Das kann in wenigen Worten geschehen. `$expl` ist entweder ein String mit einer genauen Beschreibung, was falsch gemacht wurde, oder eine Arrayreferenz mit den Seitenzahlen aus "Perl Best Practices", auf denen das gewünschte Verhalten erklärt ist.

Da in diesem Beispiel kein Programmierstil von "Perl Best Practices" behandelt wird, sind das ganz eigene Sachen, die dort ausgegeben werden.

```
my $desc = q~{ nicht in der Zeile des
   Schleifenkopfes.~;
my $expl = q~Die öffnende Klammer muss in
   der Zeile des Zeilenkopfes stehen, also

  for my $var ( ... ){
      ...
  }

anstatt

  for my $var ( ... )
  {
      ...
  }
~;
```

Die new-Methode sollte nur überschrieben werden, wenn der Nutzer Argumente an die Policy übergeben können soll. Das ist hier aber nicht notwendig.

Mein Profil

Jede Policy hat ein paar Standardsachen, die ein gewisses Profil für die Policy darstellen. Dazu gehören neben der Kurzbeschreibung in `$desc` und der genauen Beschreibung in `$expl` auch ein paar Subroutinen, die angeben, für welche PPI-Nodes die Policy angewendet werden soll, zu welchen Themes die Policy gehört und natürlich für welchen Severity-Level die Policy angewendet werden soll.

Die Angabe des PPI-Knotens ist wichtig, damit die Policy auch nur für ganz bestimmte Code-Fragmente angewendet wird. Außerdem bedeutet es einen Geschwindigkeitsvorteil, wenn nicht alle Policies für alle Knoten angewendet werden.

Der Knoten wird durch die Subroutine `applies_to` bestimmt. Die `for`-Schleife ist in `PPI::Statement::Compound`, also muss dies in der Subroutine eingetragen werden.

```
sub applies_to{ return '
   PPI::Statement::Compound'}
```

Über die Angabe des Themes können Anwender bestimmte Policies ein- oder ausschalten. Dies ist ganz praktisch, wenn man für mehrere Kunden arbeitet und alle etwas unterschiedliche Regelungen haben.

Dieses Beispiel soll in den Themes "core" und "foo" enthalten sein. Die Subroutine, die das bestimmt, heißt `default_themes`.

```
sub default_themes{ return qw/core foo/ }
```

Der Severity-Level bestimmt, wie wichtig die Regel ist. Ist es eine Regel, die in jedem Programm eingehalten werden muss, dann ist das die höchste Prioritätsstufe, ist es jedoch nur eine "Schönheitsregel", dann kann eine niedrige Prioritätsstufe ausgewählt werden. Da die hier gezeigte Regel auf jeden Fall eingehalten werden soll, bekommt sie die höchste Priorität

```
sub default_severity{
      return $SEVERITY_HIGHEST
}
```

Die Variable `$SEVERITY_HIGHEST` wird vom Modul `Perl::Critic::Utils` exportiert.

Jetzt geht's los!

Bevor das große Coden losgeht, am besten nochmal genau darüber Gedanken machen, was die Policy machen soll. In diesem Fall müssen wir bedenken, dass **kein** Fehler ausgegeben werden soll, wenn der Schleifenkopf nachgestellt ist, also so etwas wie

```
print $_ for(@array);
```

Als ganz nützlich hat sich herausgestellt, vorher verschiedene Varianten von Code, die der Nutzer schreiben könnte, mit PPI zu testen und sich die Baumstruktur anzuschauen.

In den Beispielcodes auf der Webseite ist ein Beispielprogramm mit verschiedenen Schleifenvarianten enthalten. In Listing 1 ist die PDOM-Struktur einer `for`-Schleife, wie sie gültig ist, zu sehen.

```
PPI::Statement::Compound
  PPI::Token::Word      'for'
  PPI::Token::Whitespace        ' '
  PPI::Token::Word      'my'
  PPI::Token::Whitespace        ' '
  PPI::Token::Symbol            '$var'
  PPI::Token::Whitespace        ' '
  PPI::Structure::ForLoop       ( ... )
    PPI::Statement
      PPI::Token::Symbol        '@array'
  PPI::Structure::Block         { ... }
```

Listing 1

Ein kleiner Teil ist bei einer ungültigen Schleife unterschiedlich. Zwischen der `ForLoop` und dem `Block` taucht noch ein Newline auf:

```
PPI::Structure::ForLoop       ( ... )
  PPI::Statement
    PPI::Token::Symbol        '@array'
PPI::Token::Whitespace        '\n'
PPI::Structure::Block         { ... }
```

Listing 2

Die einfachste Variante, eine solche Policy zu entwickeln ist, sich alle möglichen Varianten der zu untersuchenden Struktur aufzuschreiben und mit dem Dumper von PPI anzuschauen. Dann können die Unterschiede besser herausgearbeitet werden. Für diesen Artikel gibt es bei den Codes auf der Webseite das Skript `show_pdoms.pl`, das genau dieses zeigt.

Bei dem hier gezeigten Beispiel fällt auf, dass bei einer Regelverletzung immer ein `\n` zwischen der `For Loop` und dem `Block` erscheint.

In Listing 3 ist die Methode zu sehen, die für die eigentliche Regelüberprüfung zuständig ist.

Die Methode, die die Überprüfung enthält, heißt `violates` und bekommt von `Perl::Critic` automatisch drei Parameter übergeben: das `Perl::Critic`-Objekt, das entsprechende `PPI`-Element, auf das die Regel angewendet wird und das komplette `PPI`-Dokument (die Datei oder das Snippet, auf das `Perl::Critic` angewendet wird).

Die Policy soll nur auf `for`-Schleifen angewendet werden. Deshalb muss als erstes überprüft werden, ob es sich tatsächlich um eine solche Schleife handelt. Dazu wird überprüft, ob das erste `child`-Element ein 'for' ist.

Danach werden die Positionen von `Block`, `ForLoop` und Newline festgestellt. Nur in dem Fall, in dem ein Newline zwischen `Block` und `ForLoop` ist, wird ein Fehler ausgegeben.

Und schon ist die erste eigene Regel implementiert. Bei einer `for`-Schleifen, die gegen die Regel verstößt, wird in Zukunft ein Fehler ausgegeben, wie er in Listing 4 zu sehen ist.

Renée Bäcker

```
 1 sub violates{
 2     my ($self,$elem,$doc) = @_;
 3
 4     my $base = $elem->schild(0);
 5     return unless $base eq 'for';
 6
 7     my ($list,$newline,$block);
 8
 9     my @children = $elem->children;
10
11     for my $i ( 0 .. $#children ){
12         my $child = $children[$i];
13
14         if( $child->isa( 'PPI::Structure::Block' ) ){
15             $block = $i;
16         }
17         elsif( $child->isa( 'PPI::Structure::ForLoop' ) ){
18             $list = $i;
19         }
20         elsif( $child->isa( 'PPI::Token::Whitespace' ) and
21                $child eq "\n" and
22                not $newline){
23             $newline = $i;
24         }
25     }
26
27     if( $newline and $newline > $list and $newline < $block){
28         my $sev = $self->get_severity;
29         return Perl::Critic::Violation->new( $desc,$expl,$elem,$sev );
30     }
31
32    return;
33 }
```

Listing 3

```
 1 oeffnende Klammer muss in Zeile mit Schleifenkopf sein at line 20, column 5. Es
 2 soll in Zukunft also so etwas erzwungen werden wie
 3
 4   for my $var ( ... ) {
 5
 6 und nicht
 7
 8 for my $var ( ... )
 9 {
10 .
```

Listing 4

Profiler

Voreilige Optimierung ist bekanntlich eine schlimme Falle, in die vor allem Juniorprogrammierer tappen. Doch was tun, falls ein Skript eindeutig zu langsam läuft? Oft lässt sich mit wenig Aufwand viel gewinnen, wenn man den Hebel an der richtigen Stelle ansetzt.

Zunächst einmal gilt es festzustellen, in welchen Programmteilen am meisten Zeit verbraten wird. Hierzu wird das Skript `foo` mit dem Profiler `Devel::DProf` vom CPAN mit `perl -d:DProf foo` aufgerufen. Es läuft so etwas langsamer als normal, während der Profiler Daten über die abgearbeiteten Funktionen sammelt. Ein anschließender Aufruf des Programms `dprofpp` aus der `Devel::DProf`-Distribution liest dann die vom Profiler angelegte Datei `tmon.out` aus und zeigt den Inhalt wie in Abbildung 1 gezeigt an.

Dort ist ersichtlich, dass das Skript 96.6% der Zeit in der Funktion `expensive()` verbringt. Rechts in Abbildung 2 ist der Source Code des Skripts zu sehen. Die Funktion versucht jeweils 1000 Mal vergeblich, einen regulären Ausdruck mit einem String in Einklang zu bringen.

Um herauszufinden, in welchen Zeilen einer Funktion ein Skript CPU-Zeit vergeudet, kommt ein Line-Profiler zum Einsatz. Das Modul `Devel::SmallProf` vom CPAN analysiert beliebige Skripte zur Laufzeit und zeigt hinterher genau an, wie lange der Interpreter in jeder Zeile des Skripts verharrte.
Hierzu wird das Skript mit `perl -d:SmallProf foo` aufgerufen. Zu beachten ist, dass der Line-Profiler `Devel::SmallProf` erheblich langsamer ist, sodass man eventuell nur Teile davon untersuchen kann. Das Ergebnis liegt nach Ablauf des Skripts fertig formatiert in der Datei `smallprof.out` vor.

```
mschilli@mybox:~/DEV/articles/profiler/eg
$ perl -d:DProf foo
$ dprofpp
Total Elapsed Time = 0.107958 Seconds
  User+System Time = 0.117958 Seconds
Exclusive Times
%Time ExclSec CumulS #Calls sec/call Csec/c  Name
 96.6   0.114  0.114   1000   0.0001 0.0001  main::expensive
 0.00       - -0.000      1        -      -  strict::bits
 0.00       - -0.000      1        -      -  strict::import
 0.00       - -0.000      1        -      -  main::BEGIN
$
```

Abb. 1

Abbildung 2 zeigt, dass die Zeile mit dem Regex-Match insgesamt 10.000 Mal durchlaufen wurde. Die Anzahl der *wallclock seconds* entspricht der Anzahl der tatsächlich verstrichenen Sekunden, wenn man alle Aufrufe der Zeile zusammenrechnet. `CPU seconds` hingegen geben die verbrauchte Prozessorzeit an. Bei einem `sleep(1)`-Befehl wäre zum Beispiel der erste Wert ziemlich genau eine Sekunde pro Aufruf, während die verbrauchte CPU-Zeit nahezu Null wäre.

Wer übrigens meint, der regulären Ausdruck `/$m1/` ließe sich durch den Modifizierer `/o` beschleunigen, täuscht sich. `/o` steht für *once* und bewirkt, dass ein regulärer Ausdruck, der eine Variable enthält, nur einmal kompiliert wird. Mit ihm nimmt Perl dann an, dass sich der Wert der Variablen nicht ändert. Allerdings merkt sich das Perl beim zuletzt bearbeiteten regulären Ausdruck schon automatisch. In einer Schleife wie in der Funktion `expensive()` bringt das also nichts.

Mit einem Haudegen-Trick lässt sich das Skript allerdings erheblich beschleunigen: Wenn man mit

```
use Memoize;
memoize('expensive');
```

festlegt, dass die Funktion `expensive` immer die gleichen Werte zurückliefert, wenn man sie mit den gleichen Argumenten aufruft, wird sie nur noch einmal aufgerufen und die restlichen neun Mal 'weiß' `Memoize` bereits das Ergebnis. Und schon läuft das Programm 10x schneller!

Mike Schilli

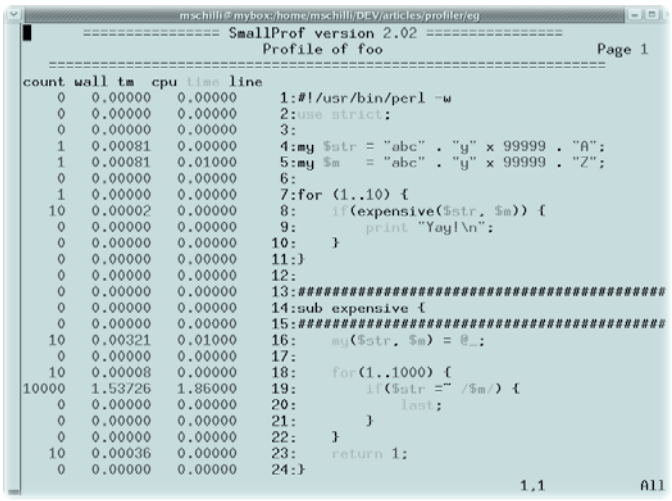

Abb. 2

Schweizer Team ist Sieger in der Kategorie Perl beim "Plat_Forms" Web-Programmier-Wettbewerb

Deutscher Perl-Workshop gratuliert – Besonders der kompakte und leicht erweiterbare Code beeindruckt

Perl erzeugt kompakten und einfach zu erweiternden Code – das bestätigt die Studie "Plat_Forms 2007: The Web Development Platform Comparison" der Freien Universität Berlin.
"Der Entwicklungszyklus ist mit Perl besonders schnell und die Sprache ist sehr flexibel. Für Unternehmen, die auf schnelle Entwicklung und leistungsfähige Software Wert legen, stellt Perl eine echte Alternative dar", so Cedric Bouvier vom schweizer Team Etat de Genève/Optaros, das in der Kategorie Perl den Wettbewerb Plat_Forms 2007 gewonnen hat.
Die anderen Perl-Teams – plusW aus Deutschland und Revolution Systems aus den USA – folgen mit knappem Abstand, die Entscheidung ist der Jury schwer gefallen.
Die teilnehmenden Perl-Teams hoffen, dass die weitreichenden Fähigkeiten der Perl-Plattform zur professionellen Web-Entwicklung nun mehr Verbreitung finden. "Dazu könnten auch Fachhochschulen und Universitäten beitragen, indem sie Perl häufiger als bisher in ihre Lehrpläne aufnehmen", so Dami Laurent, ebenfalls vom Team Etat de Genève/Optaros.

Der Deutsche Perl-Workshop gratuliert den erfolgreichen Teams zu ihrer guten Arbeit.

Perl wird zur Web-Entwicklung beispielsweise bei der Business-Plattform XING (vormals openBC), beim Nachrichten-Portal heise online und der Technik-Community Slashdot eingesetzt.

Perl 6
Der Himmel für Programmierer - Teil 2

Perl 6 ist wesentlich mehr als die nächste Version nach Perl 5. Es ist eine neue Generation Perl, die sich aus der Idee entwickelt, alle Aspekte (wie Syntax, Interpreter, CPAN und Kultur) von der Gemeinschaft vollständig erneuern zu lassen.
Diese Idee reifte in den letzten sieben Jahren zu einem vielfältigen Netzwerk aus Projekten heran, die ich in der vorigen Ausgabe, im ersten Teil dieses Artikels, vorgestellt habe. Dabei beschrieb ich auch Pugs, einen Interpreter, der bereits wesentliche Teile der Perl 6-Syntax versteht und sie langsam, aber recht stabil ausführen kann. Da auch die neue Syntax in groben Zügen ausgeknobelt ist und oft nur noch in kleineren Details geändert wird, kann es für Interessierte jetzt schon sinnvoll sein, die Sprache zu erlernen und auszuprobieren. Dieser Artikel möchte dabei helfen, indem er vor allem neue Elemente und die Veränderungen gegenüber Perl 5.8 anhand von Beispielen erläutert und auch erklärt, welche Überlegungen zu diesen Änderungen geführt haben.

Der Ansatz

Beginnen wir mit den Überlegungen. Wie ein Puzzelrahmen helfen sie das Gesamtbild zu sehen und erleichtern es auch zu begreifen, wie einzelne Entscheidungen sich in das Gesamtbild fügen. Diese Überlegungen sind um so wichtiger, da Perl 6 nicht nur die umfangreichste, sondern auch die durchdachteste Version bisher ist. Immer wieder schraubte Larry Wall nach langen Debatten an den Details, damit die Anzahl der Regeln und Ausnahmen so gering wie möglich bleibt, alle Teile ihren logischen Platz finden und alles aus einfachen, wiederverwendbaren Elementen besteht, die dem menschlichen Denken entsprechen sollen. (Oder zumindest was sich Verrückte wie Larry und Damian darunter vorstellen.) Damit wird Perl mit seinen 20 Lenzen nicht nur eleganter, aufgeräumter und erwachsener, sondern wirkt auch immer mehr wie eine eigenständige Sprache und weniger nach einer Mischung aus UNIX-shell und C.

Natürlich gab es nebenbei noch viel Unrat aufzuräumen, der sich in 12 Jahren Perl 5 und darüber hinaus angesammelt hat. Die alte Schreibweise für “Paket::sub()“ (“Paket‘sub()“) verwirrt heute mehr, als das sie benutzt oder gebraucht wird. Die bisherige “do {} while“-Schleife wartet schon lange auf eine Reparatur, da die Sprungbefehle für Schleifen wie “redo“, “next“ und “last“ darin unbekannt sind. Und selbt nützliche Dinge wie Formate werden heute wesentlich seltener gebraucht, und könnten in ein Modul ausgelagert werden.

Selbstverständlich fügte man auch viele interessante und praktische Dinge hinzu, damit der Spruch auf Perls alter Manpage wieder stimmt: “Perl vereint alle coolen Features anderer Sprachen, ohne ihre uncoolen Begrenzungen.“ Aber genauso wichtig ist, dass im üblichen “höher, schneller, breiter“-Wahnsinn eines großen Rewrites die Perlphilosophie bewahrt wurde, wenn auch in leichter Neuinterpretation.

Mehr als ein Weg

Perl Grundgesetz Artikel 1: Es gibt mehr als einen Weg (TIMTOWTDI) - ist natürlich unantastbar und wurde lediglich ausgebaut. Aus noch mehr Operatoren, Befehlen und Konstrukten, deren Funktionen sich überlappen, kann ein Programmierer sich jetzt passendes aussuchen. Auch werden nun noch mehr Programmierstile gleichberechtigt unterstützt und Perl erreicht auch so gleichzeitig volle “Buzzwordkompatibilität“. Neben verbesserter funktionaler, deklarativer und objektorientierter Programierart gibt es nun

auch aspektorientierte Programmierung (AOP - hier roles genannt) und "design by contract" (einigen sicher aus eiffel bekannt). Jeder kann so frei nach Vorlieben, Problemstellung oder Fähigkeiten wählen und kombinieren, ohne das sich die Sprache in den Weg stellt. Das ist es, was Liebhaber an Perl schätzen. Ihre Ansichten werden ernst genommen und nicht bevormundet. Sowohl Anfänger als auch Virtuosen unterstützt Perl in ihrem Programmierstil und ermöglicht es, frei nach eigenem Antrieb dazulernen zu können.

Neu in Perl 6 ist daran nur die Erkenntnis, daß Freiheit auch beinhaltet, sich gegen Freiheiten entscheiden zu können. Während Perl 5 alles möglichst offen hält, gibt es jetzt echte private Klassenvariablen, die von außerhalb des Objektes unzugänglich sind. Auch kann man den Perlinterpreter jetzt anweisen, Fehlermeldungen auszugeben, wenn die Parameter an eine Subroutine (sub) nicht in gewünschter Anzahl oder Typ übergeben wurden. Man könnte sogar mit einem Satz an LISP-artigen Macros Perl 6 in ein ultrastriktes Python 4000 verwandeln (falls dies jemand wünscht). Auch das wäre perlartig, denn es ist ein Weg und Perl möchte alle anbieten.

Apropos Signaturen (Definition der Parameter, die eine Subroutine erwartet) und Objektorientierung: dort hat sich besonders viel geändert. Nicht, weil Perl damit bisher seltsame Wege ging und @larry sich nun mehr anpassen will, sondern weil der alte Weg mehr Tipparbeit aufbürdet.

Perl5:

```
sub pythagoras {
 my ($a, $b) = @_;
 return sqrt( $a**2 + $b**2 );
}
```

Perl 6:

```
sub pythagoras ($a, $b) {
 return sqrt( $a**2 + $b**2 );
}
```

Die gute Nachricht für Liebhaber des alten Weges ist: Auch das erste Beispiel ist vollständig gültiger Perl 6-Code. In der OOP spart man sogar noch weit mehr Anschläge, wie das Minimalbeispiel mit einer kombiniertem Getter- und Setter-Funktion zeigt (siehe Listing 1 für Perl 5 und Listing 2 für Perl 6).

Ein weiterer Grund für diese Änderungen waren die parallelen Lösungsansätze für die Übergabe komplexer Daten an Subroutinen, oder für eine elegante Erzeugung und Verwaltung objektorientierter Strukturen. Die kamen oft nur schlecht miteinander aus und wurden nicht selten nur mangels vorgegebener Standardwege geschaffen. Strukturlosigkeit ist auch keine Freiheit, und Perl 6 bietet daher auch auf dem Gebiet sowohl den Autopiloten, der einem alles abnimmt und dafür nur Standard kann und vorgibt, als auch das Buschmesser für den eigenen Dschungelpfad. Aber auch das war schon vorher ein Perl-Grundsatz ("make easy things easy and hard things possible").

Eine weitere Dimension an Wahlmöglichkeiten ist die Hardwarenähe. Perl 6 lernte so gut wie alle Schmankerl der hoch abstrakten, funktionalen Programmierung, kennt nun aber auch bei Bedarf Datentypen, was Programme schneller und sparsamer machen kann. Damit kann ein einmal geschriebener Algorithmus fix noch etwas nachgetrimmt werden und Perl ist somit auch in manchen der Fälle einsetzbar, in denen auch Perl-Freunde auf C zurückgegriffen hätten.

```
class bambus { has $.length is rw }
my $pflanze = bambus.new;
$pflanze.length = 12;
```

Listing 2

```
1  package bambus;
2
3  sub new{
4    bless {length => 0}, shift;
5  }
6
7  sub length {
8    $self = shift;
9    $length = shift;
10   if (defined $length) { $self->{$length} = $length }
11   else { $self->{$length} }
12 }
13 package main;
14 my $pflanze = bambus->new;
15 $pflanze->length(12);
```

Listing 1

Legosteine der Perlwelt

Wenden wir uns nach den Grundprinzipien nun detaillierter den semantischen (Wortbedeutung) und syntaktischen (Schreibweise) Regeln der neuen Sprache zu. Die hat der Herr Wall (als studierter Linguist) vornehmlich seinen Betrachtungen natürlicher Sprachen entnommen und größtenteils schon früher angewendet, um es Programmierern zu ermöglichen, sich in Perl fast so intuitiv auszudrücken wie in ihrer Muttersprache. Eines der Mittel, um dies zu erreichen, sind einfache, universell einsetzbare Konstrukte, die stets die gleiche Idee umsetzen, selbst wenn sie in manchem Zusammenhang (formal gesehen) etwas Anderes tun. So wie etwa im Deutschen das kleine Wörtchen "und", mit dem sich Worte aller Art und auch ganze Satzteile logisch verknüpfen lassen. Dem folgend hat Perl seit seinem Beginn Skalarvariablen, die alle klassischen Datentypen speichern können, oder Arrays, die man auch einfach als Stack oder Queue verwenden kann, wofür man z.B. in Java extra Klassen bräuchte.

Perl 6 führt dieses Prinzip weiter, etwa mit dem "vergleich das!-Operator (im original: "smartmatch"), der "~~" geschrieben wird und der, je nach Kontext und Typ der zugewiesenen Daten, nicht nur die Funktion von "=~", "==", "eq" übernehmen kann, sondern auch Arrays oder Hashes vergleicht und sogar Auskunft gibt, ob ein Objekt diese Methode hat, jene Role vollführt und vieles mehr.

Perl 6 bekam ebenfalls neue Metaoperatoren, die einfachen Operatoren erweiterte Arbeitsweisen geben. Das Prinzip gibt es seit den alten C-Tagen, seit ein Operator vor dem "=" selbstzuweisend wird, also das Ergebnis dem ersten Operanden zuweist.("$a +=5" statt "$a = $a + 5") Die neuen Metaoperatoren (Hyper- ("<<" und ">>"), Cross- ("x x") und Reductionoperator ("[ ]")) helfen jedoch, einfache Operatoren auf Listen anzuwenden. Hyperoperatoren parallelisieren die Verarbeitung von Arrays.

Perl 5:

```
sub add_arrays {
  my @a1 = @{shift;};
  my @a2 = @{shift;};
  my @summen;
  $summen[$_] = $a1[$_] + $a2[$_]
   for 0..($#a1 > $#a2 ? $#a1 : $#a2);
  return @summen;
}
```

Perl 6:

```
sub add_arrays (@a1, @a2) { @a1 >>+<< @a2 }
```

Der ternäre Operator deutet an, was der Hyperoperator bei Arrays ungleicher Längen tut, aber auch mehrdimensionale Arrays können Metaoperatoren abarbeiten. Bei einem Input von "@a1 = (1,2,3)" und "@a2 = (2,3,4,5)" hieße das Ergebnis "(3,5,7,5)". Jetzt ein Beispiel um den Kreuzoperator zu veranschaulichen

Perl 5:

```
sub combine_arrays {
  my @a1 = @{shift;};
  my @a2 = @{shift;};
  my @ergebnis;
  for my $a1 (@a1){
    for my $a2 (@a2){
       push @ergebnis, $a1.$a2;
    }
  }
  return @ergebnis;
}
```

Perl 6:

```
sub combine_arrays (@a1, @a2) {@a1 x~x @a2}
```

Bei einem Input von "@a1 = ('a','b','c')" und "@a2 = (2,3)" hieße das Ergebnis "(a2,b2,c2,a3,b3,c3)". Genau, die Tilde ("~") verknüpft jetzt Strings weil der Punkt, wie gezeigt, in die OOP-Branche gegangen ist. Nun ein letztes Beispiel, welches die Fakultät berechnet wie z.B.: "3! = 1*2*3".

Perl 5:

```
sub fakultaet {
  my $n = shift;
  my $ergebnis = 1;
  $ergebnis *= $_ for 2..$n;
  $ergebnis;
}
```

Perl 6:

```
sub fakultaet (Int $n) { [*] 1..$n }
```

In Perl finden sich viele dieser universellen Grundprinzipien. Das häufigste und unscheinbarste ist vielleicht der Codeblock, der immer von 2 geschweiften Klammern umfasst wird. Egal ob es ein anonymer Block ist, eine "if"-Anweisung, ein Modul, eine Subroutine oder ein Objekt ist, in Perl 6 gelten überall die gleichen Regeln. Schlüsselwörter wie "sub" oder "if" verändern nur das Verhalten etwas. Dadurch wird die Sprache berechenbar und Anfänger können sich deswegen mit einer Art Fremdendeutsch a la "Ich habe gemacht lange Urlaub." durchschlagen, ohne das es ihnen der Interpreter übel nimmt.

Sag es genau

Könner drücken sich natürlich genauer aus, weil sie damit viel mehr in weniger Worten aussagen können. Worte mit ähnlicher Aussage, wie “gehen“, “schreiten“ und “schlendern“ können ganz andere Bilder in der Vorstellung der Zuhörer entstehen lassen. Analog gibt es in Perl 5 “for“, “map“ und “grep“, die zwar alle 3 über Arrays iterieren, aber ihre bewußte Verwendung läßt den Perlkundigen erahnen, in welche Richtung der der folgende Algorithmus geht und er kann schneller verstehen, was der Code beabsichtigt. Wenn der passende Befehl gewählt wurde, deckt der Mechanismus, den er liefert, einen großen Teil der gewünschten Funktionalität ab und es bleibt nur sichtbar, was in diesem Fall das Besondere ist. Dieses Prinzip war auch bei den Metaoperatoren erkennbar und führte auch dort zu sehr kurzem, ausdrucksstarkem und lesbaren Code. Vorausgesetzt man ist sich nicht zu schade, die Worte zu lernen.

Denn die sind oft sehr einfach zu merken. Ein Modul beginnt nämlich jetzt mit “module“, eine Klasse mit “class“ und Klassen, deren Methoden nur Regexen ausführen sind Grammatiken und “grammar“ genannt. “package“ bleibt zur Erstellung von Namensräumen und alles was für ein Package gilt, ist auch für Module und Klassen gültig. Nur bieten die noch zusätzliche Funktionen, die das schreiben erleichtern. Die Vorzüge des “class“-Befehls liegen, wie schon demonstriert, im automatisch erstellten Konstruktor, Destruktor, Gettern und Settern, aber auch das Klonen für Prototypen-basierende OOP bekommt frei Haus. Selbstredend markiert “module“ einen Namensraum, der ausgewählte Variablen und Routinen in andere Namensräume laden kann, oder ihnen anderweitig zur Verfügung stellt. Viel Funktionalität, die heute nur durch diverse Exporter-Module erhältlich ist, gehört in Perl 6 zu den selbstverständlichen Möglichkeiten jedes Moduls.

Ein weiteres Beispiel für solche spezialisierten Anweisungen, sind in Perl 6 die Befehle “regex“, “rule“ und “token“. Sie ähneln etwas dem “sub“-Befehl, der einem Codeblock einen Namen zuweisen kann oder der in Perl 6 auch dazu dienen kann, eine Referenz auf diesen Codeblock zu bilden, die dann per Zuweisung in einer Variable gespeichert werden kann. Der “Unterschied“ ist, daß diese Befehle Blöcke einleiten in die reguläre Ausdrücke gehören, die dann auch benannt oder gespeichert werden können. Token sind nur für einfache Buchstabenmuster, mit einfachen Metazeichen reserviert, Rules dürfen komplexer werden und erlauben beliebiges Einfügen von Leerzeichen, aber kein Backtracking. Regexen bieten alle rechenintensiven Operationen wie Rück- und Vorschau.

Die beschriebene deklarative Wirkung der Befehle würde aber verloren gehen, wenn sie sich zu ähnlich wären. Deshalb achtet $larry sehr genau darauf, Verwechslungen zu vermeiden. Z.B. wurde die Funktion des x-Operator in 2 verschiedene Operatoren getrennt. Die Funktion im Skalarkontext blieb gleich (“ahaaha“ = “aha“ x 2), aber die Funktion im Listenkontext ((‘aha‘,‘aha‘) = ‘aha‘ x 2) wird jetzt mit xx geschrieben. Dies war jedoch nicht nur ein Tribut an die Bequemlichkeit, nicht mehr den Kontext eines Ausdruckes herausfinden zu müssen, sondern bietet auch neue Möglichkeiten wie: (‘und‘,‘zack‘) xx 2 = (‘und‘,‘zack‘,‘und‘,‘zack‘).

Eine ähnliche, aber gefährlichere Falle barg der Bereichsoperator. Oft wird er etwa so benutzt (for (1..9) {}), aber nicht viele wissen noch, daß er im Skalarkontext zum Flipflop-Operator wird. Innerhalb einer Schleife lassen sich damit zwar auch wie in awk und sed Bereiche herausfiltern:

Perl 5:

```
while (<>) {
# gibt zeilen von  start bis stop-Marke aus
print if /start/ .. /stop/;
...
```

Da die Arbeitslogik dieses Operators mehr mit verketteten Vergleichen (1 < $a < 4), die jetzt auch in Perl möglich sind, zu tun hat, als mit einem Zahlenbereich, heißt der Flipflop-Operator auch jetzt ff. Genauer gesagt: ‘..‘ wird zu ‘ff‘ (awk-Verhalten) und ‘...‘ => ‘fff‘ (ähnlich sed). Das Beispiel in Perl 6 ginge dann etwa so:

Perl 6:

```
for =$fh {
  # gibt zeilen von start bis stop aus
  print if /start/ ff /stop/;
...
```

Filehandles werden in Skalaren gespeichert ($fh). Das “=“ davor hat die gleiche Funktion wie das alte “<>“ - es gibt bei jedem Aufruf die nächste Zeile aus diesem Inputstream, aber nur im ‘lazy list‘-Kontext. Ein anderer Kontext, wie ihn z.B. “while“ forciert, würde gleich alle Zeilen aus der Datei quetschen und man könnte die Zeilen nicht nicht einzeln untersuchen. Deswegen benutzte ich in dem Beispiel “for“. Doch was macht jetzt der Bereichsoperator im Skalarkontext? Er hilft zu überprüfen, ob ein Skalarwert in einem Bereich liegt. Das sieht dann so aus, wie in Listing 3 dargestellt.

```
3 ~~ 2 .. 5 # ist wahr
3 ~~ 5 .. 2 # nicht wahr, leerer Bereich
5.5 ~~ 2 .. 5 # nicht wahr, liegt außerhalb des Bereiches
5 ~~ 2^..^5 # nicht wahr, Schranken sind ausgenommen
```

Listing 3

```
for @a -> $a { # jede runde den nächsten Wert aus @a nach $a
... # $a ist im Block bekannt (wie mit my)

for zip(@a; @b) -> $a { # 1 Wert je Runde abwechselnd nach $a

for zip(@a; @b) -> $a, $b { # jede Runde 2 parallel

for @a Z @b -> $a, $b { # andere Schreibweise
```

Listing 4

Natürliche Prominenz und visuelle Reize

Zurück zu den Prinzipien natürlicher Sprachen. Ich erwähnte, dass ein sorgfältig akzentuiertes Wort eine sehr genaue Situation in das Bewusstsein rufen kann. Das wirkt umso besser, je näher dieses Wort an Anfang des Satzes steht. Beispiel: “Abends, als die rotglühende Sonne hinter die Tannenwipfeln zu versinken begann, ging ich den mir wohl vertrauten Weg, hinunter zum Bach.“ Der Satz ist zwar recht lang, aber insgesamt erträglich zu lesen, weil bereits das erste Wort die Szene umreißt, der Leser ahnt was kommt und er wird von Detail zu Detail geführt, ohne vor- und zurückzuschauen. In vielen Bereichen ist es selbstverständlich, den Satz, oder die Sinneinheit, mit einer klaren Ansage zu beginnen. Alle genannten Blockmodifikatoren wie “module“, “if“, “while“, “map“, “sub“, “given“ sind Auftakt für eine bestimmte logische Struktur. Nur bei den regulären Ausdrücken hatte man bisher zuerst an den Anfang zu sehen, dann an das Ende (für die Optionen) und erst dann konnte man die Regex verstehen. Um das zu vereinfachen, sind die Optionen nach vorne gewandert und das Komplizierte nach hinten, wo es am wenigsten belastet:

Perl 6:

```
# tauscht jedes i gegen ‘bebe‘ aus
$text ~~ s:g/ i / bebe /;
```

Schwer verwechselbare Schlüsselworte am Anfang einer Sinneinheit haben auch eine visuelle Qualität, die durch gute Einrückungen und Leerzeichen voll zur Geltung kommt. In vielen Situationen, so auch beim Lesen oder Suchen im Quellcode, suchen Menschen nach schnellen graphischen Orientierungspunkten, und Perl 6 versucht mit einigen weiteren Mitteln, dieser natürlichen Neigung der Menschen entgegen zu kommen. Zum einen sind einige Operatoren auch als Piktogramme zu verstehen. Feed-Operatoren (<==, <<==, ==>, ==>>) kann man fast ansehen, dass sie Röhren (Pipes) sind, in denen Datenströme entlangfließen. Sie bieten eine Möglichkeit eine Schwartz‘sche Transformation etwas anschaulicher zu schreiben. Das zweite Beispiel benutzt lediglich mehrere Quellen:

Perl 6:

```
@result <== map { ... }
<== grep{ ... }
<== map { ... }
<== @data;

@result2<== map { ... }
<== grep{ ... }
<== map { ... }
<<== @data;
<<== @data2;
```

Ein anderes Piktogramm ist neben dem wohlbekannten Hashkonstruktor (‘=>‘), der einfache Pfeil nach rechts (‘->‘). Dies ist eine alternative Schreibweise für einfache Signaturen, wenn lediglich ein paar Werte in einen beliebigen Block “hineingeworfen“ werden sollen. Das Konstrukt nennt sich dann ‘pointy block‘ und findet seine Anwendung z.B. bei “for“-Schleifen, da ‘for‘ nur über einen array iteriert (siehe Listing 4 für Perl 6.

Wer genau hinsieht, merkt, dass das “Z“ nicht nur für zip(Reißverschluss) steht, sondern auch ein Piktogramm ist, welches die Arbeitsweise des Operators anzeigt (zuerst aus linkem, dann aus rechtem Array, zurück zum linken aber nächsten ...).

Es läßt sich auch an der Länge und Form der Perl 6-Operatoren einiges ablesen. Streng nach Huffman-Coding sind die wichtigsten am kürzesten und die seltener gebrauchten um so länger. Im Fall der Metaoperatoren soll die Länge auch eine Warnung sein: “Achtung, dies ist kein gewöhnlicher Opera-

tor“. Ähnliches gilt wohl auch für private Klassenvariablen, die mit ihrem Ausrufungszeichen als Twigil etwas seltsam und exponiert aussehen ($objekt!private_var). Larry meinte dazu nur: “Seltsame Dinge sollten auch seltsam aussehen“.

Weltsprache Perl

Auffällig an Perl 6 ist auch ein Hang zum “simple english“. Viele neue Befehle entstammen dem englischen Alltagswortschatz. Worte wie ‘given‘, ‘when‘, ‘is‘, ‘has‘, ‘does‘, ‘what‘, ‘repeat‘, ‘but‘, ‘gather ‘und ‘take‘ sind nicht schwer zu merken und vermitteln auch einem Englisch-Anfänger eine Ahnung, was der zugehörige Befehl machen könnte. Dies ist nicht völlig neu, da bereits früher Begriffe wie ISA, use, no und einiges mehr in Perl auftauchten. Aber trotzdem bekommt Perl durch diese einfache Benennung und die klare Grammatik eine unverkrampfte Ausstrahlung. Hier heißt eine Methode ‘method‘ und eine Subroutine ‘sub‘. Seit mir das geläufig ist, wundert es mich, daß in anderen Sprachen eine Routine mit ‘def‘ definiert wird, ohne das irgendwo steht was definiert wird. Perl ist wirklich angenehm explizit und deklarativ geworden.

Bedürfnisse prägen Sprachen

Bisher schrieb ich von ein bis wenigen Menschen, die versuchen, eine Sprache nach natürlichen Gesetzmäßigkeiten zu gestalten. Was für ein Widerspruch, werden doch natürliche Sprachen durch ihre vielen Nutzer geformt. Aber es gibt in Perl 6 mindestens 2 Fälle, in denen Larry entgegen seine Bestrebungen handelte und dem Druck der Nutzerwünsche nachgegeben hat. Das sind zum einen Kommentare, die über mehrere Zeilen gehen und eine “case“-Anweisung. Nach dem Motto “wenn dann richtig“, fand er in beiden Fällen Lösungen, die wirklich mächtig sind.

Perl 6:
```
# "Bugs is gonna to bite"
say "Bugs is #{[ ??? ]}gonna to bite!";
say "Bugs is#(
(()) lala
# "Bugs is gonna ... now"
) biting you right now";
```

Die Raute kommentiert wie eh und je den Rest der Zeile. Fügt man ihr jedoch unmittelbar öffnende Klammern an, wird kommentiert, bis die gleiche Anzahl und Art schließender Klammern in umgekehrter Reihenfolge vorkommt. Perl zählt auch die Klammern innerhalb des Kommentars, so daß auch dort geklammert werden kann, ohne Einfluß auf das Kommentarende, solang im Kommentar auch zu jeder schließenden auch eine öffnende Klammer gehört.

Der “given-when“-Befehl sollte bekannt sein, da er in der ersten Ausgabe mit Perl 5.10 vorgestellt wurde. Er funktioniert ähnlich einer “for“-Schleife mit einem Durchlauf. Der Wert nach “given“ setzt den Inhalt der Kontextvariable ($_). Die “when“-Klauseln führen einen smartmatch (~~) gegen $_ aus. Dadurch sind die Klauseln sehr mächtig und auch in einer “for“-Schleife verwendbar. Ein großer Unterschied zur C-Variante ist ebenfalls der “continue“-Befehl, der ein Verlassen des “given“-Blocks nach Ablauf der Klausel verhindert. Ein “break“ gibt es nicht.

Perl 6:
```
given $ratezahl {
when 12|7 { say 'eine Märchenzahl' }
say 'hm, 7 o. 12 wars schon mal nicht.';
when 9 { say 'verrat ich nicht'; continue }
when 1..6 { say 'kleiner als 7' }
default { say 'ich komm nicht drauf.'}
}
```

Unnatürliche Einflüsse

Perl 6 besitzt auch gänzlich unnatürliche Einflüsse, vor allem den menschlichen Ordnungswahn und den Zwang mancher Programmierer, alles in Objekten einzuordnen.
Ja, die schlechte Kunde zuerst, Perl 6 ist mindestens so objektorientiert wie Ruby. Jede Zahl und jeder String ist ein Objekt. “@array.sort“ und “text“.print sind ab jetzt gültiges Perl. Aber nun die gute Nachricht, die alte Schreibart bleibt erhalten. Und noch besser: mit einer besonderen Schreibweise läßt sich jedes objektorientiert programmierte Modul prozedural benutzen:

Perl 6:
```
$hacker.feed('Pizza and cola');
feed $author: 'Kuchen and Kräutertee';
```

```
$*ARGS # Argumente (Parameter) an das Script
$*IN # Standarteingabe
$*OUT # Standartausgabe
$*ERR # Standartfehlerausgabe
$*PERLVER # aktuelle Perlversion
$*OS # Auf welchem Betriebsystem lauf ich gerade?

$?OS # Für welches Betriebsystem kompiliert?
$?PARSER # Welche Grammar wurde für aktuelle Zeile benutzt?
@?BLOCK # In welchen Blöcken bin ich gerade?
$?LINE # In welcher Zeile bin ich?
```

Listing 5

Der Ordnungswahn war allerdings dieses mal berechtigt. Sämtliche Sondervariablen, deren Namen oft nur ein Zeichen kurz war und deren Namensraum sich über alle Sonderzeichen erstreckte, haben zwei gesunde Namen bekommen: eine Abkürzung und eine ausgeschriebene Form. Und eine Sekundäre Sigil (Twigil), damit sie in ihrem Sondernamensraum bleiben. Spezielle Variablen, deren Inhalt zur Kompilierungszeit fest steht, besitzen als Twigil ein Fragezeichen und die Veränderlichen, wie alle anderen globalen Variablen auch, einen Stern. Einige Beispiele (Perl 6) in Listing 5.

Epilog

Wie hoffentlich zu merken war, bietet Perl 6 viel neues und interessantes. Es ist definitiv zu viel, um es auf einmal zu präsentieren und auch zu viel, um alles auf einmal zu verstehen. Deswegen möchte ich in den folgenden Ausgaben, in wesentlich kürzeren Artikeln, jeweils ein kleines Teilgebiet noch gründlicher behandeln und nebenbei auch aktualisieren, was hiervon dann veraltet ist.

Herbert Breunung

TPF-Ticker

Parrot Grant

Dave Rolsky berichtete über die Parrot Releases Februar bis einschließlich April. In dieser Zeit haben Allison Randal und Jonathan Worthington das Objekt Design von Parrot abgeschlossen. Im April-Release von Parrot wurde das Design auch größtenteils schon implementiert.

Die Unterstützung für viele Sprachen, wie zum Beispiel Lua, PHP, Tcl und Perl6, wurde stark verbessert.

Jesse Vincent hat das Amt des Parrot-Projektleiters abgegeben - macht das aber noch weiterhin für Perl6 - und Will Coleda hat dieses Amt übernommen.

VB2Perl
Microsoft Excel im Büro

Microsoft Excel wird fast in jedem Büro vielseitig eingesetzt. Es ist ein weitverbreitetes Werkzeug für die Sammlung, Aufbereitung und Präsentation von Daten. Excel ist im Allgemeinen für Nichtprogrammierer zugänglicher als Perl.

Die beiden Programme ergänzen sich sehr gut und arbeiten auch recht gut zusammen. So kann Perl unter Verwendung des Moduls `Spreadsheet::ParseExcel` Excel-Dateien lesen und Komma-, Semikolon- oder Tab-separierte Dateien schreiben. Das Excel Dateiformat wird auch vom Modul `Spreadsheet::WriteExcel` unterstützt.

Die Zusammenarbeit könnte aber noch besser sein, insbesondere für das Umformatieren von Tabellen, das Zusammenführen von Daten aus mehreren Tabellenblättern und das Erstellen von Grafiken. Hier muss oft in Excel Hand angelegt werden.

Diese manuelle Nacharbeiten sind nicht nur langweilig, sondern auch fehleranfällig und langsam. Insofern ist eine Automatisierung erstrebenswert - alles was ich manuell wiederholen kann, kann die Maschine besser wiederholen.

Automation von Excel

Für die Automation von Excel und der ganzen Office Produktlinie hat Microsoft die OLE-API spezifiziert. Excel ist komplett über OLE steuerbar, und Perl hat über das Modul `Win32::OLE` Zugriff auf diese Steuerung.

Die OLE API

OLE selbst ist eine API, die es ermöglicht, Methodenaufrufe an Objekte über Prozessgrenzen und auch Maschinengrenzen hinweg abzusetzen, solange nur der Name der Objektmethode und eventuell die Namen der Parameter bekannt sind. Für Perl-Programmierer ist das wenig neu - der Name einer Methode muss nicht von Anfang an festgelegt sein:

```
$objekt->$methode( @parameter );
```

Der Wald vor lauter Bäumen

Es gibt nur ein kleines Problem - man muss den richtigen Namen der Objektmethode oder -eigenschaft kennen, bevor man auf diese zugreifen kann. Microsoft Office ist eine sehr grosse Sammlung von sehr komplexen Programmen. Erfreulicherweise ist auch die zugehörige Dokumentation sehr groß, was sich aber wiederum als Nachteil erweist, wenn man etwas nachschlagen möchte. Insbesondere vor dem Hintergrund, dass die meisten Office-Aufgaben keine expliziten Programmieraufgaben sind, lohnt sich der Zeitaufwand des Studiums der Dokumentation nicht unbedingt - erst durch die Wiederholung amortisiert sich ein Programm.

Der Makro Recorder

Das Problem der Komplexität und gewünschten Wiederholbarkeit findet auch schon bei Microsoft eine Lösung - den Office Makro-Recorder. Mit ihm kann man Vorgänge aufzeichnen, abspeichern und bei Bedarf wieder ablaufen lassen. Weiterhin kann man den gesamten aufgezeichneten Vorgang als Visual Basic Quellcode anschauen und modifizieren.

Diese Fähigkeit der Abänderung und Parametrisierung des Aufgezeichneten ist es, die es uns ermöglicht, auf die Doku-

mentation weitestgehend zu verzichten. Wir zeichnen den gewünschten Vorgang einfach auf und übersetzen den so erhaltenen Basic Code nach Perl.

Beispiel 1

Täglich sollen in eine Datei Zeilenüberschriften eingefügt werden und diese Datei dann an einen Mailverteiler weitergeleitet werden.
Diese Aufgabe ist offenbar eintönig und wenig zeitaufwendig, es lohnt sich aber trotzdem oder gerade deswegen, sie zu automatisieren. Damit werden auch solche Klippen wie Urlaub oder Krankheit umschifft. Dazu ist es nötig, ein letztes Mal den Vorgang manuell durchzuführen, mit eingeschaltetem Makro-Recorder:

- Makroaufzeichnung einschalten
- Datei öffnen
- Leere Zeile ganz oben einfügen
- Spaltenüberschriften eintragen
- Datei speichern
- Makroaufzeichnung anhalten
- Makro im VB-Editor öffnen (`ALT-F11`)
- `Modul1` anschauen

Im Ergebnis erhält man Visual Basic Code (Listing 1)

```
'Option Explicit 1
Rows("1:1").Select
Selection.Insert Shift:=xlDown
ActiveCell.FormulaR1C1 = "Sprache"
Rows("1:1").Select
Range("B1").Activate
ActiveCell.FormulaR1C1 = "Problem"
Rows("1:1").Select
Range("C1").Activate
ActiveCell.FormulaR1C1 = "Zeilen"
Range("A1").Select
ActiveWorksheet.SaveAs "foo.xls"
```

Listing 1

Liest man den Code durch, kann man die einzelnen Schritte der Eingabe klar nachvollziehen. Es sind offenbar keine Schritte, wie sie ein Programmierer programmieren würde. Die Schritte beschreiben eben die schrittweise Manipulation von Excel durch einzelne Funktionsaufrufe. So wird in den ersten zwei Zeilen eine neue Zeile in das Tabellenblatt eingefügt, indem erst die oberste Zeile markiert wird und dann nach unten verschoben wird. Dann wird jeweils die Markierung über die einzelnen Zellen geschoben und der jeweilige Text eingetragen. Ein echtes Programm wird auf die "globalen" Variablen `Active*` verzichten und stattdessen "echte", lokale Variablen verwenden.

Exkurs: Excel aus Perl Sicht

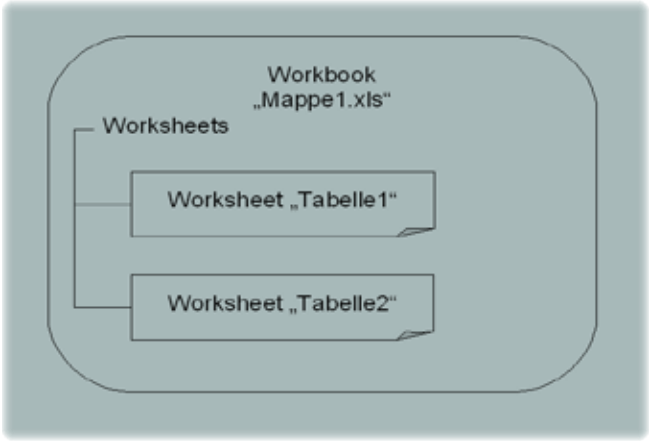

Abb. 1

Die Idee zur Struktur von Excel ist ein "Buch" mit vielen "Blättern". Das Programm Excel kann mehrere Dateien öffnen. Jede Excel-Datei `.xls` entspricht dabei einem Buch ("Workbook"). Jedes Buch enthält wiederum ein oder mehrere Tabellenblätter ("Worksheet"). Diese Tabellenblätter - beziehungsweise die Zellen ("Cell") auf diesen - sind die Dinge, die unsere Perl-Skripte am meisten bearbeiten werden.

Für diejenigen, die mit der Basic Syntax oder insbesondere der Syntax von Visual Basic nicht vertraut sind, ist am Ende des Artikels ein kurzer Überblick über die Feinheiten gegeben.

VB nach Perl übersetzen

Im Wesentlichen ist die Übersetzung von Visual Basic nach Perl geradlinig möglich. Es wäre sogar möglich, die Übersetzung durch ein Programm zu automatisieren, aber die semantische Übersetzung von `Active*` in echte Variablen und die Umstellung bzw. Parametrisierung des Codes haben es mir bisher zu aufwendig erscheinen lassen. Die folgenden Regeln erzeugen fast lauffähigen übersetzten Code:

```
1  use Win32::OLE qw(in);
2  Win32::OLE->Warn(3); # croak() bei Fehler
3
4  my $Excel = Win32::OLE->new('Excel.Application', 'Close');
5  my $Worksheets = $Excel->ActiveWorkbook->Worksheets;
6  for my $Worksheet (in ($Worksheets)) {
7      $Worksheet->SaveAs(...);
8  };
9  $Workbook->Close();
```

Listing 2

- Punkt nach `->`
- `Dim Foo As Bar` nach `my $foo;`
- `Set Plonk = X` nach `$plonk = $x`
- `Active...` nach `my $foo =`
- `Foo := xlBar` nach `{ Foo => xlBar }`

Gewappnet mit diesen Regeln können wir eine erste Übersetzung des Codes zum Speichern einer Datei probieren:

Visual Basic Code

ActiveWorksheet.SaveAs "foo.xls"

Die direkte Übersetzung sieht aus wie folgt:

```
my $Excel=Win32::OLE->new('Excel.Application');
my $Worksheet=$Excel->ActiveWorksheet;
$Worksheet->SaveAs("foo.xls");
```

Eine bessere Übersetzung mit Fehlerüberprüfung, die auch nicht nur das aktive, sondern alle Tabellenblätter speichert, ist in Listing 2 dargestellt.

Die Import-Zeile von `Win32::OLE` importiert die `in` Funktion, welche Visual-Basic Arrays Perl-kompatibel macht. Mit `Win32::OLE->Warn(3)` werden Fehler auf der Excel-Seite zu `die` in Perl umsgesetzt. In der Schleife werden dann alle Tabellenblätter der aktiven Datei als einzelne Dateien exportiert. Zuletzt wird die aktuell bearbeitete Datei geschlossen.

Um jetzt die einzelnen Tabellenblätter zum Beispiel als CSV-Dateien zu speichern, könnte man die richtige Stelle in der langen Parameterliste von `SaveAs` suchen, oder einfach benannte Parameter verwenden, wie sie auch in Perl üblich sind. Benannte Parameter haben in VB die folgende Form:

```
ActiveWorksheet.SaveAs "foo.csv", \
      FileFormat := xlFormatCSV
```

In Perl beziehungsweise unter `Win32::OLE` werden sie als Hashreferenz am Ende der Parameterliste übergeben:

```
...
$Worksheet->SaveAs("foo.csv",
      { FileFormat => xlFormatCSV });
```

Die speziellen Werte für die Konstanten wie zum Beispiel `xlFormatCSV` findet man entweder über Google oder eine andere Suchmaschine, oder man importiert diese Konstanten aus Excel mittels

```
use Win32::OLE::Const 'Microsoft Excel';
 ...
$Worksheet->SaveAs("foo.csv",
       { FileFormat => xlFormatCSV });
```

Ein etwas komplexeres Beispiel:

Präsentieren von CSV-Daten

Als etwas komplexere Darstellung der Fähigkeiten wollen wir uns eine Präsentation mit Diagrammen anschauen. Dazu werden die einzelnen Schritte mit dem Makro-Rekorder aufgezeichnet und dann in ein Perl-Programm übersetzt.

Als erstes sollen Spaltentitel in die Tabelle eingefügt werden. Nachdem die Datei geöffnet ist, werden die Titel eingefügt. Listing 3 zeigt den Visual Basic Code und den dazugehörigen Perl-Code.

```
Rows("1:1").Select
Selection.Insert Shift:=xlDown
ActiveCell.FormulaR1C1 = "Sprache"
Rows("1:1").Select
Range("B1").Activate
ActiveCell.FormulaR1C1 = "Problem"
Rows("1:1").Select
Range("C1").Activate
ActiveCell.FormulaR1C1 = "Zeilen"
Range("A1").Select
```

Listing 3

Zunächst sollen in die geöffnete Datei die Spaltentitel eingefügt werden. Aus den zwei Schritten des Markierens und darauffolgenden Verschiebens der markierten Zellen wird in unserem Code ein geschachtelter Aufruf:

```
# Zeilen einfügen
# Rows("1:1").Select
# Selection.Insert Shift:=xlDown
$sheet->Rows("1:1")
      ->Select
      ->Insert( Shift => xlDown );
```

Auf das jeweilige Markieren der neuen Zellen können wir verzichten, da wir unsere Werte direkt schreiben. Ein flexiblerer Ansatz würde Werte aus einem Array in eine Zeile beliebiger Länge in einer Schleife schreiben.

```
# ActiveCell.FormulaR1C1 = "Sprache"
# Rows("1:1").Select
$sheet->Cells("A1")->FormulaR1C1 = "Sprache";

# Range("B1").Activate
# ActiveCell.FormulaR1C1 = "Problem"
$ sheet->Cells("B1")->FormulaR1C1 = "Problem";

# Rows("1:1").Select
# Range("C1").Activate
# ActiveCell.FormulaR1C1 = "Zeilen"
$ sheet->Cells("C1")->FormulaR1C1 = "Zeilen";

# Range("A1").Select
```

Als nächstes soll aus dieser Tabelle ein Diagramm erzeugt werden. Listing 4 zeigt den Code, der aus der Makro-Rekorder-Aufzeichnung resultiert, und den Perl-Code Beim Visual Basic Code sieht man, dass bei der Aufzeichnung zwei Diagrammtypen hintereinander ausgewählt wurden, zunächst ein Säulendiagramm `xlAreaStacked` und dann ein Netzdiagramm (`xlRadar`).

Der zentrale Knackpunkt der Verarbeitung ist die Erstellung der Grafik. Da die Konstanten von Excel bereits vorgegeben sind, liefert dieser Schritt wenig überraschendes (siehe Listing 5).

Die Übersetzung der einzelnen Teile nach Perl ist einfach; es empfiehlt sich trotzdem, sie schrittweise vorzunehmen und die einzelnen Abschnitte jeweils zu testen. Den zugrunde liegenden Visual Basic Code habe ich als Kommentar belassen, um einen direkten Vergleich zu ermöglichen.

Andere Anwendungsgebiete

Microsoft Excel ist bei weitem nicht das einzige Programm mit einer OLE-Schnittstelle. Fast alle Office-Produkte haben eine tiefgehende OLE-Schnittstelle, die die Steuerung von außen ermöglicht. Eine unrühmliche Ausnahme ist Outlook, dessen Manipulationsmöglichkeiten für Kalender, Kontakte und Foren zu wünschen übrig lassen.

Die Open-Source Gemeinde meidet OLE-Schnittstellen wie der Teufel das Weihwasser, weder OpenOffice noch FireFox haben eine solche Schnittstelle.

```
Charts.Add
ActiveChart.SetSourceData Source:=Sheets("Tabelle5").Range("A3")
ActiveChart.Location Where:=xlLocationAsNewSheet
ActiveChart.ChartType = xlAreaStacked
ActiveChart.ChartType = xlRadar
```

Listing 4

```
# Charts.Add
my $chart = $workbook->Charts->Add();
#ActiveChart.SetSourceData Source:=Sheets("Tabelle5").Range("A3")
#ActiveChart.Location Where:=xlLocationAsNewSheet
#ActiveChart.ChartType = xlAreaStacked
#ActiveChart.ChartType = xlRadar
for ($chart) {
    $ _ ->SetSourceData(...);
    $ _ ->Location( {Where => xlLocationAsNewSheet});
    $ _ ->ChartType( xlRadar );
}
```

Listing 5

Perl in Excel

Manchmal hat man nur Excel und kein Perl zur Verfügung und muss sich damit behelfen. Zum Glück ist eines der wesentlichen Features von Perl, die Hashes, auch in Excel vorhanden. Die `SVERWEIS` Funktion (engl. `VLOOKUP`) ist das Nachschlagen in einer Tabelle.

Die Formel

```
=SVERWEIS(A1;B:C;2;FALSCH)
```

ist das selbe wie der folgende Perl Code:

```
my %b_c = (
    Perl => 'http://www.perl.org',
    PHP  => 'http://hardenedphp.de',
    Ruby => 'http://ruby-lang.org',
);
print $b_c{ $a1 };
```

Exkurs: Basic-Syntax

Zum Schluss noch ein kurzer Überblick über die Syntax von Visual Basic, soweit sie einem Perl Prgrammierer begegnet.

Zeilenfortsetzung

Zeilen werden umgebrochen, indem man als Umbruchsmarkierung einen Unterstrich ("_") setzt:

```
   Print _
     "Hello World"
```

Entfernt ist dies mit der Shell Syntax für Zeilenumbrüche vergleichbar:

```
   echo \
       "Hello World"
```

Kommentarzeichen

Das Kommentarzeichen in Visual Basic ist das Hochkomma (""):

```
' Dies ist ein Kommentar
```

Funktionsaufrufe

Funktionen werden wie in Perl aufgerufen und deren Ergebnis wie gewöhnlich zugewiesen:

```
X = Left("Hello World",5)
```

Prozeduraufrufe

Prozeduraufrufe geschehen in Visual Basic grundsätzlich ohne Klammern:

```
Print "Hello World"
```

Achtung! Eventuell vorhandene Klammern werden als arithmetische Klammern um den ersten Parameter interpretiert:

```
Print ("Hello World",1) ' Fehler?
Print "Hello World",1   ' "Hello World1"
```

Optionale und benannte Parameter

Lücken in der Parameterliste werden in Visual Basic durch die Vorgabewerte aufgefüllt:

```
.SaveAs "myfile.csv",,,xlFormatCSV
```

Dafür gibt es kein direktes Perl Äquivalent. Die naive Übersetzung

```
->SaveAs("myfile.csv",,,xlFormatCSV)
```

ist inhaltlich etwas anderes - die aufeinanderfolgenden Kommata werden von Perl ignoriert.

Benannte Parameter werden in Visual Basic durch das Zuweisungszeichen `:=` markiert:

```
SaveAs Filename := "myfile.csv" _
   Format => xlFormatCSV
```

Stringmanipulation

Zeichenketten werden mit dem Kaufmannsund ("&") zusammengehängt:

```
Print "Hello " & "World"
```

Anführungszeichen müssen über die C<Chr()>-Funktion in Zeichenketten eingesetzt werden:

```
Print "Er sagte " & Chr(34) & "Das war's" & Chr(34) & vbCrLf
```

Max Maischein

FrOSCon

Free & Open Source Software Conference

Besucher- Registrierung mit Verlosung

25. und 26. August 2007
FH Bonn-Rhein-Sieg

Über 60 Vorträge und Workshops zu aktuellen Themen rund um freie Software:

Betriebssysteme - Entwicklung - Administration - Sicherheit - Recht
Desktop - Open Hardware - Web 2.0 - Virtualisierung ...

Über 20 freie Projekte:

Amarok - Archlinux - BSD - CaCert - CCC ErfaKreise - CentOS - Debian
Drupal - FFII - FreeWRT - FSFE - Geeklog - GRML - KDE - Kubuntu
Linaccess - LinuxGamer - Openoffice - PHP - Sidux- Typo3 - X.org - Zope

- Keynote mit Elmar Geese (Vorsitzender Linux-Verband)
- Drupal-Subkonferenz
- Key-Signing-Party
- LPI-Prüfungen
- Hüpfburg

Member of:

OPEN EVENTS FOUNDATION

FH Bonn-Rhein-Sieg, Grantham-Allee 20, 53757 Sankt Augustin bei Bonn

Dresden Perl Mongers

Dresden.pm wurde am 1. September 2002 von Jörg Westphal gegründet. Wenn dieser Text erscheint, denken wir also bereits über unser 5-jähriges Bestehen nach. Falls ihr, liebe Leser, zum Jubiläumstreffen im September 2007 vorbei kommen möchtet, sagt Bescheid, wir freuen uns immer über Besuch, vielleicht können wir was zusammen machen.

Aus Sicht von Google sind wir eine gut platzierte Wiki-Webseite (http://dresden-pm.org) und eine Mailingliste. Dass eine gute Google-Platzierung unnützerweise dazu führt, dass man nur noch sich selbst findet, tut dem Ego keinen Abbruch.

Die Mailingliste lesen ca. 50 Personen, zu den Treffen schafft es meist nur ein harter Sozialkern von 5..7 Leuten. Wir sind Studenten, Berufsprogrammierer und normale Menschen.

Wir treffen uns jeden ersten Donnerstag im Monat ab 20.00 Uhr im Medienkulturzentrum Pentacon, nachdem wir anfangs in Gaststätten und zwischendurch in der Wir AG in der Neustadt waren. Die Historie kann man ganz gut im Tagebuch auf unserer Webseite nachlesen, nur in letzter Zeit komme ich nicht mehr so richtig zum Schreiben.

Die Treffen sind vorwiegend “social meetings“ mit stark schwankendem “tech meeting“-Anteil, wir hatten schon echte, geduldstrapazierende Freaktreffen, oft aber reden wir auch einfach nur über die Guten Alten Zeiten(tm).

Dresdner Perlmongers tummeln sich üblicherweise seit Urzeiten bei perl-community.de, in IRC-Channels oder seit 2004 auch als Orgas beim Deutschen Perlworkshop. Betreute Perl-Themen sind die deutschsprachige Perl-FAQ, ein paar CPAN-Module, ein Emacs-Mode für .pod-Files, eine Variante des cperl-mode für Perl6 und diverse Dinge beim Perlworkshop, wie z.B. die aktuelle Webseite und der Tagungsband.

Kleine Highlights sind seit 2004 der jährliche Linux-Info-Tag, auf dem wir üblicherweise einen Vortrag halten. 2006 gab‘s noch einen im Rahmen der Woche der Informatik. Ein größeres Highlight war 2005 der 7. Deutsche Perlworkshop in Dresden, bei dem wir die lokale Organisation übernommen haben.

Unsere Fetische sind Closures, YAPC-Videos und Grundsatzdiskussionen. Nach einem kleinen Logo-Wettbewerb 2005 zelebrieren wir üblicherweise das Logo, das in Abb.1 dargestellt ist.

Für Plakate verwenden wir das von Wasi zum Wettbewerb eingereichte Bild (siehe Abb. 2)

Abb. 1: Logo

Abb. 2: Plakat

Nichtvirtuell pflegen wir außerdem Beziehungen zum benachbarten Chemnitz.pm, wir waren schonmal dort und etwas öfter kommt Chemnitz.pm bei uns vorbei. Unser schönstes Erlebnis war der Ausstieg aus einem 5-Meter hoch gelegenen Fenster, als wir spätnachts unachtsamerweise keinen Schlüssel für die abgeschlossene Haustür hatten. Wenn jemals Sachsen.pm wiedererweckt wird, werden wir dieses Treffen als Gründungstreffen rückdeklarieren.

Steffen Schwigon

Benutzerauthentifizierung und -autorisation bei Catalyst

In der zweiten Ausgabe von $foo wurde eine kleine Anwendung mit Catalyst gezeigt. So wie die Anwendung aufgebaut ist, können nur die wenigen Einträge angeschaut werden, die schon in der Datenbank existieren. Für eine richtige FAQ gehört es sich aber, dass sie erweitert werden kann.

Schön ist es, wenn jeder etwas zu der FAQ hinzufügen kann. Aber bei Unternehmensinternen Seiten oder bekannten Internetseiten ist genau das nicht gewollt. Häufig werden Wiki-Projekte und Foren zugespammt. Damit das der Beispielanwendung nicht passiert, wird eine Benutzerauthentifizierung eingeführt.

Die Authentizierung überprüft nur, ob der Nutzer auch tatsächlich der ist, der er zu sein vorgibt. Bei Redaktionssystemen reicht das aber nicht. Es dürfen nur registrierte Nutzer etwas eintragen, aber löschen darf nur der Boss. Genau dafür wird die Autorisierung benötigt. Die legt fest, welcher Nutzer welche Rechte hat. Hat der Nutzer nur Leserechte für einen Artikel, darf er etwas erstellen oder darf der Nutzer vielleicht sogar einen Artikel löschen?

In diesem Artikel wird gezeigt, wie eine Zugangskontrolle mit einer htpasswd-Datei umgesetzt wird. Hier werden noch keine Funktionalitäten für eine eingeloggte Person zur Verfügung gestellt.

Um die Arbeit zu sparen, eine Anmeldung zu implementieren, werden schon zwei User in die Datenbank eingetragen: Der User “OberKellner“ mit dem Passwort “adminpass“ ist der Admin der Anwendung.

Somit sieht die htpasswd-Datei wie folgt aus:

```
OberKellner:$apr1$$1$oTgMh$XeeOFljGbeR2uaUeXJyTk0
```

In Catalyst werden die Funktionen zur Authentifizierung und der Autorisierung mit ein paar Plugins sehr schnell in eine Anwendung integriert.

Sesam öffne Dich ...

... war in einem Märchen des geheimnisvolle Passwort. Ob mit diesem Satz auch die Webanwendung “geöffnet“ werden kann, bleibt abzuwarten. Die generelle Funktion von Zutritt und Ausgang werden durch das Plugin `Catalyst::Plugin::Authentication` ermöglicht.

Da die Speicherung der Daten in einer htpasswd-Datei vorgenommen wird, wird noch das Modul `Catalyst::Plugin::Authentication::Store::Htpasswd` benötigt.

Im nächsten Schritt wird die Konfiguration der Anwendung angepasst. Die Datei perlfaq.yml wird einfach um folgende zwei Zeilen ergänzt:

```
authentication:
    htpasswd: /Pfad/zur/Passwort.Datei
```

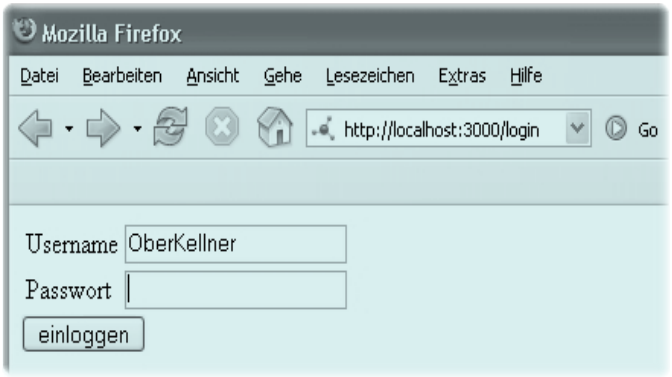

Abb. 1

In der Startklasse der Catalyst-Anwendung müssen noch die Plugins geladen werden, damit die entsprechenden Funktionalitäten verwendet werden können. Das Laden der Plugins ist in Listing 1 dargestellt.

```
1 use Catalyst qw/
2  -Debug
3  ConfigLoader
4  Static::Simple
5
6  Authentication
7  Authentication::Store::Htpasswd
8  Authentication::Credential::Password
9 /;
```

Listing 1

Im Vergleich zur Basisversion sind jetzt die drei Authentication-Plugins hinzugekommen. Das Authentication::Credential ist notwendig, wenn die Authentisierung mit einem Passwort von statten gehen soll. Auf CPAN finden sich noch viele weitere Credential-Module, mit denen eine HTTPBasic Authentisierung möglich ist oder mit Flickr-Zugangsdaten.

Jetzt sind die Grundlagen gelegt, und es kann mit den Controllern weitergehen. Das Grundgerüst dieser Controller kann mit dem `create`-Skript von Catalyst erzeugt werden (siehe Listing 2).

Das gleiche nochmal für den Login. Die Controller müssen noch angepasst werden.

Das Authentication-Modul stellt schon eine Funktion “login“ zur Verfügung, die im Login-Controller einfach aufgerufen wird (Listing 3). So muss man sich nicht selbst um die Auswertung der htpasswd-Datei kümmern. Bei einem Umstieg auf eine Datenbank muss einfach nur das Store-Modul ersetzt werden, und dann sollte es ohne weitere Code-Veränderungen funktionieren.

In der Subroutine sind nicht die Zuweisungen für den Stash enthalten. In den Codes auf der $foo-Webseite wird noch ein Login-Formular ausgegeben (Abb. 1) und bei erfolgreichem Login wird eine Seite mit Begrüßung und Logout-Button gezeigt.

In diesem Fall wird bei dem Logout einfach auf die Startseite der PerlFAQ-Anwendung verwiesen.

Autorisation

Für die Berechtigungen reicht die htpasswd-Datei nicht aus. Dazu werden in der Datenbank drei Tabellen angelegt. Eine Tabelle für den Nutzer; hier werden einfach der Username und das Passwort zusammen mit einer ID gespeichert. Die zweite Tabelle enthält die ID und den Namen der Rollen, die ein Nutzer haben kann. So eine Rolle könnte ‘Admin‘, ‘Editor‘

```
1 C:\PerlFAQ\script>perlfaq _ create.pl controller Logout
2   exists "C:\PerlFAQ\lib\PerlFAQ\Controller"
3   exists "C:\PerlFAQ\t"
4  created "C:\PerlFAQ\lib\PerlFAQ\Controller\Logout.pm"
5  created "C:\PerlFAQ\t\controller _ Logout.t"
```

Listing 2

```
1  package PerlFAQ::Controller::Login;
2
3  use base qw(Catalyst::Controler);
4
5  sub login : Global{
6       my ($self,$c) = @_;
7
8       my $user = $c->request->params->{username} || '';
9       my $pass = $c->request->params->{password} || '';
10
11       if( $c->login( $user, $pass ) ){
12           # user ist eingeloggt
13       }
14       else{
15           # Fehlschlag
16       }
17   }
```

Listing 3

oder anders heißen. Diese zwei Tabellen müssen miteinander verbunden werden. Dies geschieht über eine dritte Tabelle, die nur die ID des Users und die ID der Rolle speichert. In Abbildung 2 sind diese drei Tabellen dargestellt.
Für diese drei Tabellen werden noch die DBIx::Class-Module erzeugt, und dann können die Tabellen in der Catalyst-Anwendung verwendet werden. Hier soll die Rolle des Users bei der Begrüßung mit ausgegeben werden (siehe Listing 4).

Wenn der Nutzer eingeloggt ist, bekommt er gesagt, zu welcher Gruppe er gehört (Abb. 3).

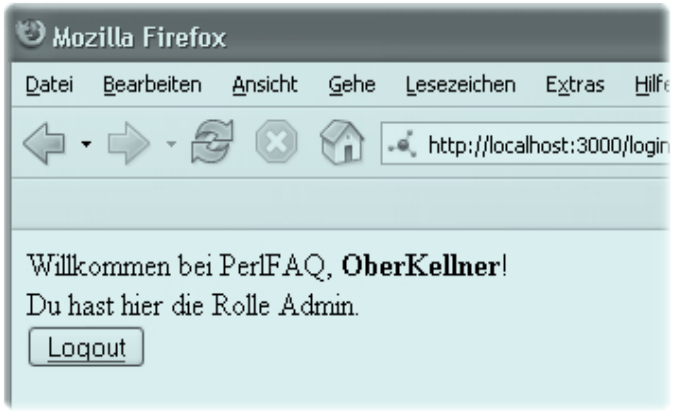

Abb. 2

Renée Bäcker

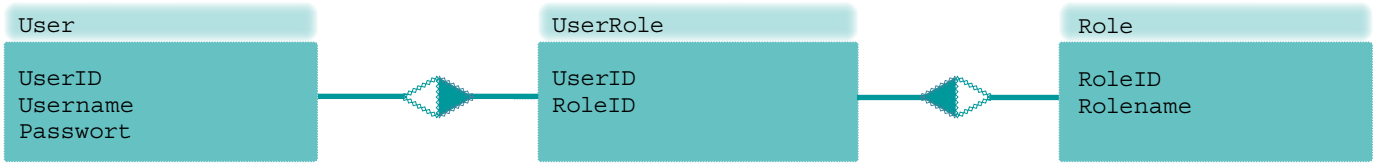

Abb. 3

```
1    sub login : Global {
2        my ( $self, $c ) = @_;
3
4        my $user = $c->request->params->{username} || '';
5        my $pass = $c->request->params->{password} || '';
6
7        $c->stash->{template} = 'login.tmpl';
8
9        if( $user and $pass ){
10           if( $c->login( $user, $pass ) ){
11               my ($usr)  = $c->model( 'DBIC::User' )->search({
12                   Username => $user,
13               });
14
15               my ($elem) = $c->model( 'DBIC::UserRole' )->search({
16                   'UserID' => $usr->UserID,
17               });
18
19               my $role = $elem->RoleID->Rolename;
20
21               $c->stash->{user}     = $user;
22               $c->stash->{rolle}    = $role;
23               $c->stash->{template} = 'success.tmpl';
24           }
25           else{
26               $c->stash->{error_msg} = 1;
27           }
28       }
29
30       $c->forward( 'PerlFAQ::View::HTC' );
31   }
```

Listing 4

Ajax mit Perl

Ajax ist keine neue Erfindung, aber seit ein paar altbekannte Technologien zusammengefasst wurden und unter dem Schlagwort “Ajax“ verbreitet werden, ist ein wahrer Hype entstanden. Spätestens seit den Google Maps ist der Durchbruch vollends gelungen.

Kaum eine große Webseite - egal of flickr.com oder xing.com - verzichtet auf Ajax-Komponenten. Der große Vorteil an Ajax ist, dass nicht immer eine komplette Seite geladen werden muss, sondern immer nur einzelne Teile beim Server angefragt werden.

Viele Webseiten, die dynamisch sind, werden mit Perl realisiert - vom einfachen Formmailer bis hin zu del.icio.us oder imdb.com. Aber auch bei diesen Seiten werden Ajax-Komponenten eingesetzt. Auf CPAN[1] finden Perl-Programmierer auch jede Menge Unterstützung für die Implementierung von Ajax-Funktionen.

Dieser Artikel soll einen Einblick in ein paar Module geben, mit denen Ajax-Funktionalitäten einfach umgesetzt werden können. Das Ganze wird am Beispiel einer ToDo-Liste gezeigt.

Hinter der ToDo-Liste steht eine einzige Tabelle, in der die Daten zu den einzelnen Aufgaben gespeichert werden. Wie die Tabelle aussieht, ist in Abbildung 1 dargestellt.

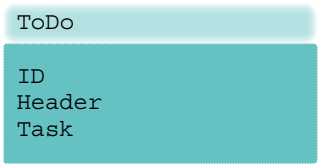

Abb. 1

Beim Start der Anwendung werden einfach nur die Titel der Aufgaben gezeigt (Abbildung 2), bei Klick auf einen Titel wird die Beschreibung der Aufgabe beim Server angefordert und dann dargestellt. Der User kann daraufhin die Beschreibung ändern und speichern. Weiterhin ist es möglich, die Aufgabe komplett zu löschen.

In diesem Artikel wird dieses Beispiel auf drei unterschiedliche Arten gelöst, jeweils mit einem anderen Modul. In den Beispielcodes auf der Webseite wird Ajax immer in einer `CGI::Application`-Anwendung eingesetzt.

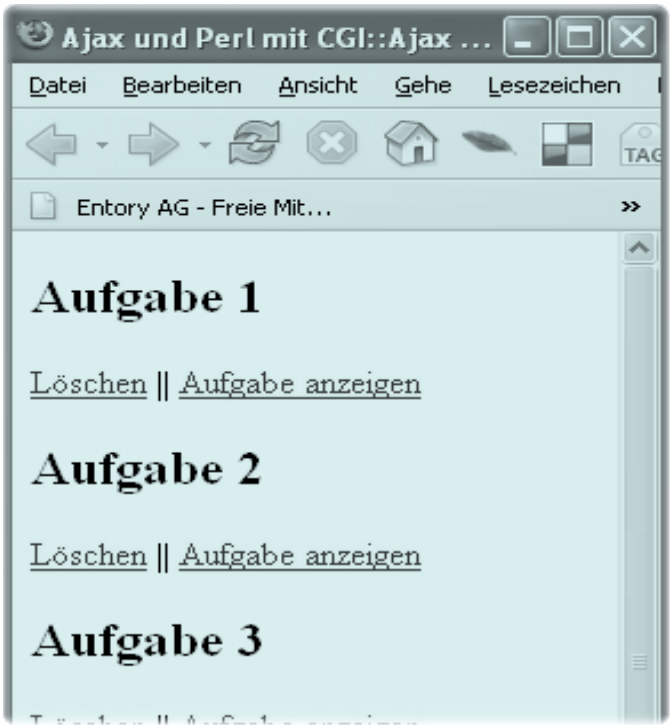

Abb. 2

CGI::Ajax

CGI::Ajax übernimmt wichtige Aufgaben. Es sorgt dafür, dass JavaScript-Funktionen auf Funktionen im Perl-Skript gemappt werden.

```
my $ajax = CGI::Ajax->new(
    'NameJSFunktion' => \&perl_function );
```

Man muss dazu keine Zeile an JavaScript schreiben und um die Ausgabe des HTML-Codes muss sich der Programmierer auch nicht kümmern. Die HTML-Ausgabe wird wie gewohnt generiert und dann einem Objekt von CGI::Ajax übergeben.

Das einzige Javascript, das vom Programmierer oder dem Designer erstellt werden muss, ist der "Auslöser" für die JavaScript-Funktionen. Im Template steht dann bei einem Link so etwas wie

```
onclick="NameJSFunktion([
 'action__delete_task','id__1'],['task1']);
```

NameJSFunktion ist der Name der JavaScript-Funktion, die bei einem Klick auf den Link ausgeführt werden soll. Als erstes wird der Funktion eine Liste mit Parametern übergeben, die dann auch als Parameter des GET-Requests an den Server übermittelt werden.

Durch dass "__" wird der Parameter-Name von seinem Wert getrennt. In diesem Fall wird zum Beispiel action=delete_task an den Server übermittelt. So können beliebige Parameter übergeben werden.

Die zweite Liste enthält die IDs von Elementen, die durch die Antwort des Servers modifiziert werden sollen. In diesem Fall gibt es ein div mit dem Namen task1, das die Antwort des Servers enthalten soll.

Ajax

Ajax ist ein Akronym für Asynchronous Java Script and XML und bezeichnet eine Sammlung von bekannten Technologien, mit der eine asynchrone Datenübertragung zwischen Client und Server möglich ist. Dadurch ist es möglich, Teile einer HTML-Seite neu zu laden, ohne dass die komplette Seite neu geladet werden muss.

Fallen

Auf ein paar kleine Fallen muss man aufpassen. So muss zum Beispiel darauf geachtet werden, dass man keine JavaScript-builtin-Funktionen verwendet. Das ist auch der Grund, warum in dem Beispiel eine Funktion delete2 anstatt delete heißt.

Unter Windows sollte man sich nicht wundern, wenn man sich mal die URLs anschaut, die für den Request benutzt werden. Da Windows die Länge von Dateinamen auf 8 Zeichen kürzt (intern), steht bei längeren Skriptnamen so etwas wie INDEX_~1.CGI als URL in der JavaScript-Funktion.

Wenn man durch den GET-Request des XMLHttpRequest-Objekts eine Funktion des CGI-Skripts aufruft, die nicht existiert, bekommt man einen "Internal Server Error" und in den Log-Dateien findet sich eine Meldung "POSSIBLE SECURITY INCIDENT!". Der Hinweis ist nicht so ganz aussagekräftig, aber wenn man es einmal weiß, findet sich der Fehler ganz schnell.

Bei der Verwendung von CGI::Application muss man etwas aufpassen. Da die Perl-Subroutinen direkt ausgeführt werden, steht das Objekt von CGI::Application nicht zur Verfügung. Deshalb muss man auf Codereferenzen verzichten und stattdessen eine "externe" URL angeben.

CGI::Ajax - Beispiel

Ein ganz einfaches Beispiel ist in Listing 1 gezeigt.
Was wird da gemacht? In den Zeilen 3-5 werden die benötigten Module geladen. strict sollte in jedem Perl-Programm enthalten sein, um viele Fehler von vornherein auszuschließen.CGI.pm wird für verschiedene kleine Aufgaben wie Herausfinden der eigenen URL und so weiter verwendet.

In Zeile 8 wird das Mapping gemacht. CGI::Ajax erzeugt eine JavaScript-Funktion jsecho, in der mit einem XMLHttpRequest-Objekt ein GET-Request gemacht wird. Die Funktion, die im CGI-Skript ausgeführt werden soll, heißt perlecho.

Die HTML-Ausgabe und die Ausgabe der Header wird in Zeile 10 gemacht. build_html erzeugt den JavaScript-Code, der in den head-Bereich des HTML-Codes eingefügt wird.

Wichtig ist noch Zeile 20, in der gesagt wird, dass nach einem Tastendruck die Funktion jsecho aufgerufen werden soll.

```
 1 #!/usr/bin/perl -w
 2
 3 use strict;
 4 use CGI;
 5 use CGI::Ajax;
 6
 7 my $cgi = CGI->new();
 8 my $ajax = CGI::Ajax->new(jsecho => \&perlecho);
 9
10 print $ajax->build_html( $cgi, html() );
11
12 sub perlecho{
13     my ($value) = @_;
14     return $value;
15 }
16
17 sub html{
18     return q~
19 <html><body>
20     <input type="text" id="echofield" onkeyup="jsecho(['echofield'],['test'])">
21     <div id="test"></div>
22 </body></html>
23     ~;
24 }
```

Listing 1

Das `'echofield'` sagt aus, dass der Wert des HTML-Elements mit der ID `'echofield'` an den Server übergeben werden soll. Das Ergebnis wird dann in dem HTML-Element mit der ID `'test'` dargestellt.

Zwischenbilanz CGI::Ajax

Das Modul ist sehr gut für relativ einfache Ajax-Anwendungen geeignet. Sobald es in andere Frameworks integriert werden soll, ist "Spieltrieb" gefordert. Nicht alles ist dokumentiert, aber viele andere hatten wahrscheinlich schon ähnliche Probleme und die Lösungen sind im Internet zu finden.

HTML::Prototype

Prototype ist ein sehr bekanntes Framework für Ajax. Auch hierfür gibt es auf CPAN ein entsprechendes Modul, das den Programmierer bei seiner Arbeit unterstützt. Laut Dokumentation ist es zwar hauptsächlich für die Verwendung in Catalyst gedacht, aber das Modul kann auch mit anderen Webframeworks verwendet werden - wie hier am Beispiel von CGI::Application gezeigt wird.

Das JavaScript der Prototype-Bibliothek wird direkt mit dem Modul mitgeliefert. Man kann es dann in die HTML-Ausgabe auf zwei unterschiedliche Wege einbinden. Zum einen kann das JavaScript in jede HTML-Ausgabe direkt geschrieben werden, oder das JavaScript wird erst in eine eigene Datei geschrieben und dann mit dem <link>-Tag in das HTML eingebunden. Die erste Variante hat den Vorteil, dass immer das aktuelle JavaScript verwendet wird, die zweite Variante hat einen Geschwindigkeitsvorteil, da nicht immer das komplette JavaScript mit heruntergeladen werden muss.

Mit `HTML::Prototype` kann das Schreiben von JavaScript komplett vermieden werden, was für JavaScript-Einsteiger sicherlich von Vorteil ist.

Die Prototype-Bibliothek wird mit

```
print $prototype
    ->define_javascript_functions;
```

ausgegeben.

Die meisten Methoden von `HTML::Prototype` liefern den JavaScript-Bereich für die Ajax-Funktionalität, während die HTML-Elemente im Template definiert werden müssen. Es gibt allerdings einige Ausnahmen - die Funktionen, die einen JavaScript-Event-Handler verwenden (zum Beispiel `link_to_remote`), liefern das komplette HTML-Element mit dem zugehörigen JavaScript.

Das Beispiel von oben sieht mit HTML::Prototype aus wie in Listing 2.

Der wichtige Teil ist von Zeile 16 bis 25. Davor wird ein neues CGI-Objekt erzeugt, der Header ausgelesen und die Parame-

ter geparst. Wenn es einen ‘action‘-Parameter gibt, wird der Wert des Parameters ‘test‘ zurückgeliefert. Das ist die Aktion, die die Ajax-Funktionalität abfängt und auf der Seite darstellt.

Ein observe_field ist ein Textfeld, dass in gewissen Abständen - über ‘frequency‘ einstellbar - die URL aufruft, die mit dem Parameter ‘url‘ festgelegt wird. Der erste Parameter für das observe_field ist die ID des Textfeldes, dessen Inhalt “überwacht“ werden soll. Über das ‘with‘ wird festgelegt, was an die URL übergeben werden soll.

In Zeile 22 wird die Prototype-Bibliothek in eine Variable gespeichert. Die HTML-Ausgabe wird in Zeile 23 gemacht. Dabei wird das gesamte JavaScript in den head-Bereich des HTML geschrieben und die JavaScript-Funktion, die das Textfeld “überwacht“, wird unterhalb des Feldes eingesetzt.

Genauso übersichtlich und einfach sind alle weiteren Funktionen des Moduls nutzbar. Mittlerweile ist ein Großteil der Prototype-Funktionen auch in dem Modul umgesetzt.

Fallen von HTML::Prototype

Trotz der umfangreichen Dokumentation ist es an manchen Stellen etwas unklar, wie man sein Ziel erreichen kann. AnnoCPAN ist hier eine ganz gute Hilfe. Wenn Parameter an eine URL übergeben werden soll (zum Beispiel `http://localhost/script.cgi?action=test`), muss bei den Optionen der Parameter `with` eingestellt werden. Dabei ist darauf zu achten, dass ein ein gequoteter String im JavaScript ankommt. Deshalb sieht es zum Beispiel so aus:

```
with    => "'action=show_detail;id=$id'",
```

Zwischenbilanz HTML::Prototype

Mit diesem Modul kann man die Arbeit mit JavaScript auf ein absolutes Minimum reduzieren. Allerdings hat dieses Modul den Nachteil, dass die Dokumentation an einigen Stellen sehr dürftig ist. Viele Sachen müssen einfach ausprobiert werden, aber gerade für Anfänger ist es schwierig, alle Funktionalitäten zu nutzen.

Fazit

Mit `CGI::Ajax` gibt es ein Modul, das für Ajax-Einsteiger sehr gut geeignet ist. Die Einbindung in Frameworks ist nicht optimal, aber mit ein paar kleinen Tricks funktioniert auch das ganz annehmbar.

Wer sich mit Ajax und der Prototype-Bibliothek auskennt, der wird sich mit `HTML::Prototype` anfreunden können. Die dürftige Dokumentation wird durch die umfangreiche Funktionalität wieder wettgemacht.

Renée Bäcker

```
 1 #!/usr/bin/perl -w
 2
 3 use strict;
 4 use CGI;
 5 use HTML::Prototype;
 6
 7 my $cgi = CGI->new();
 8 print $cgi->header();
 9
10 my %params = $cgi->Vars;
11
12 if( $params{action} ){
13     print $params{test};
14 }
15 else{
16     my $field = $pt->observe_field('testid',{
17                         update => 'view',
18                         url    => 'html_prototype_test.cgi',
19                         with   => "'action=echo;test='+value",
20                         frequency => 1,
21                   });
22     my $proto = $pt->define_javascript_functions;
23     print qq~<html><head>$proto</head>
24       <body><input type="text" id="testid"><div id="view"></div>
25       $field</body></html>~;
26 }
```

Listing 2

There's More Than One Way to Get There

Conference Theme
Social Perl

Getting to

YAPC
EU::2007

http://vienna.yapceurope.org

28th to 30th August 2007

Vienna, Austria

at the

University of Economics and Business Administration

VIENNA'PM

Larry Wall, Damian Conway,
Mark Jason Dominus
and many more

Existiert eine Subroutine?

In der Programmierung kommt es häufiger vor, dass man wissen möchte ob eine Subroutine überhaupt existiert bevor man sie ausführt. Das ist vor allem dann von Bedeutung, wenn man auf fremden Code, wie zum Beispiel Plugins, zugreifen will. Hier werden einfach zwei Möglichkeiten gezeigt, wie man diese Überprüfung gestalten kann.

defined &sub

```
1 #!/usr/bin/perl
2
3 use strict;
4
5 print "yes hallo" if defined &Test::hallo;
6 print "yes test" if defined &Test::test;
7 print "yes, can do it"
8     if defined &can _ do _ it;
9
10 sub can _ do _ it{ 1 }
11
12 package Test;
13
14 sub test{ 1 }
```

Eigentlich ist man es gewohnt, dass `&Test::test` die Subroutine ausführt, aber bei der Überprüfung mit `defined` passiert das nicht.

can

Der zweite Weg ist `can` aus dem Modul `UNIVERSAL`. Das wird meistens bei Klassen und Objekten verwenden.

```
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5
6  print "yes hallo" if Test->can('hallo');
7  print "yes test" if Test->can('test');
8
9  package Test;
10
11 sub test{ 1 }
```

can kann auf drei verschiedene Wege aufgerufen werden. Wie gezeigt über die Klasse, dann geht das auch mit dem Objekt - also `$obj->can('test')` oder als Funktion `can( VAL, METHOD)`. Der Aufruf als Funktion ist allerdings nicht empfohlen (siehe auch `UNIVERSAL::can`). Auch für das "Hauptprogramm" funktioniert es:

```
if( main->can('can _ do _ it') ){
  print "yes, can do it\n" ;
}

sub can _ do _ it{ 1 }
```

Als Rückgabewert bekommt man eine Referenz auf die Subroutine, so dass dann auch die die Subroutine ausgeführt werden kann:

```
my $ref = Test->can('test');
if( $ref ){
     $ref->();
}
```

Diese Methoden funktionieren jedoch nur dann, wenn die Subroutine tatsächlich existiert. Stellt eine Klasse oder ein Objekt eine Funktionalität über AUTOLOAD zur Verfügung, dann wird das mit diesen beiden Methoden nicht erkannt.

Renée Bäcker

Neue Perl-Podcasts

Perlcast.com

brian d foy über Benchmarks

Dieser Perlcast ist kein Interview, sondern eine Aufnahme eines Perlmonger-Meetings von LosAngeles.pm. Es ist also ein ziemlich langer Perlcast! Gleichzeitig ruft Josh McAdams Perlmonger-Gruppen auf, Vorträge aufzunehmen und ihm zu schicken.

brian d foy über Benchmarks (Teil II)

Auch diese Aufnahme ist kein Interview, sondern die MP3-Version des Lightning Talks von brian d foy auf dem "Nordic Perl-Workshop".

Curtis "Ovid" Poe über "Testing with Perl"

Josh McAdams hat auf dem Nordic Perl-Workshop mehrere Interviews geführt. Eines davon mit Ovid über Testen mit Perl. In dem Interview geht es hauptsächlich um das "Test Anything Protocol" (TAP). Ovid stellt dabei auch einige Neuerungen bei der Ausgabe von Testergebnissen vor.

"DBIx::Class" mit Matt S Trout

Wer SQL in seinen Programmen vermeiden möchte, greift häufig zu einem Modul, das sogenanntes Object-Relational-Mapping beherrscht. DBIx::Class ist ein solches Modul. Matt S Trout erzählt einiges über die Hintergründe von DBIx::Class.

"Lerning Perl6" von brian d foy

Eine weitere Aufnahme vom Nordic Perl-Workshop: brian d foy hält einen Vortrag über "Lerning Perl6". In seinem 30-Minuten-Vortrag geht foy auf die Syntax von Perl6 ein. Dabei geht es nur um grundlegende Dinge wie Variablen, Schleifen und so weiter.

CPAN News - 6 neue Module

Lyrics::Fetcher::LyricWiki

... ist ein Modul für Musikfreunde oder diejenigen, die gerne im Auto oder unter der Dusche singen. Doch wie war doch gleich der Text des Liedes? Was singen die in der 3. Strophe? Mit diesem Modul kann man einfach die Songtexte runterladen. Leider finden sich viele aktuellen Songtexte nicht in dem Wiki.

Hier aber mal ein Beispiel für ein Lied, das mir ganz gut gefällt. Die Zeile 10 kann man auch etwas anders gestalten, wenn der "Fetcher" flexibler gestaltet werden soll - und eventuell noch von anderen Quellen die Texte bezogen werden. Dann muss es `use Lyrics::Fetcher` heißen und `Lyrics::Fetcher->fetch` dem `fetch` muss dann als letzter Parameter noch mittels `'LyricWiki'` mitgeteilt werden, dass der Songtext aus dem LyricWiki geholt werden soll.

```
1 #!/usr/bin/perl
2
3 use strict;
4 use warnings;
5 use Lyrics::Fetcher::LyricWiki;
6
7 my $band='Pink';
8 my $song='Dear Mr. President';
9
10 my $text=Lyrics::Fetcher::LyricWiki->fetch(
11     $band,
12     $song,
13 );
14
15 print $text;
```

GD::Chart::Radial

Nicht nur Manager mögen Charts, denn häufig sagt ein Bild mehr als 1000 Worte. Wer Charts automatisiert erstellen möchte, kann auf die verschiedenen GD::Chart::*-Module zurückgreifen. Auf CPAN ist auch eine neue Version vom Modul GD::Chart::Radial zu finden, mit dem sich sehr gut Netzdiagramme erstellen lassen. Mit dem Beispielcode wird ein Diagramm mit drei Achsen und zwei Linien gezeichnet. Die erste Arrayreferenz in `@data` gibt an, was die Maximalwerte der Achsen sind. Danach wird für jede Linie eine Arrayreferenz angegeben. Das Modul kann noch weitere Arten von Diagrammen darstellen, was über die `set`-Methode mit dem Parameter `style` eingestellt wird.

```
1 #!/usr/bin/perl
2
3 use strict;
4 use warnings;
5 use GD::Chart::Radial;
6
7 my @data  = ([4   , 3.5, 3   ], #axis
8              [1.2, 3.1, 2.2], #red
9              [2.3, 1.5, 0.3], #blue
10 );
11 my $chart = GD::Chart::Radial->new(400,400);
12 $chart->set( title => 'Testchart',);
13 $chart->plot(\@data);
14
15 open my $fh, '>', 'testchart.png' or die $!;
16 binmode $fh;
17 print $fh $chart->png;
18 close $fh;
```

Image::Magick::Thumbnail::PDF

In manchen Fällen braucht man eine kleine Abbildung einer PDF-Datei. Für solche Fälle ist genau dieses Modul gemacht. Man benötigt dazu ImageMagick und ein paar kleinere Module und dann kann es losgehen.

Der Beispielcode zeigt, wie man ein Vorschaubild von der ersten Seite eines PDF-Dokuments bekommt. Man kann dem Modul aber auch mitteilen, von welcher Seite das Bild erstellt werden soll. Über das `restriction` wird die maximale Seitenlänge eingestellt.

In der Dokumentation steht, dass ein absoluter Pfad zur PDF-Datei angegeben werden muss, in einem Test hat es aber auch mit relativen Pfaden funktioniert.

```
1 #!/usr/bin/perl
2
3 use strict;
4 use warnings;
5 use Image::Magick::Thumbnail::PDF
6       'create_thumbnail';
7
8 my $pdf  = 'Fraunhofer_Abstract.pdf';
9 my $path = create_thumbnail(
10     $pdf,
11     { restriction => 300 },
12 );
13 print $path;
```

Params::Check::Faster

Benutzereingaben sollte man niemals trauen. Eine Überprüfung ist daher sehr wichtig. Ein Modul, das man dazu verwenden kann, ist Params::Check::Faster. Dazu legt man in einem Hash an, in dem alles notwendige festgelegt wird: Welche Werte wird es geben, was ist erlaubt, soll es in eine Variable gespeichert werden? In einem zweiten Hash sind die Benutzereingaben gespeichert. Beides wird der Funktion check übergeben. Der Umgang mit Fehlern ist etwas gewöhnungsbedürftig bzw. die Fehlermeldungen helfen nicht immer weiter, aber das entwickelt sich hoffentlich noch etwas.

```
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5  use Params::Check::Faster
6     qw(check last_error);
7  use Data::Dumper;
8
9  my $magazine;
10 my %profile = (
11     abotypes => {
12       default => 'JahresAbo',
13       allow   => ['JahresAbo', 'OnlineAbo' ],
14     },
15     magazine  => {
16       allow    => qr/Perl/,
17       store    => \$magazine,
18     },
19 );
20
21 my %input = (
22     abotypes => 'JahresAbo',
23     magazine  => 'The Perl Review',
24 );
25
26 my $var = check( \%profile,
27         \%input, 1 ) or print last_error();
28 print Dumper($var), " && ", $magazine;
```

Parse::BooleanLogic

Ich habe zwar noch keine richtige Verwendung für das Modul gefunden, aber es gibt sicherlich den ein oder anderen Anwendungsfall, bei dem es nützlich werden könnte. Das Modul parst einen Ausdruck wie `X AND ( Y=10 OR X=5 )` und erstellt daraus einen Baum. Es können auch Subroutinen angegeben werden, die bei bestimmten "Events" wie "öffnende Klammer" ausgeführt werden sollen.

Leider ist die Dokumentation nicht wirklich gut, und es sind noch ein paar Fehler enthalten. Durch Testen und einen Blick in das Testskript findet man aber heraus, wie das Modul angewendet werden muss.

```
1 #!/usr/bin/perl
2
3 use strict;
4 use warnings;
5 use Parse::BooleanLogic;
6 use Data::Dumper;
7
8 my $string='X < 8 AND (Y = 10 OR Z = 5)';
9 my $parser=Parse::BooleanLogic->new();
10 my $tree = $parser->as_array( $string );
11 print Dumper( $tree );
```

Games::Sudoku::Solver

Wer seine Kumpels beeindrucken will, wie schnell man doch auch sehr schwierige Sudokus lösen kann, sollte dieses Modul installieren. Es löst die Rätsel rekursiv. Allerdings geht so der Spass am Sudoku-Rätseln verloren.

```
1 #!/usr/bin/perl
2
3 use strict;
4 use warnings;
5 use Games::Sudoku::Solver qw(:Minimal);
6
7 my @sudoku = qw(
8     4 7 0 0 0 8 0 0 6
9     0 9 6 0 5 4 1 7 0
10     8 0 2 6 9 0 0 0 0
11     0 0 0 0 0 1 0 0 4
12     1 0 0 0 0 0 0 2 9
13     0 5 0 7 0 0 0 3 0
14     0 8 0 0 1 0 0 5 0
15     5 6 4 3 0 2 0 0 0
16     0 0 0 4 0 0 0 6 0
17 );
18
19 my (@field, @solution);
20
21 sudoku_set( \@field, \@sudoku );
22
23 my $anzahl = sudoku_solve( \@field,
24     \@solution );
25 for( 0..$anzahl-1 ){
26     sudoku_print $solution[$_];
27 }
```

Termine

August 2007

08.	Treffen Hamburg.pm
	Treffen Cologne.pm
13.	Treffen Ruhr.pm
16.	Treffen Darmstadt.pm
	Treffen Erlangen.pm
23.	Perl Best Practices (ETH Zürich)
24.	Perl Best Practices (ETH Zürich)
25.	FrOSCon 2007
	Perl 6 Update (ETH Zürich)
26.	FrOSCon 2007
28.	Treffen Bielefeld.pm
	YAPC::Europe 2007 (Wien)
29.	Treffen Berlin.pm
	YAPC::Europe 2007 (Wien)
30.	YAPC::Europe 2007 (Wien)

September 2007

04.	Treffen Frankfurt.pm
	Treffen Stuttgart.pm
06.	Treffen Dresden.pm
10.	Treffen Ruhr.pm
12.	Treffen Hamburg.pm
	Treffen Cologne.pm
19.	RailConf 2007 (Berlin)
	Treffen Darmstadt.pm
	Treffen Erlangen.pm
25.	Treffen Bielefeld.pm
26.	Treffen Berlin.pm

Hier finden Sie alle Termine rund um Perl.

Natürlich kann sich der ein oder andere Termin noch ändern. Darauf haben wir (die Redaktion) jedoch keinen Einfluss.

Uhrzeiten und weiterführende Links zu den einzelnen Veranstaltungen finden Sie unter

http://www.perlmongers.de

Kennen Sie weitere Termine, die in der nächsten Ausgabe von $foo veröffentlicht werden sollen? Dann schicken Sie bitte eine EMail an:

termine@foo-magazin.de

Oktober 2007

02.	Treffen Frankfurt.pm
	Treffen Stuttgart.pm
04.	Treffen Dresden.pm
08.	Treffen Ruhr.pm
10.	Treffen Hamburg.pm
	Treffen Cologne.pm
18.	Treffen Darmstadt.pm
	Treffen Erlangen.pm
30.	Treffen Bielefeld.pm
31.	Treffen Berlin.pm

LINKS

http://www.Perl-community.de

Perl-Community.de ist eine der größten deutschsprachigen Perl-Foren. Hier ist aber nicht nur ein Forum zu finden, sondern auch ein Wiki mit vielen Tipps und Tricks. Einige Teile der Perl-Dokumentation wurden ins Deutsche übersetzt. Auf der Linkseite von Perl-Community.de findet man viele Verweise auf nützliche Seiten.

http://www.pm.org

Das Online-Zuhause der Perl-Mongers. Hier findet man eine Übersicht mit allen Perlmonger-Gruppen weltweit. Außerdem kann man auch neue Gruppen gründen und bekommt Hilfe...

http://www.perl-workshop.de

Der Deutsche Perl-Workshop hat sich zum Ziel gesetzt, den Austausch zwischen Perl-Programmierern zu fördern. Der 10. Deutsche Perl-Workshop findet im Februar 2008 in Erlangen statt.

http://www.perl-foundation.org

Die Perl-Foundation nimmt eine führende Funktion in der Perl-Gemeinde ein: Es werden Zuwendungen für Leistungen zugunsten von Perl gegeben. So wird z.B. die Bezahlung der Perl6-Entwicklung über die Perl-Foundationen geleistet. Jedes Jahr werden Studenten beim “Google Summer of Code“ betreut.

http://www.Perl.org

Auf Perl.org kann man die aktuelle Perl-Version downloaden. Ein Verzeichnis mit allen möglichen Mailinglisten, die mit Perl oder einem Modul zu tun haben, ist dort zu finden. Auch Links zu anderen Perl-bezogenen Seiten sind vorhanden.

// SEIBERT / MEDIA

„Statt mit blumigen Worten umschreiben unsere Programmierer den Job so:
Apache, Catalyst, CGI, DBI, JSON, Log::Log4Perl, mod_perl, SOAP::Lite, XML::LibXML, YAML“

Professionell Perl programmieren lernen?

Wir suchen Einsteiger als Perl-Programmierer/innen (Vollzeit/Praktikum)

//SEIBERT/MEDIA besteht aus den drei Kompetenzfeldern Technologie, Consulting und Design und gehört zu den erfahrenen und professionellen Internet-Agenturen in Deutschland. Wir entwickeln seit 1996 mit heute knapp 50 Mitarbeitern Intranets, Extranet-Systeme, Web-Portale aber auch klassische Internet-Seiten. Seit 2005 konzipiert unsere Designabteilung hochwertige Unternehmensauftritte und kommunikative Konzepte. Beratung im Bereich Online-Marketing und Usability runden das Leistungsportfolio ab.

Ihre Aufgabe wird sein, in unserer Entwicklungsabteilung im Team komplexe E-Business Applikationen zu entwickeln. Dabei ist objektorientiertes Denken genauso wichtig, wie das Auffinden individueller und innovativer Lösungsansätze, die gemeinsam realisiert werden.

Wir freuen uns auf Ihre Bewerbung unter http://www.seibert-media.net/jobs/.

//SEIBERT/MEDIA GMBH
RHEINGAU PALAIS / SÖHNLEINSTRASSE 8
65201 WIESBADEN
T. +49 611 20570-0 / F. +49 611 20570-70
BEWERBUNG@SEIBERT-MEDIA.NET
HTTP://WWW.SEIBERT-MEDIA.NET/JOBS/

FREAKZEIT im Linuxhotel

DAS Wochenend-Reiseziel für alle, die Spaß an Computern haben

Wo andere nach Hamburg ins Musical oder in den Schwarzwald zur Wellness fahren, treffen sich Computerfreaks nun jedes Wochenende im Linuxhotel, um zusammen mit Gleichgesinnten am PC zu arbeiten.

Es erwartet Sie eine ruhige, konzentrierte Umgebung mit Leihnotebooks zum Testen kritischer Installationen, Seminarraum, Technik, Bibliothek, Probeservern, WLAN, 230V-Steckdosen selbst im 25.000qm großen Privatpark und vielem mehr. Auch Vollpension und Getränke sind enthalten, bei schönem Wetter wird gegrillt, wenn's kalt ist, geht es in die Sauna. Fahrräder stehen bereit, kleine Modellflieger, Fitnessraum, eine Wii, und vor allem all' der Kleinkram, den man für erfolgreiche Computerarbeiten braucht.

Warum nicht 'mal ein Wochenende unter Computerfreaks? Jeder arbeitet an seinem Thema, aber man schaut sich zwanglos über die Schultern und hilft sich gegenseitig!

Die Freakzeit-Wochenenden sind ein Experiment, denn eigentlich lebt das Linuxhotel von sehr zufriedenen Unternehmen und Organisationen, die ihre Mitarbeiter bei uns schulen lassen (wir haben eines der breitesten Kursprogramme in Deutschland, siehe www.Linuxhotel.de).

Doch die IT-Branche ist etwas Besonderes! Viele Profis haben wirklich Spaß an ihrer Arbeit, und es ist nicht einzusehen, warum z.B. Musical-Freunde oft mehrfach pro Jahr von weit her zu ihren Theatern fahren, doch für Computerfreaks gibt es nichts Vergleichbares!

Das Linuxhotel soll der Ort werden, an dem man jedes Wochenende interessante Leute aus unserer Branche zwanglos treffen kann. Machen Sie mit, schnappen Sie sich Ihr Notebook, und melden sich für irgendeines der kommenden Wochenenden an! Siehe www.Linuxhotel.de und auf „Freakzeit“ klicken.