Ausgabe 1/2010 – www.foo-magazin.de - 6,00 €

$foo

PERL MAGAZIN

Performance großer Webseiten
Wer langsam ist, wird verlassen

Visualisierungen
mit Perl und ImageMagick

Lotus Notes/Domino
Perl(en) mit Java

Nr 13

TPF

The Perl Foundation

- Perl und Enterprise-Anwendungen
- Perl-Projekte (Foswiki, OTRS, u.a.)
- u.v.m

CeBIT

02.-06. März 2010

Halle 2

Open Source Project Lounge

VORWORT

2010 kann kommen!

Marketing von Perl...

... wird in diesem Jahr stärker betrieben als jemals zuvor. Im Oktober 2009 wurde bei der Perl Foundation ein Marketing Komitee gegründet (siehe auch das Interview mit Karen Pauley), das unter anderem für das "Marketing" verantwortlich ist.

Ziel ist es, dass Perl wieder mehr in der Öffentlichkeit erscheint und zu zeigen, wie sexy Perl ist - oder sein kann.

Auf der CeBIT wird die Perl Foundation in der "Open Source Project Lounge" vertreten sein. Damit wird mal ein ganz anderes Publikum angesprochen als es bisher üblich war. Perl ist ja auch für Business Anwendungen geeignet. XING ist so ein Beispiel, dass auch bei sehr großen Anwendungen Perl eingesetzt werden kann - und wir sind froh, dass Dr. Johannes Mainusch von XING einen Artikel für diese Ausgabe beigesteuert hat.

Aber nicht nur die Präsenz auf Konferenzen und Messen spielt beim Marketing eine Rolle - auch das Aussehen von Webseiten. Leo Lapworth hat mit einem Team (fast) alle Seiten unter perl.org neu gestaltet und diesen einen schöneres Aussehen verpasst.

Wer beim Marketing von Perl mitmachen möchte, kann sich auf der Mailingliste des Marketing Komitees unter tpf-marketing-subscribe@perl.org eintragen.

*Release von Rakudo **

Ein weiteres Highlight in diesem Jahr dürfte das Release von Rakudo * (Perl 6 auf Parrot) sein. Damit dürften noch einige Programmierer mehr mit Perl 6 "spielen".

Das wird hoffentlich zu mehr Feedback an die Kernentwickler um Patrick Michaud und Jonathan Worthington führen.

Man darf gespannt sein, wie sich das entwickelt...

Aber jetzt wünsche ich Ihnen erstmal viel Spaß mit der 13. Ausgabe von "$foo - Perl-Magazin" und ein frohes Neues Jahr.

Renée Bäcker

Die Codebeispiele können mit dem Code

ncje2n

von der Webseite www.foo-magazin.de heruntergeladen werden!

Alle weiterführenden Links werden auf del.icio.us gesammelt. Für diese Ausgabe: http://del.icio.us/foo_magazin/issue13.

IMPRESSUM

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

INHALTSVERZEICHNIS

ALLGEMEINES

ANWENDUNG

PERL

MODULE

INTERVIEW

NEWS

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 Workshop und Mitglied im Orga-Team des deutschen Perl-Workshops

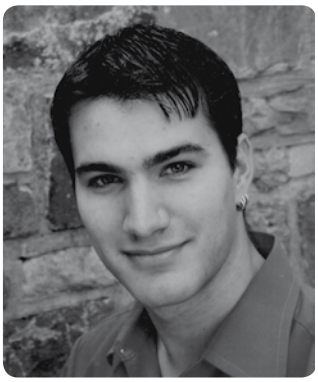
Ferry Bolhár-Nordenkampf

Ferry kennt Perl seit 1994, als sich sein Dienstgeber, der Wiener Magistrat, näher mit Internet-Technologien auseinanderzusetzen begann und er in die Tiefen der CGI-Programmierung eintauchte. Seither verwendet er – neben clientseitigem Javascript – Perl für so ziemlich alles, was es zu programmieren gibt; auf C weicht er nur mehr aus, wenn es gar nicht anders geht – und dann häufig auch nur, um XS-Module für Perl schreiben. Wenn er nicht gerade in Perl-Sourcen herumstöbert, schwingt er das gerne das Tanzbein oder den Tennisschläger.

Herbert Breunung

Ein perlbegeisteter Programmierer aus dem ruhigen Osten, der eine Zeit lang auch Computervisualistik studiert hat, aber auch schon vorher ganz passabel programmieren konnte. Er ist vor allem geistigem Wissen, den schönen Künsten, sowie elektronischer und handgemachter Tanzmusik zugetan. Seit einigen Jahren schreibt er an Kephra, einem Texteditor in Perl. Er war auch am Aufbau der Wikipedia-Kategorie: "Programmiersprache Perl" beteiligt, versucht aber derzeit eher ein umfassendes Perl 6-Tutorial in diesem Stil zu schaffen.

Alexander Becker



Alexander Becker befindet sich in den letzten Zügen seines Computerlinguistik-Studiums und schreibt seit dem zweiten Millennium CGI-Skripten mit dem MVC-Framework CGI::Application (neuerdings auch Titanium genannt). Eine Hälfte jeder Woche arbeitet er für eine IT-Firma in Luxemburg. Den Rest der Zeit vertreibt er sich wahlweise mit Kursen an der Uni, ersten Versuchen als CPAN-Autor oder dem Verdreschen von Beachvolleyballbällen.

Arnd Koch

Der 31 jährige Kieler ist Programmierer aus Leidenschaft, seit 1997 schreibt er Software und hat dabei im Laufe der Jahre eine besondere Vorliebe für Perl entwickelt. Er arbeitet als Software-Entwickler bei der assono GmbH (http://www.assono.de) vorwiegend an IBM Lotus Notes/Domino basierenden Lösungen.

Max Kossatz

geb. 1970, war Mitte der 90er Jahre in New York federführend am Aufbau eines Internetproviders für Kulturschaffende beteiligt. Seit seiner Rückkehr nach Österreich war er in verschiedenen Aufsichtsrats-, Management- und Geschäftsführungspositionen im Bereich Internet, Mobile Services sowie Retail tätig. Studium an der Universität für Angewandte Kunst, Peter Weibel. Teilnehmer an verschiedenen nationalen und internationalen Ausstellungen, unter anderem der Biennale in Venedig. Max Kossatz ist seit 2005 Mitgründer, Mitbesitzer und Geschäftsführer der Domsich Kossatz & Steinberger Beratungs OEG und betreibt unter „Wissen belastet“ (http://wissenbelastet.com) einen privaten Weblog. Domsich Kossatz & Steinberger Beratungs OEG beschäftigt sich mit Markt und Meinungsbeobachtung im Internet durch Analyse von Webseiten, Foren, Blogs und Social Communities.

Dr. Johannes Mainusch

Dr. Johannes Mainusch, Vice President Operations, XING AG. Als Vice President Operations betreut Dr. Johannes Mainusch den Betrieb und die technische Entwicklung der XING-Plattform, einer High-Traffic Webseite mit über 8.3 Millionen Mitgliedern (Stand 11/2009) weltweit. Mit dem Fokus auf iterativen Projektmethoden war er zuvor sowohl für Lufthansa-Systems als auch beratend für zahlreiche Industrie- und Logistikunternehmen tätig.

Wer langsam ist, wird verlassen
Performance großer Webseiten
- Teil 1 -

Menschen werden von langsamen Webseiten frustriert und wenden sich ab. Dieser Artikel beschäftigt sich mit Performance von Webseiten und ihrer Wahrnehmung von Performance durch Menschen. Es werden Maßnahmen zur Verbesserung der Performance bei XING und deren Wirkung diskutiert. Dabei wird auf folgende Dinge eingegangen: Warum ist gute Performance wichtig? Wie wird die Verantwortung zur guten Performance richtig verankert? Wie wird die Performance richtig gemessen? Exemplarisch wird erklärt, welche Maßnahmen durchgeführt werden können.

Über XING

Seit über zwei Jahren arbeite ich bei XING und wundere mich immer wieder über die Größe der Plattform im Vergleich zu dem, was ich in den Jahren vorher in verschiedenen Unternehmen gesehen habe. XING ist einfach eine große Web-Applikation, mit über 200.000 Benutzern, die jede Stunde vorbeikommen. Das sind etwa 3 Fußballstadien voller Menschen, die auf jeden Klick auf der Website schnell eine Antwort haben möchten. Damit sind wir laut Alexa (http://www.alexa.com/topsites/countries/DE) eine der 20 größten Sites in Deutschland.

Abbildung 1: Die Kundennutzen der XING-Plattform.

XING ist das führende europäische Online-Business-Netzwerk. Die Idee: Jeder kann sicher seine berufsbezogenen Daten hinterlegen, seine Kontakte und sein Netzwerk pflegen, passende Jobs finden, Informationen zu Firmen recherchieren, sich mit Experten austauschen und vieles mehr. XING begleitet und unterstützt also die Karierre des jeweiligen XING-Mitglieds. Stand 11/2009 sind in XING über 8.300.000 Mitglieder registriert. Auch die Tatsache, dass Web-Technologie nicht bloß ein Kostenfaktor in einem großen Unternehmen sein muss, fasziniert mich immer wieder aufs Neue. XING finanziert sich überwiegend aus den monatlichen Beiträgen der über 660.000 Premium-Mitglieder. D.h. ich als IT-Mitarbeiter bin nicht ein Kostenfaktor, sondern zentraler Bestandteil eines Business-Modells, das sich trägt.

Ein Blick unter die Motorhaube - die Technik

XING basiert auf MySQL als Datenbank, Apache als Webserver, perl mit fastCGI und Ruby on Rails mit Phusion Passenger als Applikationsschicht, SOLR für die Suche. Der Kern der Plattform wurde vor über sechs Jahren als openbc.com entwickelt und seitdem ständig erweitert. Seit ca. drei Jahren werden viele neue Features in Ruby on Rails erstellt. Dazu gehören die Bereiche Jobs, Unternehmen, Events und einiges mehr.

Hier einmal einige Zahlen zu XING:

- über 260 Mitarbeiter, Firmenzentrale in Hamburg

- > 300.000 Zeilen Perl-Code
- > 40.000 Zeilen Ruby-on-Rails-Code
- XING-Engineering: 15 Perl-Entwickler, 25 Ruby on Rails Entwickler, 8 davon in Spanien, 12 Frontend-Entwickler, 7 Admins, 5 interne Admins, 12 QA-Mitarbeiter
- wir liefern unseren Inhalt aus zwei parallel arbeitenden Rechenzentren in Hamburg aus.

Wir releasen Mittwochs... Jeden Mittwoch. 50 Mal im Jahr. Die Menge Angst, die man pro Jahr von Releases hat ist eine Konstante! Wir haben diese Konstante durch 50 geteilt. Wir sind geübt. Wir haben das bewusst in die Arbeitszeit und damit auch in eine Zeit gelegt, in der viele Benutzer auf der Plattform sind, denn so können wir sofort und in voller Besetzung auf Mitgliederfragen antworten.

Wussten Sie als XING-Benutzer das? Besser nicht, denn das würde ja bedeuten, dass unsere Releases störungsfrei und somit unbemerkt geschehen. Klar gibt es noch manchmal Störungen, aber wir werden immer besser darin, die Releases ohne Ruckler hinzubekommen. Mut, Ausdauer und die schrittweise Verbesserung unsere Prozesse helfen uns und sind inzwischen Bestandteil unserer täglichen Arbeit...

Der Mensch und die Maschine - was geschieht vor dem Klick

Der Mensch und die Maschine interagieren, etwa ein Benutzer bei XING. Was sind die Erwartungen des Menschen im Umgang mit Technik? Um das zu erörtern stellen sie sich folgende Frage: Was ist das elektrische Gerät, das Sie und fast alle Menschen zuerst im Leben bedient haben?

Der Lichtschalter. Und diese erste Erfahrung im Leben prägt und setzt Maßstäbe. Sollte das Licht beim Umklappen des Lichtschalters ausbleiben, so meldet das Gehirn 'Licht kaputt!'. Diese Erkenntnis trifft das Gehirn innerhalb weniger Millisekunden. Ein spontanter Feldversuch mit meinem iPhone und einem Programm namens 'ReactionTest' ergaben, dass die von mir getesteten Probanden auf optische Signale innerhalb von 0,25 bis 0,3 Sekunden reagieren. Die reine Wahrnehmung von optischen Reizen dürfte dabei um einiges schneller sein, da bei dem von mir durchgef ührten Test auch eine mechanische Reaktion nötig ist (Tippen auf den iPhone-Bildschirm).

These: In der Mensch-zu-Mensch-Interaktion wird im Dialog für eine reibungsfreie Kommunikation eine Reaktionszeit von weniger als 0,25 Sekunden als Wert angesetzt, d.h., der Mensch erwartet von seinem Dialogpartner ein Response-Verhalten mit einer Antwortzeit von 0,25 Sekunden. Größere Werte wirken sich auf die Kommunikation störend aus.

Schlechte Performance nervt im wahrsten Sinne des Wortes. Unser Hirn beginnt mit der nächsten Signalverarbeitung, ist dann wieder verwirrt, missinterpretiert sogar im schlimmsten Falle die Signale als Störung oder löst Alarm aus.

Google und Bing haben das Verhalten von Benutzern bei schlechter Performance quantitativ untersucht und sind dabei zu folgenden erstaunlichen Ergebnis gekommen (siehe Abbildung 3).

Schon bei einer Verzögerung der Auslieferung der Web-Inhalte um 200ms, also einer Fünftelsekunde, verlassen Benutzer die Website. Und selbst nachdem diese künstlich eingeführte Kommunikationsstörung wieder beseitigt wurde, kehren nicht alle Benutzer zurück.

Fazit: Schlechte Performance beginnt schon bei sehr kleinen Verzögerungen im Bereich von 0,2 Sekunden und vertreibt Benutzer nachhaltig.

Abbildung 2: XING betreibt über 300 Server in zwei Rechenzentren. Ziel ist es, dieWebseiten dem Benutzer innerhalb von 2 Sekunden auszuliefern.

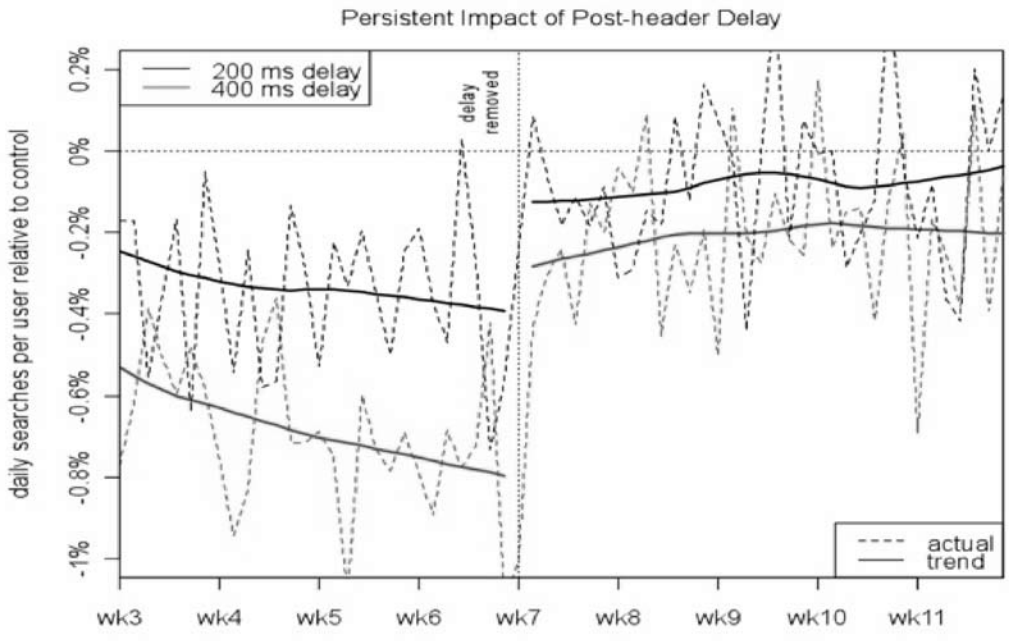


Abbildung 3:
Googles Test: Google hat in einem A/B-Testverfahren die Antworten aus Suchanfragen für Benutzergruppen im Vergleich zum Normalverhalten um 200 bzw. 400 Millisekunden verzögert ausgeliefert. Nach einigen Wochen zeigt sich, dass diese Benutzer beginnen, die google Site zu verlassen. Auch nachdem die künstliche Störung behoben wurde, kehren nicht alle ursprünglichen Benutzer zurück.

Nicht funktionale Anforderungen und ein neues Relevanzmaß für Webseiten

Traditionell wird die Relevanz von Webseiten, also die Nützlichkeit für den Benutzer, lediglich über die Inhalte auf der Seite definiert. Das spiegelt sich auch in der Produktion von Webseiten wider, Performance wird häufig nicht einmal in die nicht funktionalen Anforderungen von Produkten aufgenommen, oder falls Performance aufgenommen wird, dann nicht in einfach messbarer Form. Für die Definition einer Performance Anforderung für ein Web-Produkt hat sich folgende Formulierung bewährt:

Die ausgelieferten Webseiten soll im Browser beim Benutzer innerhalb von 2 Sekunden in 95% der Fälle erfolgen. Eine Seite gilt als ausgeliefert, wenn das Javascript-Event Dom-Ready ausgelöst wird. [1]

Dabei können die zwei Sekunden und die 95% durch die entsprechenden Wunschfaktoren ersetzt werden. Man bedenke immer, dass zwei Sekunden für einen Menschen eine sehr langsame Reaktionszeit darstellt. Der technische Aufwand kann hingegen schon sehr hoch sein, um wenigstens diese Anforderung zu erreichen. Die Listing 1 zeigt, dass wir bei XING von diesem Ziel noch ein Stück entfernt sind, derzeit (Stand 1.12.2009) werden 95% der Zugriffe innerhalb von fünf Sekunden abgearbeitet.

Wichtig ist, dass auch bei so einer Formulierung der Anforderungen das definierte Ziel SMART ist, also spezifisch, messbar, angemessen, realistisch, terminiert. Das bedeutet für Performance:

Spezifisch heißt in diesem Fall, dass die Performance im Browser des Benutzers gemessen wird, also nicht irgendwo im Rechenzentrum, sondern dort, wo der Kunde sitzt. Die Verantwortung für die Auslieferung am Browser des Benutzers wird besonders im Kunden-Lieferanten-Umfeld häufig mit dem Hinweis abgewiesen, dass letzte Strecke zum Benutzer nicht im Einflussbereich des Betriebes liegt. Dann kann es geschehen, dass dieses Ziel als 'nicht realistisch beeinflussbar' fallen gelassen wird. Das ist allerdings dann für die Performance fatal, wie wir später noch sehen werden.

Messbar bedeutet 95% in zwei Sekunden. Indem einfach (!) am Client später gezählt wird, wie viele Auslieferungen länger dauern als zwei Sekunden, kann diese Anforderung überprüft werden [2].

Angemessen entspricht 'herausfordernd/ehrgeizig'. Knackige Ziele helfen der Verbesserung. Zwei Sekunden am Client für 95% der Antworten ist für XING ein ehrgeiziges Ziel. Derzeit werden bei XING nur ca. 76% aller Zugriffe innerhalb von zwei Sekunden entsprechend der oben genannten Anforderung erbracht. Hier gibt es also noch einiges Verbesserungspotenzial.

Ladezeit	% Anteil ausgelieferter Seiten, 01/12/2009	dito, 09/12/2009
innerhalb 1 Sekunde	45.0%	47,2%
innerhalb 2 Sekunden	76.1%	89,0%
innerhalb 3 Sekunden	87,4%	93,2%
innerhalb 4 Sekunden	92,2%	95,5%
innerhalb 5 Sekunden	94,6%	97,0%
innerhalb 6 Sekunden	96,2%	97,8%
innerhalb 7 Sekunden	97,0%	98,3%
innerhalb 8 Sekunden	97,6%	98,6%
innerhalb 9 Sekunden	98,1%	98,8%
innerhalb 10 Sekunden	98,6%	99,0%

Listing 1

Listing 1: Verteilung der XING-Antwortzeiten über alle Browser und Webseiten, Stand 01/12/2009 und 09/12/2009. Sichtbar ist eine Verbesserung, die wir auf die Optimierung der Einbindung von Trackinpixeln und auf die native Unterstützung des Internet-Explorers 8 zurückführen. Der Anteil der XING Nutzer mit Internet-Explorer 8 liegt derzeit bei 16%.

Realistisch bedeutet 'erreichbar'. Somit wird über 'angemessen' und 'realistisch' ein Zielkorridor definiert. Der Bogen darf halt auch nicht überspannt werden.

Terminiert beantwortet dabei die Frage, bis wann das Ziel erreicht sein soll. In einem dynamisch wachsenden Umfeld mit wöchentlichen Änderungen ist das eine der am schwierigsten zu formulierenden Anforderungen, da jedes wöchentliche Release mit neuen Features auch neue Performance-Herausforderungen mit sich bringt. Bei XING planen wir dieses Ziel in 2010 zu erreichen.

So eine Performance-Anforderung ist nur zu erreichen, wenn im ganzen Unternehmen von der Produktentwicklung bis zum Betrieb, von der Strategie bis zum Administrator bei allen Beteiligten die entsprechende Awareness geschaffen wird. Als Hilfsmittel dafür plädiere ich für ein neues Relevanzmaß für Webseiten, dass sich nicht nur auf die angebotenen Inhalte bezieht, sondern auch die Auslieferungsgeschwindigkeit berücksichtigt. Mathematisch könnte man das folgendermaßen festhalten:

$$\text{Relevanz} \sim \frac{\text{Inhalt}}{\text{Auslieferungsgeschwindigkeit}}$$

Formel 1

Im Klartext: **Je schneller die Webseiten ausgeliefert werden, desto relevanter erscheinen sie dem Benutzer.** 1999 hat eine kleine Firma namens Google den Suchgiganten Altavista zunächst in die Schranken verwiesen und dann in die Bedeutungslosigkeit verbannt. Und dass, was heute noch den meistem Personen in Erinnerung daran ist, ist die enorme Geschwindigkeit, mit der die Inhalte auf dem Bildschirm erschienen. Während eine einfache Suche bei Altavista gut und gern fünf Sekunden dauern konnte, hatte Google schon innerhalb einer Sekunde das Ergebnis parat.

Auch Google hat sich inzwischen dazu entschlossen, die Auslieferungszeit als ein wesentliches Kriterium für das Ranking der Seiten zu verwenden. Damit wird die Web-Performance ganz automatisch zu einem wesentlichen Faktor für Webseiten mit gutem Ranking.

Browser und Server - Was geschieht nach dem Klick

Was passiert nach dem Klick im Browser? Klar, die zum Hyperlink gehörende Webweite erscheint... Nach einiger Zeit...

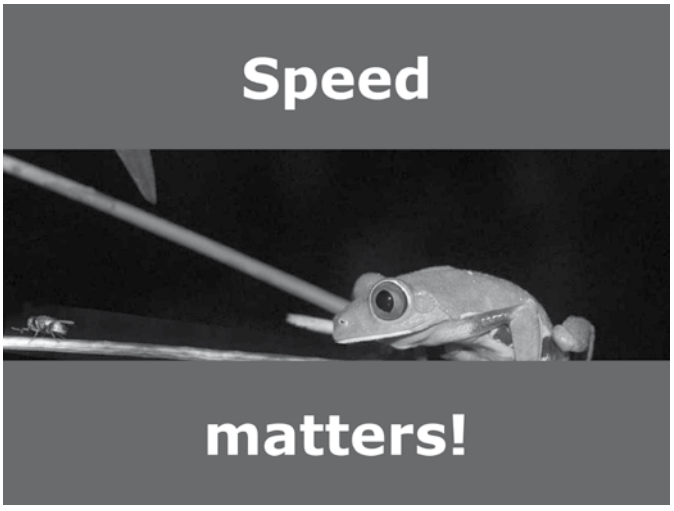


Abbildung 4: Speed matters - nicht nur in der Natur...
(Quelle: sxc.hu, Montage: Dr. Johannes Mainusch

```
1
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML ...>
3 <html xmlns=\"http://www.w3.org/1999/xhtml" xml:lang="de" lang="de">
4 <head>
5 ...
6 <link rel="stylesheet" .... href="https://..../general.min.css" />
7 <script src="https://www.xing.com/js/v/23/lib.min.js" ....></script>
8 <script src="https://www.xing.com/js/v/43/general.min.js" ....></script>
9 ...
10 </head><body >
11 <div id="container" >
12 ...
13 <img src="/img/pg_minilogo.png" height="25" />
14 <img id="photo" src="/img/users/f/3/0/00f9e12be.3155363,1.jpg" ... />
15 ...
16 </body></html>
```

Listing 2

Aber was geschieht nun wirklich en Detail? Wenn Performance von Websites verbessert werden soll, ist das genau die richtige Frage. Nur ein tiefes Verständnis und Interesse für noch unbekannte Details sind die Schlüssel zum Erfolg beim Optimieren von Performance.

Eine der, wie ich finde, nach wie vor schönsten Erklärungen hat der WDR mit der Sendung mit der Maus gebracht. Hier wird in ca. 18 Minuten erklärt, was zwischen dem Klick und der Auslieferung der Seite geschieht. Nun mag die Erklärung für Web-Experten etwas zu simpel erscheinen, jedoch müssen wir uns immer wieder klar machen, dass für gute Performance alle in einem Unternehmen Verantwortung tragen müssen, und Themen wie DNS-Auflösung oder Browser-Caching, oder Cookies sind kompliziert und in der Regel für nicht Experten total unverständlich. Performance aber muss verständlich werden!

Daher bitte keine Angst vor einfachen Erklärungen. Eine gut illustrierte Erklärung findet sich auf Abbildung 5. Im Wasserfall-Graphen einer der XING-Webseiten wird deutlich, dass nacheinander folgende Dinge geschehen:

• Der Benutzer klickt auf einen Link, auf ein Bookmark oder gibt eine URL ein und tippt die Return-Taste. Der Browser sendet daraufhin einen HTTP-Request an die entsprechende Webseite. Beispielsweise auf *https://www.xing.com*.

• Zunächst einmal wir der Domainname www.xing.com in die zugehörige IP-Adresse übersetzt. In diesem Fall erfolgt die Auflösung auf die IP-Nummer 62.96.140.140. Anschließend wird zwischen Browser und Server die https-Verschlüsselung ausgehandelt. Im Wasserfall-Graphen sind diese beiden Schritte in der zweiten Zeile auf Abbildung 5. Die dafür benötigte Zeit kann variieren und sollte in der Regel deutlich unter 100 Millisekunden liegen. (Schon dieser erste Schritt ist im Detail wesentlich komplexer, aber für unsere Betrachtung reicht es soweit erst einmal aus.)

• Nun erfolgt auf der Serverseite der Zusammenbau der ersten HTML-Seite. Diese wird, sobald sie fertig ist, ausgeliefert [2]. Das fertige HTML-Dokument kommt beim Browser an. Allerdings fehlen hier natürlich alle Bilder, CSS-Dateien, Javascript-Dateien, externer Content wie etwa Werbung oder Zählpixel, also alles, was nun nachgeladen werden muss. Diese Dinge stehen im Ergebnis der ersten HTML-Seite, hier beispielhaft 'index.html' genannt, etwa in folgender Form (mit vielen Auslassungen) - siehe Listing 2.

• Jetzt werden nacheinander alle 'Assets' der Webseite abgerufen, also alle Dokumente, auf die in der ersten index.html-Datei verwiesen wird. Hier werden also alle Dokumente referenziert, die nachgeladen werden müssen, damit die Seite komplett angezeigt werden kann. Im in Abbildung 5 gezeigten Fall handelt es sich um über 60 (!) nachgeladene Dateien. Diese sehr große Anzahl ist schon der erste Schritt in Richtung schlechte Performance, denn selbst wenn jede einzelne dieser Dateien im Durchschnitt nur 100 Millisekunden zum Laden braucht, und selbst unter der Annahme, dass immer zwei Dateien gleichzeitig von einer Domäne geladen werden können [3,4] wird die Auslieferungszeit bei

$$\frac{60 \text{ Dateien} \times 100 \text{ Millisekunden/Datei}}{2} = 3 \text{ Sekunden}$$

liegen.

Nach dem Klick werden also zunächst die dynamischen Inhalte der Website beispielsweise in Form einer 'index.html' bereit gestellt. Anschließend interpretiert der Browser diese

Datei und lädt alle notwendigen Assets nach. Das alles geschieht innerhalb weniger Sekunden. Für gute Performance lässt sich folgende Grundregel ableiten:

```
Gute Performance = weniger Elemente
  • mehr Elemente parallel
  • schnelle Ladezeiten der einzelnen Elemente
```

Formel 2

Eine Website, die diese Regeln eisern befolgt, ist Google. Schon der erste Blick auf den Wasserfall-Graphen von Google mit Yslow zeigt das. Kurz und schnell (siehe Abbildung 6) mit nur der index.html, fünf Asssets und alles innerhalb von 0,3 Sekunden.

Es gibt im Internet und in Buchform zahlreiche Veröffentlichungen, Vorträge, Videos und Browser-Plugins zu dem Thema. Nachfolgend einige Betrachtungen von uns.

Der Wasserfall-Graph zeigt auch, dass der größte Teil der Zeit bei der Auslieferung von Websites nicht im Backend bei der Berechnung des dynamischen Inhalts der Seite anfällt, sondern bei der Auslieferung der Seiten zum Browser. Hier gilt es zu optimieren. Eine gute Hilfe dafür stellen Werkzeuge wie Yslow oder webpagetest.org dar. Yslow vergibt beispielsweise nach der Analyse einer Webseite Zensuren nach dem amerikanischen Schulnotenprinzip und verweist auf technische Möglichkeiten der Verbesserung, die teilweise einfach zu erreichen sind.

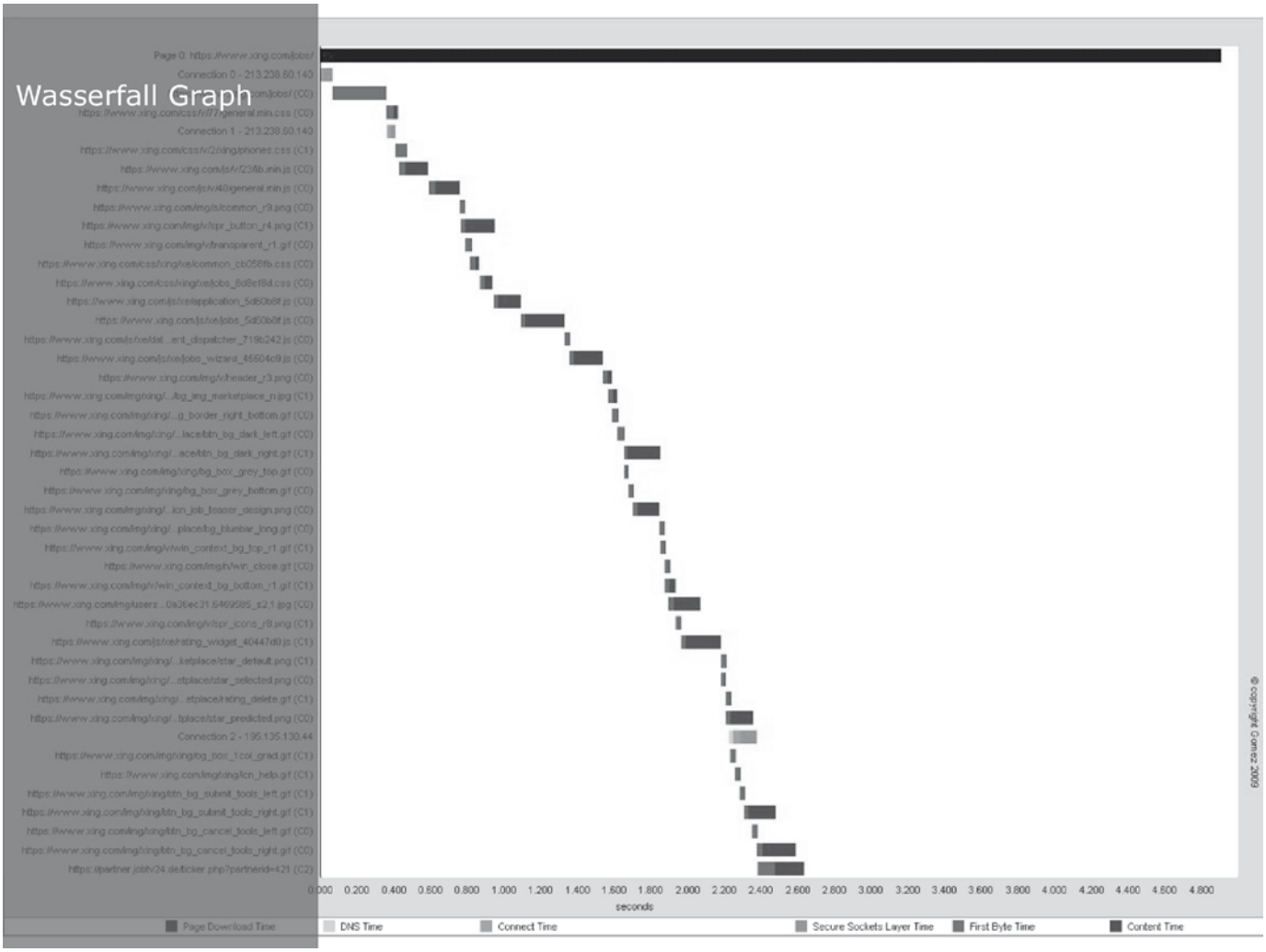


Abbildung 5: Im Wasserfall-Graph wird dargestellt, wie die einzelnen Objekte einer Website nacheinander geladen werden. Dabei werden auf der x-Achse die Zeit und auf der y-Achse die Dokumente eingetragen, die zur vollständigen Darstellung der Webseite nacheinander geladen werden müssen. Hier ein Ausschnitt des Wasserfall-Graphen einer XING-Seite aufgezeichnet durch den GOMEZ-Service.

Performance Awareness Performance-Verantwortung im Unternehmen

Performance im Unternehmen, zumal in einem Unternehmen mit 50 Releases im Jahr und vielen wichtigen Business-getriebenen Vorhaben ist ein flüchtiges Ziel. Es braucht Überzeugung und Sturheit, Überzeugungskraft, Teamgeist, denn an der guten Performance müssen alle Abteilungen mitarbeiten und dafür vorausdenken. So erfordert beispielsweise die Umsetzung einer neuen Gestaltung der Website aus Performance-Gesichtspunkten von Anfang an, dass alle daran mitmachen. Performance muss im Unternehmen als Business-Ziel verankert werden.

Das ist schwer zu erreichen, da es für Performance keinen einzelnen Business-Verantwortlichen geben kann. Gute Performance zahlt nie direkt auf ein Business-Ziel ein. Weiterhin ist das Thema derart technisch in der Umsetzung und Umfasst alle Produkt und Engineering Bereiche von der Admin bis zum Frontend-Entwickler, dass es auf auf der Seite der Produktentwicklung keinen direkt verantwortlichen Bereich gibt. Daher wird das Thema in den meisten Unternehmen vernachlässigt. **Performance hat keinen Verantwortlichen im Unternehmen**.

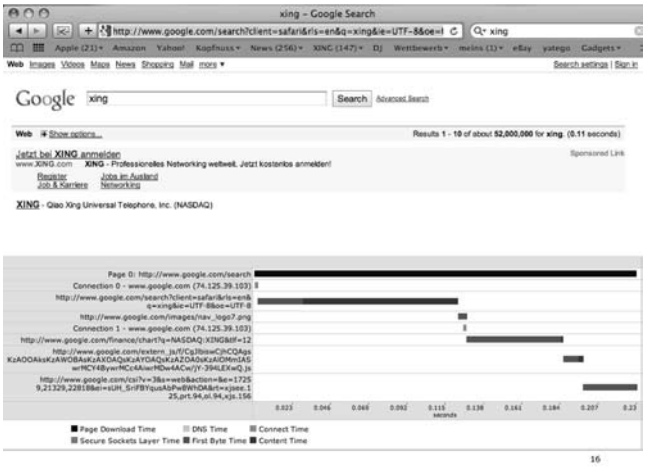


Abbildung 6: Der Wasserfall-Graph von Google mit dem Firefox-Browser und Yslow betrachtet.

Das gilt es zu ändern. Ein perfomantes Unternehmen verankert Performanceziele übergreifend und schafft Awareness bei allen Beteiligten. Bis es soweit ist, brauchen die Performatoren [5] des Unternehmens einen langen Atem, Ausdauer, einen hohe Frustschwelle, Selbstmotivation und vor allen eine freundliche und durchschlagende Überzeugungskraft.

Dr. Johannes Mainusch mit Beiträgen von Christopher Blum und Björn Kaiser

Freuen Sie sich auf die Fortsetzung dieses Artikels in Ausgabe 14 des $foo-Magazins.

[1] Dom-ready bedeutet bei XING, dass die Auslieferung der eingebundenen Werbung und von Zählpixeln nicht mitgezählt wird.

[2] Hier gibt es Techniken und Tipps zur Performance-Optimierung, Stichwort: early page flushing.

[3] Moderne Browser sind in der Lage, mehrere Verbindungen gleichzeitig zu einer Domäne aufzubauen. Die Default-Einstellung bei den meisten Browsern liegt bei zwei Verbindungen zur Domäne, der Internet-Explorer 8 macht als Default-Einstellung bis zu acht Verbindungen zur ausliefernden Domäne auf. Ein Blick auf Abbildung 5 zeigt die jeweils zwei parallel geöffneten Verbindungen.

[4] Performance-Tipp: Durch Auslieferung von Assets von weiteren Subdomänen kann die Anzahl der gleichzeitig auslieferbaren Dateien erhöht werden. Das hat zur Folge, dass bei schnellen Internetverbindungen der Wasserfall-Graph mehr parallele Element enthält, und somit die Gesamtladezeit schrumpft. Beispiel: m1.xing.com, m2.xing.com

[5] Performator: Protagonist der guten Performance des/der Produkte eines Unternehmens.

Lotus Notes / Domino - Perl(en) mit Java

Schnell ein paar Informationen aus Notes-Anwendungen extrahieren und aufbereiten - und dazu Perls Stärken effektiv nutzen? Hier wird anhand eines Beispiels gezeigt, wie man mit Perl auf die Datenbanken bzw. Anwendungen eines Domino-Servers zugreifen kann.

Leider gibt es derzeit keinen direkten Weg, um den Domino Server mit Perl anzusprechen, allerdings kann man dieses Manko umgehen: hierzu kommt das CPAN-Modul `Inline::Java` von Patrick LeBoutillier zum Einsatz. Es ermöglicht die Nutzung von Java-Klassen in Perl Programmen und ist dabei so transparent, dass man so gut wie keinen Java-Code schreiben muss. Perl übernimmt dabei die gesamte Interaktion mit der Java-VM und bildet Java- auf Perl-Objekte ab.

Setup

Da wir den Domino Server über Java ansteuern, benötigen wir zunächst mal ein Java Development Kit (JDK) und die Datei `NCSO.jar`. In dieser sind alle Java-Klassen enthalten, die zum Ansprechen des Servers erforderlich sind. Die Datei ist im Domino-Data Verzeichnis zu finden unter `domino/java/`. Auf dem Domino Server muss ferner auch der "DIIOP" (Domino Internet Inter-ORB Protocol)-Task gestartet sein.

Natürlich brauchen wir auch Perl, einen laufenden Domino-Server und einen (Web)Benutzer-Account mit Passwort. Die Notes-Anwendung für die Beispiele ist erhältlich unter *http://www.assono.de/blog/d6plinks/Foo-Artikel-Notes-und-Perl*. Sie basiert auf dem, bei OpenNTF erhältlichen, *"assono Framework 2"* (*http://www.openntf.org/Projects/pmt.nsf/ProjectLookup/assono+Framework+2*).

Inline::Java installieren

Das Paket `Inline::Java` sollte nicht über die CPAN-Shell installiert werden, da ggf. die Konfiguration anpasst werden muss. Details dazu kann man aus dem Readme entnehmen. Hat man dies soweit erledigt, wird das Paket übersetzt und installiert mit:

```
perl Makefile.PL
  J2SDK=/usr/lib/jvm/java-6-sun/
make
make test
make install
```

Der Pfad des JDK muss so angepasst werden, dass er zu dem jeweiligen System und der zu verwendenden JDK-Version passt.

Bei etwaigen Warnungen bezüglich fehlender Pakete sollten zunächst alle Paket-Abhängigkeiten aufgelöst und fehlende Pakete installiert werden.

Die JNI-Extension produziert (zumindest mit meiner Konfiguration mit Java 6) hin und wieder Abstürze, kann aber auch einfach deaktiviert werden bzw. braucht gar nicht mit installiert zu werden.

Die Verbindung testen

Nun, da unsere Konfiguration soweit steht, sollten wir auch gleich einmal testen, ob auch alles funktioniert. Ein einfaches Grundgerüst für ein Programm könnte etwa so aufgebaut sein:

```
use strict;
use warnings;
use Inline (
            Java         => 'STUDY',
            CLASSPATH    => 'NCSO.jar',
            STUDY        =>
              ['lotus.domino.NotesFactory'],
            AUTOSTUDY    => 1
           );
```

Über `STUDY` teilen wir `Inline::Java` mit, welche Java Klassen in Perl zur Verfügung gestellt werden sollen. In unserem Falle reicht eigentlich die *"NotesFactory"*- Klasse, da diese zum Erzeugen der Session benötigt wird. Die weiteren Klassen werden dann je nach Bedarf ebenfalls in Perl-Objekten abgebildet (`AUTOSTUDY`). Sofern die Datei `NCSO.jar` nicht schon über den Pfad in der Umgebungsvariable `CLASSPATH` gefunden wird, kann man hier auch den Pfad angeben.

```
my $server   = "localhost";
my $user     = "benutzer";
my $pass     = "einpasswort";
my $mailfile = "mail/$user.nsf";
my $session;
my $maildb;
```

Für den Verbindungsaufbau werden einige Informationen benötigt: Server-Name, Benutzer und Passwort (siehe Listing 1)

Zum Abfangen von Laufzeit-Fehlern wird hier alles in einem `eval`-Block gekapselt. Die Session wird erzeugt und damit sind wir am Domino Server angemeldet. Im Anschluss wird versucht die Mail-Anwendung zu öffnen und eine entsprechende Status-Meldung wird ausgegeben. Abschließend werden die erzeugten Objekte der Ordnung halber wieder freigegeben.

```
if($@){
    if(ref($@)){
        print $@->toString(), "\n";
    }else{
        print "Fehler:", $@, "\n";
    }
}
```

Aufgetretene Fehler werden unterschiedlich behandelt. Sollte der Fehler von `Inline::Java` geworfen werden, so enthält `$@` eine Objekt-Referenz auf das Fehler-Objekt, andernfalls ist es ein einfacher String.

`Inline::Java` stellt darüber hinaus die Funktion `caught` zur Verfügung. Mit dieser können unterschiedliche Exceptions nach Namen behandelt werden. Für Notes/Domino ist dies in der Regel `NotesException`.

Wenn das obige Programm wie gewünscht läuft, ist es geschafft - wenn nicht so gibt die Fehlermeldung Auskunft über den möglichen Grund.

Zugriff auf Ansichten und Dokumente

Im ersten Schritt hatten wir uns darauf beschränkt, eine Verbindung aufzubauen und anschließend eine Notes-Anwendung zu öffnen. Damit lässt sich zwar schon einiges machen, aber es ist nicht sehr nützlich. Erst mit dem Zugriff auf Ansichten und Dokumente in der Notes-Anwendung kommen wir an die eigentlichen Inhalte.

Für diesen Zweck verwenden wir eine Notes-Anwendung, in welcher Informationen zu einem bestimmten Host bzw. zu einer bestimmten IP Adresse gespeichert werden. Der Aufbau ist dabei so gehalten, dass die Anwendung leicht modifizierbar und somit auch für andere Zwecke verwendbar ist.

Für dieses Beispiel prüfen wir mit `Net::Ping`, ob auf ein bestimmter Dienst auf dem angegeben Host erreichbar ist. Die Verfügbarkeitstests können mit dem Notes Client erstellt werden. Das Perl Programm kann dann z.B. periodisch gestartet werden und anstehende Anforderungen abarbeiten.

```
eval{
    $session  = lotus::domino::NotesFactory->createSession($server, $user, $pass);
    $maildb   = $session->getDatabase("", $mailfile, 0);

    if($maildb->isOpen()){
        print "Datenbank erfolgreich geöffnet!\n";
    }else{
        print "Datenbank wurde nicht geöffnet :-(\n";
    }

    $maildb->recycle();
    $session->recycle();
};
```

Listing 1

Dokumente auswählen

Zuerst müssen die noch nicht verarbeiteten Dokumente gewählt werden. Dazu wird zunächst eine Ansicht geöffnet, die bereits so gestaltet ist, dass nur die relevanten (neuen) Dokumente angezeigt werden. Es gibt noch weitere Möglichkeiten, an die Dokumente zu kommen, z.B. kann man mit der Methode `FTSearch` der `database`-Klasse eine Volltextsuche durchführen. Komplexere Aufgaben lassen sich auch über die Klasse `ViewNavigator` lösen. Es gibt hier noch Raum für Optimierungen, performanter wäre z.B. das Auslesen der benötigten Werte über die Methode `getColumnValues` der `ViewEntry`- Klasse und die Verwendung von `getFirstEntry` bzw. `getNextEntry` anstatt `getNthEntry`.

```
$view = $database->getView($viewname);
$vec  = $view->getAllEntries();
```

Aus der Ansicht erhalten wir die `ViewEntryCollection` in der alle relevanten Dokumente enthalten sind. Die einzelnen Elemente der Collection sind vom Typ `ViewEntry`. Aus diesen erhalten wir das eigentliche Notes-Dokument:

```
for($i=1; $i <= $vec->getCount(); $i++){
    $viewentry = $vec->getNthEntry($i);
    $document  = $viewentry->getDocument();
```

Werte aus Feldern auslesen

Um Werte aus Feldern auszulesen, verwendet man je nach Feld-Typ die Methode `getItemValue` (bzw. `getItemValueString`, `getItemValueInteger`, ...) des Dokument-Objekts. Wenn man alle Felder des Dokuments verarbeiten will, so kann man auch `getItems` verwenden, um ein Array zu erhalten. In unserem Fall sind es jedoch nur drei Felder, die benötigt werden, wobei das Kommentar-Feld nur auf der Konsole ausgegeben wird.

```
$current_comment = $document
   ->getItemValueString($comment_field);
$current_host    = $document
   ->getItemValueString($host_field);
$current_port    = $document
   ->getItemValueInteger($port_field);
```

Felder befüllen und das Dokument speichern

Zum Überprüfen der Verbindung wird zunächst der Port gesetzt und dann die Methode `ping` des `Net::Ping`-Objektes ausgeführt (der Name ist etwas irreführend, es wird hier natürlich kein ICMP-`Echo Request` gesendet, sondern eine Verbindung via TCP aufgebaut).

```
$ping->port_number($current_port);
if ($ping->ping($current_host, $timeout)){
  ...
```

Nachdem festgestellt wurde, ob der gewählte Dienst verfügbar ist, werden die Ergebnisse in das Notes-Dokument geschrieben. Eine kleine Falle gibt es an dieser Stelle: Wenn kein Feld mit dem angegeben Namen existiert, wird einfach ein Neues angelegt. Man könnte also sagen, dass es eher ein `ReplaceOrCreateNew` ist, daher sollte man an dieser Stelle besonders auf richtige Schreibweise achten. Feldnamen sind übrigens in Notes/Domino nicht case-sensitiv.

Der Methode `save` können mehrere Argumente mitgegeben werden. Wird das erste Argument auf `true` gesetzt, so wird das Speichern des Dokumentes erzwungen. Wenn das gleiche Dokument bereits anderweitig geöffnet wurde, wird ein Konflikt-Dokument erstellt. Die beiden anderen Argumente sind zuständig für das Erstellen eines Antwortdokuments bzw. markieren das Dokument als gelesen.

```
$document->replaceItemValue(
           $status_field, $status_value);
$document->replaceItemValue(
           $result_field, $result_value);
$document->save(1);
```

Sobald das Dokument gespeichert ist, ist auch in der Notes-Anwendung die Änderung zu sehen. Der Vorgang ist damit eigentlich abgeschlossen - es muss nur noch aufgeräumt werden.

Explizites Freigeben der Objekte

Es sollte darauf geachtet werden, alle Notes-Objekte explizit freizugeben, sobald sie nicht mehr benötigt werden. Dies ist erforderlich, da der Java-Garbage-Collector nur den Speicher der Java-Objekte freigibt, nicht jedoch den Speicher auf dem

Domino-Server. Jedes Notes-Objekt hat zu diesem Zweck eine Methode `recycle`.

```
$vec->recycle()          if(ref($vec));
$view->recycle()         if(ref($view));
$database->recycle()     if(ref($database));
$session->recycle()      if(ref($session));
```

Fehlerbehebung

Wenn bereits der Verbindungstest mit folgender Meldung fehlschlägt:

```
NotesException: Could not get IOR from
Domino Server:
java.io.FileNotFoundException:
http://<server>/diiop_ior.txt
```

so liegt dies daran, dass die Datei `diiop_ior.txt` nicht auf dem Server gefunden werden konnte. Wo diese sich auf dem Dateisystem des Servers befindet, erfährt man durch Eingabe des folgenden Befehls in der Serverkonsole:

```
tell DIIOP show config
```

Die Berechtigungen der Datei sollten so gesetzt sein, dass diese von jedem gelesen werden kann. Ferner ist erforderlich, dass der HTTP- und der DIIOP-Task des Domino-Servers gestartet sind.

Ein weiterer Stolperstein ist das Fehlen des Hostnames im Serverdokument unter `Internetprotocols/DIIOP`. Dies führt dazu, dass versucht wird, die Verbindung über `localhost` aufzubauen, was auf Client-Seite natürlich fehlschlägt:

```
NotesException: Could not open Notes session
: org.omg.CORBA.COMM_FAILURE
: java.net.ConnectException
: Connection refused:  connect Host
: 127.0.0.1 Port: 60148
```

Weitere Informationen zur Problemlösung kann man z.B. in Teil 2 von *"Java access to the Domino Objects" (http://www.ibm.com/developerworks/lotus/library/ls-Java_access_pt1/index.html)* finden.

Fazit

Sicherlich gibt es andere Wege, Anwendungen für Notes/Domino zu entwickeln aber der Charme dieser Lösung liegt ganz klar in der Flexibilität, die durch Perl dazugewonnen wird. Gerade für die Vernetzung verschiedener Systeme ist Perl der *"Klebstoff"* schlechthin. Für einen Administrator, der Domino Server in seinem Netzwerk hat, mag es sich als sehr nützlich erweisen, für den alltäglichen Bedarf das eine oder andere Script zu entwickeln.

Arnd Koch, assono GmbH

Visualisierungen mit Perl und ImageMagick

Visualisierungen wie z. B. die BBC in ihrer Serie "Britian from above" gemacht hat (siehe http://www.bbc.co.uk/britain-fromabove/stories/visualisations/taxis.shtml) faszinieren.

Die Idee ist, das öffentliche Netz der Wiener Linien dementsprechend darzustellen, um zu zeigen wo im Laufe eines Tages Autobusse, Straßenbahnen und Nachtautobusse in Haltestellen stehen und wie viele Busse bzw. Straßenbahnen zu welcher Uhrzeit unterwegs sind.

Das Problem ist nur, dass es keine fertigen Daten der Haltestellen, ihrer Geoposition und den Fahrplanzeiten der Autobusse, Straßenbahnen und Nachtautobusse für Wien gibt.

Zum Glück gibt es die barrierefreie Webseite der Wiener Linien (http://www.wienerlinien.at/itip/bf/), die es relativ einfach ermöglicht, zumindest die Haltestellen, die Linien und die Fahrplanzeiten zu ermitteln (die Vorgehensweise würde diesen Artikel sprengen und ist auch von Ort zu Ort verschieden).

Um zu den Haltestellen die notwendige Geo-Position zu ermitteln hat sich die Googles Maps-API bewährt. Das folgende Skript (ich entschuldige mich jetzt schon für meinen schlechten Programmierstil) ermittelt die entsprechenden Geo-Koordinaten (die Table in der Datenbank hat den Namen "oeffi"):

```
#!/usr/bin/perl
#-----------------------------------------------------------------------
# Geo-Koordinaten ermitteln
# By Max Kossatz, 2009
# http://wissenbelastet.com
# Lizenz: Creative Commons BY-NC
# http://creativecommons.org/licenses/by-nc/3.0/at/
#-----------------------------------------------------------------------

# Für die Datenbank-Anbindung wird in diesem Beispiel MySQL
# verwendet, kann aber auch jede andere Datenbank sein.
use DBI;

# Die Geokoordinaten werden mittels Googles Geo-Coder ermittelt.
# Hierzu wird ein Google Maps-API Key benötigt, nähere Infos siehe
# hier: http://code.google.com/intl/de/apis/maps/signup.html
use Geo::Coder::Google;
my $geo = Geo::Coder::Google->new(
    apikey   =>'GOOGLE_MAPS_API_KEY',
    language =>'de',
);

# Verbindung zur Datenbank herstellen.
# DATENBANK, USERNAME und PASSWORT entsprechend anpassen.
my $dsn = "DBI:mysql:database=DATENBANK;mysql_enable_utf8=1";
my $dbh = DBI->connect($dsn,"USERNAME","PASSWORT");

# Ungebufferte Ausgabe zum Mitloggen.
$|=1;
```

Listing 1

```
# Da eine Haltestelle von mehreren Linien angefahren wird ist in Wien nur die
# Haltnummer als eindeutiges Kriterium verwendbar.
$db = $dbh->prepare(
  "SELECT distinct haltnummer from oeffi where plz=0 order by haltestelle asc"
);

$db->execute;

while($haltnummer=$db->fetchrow_array) {

# Aufgrund der Haltnummer den Stationsnamen aus der Datenbank holen.
    $db1 = $dbh->prepare(
      "select haltestelle from oeffi where haltnummer=$haltnummer limit 0,1"
    );

    $db1->execute;

    $haltestelle = $db1->fetchrow_array;

# Diese Haltestelle an den Google Geocoder übergeben. Als "Trick" wird vor den
# Haltestellennamen "Haltestelle " und nachher ", Wien" geschrieben, mit den
# Wiener Daten hat dies sehr gut funktioniert.
    my $location = $geo->geocode(
        location => 'Haltestelle '.$haltestelle.", Wien"
    );
    $longi   = $location->{'Point'}->{'coordinates'}->[0];
    $lati    = $location->{'Point'}->{'coordinates'}->[1];
    $plz     = "";
    $address = $location->{'address'};

# Um die Daten überprüfen zu können wird die Postleitzahl der ermittelten Geo-
# Position ermittelt, falls diese nicht vorhanden ist wird die Geo-Abfrage
# nochmals ohne "Haltestelle " durchgeführt.
    $address =~ /(\d\d\d\d) Wien/;
    $plz = $1;
    if (!$plz) {
        $location = $geo->geocode( location => $haltestelle.", Wien");
        $longi    = $location->{'Point'}->{'coordinates'}->[0];
        $lati     = $location->{'Point'}->{'coordinates'}->[1];
        $plz      = "";
        $address  = $location->{'address'};
        $address  =~ /(\d\d\d\d) Wien/;
        $plz      = $1;
    }

# Die Postleitzahl wird, damit man dann einfacher die Ergebnisse überprüfen
# kann, in der Datenbank mit abgespeichert.
    if (!$plz) { $plz = 0; }
    $dbh->do(
        "update oeffi set longi=$longi, lati=$lati, plz=$plz where haltnummer=$haltnummer"
    );
}
```

Listing 1 (Fortsetzung)

Die "haltnummer" (siehe Datenbankstruktur, Abbildung 1) ist bei den Wiener Linien die eindeutige Nummer einer Haltestelle und einer bestimmten Linie, da ja mehrere Linien eine Haltestelle anfahren können. Insgesamt sind über 500.000 Bewegungsdaten in der Datenbank.

Da der Google Geo-Coder nicht immer den richtigen Ort findet, ist es ratsam die Koordinaten nach der Ermittlung händisch zu überprüfen. Einerseits geht das sehr gut mit der Postleitzahl, falls diese nicht z.B. zu Wien gehört muss man nachbessern. Auch hat es sich bewährt, die nördlichsten, östlichsten, südlichsten und westlichsten Koordinaten zu begutachten und z.B. in Google Maps oder Openstreetmap zu kontrollieren, ob diese an der richtigen Position sind.

Falls Koordinaten nicht stimmen, am besten mit Google Maps oder Openstreetmap diese händisch nachbessern.

Um diese Daten jetzt zu animieren sind mehrere Layers notwendig, die dann mit einer Videoschnittsoftware (in diesem

Datenbank

id	linie	haltestelle	plz	uhrzeit	haltnummer	lati	longi
72475	74A	Hofmannsthalgasse	1030	1800	290	48,185025	16,399887
72543	74A	Eslarngasse	1030	1800	295	48,196706	16,395841
72644	O	Gudrunstraße	1100	1800	300	48,178069	16,376543
73008	2	Dresdner Straße U	1200	1800	318	48,23718	16,380177
73108	31	Friedrich-Engels-Platz	1200	1800	321	48,243988	16,379894
73193	2	Traisengasse S	1200	1800	324	48,233504	16,383406
73257	2	Heinestraße, Taborstraße	1020	1800	328	48,222281	16,382335
73439	18	Südtiroler Platz S U	1040	1800	340	48,186895	16,373736
73475	O	Columbusplatz	1100	1800	341	48,181963	16,373684
73641	5	Rauscherstraße	1200	1800	349	48,230848	16,375029

Abbildung 1:
Datenbankstruktur

```
#!/usr/bin/perl
#-----------------------------------------------------------------------
# Erstellung der Grafiken mit der eingeblendeten Uhrzeit
# By Max Kossatz, 2009
# http://wissenbelastet.com
# Lizenz: Creative Commons BY-NC
# http://creativecommons.org/licenses/by-nc/3.0/at/
#-----------------------------------------------------------------------

# Für die Datenbank-Anbindung wird in diesem Beispiel MySQL verwendet, kann aber auch jede
# andere Datenbank sein.
use DBI;

# Perl Modul für Image Magick, siehe http://www.imagemagick.org/
use Image::Magick;

# Verbindung zur Datenbank herstellen.
# DATENBANK, USERNAME und PASSWORT entsprechend anpassen.
my $dsn="DBI:mysql:database=DATENBANK;mysql_enable_utf8=1";
my $dbh=DBI->connect($dsn,"USERNAME","PASSWORT");

# Ungebufferte Ausgabe zum Mitloggen.
$|=1;

# Folder in dem die fertigen Grafiken gespeichert werden sollen.
$folder="uhrzeit/";

# Nummer der Grafik innerhalb der Sequenz, beginnend mit 1.
$filenummer=1;

# Die Uhrzeiten aus der Datenbank holen.
my $sth= $dbh->prepare(
  "select distinct uhrzeit from oeffi order by uhrzeit asc"
);
$sth->execute;
$uhrzeit=$sth->fetchrow_array;

# Da die erste Uhrzeit "0" ist, dient $a dazu um nicht die while-Anweisung sofort abzubrechen.
$a=1;

while($uhrzeit=$sth->fetchrow_array || $a) {

# Uhrzeit "0" geschafft, $a auf "0" setzen.
    $a=0;

# Grafik erstellen, Hintergrundfarbe Grün um im Videoschnitt-Programm diese Farbe Transparent
# machen zu können.
    $image=Image::Magick->new;

    $image->Set(size=>'1115x812');
    $image->ReadImage('xc:green');
```

Listing 2

```
# Umwandlung der Uhrzeit aus der Datenbank in das Format HH:MM.
    $uhrzeit=sprintf '%04s',$uhrzeit;
    $uhrzeit=~/(\d\d)(\d\d)/;
    $puhr=$1.":".$2;

# Uhrzeit in die Grafik eintragen.
    $image->Annotate(
        text      => $puhr,
        x         => 520,
        y         => 40,
        pointsize => 30,
        fill      => 'black',
        stroke    => '#444444',
    );

# Grafik abspeichern, Filenummer erhöhen.
    $image->Write('png:'.$folder.$filenummer.'.png');
    $filenummer++;
}
```

Listing 2 (Fortsetzung)

Beispiel Sony Vegas) übereinandergelegt werden. Diese Layers bestehen aus Einzelbilder, die dann zu einem Video mit 25 Bildern/Sekunde zusammengebaut werden.

Der erste Layer ist die Uhrzeit, beginnend mit "00:00" und endend mit "23:59", das sind 1440 Grafiken. Das entsprechende Skript dazu in Listing 2.

So sehen die fertigen Einzelbilder des ersten Layers aus, wie in Abbildung 2 dargestellt.

Als zweiter Layer wird die Balkengrafik, die anzeigt wie viele Busse, Straßenbahnen und Nachtbusse zu jeder Uhrzeit in einer Station stehen, generiert (siehe Abbildung 3). Das passende Skript dazu auf der nächsten Seite (Listing 3).

Der dritte Layer sind jeweils die Fahrzeuge als Punkte, die zu der entsprechenden Uhrzeit in den jeweiligen Haltestellen stehen. Da die Datenbank keine Ahnung davon hat wer vorher oder nachher in einer Station steht, bzw. wo z. B. ein Bus hinfährt, nachdem er in einer Haltestelle stand, ist es schwierig den "Flow" der Linien zu visualisieren.

Um zumindest eine Art "Nachzeichner"-Effekt zu erreichen, wird ein Gaussian Blur über jedes Bild gelegt und, nachdem die Grafik gespeichert ist, diese um 20% abgedunkelt und wieder als Hintergrundgrafik für die nächsten Uhrzeit verwendet. So entsteht mit der Zeit der gewünschte Effekt. Das entsprechende Skript dazu in Listing 4.

Das Resultat des Skriptes für den dritten Layer ist in Abbildung 4 zu sehen. Das Schwarz wird in der Videoschnittsoftware durch die Hintergrundgrafik ersetzt (funktioniert besser als in Image Magick).

Abbildung 2: Erster Layer mit der eingeblendeten Uhrzeit

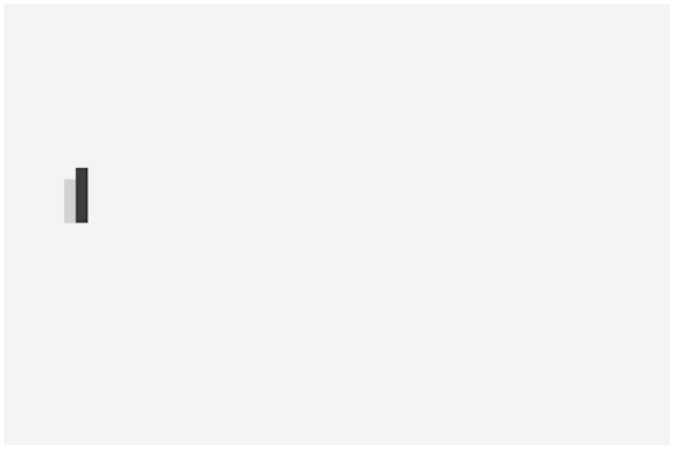


Abbildung 3: Beispiel des zweiten Layers mit den bis zu drei Balkengrafiken (Nachtautobusse fahren nur in der Nacht).

```
#!/usr/bin/perl
#-----------------------------------------------------------------------
# Erstellung der Balkengrafik
# By Max Kossatz, 2009
# http://wissenbelastet.com
# Lizenz: Creative Commons BY-NC
# http://creativecommons.org/licenses/by-nc/3.0/at/
#-----------------------------------------------------------------------

# Für die Datenbank-Anbindung wird in diesem Beispiel MySQL
# verwendet, kann aber auch jede andere Datenbank sein.
use DBI;

# Perl Modul für Image Magick, siehe http://www.imagemagick.org/
use Image::Magick;

# Verbindung zur Datenbank herstellen.
# DATENBANK, USERNAME und PASSWORT entsprechend anpassen.
my $dsn="DBI:mysql:database=DATENBANK;mysql_enable_utf8=1";
my $dbh=DBI->connect($dsn,"USERNAME","PASSWORT");

# Ungebufferte Ausgabe zum Mitloggen.
$|=1;

# Folder in dem die fertigen Grafiken gespeichert werden sollen.
$folder="graphen/";

# Nummer der Grafik innerhalb der Sequenz, beginnend mit 1.
$filenummer=1;

# Uhrzeiten aus der Datenbank holen.
my $sth= $dbh->prepare("select distinct uhrzeit from oeffi order by uhrzeit asc");
$sth->execute;
$uhrzeit=$sth->fetchrow_array;

# Da die erste Uhrzeit "0" ist, dient $a dazu um nicht die while-
# Anweisung sofort abzubrechen.
$a=1;
while($uhrzeit=$sth->fetchrow_array || $a) {
    if (!$uhrzeit) { $uhrzeit='0000'; }

# Uhrzeit "0" geschafft, $a auf "0" setzen.
    $a=0;

# Leere Grafik erstellen, Hintergrundfarbe Gelb um diese Farbe im
# Videoschnittprogramm Transparent machen zu können.
    $image=Image::Magick->new;
    $image->Set(size=>'1115x812');
    $image->ReadImage('xc:yellow');

# Zuerst die Autobuslinien, die in Wien auf "A" enden und in der Grafik Rot
# dargestellt werden.
    $rot = $dbh->prepare(
        "select count(id) from oeffi where uhrzeit=$uhrzeit and linie like '\%A'"
    );
    $rot->execute;
    $rot = $rot->fetchrow_array;

# Der Balken kann maximal 400 Pixel hoch sein und ist 20 Pixel breit.
    if ($rot>0) {

# Berechnung der Balkenhöhe.
        $rot = 400-int(200/500*$rot);

# Balken zeichnen.
        $image->Draw(
            stroke    => '#FF0000',
            fill      => '#FF0000',
            primitive => 'rectangle',
            points    => '100,400 119,'.$rot,
        );
    }
```

Listing 3

```
# Dann die Straßenbahnen, die in Wien nicht auf "A" enden und nicht mit "N"
# beginnen, werden in der Grafik Blau dargestellt.
    $blau = $dbh->prepare(
        "select count(id) from oeffi where uhrzeit=$uhrzeit
             and linie not like '\%A' and linie not like 'N\%'"
    );
    $blau->execute;
    $blau = $blau->fetchrow_array;
    if ($blau>0) {
        $blau = 400 - int(200/500*$blau);
        $image->Draw(
            stroke    => '#0000FF',
            fill      => '#0000FF',
            primitive => 'rectangle',
            points    => '120,400 139,'.$blau,
        );
    }

# Zum Schluß die Nachtautobusse, die in Wien mit "N" beginnen und in der Grafik
# Grün dargestellt werden.
    $gruen = $dbh->prepare(
        "select count(id) from oeffi where uhrzeit=$uhrzeit
            and linie like 'N\%'"
    );
    $gruen->execute;
    $gruen = $gruen->fetchrow_array;
    if ($gruen>0) {
        $gruen=400-int(200/500*$gruen);
        $image->Draw(
            stroke    => '#00FF00',
            fill      => '#00FF00',
            primitive => 'rectangle',
            points    => '140,400 159,'.$gruen,
        );
    }

    $image->Write('png:'.$folder.$filenummer.'.png');
    $filenummer++;
}
```

Listing 3 (Fortsetzung)

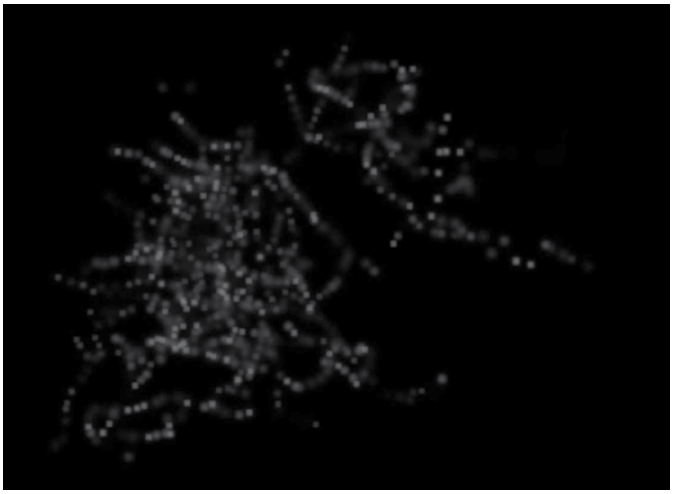


Abbildung 4: Fahrzeuge

Abbildung 5: Hintergrundgrafik mit einer Karte von Wien

Der letzte Layer ist eine Google Maps oder Openstreetmap Grafik, die in diesem Beispiel (Abbildung 5) Wien zeigt.

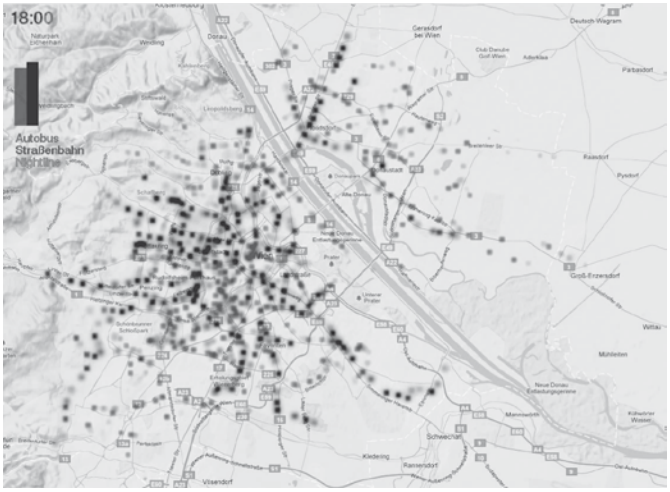


Abbildung 6: Fertiges Standbild

Diese vier Layer werden als Einzelbilder von 12:00 Mittag bis 11:59 Mittag des nächsten Tages in die Videoschnittsoftware eingespielt, sowie übereinandergelegt und ergeben das folgendes Bild (Abbildung 6) die Texte wurden mit der Videosoftware eingefügt).

Da die 1440 Bilder mit 25 Bilder pro Sekunde abgespielt werden ergibt sich somit ein 60 Sekunden langen Video.

Das fertige Video ist unter: *http://wissenbelastet.com/2009/09/22/visualisiert-24-stunden-wiener-linien/* abrufbar.

Fazit: Image Magick und Perl sind ein wunderbares Team bei der Visualisierung von Daten!

Max Kossatz

```
#!/usr/bin/perl
#----------------------------------------------------------------------
# Berechnung der Punkte auf der Wien-Karte
# By Max Kossatz, 2009
# http://wissenbelastet.com
# Lizenz: Creative Commons BY-NC
# http://creativecommons.org/licenses/by-nc/3.0/at/
#----------------------------------------------------------------------

# Für die Datenbank-Anbindung wird in diesem Beispiel MySQL
# verwendet, kann aber auch jede andere Datenbank sein.
use DBI;

# Perl Modul für Image Magick, siehe http://www.imagemagick.org/
use Image::Magick;

# Verbindung zur Datenbank herstellen.
# DATENBANK, USERNAME und PASSWORT entsprechend anpassen.
my $dsn="DBI:mysql:database=DATENBANK;mysql_enable_utf8=1";
my $dbh=DBI->connect($dsn,"USERNAME","PASSWORT");

# Ungebufferte Ausgabe zum Mitloggen.
$|=1;

# Folder in dem die fertigen Grafiken gespeichert werden sollen.
$folder="bilder/";

# Nummer der Grafik innerhalb der Sequenz, beginnend mit 1.
$imagenummer=1;

# Diese vier Koordinaten definieren die Ecken des Rechteckes in dem sich die Geo-
# Koordinaten befinden, um diese entsprechend auf die Hintergrundgrafik umrechnen zu können.
# Diese Koordinaten findet man z. B. auf http://openstreetmap.org
# Als Beispiel wird Wien verwendet.

# Westlichster Punkt des Rechtecks
$longimin=16.258322;
```

Listing 4

```
# Östlichster Punkt des Rechtecks
$longimax=16.580219;

# Südlichster Punkt des Rechtecks
$latimin=48.128469;

# Nördlichster Punkt des Rechtecks
$latimax=48.298805;

# Berechnung Breite des Rechtecks
$longidiff=$longimax-$longimin;

# Berechnung der Höhe des Rechtecks
$latidiff=$latimax-$latimin;

# Berechnung der Mitte des Rechtecks, wichtig für das Koordinaten-System
$longimitte=($longimin+$longimax)/2;
$latimitte=($latimin+$latimax)/2;

# Die breite der fertigen Grafik
$breite=1000;

# Das Seitenverhältnis für die fertige Grafik
$aspect=4/3;

# Die Höhe der Grafik wird über das Seitenverhältnis und die Höhe bzw. Breite des Geo-Raums
# definiert.
$hoehe=int(($latidiff/($longidiff/$breite))*$aspect);

# Berechnen des Verhältnisses von Breite bzw. Höhe pro dargestelltem Pixel.
$xpixel=$longidiff/$breite;
$ypixel=$latidiff/$hoehe;

# Damit die äussersten Haltestellen nicht abgeschnitten werden, wird der Bildbereich um
# diesen Faktor vergrößert, sozusagen ein "Rahmen" um die Darstellungsfläche.
$size=1.1;

# Berechnung der neuen Breite und Höhe der Grafik.
$neue_breite=int($breite*$size);
$neue_hoehe=int($hoehe*$size);

# um zu wissen wann eine neue Uhrzeit (=neue Grafik) anfängt.
$uhrzeit_alt=0;

# ImageMagick Image erstellen mit der berechneten Breite und Höhe.
$image=Image::Magick->new;
$image->Set(size=>$neue_breite.'x'.$neue_hoehe);

# Image ist Schwarz mit Transparenzfarbe Schwarz.
$image->ReadImage('xc:black');
$image->Transparent('black');

# Um den Nachzieheffekt zu erreichen brauchen wir mindestens 2x24 Stunden,
# da am Anfang noch zu wenig Punkte in der Grafik sind um einen schönen Effekt
# zu erreichen. $durchlauf zählt die Durchläufe.
$durchlauf=1;

while($durchlauf<3) {

# Die entsprechenden Daten (Geo-Position, Uhrzeit, und Line) von der Datenbank holen.
    my $sth= $dbh->prepare(
        "select longi,lati,uhrzeit,linie from oeffi order by uhrzeit asc"
    );
    $sth->execute;

    while(($longi,$lati,$uhrzeit,$linie)=$sth->fetchrow_array) {
```

Listing 4 (Fortsetzung)

```
# Eine neue Uhrzeit bedeutet, die alte Grafik mit einem Gaussion-Blur zu
# versehen (damit entsteht dieser Nachzieh-Effekt). Dann wird eine Kopie der
# Grafik erstellt, um mit Normalize wieder die volle Bandbreite des Fabraumes
# zu nutzen. Diese Kopie wird auch abgespeichert, während das Original mit 20%
# Schwarz abgedunkelt wird, um diesen Nachzieheffekt zu verstärken.
# Natürlich wird auch die neue Uhrzeit gespeichert und die Imagenummer erhöht.
        if ($uhrzeit_alt!=$uhrzeit) {
            $uhrzeit_alt=$uhrzeit;
            $image->GaussianBlur(radius=>4, sigma=>1000);
            my $temp1=$image->clone();
            my $temp1->Normalize();
            $temp1->Write('png:'.$folder.$imagenummer.'.png');
            $image->Colorize(fill=>'black',opacity=>'20%');
            $imagenummer++;
        }

# Die entsprechende X und Y Koordinate in der Grafik für die Geoposition
# berechnen, mit leichten Anpassungen.
        $x=int((($longi-$longimin)/$xpixel)+int(($neue_breite-$breite)/2))+25;
        $y=int((($latimax-$lati)/$ypixel)+int(($neue_hoehe-$hoehe)/2)*$aspect)+5;

# Wenn in Wien eine Linie auf "A" endet ist sie ein Autobus und bekommt die Farbe Rot, wenn
# die Linie mit "N" anfängt bekommt sie die Farbe Grün und Straßenbahnen erhalten die Farbe
# Blau.
        $color="#0000FF";
        if ($linie=~/A/) { $color="#FF0000";}
        if ($linie=~/N/) { $color="#00FF00"; }

# Damit die Punkte sichtbarer sind wird ein Kreis gezeichnet der einen Radius von
# 3 Pixel hat.
        $xmin=$x-1;
        $xmax=$x+1;
        $ymin=$y-1;
        $ymax=$y+1;
        $image->Draw(
            fill      => $color,
            primitive => 'circle',
            points    => $xmin.','.$ymin.' '.$xmax.','.$ymax,
        );
    }
    $durchlauf++;
}
```

Listing 4 (Fortsetzung)

ANWENDUNG

Kalenderdateien aus Webkalendern erstellen

Ein interessanter Kalender im Web, aber keine iCal-Datei. Schade, da muss man wohl alles abtippen, um es in den eigenen Kalender zu übernehmen. Oder auch nicht! Für den Deutschen Perl-Workshop im letzten Jahr, wollten die Orgas eine Kalenderdatei mit den Vorträgen zur Verfügung stellen. Damit ich das nicht abtippen muss, habe ich ein kleines Skript geschrieben, dass diese Dateien erzeugt.

Als erstes wird der Zeitplan des Workshops geholt. Dafür benutze ich einfach `LWP::Simple` mit der `get` Methode

```
my $url = 'http://www.perl-workshop.de/
           de/2009/zeitplan.html';
my $content = LWP::Simple::get( $url );
```

Danach müssen erstmal die Informationen aus der Webseite geholt werden. Ich persönlich bevorzuge für solche Aufgaben das Modul `Web::Scraper`, da es einfach zu benutzen ist und der Perl-Code schlank und lesbar bleibt. In der Ausgabe 7 von $foo habe ich das Modul schonmal genauer vorgestellt.

```
my $scraper = scraper {
  process 'div[class="zeitplan"]',
    'tage' => scraper {
      process 'table',
        'vortraege[]' => scraper {
          process 'td',
            'eintraege[]' => 'TEXT';
        };
      };

      result 'tage';
};

my $all_days = $scraper->scrape( $content )
                       ->{vortraege};
```

Aber das Modul soll hier ja nicht Hauptthema sein, deshalb nur eine kurze Erläuterung: Der Zeitplan an sich ist innerhalb eines `div`s mit der CSS-Klasse "zeitplan". Die Vorträge wiederum sind in einer Tabelle (eine Tabelle pro Tag) und alle benötigten Informationen in den Zellen der Tabelle. Dazu muss man noch wissen, dass immer vier Tabellenzellen zusammen die Informationen zu einem Vortrag liefern: Beginn, Dauer, Vortragender und Titel.

Danach beginnt die eigentliche Arbeit für die Kalenderdatei. Damit ich mich nicht näher mit dem ICal-Format beschäftigen muss, benutze ich das Modul `Data::ICal`. Für die Zeit-Felder empfiehlt es sich, noch das Modul `Date::ICal` zu verwenden, da man sonst genau wissen muss, wie Zeitangaben im ICal-Format aussehen. Ich bin da anfangs drübergestolpert, und habe mich dann gewundert, warum mein Kalender bei den Zeiten nur Murks angezeigt hat.

Als erstes muss ein neues ICal-Objekt erzeugt werden, das dann die komplette Kalenderdatei repräsentiert.

```
my $ical_all = Data::ICal->new;
```

Danach werden die einzelnen Tage respektive Tabellen durchgegangen und dort wiederum alle Einträge (Listing 1). Wie schon weiter oben geschrieben, gehören immer vier Zellen zu einander, die dann die Informationen über den Vortrag ergeben.

Die Zeilen, die "--" enthalten werden rausgenommen, da die Pausen nicht mit in der Kalenderdatei auftauchen sollen. Danach werden die Zeitangaben berechnet. Die Funktion `_get_epoche` berechnet mit Hilfe der Funktion `timelocal` aus `Time::Local` die Epochensekunden für einen Vortrag.

Für die einzelnen Einträge innerhalb der Kalenderdatei wird ein Event-Objekt erzeugt, das als Attribut eine Zusammenfassung, Startzeitpunkt und Endzeitpunkt hat. Für dieses Event-Objekt benötigen wir auch das `Date::ICal`-Modul und die Epochensekunden. Danach muss das Event noch mittels `add_entry` zu der Kalenderdatei hinzugefügt werden.

Zum Schluss werden die Kalenderinformationen in eine .ics-Datei geschrieben und schon ist die Kalenderdatei fertig.

```
open my $fh, '>', 'gpw2009.ics' or die $!;
print {$fh} $ical_all->as_string;
close $fh;
```

Natürlich muss das Skript für einen anderen Webkalender entsprechend angepasst werden. Aber das Beispiel zeigt, dass mit `Data::ICal` leicht Kalenderdateien erstellt werden können.

Renée Bäcker

```
for my $i ( 0 .. $#$all_days ){
    my $date = $days[$i];
    my $day  = $all_days->[$i];

    my (undef,$entries) = %$day;
    while( @$entries ){
        my $time      = shift @$entries;
        my $duration  = shift @$entries;
        my $presenter = shift @$entries;
        my $title     = shift @$entries;

        next if $presenter =~ /--/;

        my $dtstart =  _get_epoche( $date, $time );
        $duration   =~ s/\s*min\s*(?:\(Tutorial\))?//;
        $duration   = 30 if $duration =~ /BOF/;

        my $event   = Data::ICal::Entry::Event->new;
        $event->add_properties(
            summary => sprintf( "%s: %s", $presenter, $title ),
            dtstart => Date::ICal->new( epoch => $dtstart )->ical,
            dtend   => Date::ICal->new( epoch => $dtstart + ( $duration * 60 ) )->ical,
        );

        $ical_all->add_entry( $event );
    }
}
```

Listing 1

Perlfragen.de

Das Team von "$foo - Perl Magazin" stellt auf *http://perlfragen.de* eine Übersetzung der aktuellen Perl FAQ zur Verfügung. Damit soll Perl-Einsteigern ein Anlaufpunkt gegeben werden. Da noch nicht viel übersetzt ist, wird es vorübergehend einen Deutsch-Englisch-Mischmasch geben, der aber nach und nach abgebaut wird. Mithilfe ist natürlich gerne gesehen.

Wir versuchen die möglichst aktuelle Perl FAQ zu verwenden und schauen uns einmal im Monat die Änderungen am Original an. Wer selbst einen Blick auf das Original werfen will, kann sich die Quellen unter *http://github.com/briandfoy/*perlfaq anschauen. Die Links in den perlfaq-Dokumenten wurden für die Übersetzung nicht angepasst. So finden sich im Abschnitt "Wie kann ich etwas zu den perlfaq beitragen" die Links auf das Repository der englischen perlfaq.

Die Übersetzungen stellen wir auch im Wiki von Perl-Community.de und auf Github unter *http://github.com/reneeb/perlfaq_de* zur Verfügung. So kann jeder die Quellen holen und an einer eigenen Version arbeiten und/oder Fehler beseitigen.

Perl hilft der Feuerwehr

... jedenfalls bei Übungen. In diesem Artikel - der in der nächsten Ausgabe von $foo fortgesetzt wird - möchte ich ein Programm vorstellen, mit dem unsere Feuerwehr manche Übungen aufwertet. Manche Situationen kann man ohne großen finanziellen und organisatorischen Aufwand nicht realistisch üben. Deshalb habe ich ein Programm geschrieben, mit dem man Einsätze mit Brandmeldeanlagen üben kann - ohne dass man eine teure Brandmeldeanlage braucht. Wer dann doch noch etwas Geld (< 200 EUR) investieren will und kann, kann die Situation noch realistischer Darstellen.

Eine Brandmeldeanlage ist ein System, bei dem verschiedene Melder wie z.B. Rauchmelder und Druckknopfmelder (die roten Kästen, bei denen man im Brandfall die Scheibe einschlagen und den Knopf drücken soll) an eine zentrale Steuereinheit angeschlossen ist, die die Meldungen auswertet, den Feueralarm im Gebäude auslöst und automatisch die Feuerwehr alarmiert.

Die Feuerwehr hat in dem Gebäude dann verschiedene Elemente, um zu wissen, welcher Melder einen Brand gemeldet hat. Diese Elemente nennt man Anzeigetableau und Bedienfeld. Das ist deutschlandweit einheitlich. In meinem Programm werden diese Elemente in einer `WxPerl`-GUI dargestellt und ist Bestandteil in diesem Teil des Artikels.

Abbildung 1: Foto eines Anzeigetableaus mit "logischer" Einteilung für die GUI-Entwicklung

Um realistischer üben zu können, benötigen wir außerdem einen Melder, der durch Mitarbeiter ausgelöst werden kann, ohne dass die "richtige" Brandmeldeanlage losgeht. Wie dieser Melder in das System eingebunden wird, wird in der nächsten Ausgabe gezeigt.

"Designen" der Oberfläche

Das Aussehen der Elemente für die Feuerwehr ist in einer Norm geregelt, also brauche ich mir darüber keine großen Gedanken zu machen. Einen kleinen Nachteil hat das vorgegeben "Layout" aber: Ich darf es nicht anpassen. Bevor ich mit der Programmierung loslege, muss ich also erstmal wissen, welche Layoutsachen ich brauche. Ein Foto einer echten Anlage hilft. Mit `Gimp` zeichne ich mir dann Hilfslinien ein, so dass man schneller sieht, welche `Sizer` (wird Herbert Breunung in den nächsten Ausgaben noch genauer erklären) man benötigt. Zusätzlich kann auf den Fotos vermerkt werden, welche Elemente es gibt (`Label`, `Bitmap`, `Button`,...) (siehe Abbildung 1).

Das eignet sich dann hervorragend als "Bauanleitung" während der Programmierung.

Perl beschleunigt die Entwicklung

Für die Umsetzung des Programms verwende ich `Kephra` und `Padre`, zwei Editoren, die in Perl geschrieben sind - ebenfalls mit `WxPerl`. Diese Editoren haben den Vorteil, dass ich relativ einfach Plugins schreiben kann, die mir das Leben vereinfachen. Unter anderem habe ich ein Plugin geschrieben, mit dem das `:everything` beim Importieren von

Konstanten durch alle tatsächlich benutzten Konstanten ersetzt. Das bringt Speichervorteile.

Mit einem Skript, das `Module::Starter` benutzt, wird die Grundstruktur von `WxPerl`-Anwendungen erstellt, so dass es dann relativ leicht ist, das ganze auch auf CPAN zu stellen. Und durch Templates wird viel Schreibarbeit abgenommen.

Das Programm

Das Programm ist möglichst modular, damit Änderungen am Layout seperat von Änderungen an Aktionen vorgenommen werden können. Auch allgmeine Hilfsfunktionen sind ausgelagert. Damit verschiedene Meldertypen oder evtl. auch mal andere Medien als Auslöser implementiert werden können, wurde diese Logik in Plugins ausgelagert.

Unterstützung durch Tools

Nachdem Herbert Breunung (der auch das WxPerl-Tutorial für $foo schreibt) auf verschiedenen Veranstaltungen von `XRC` gesprochen hat und als Design-Tool `wxFormBuilder` empfohlen hat, habe ich die Oberfläche mit diesem Tool erstellt. Das Tool ist zwar nicht direkt für Perl gedacht - es unterstützt die Code-Generierung für C++ und Python - aber man kann das Layout als `XRC` exportieren.

Da ich die Oberfläche auch als "Projektdatei" für dieses Tool speichere, fallen zukünftige Änderungen am Aussehen der Anwendung nicht besonders schwer.

Nach etwas Ausprobieren, ging die Arbeit mit `wxFormBuilder` ganz gut von der Hand und das Gute ist, dass man das Ergebnis gleich sieht (Abbildung 2).

Im linken Teil des Tools sieht man die Oberfläche als Baumstruktur. Hier kann man die Elemente auch beliebig hin und her schieben. In der Mitte ist die "fertige" Oberfläche zu sehen und rechts kann man die Eigenschaften für jedes Element bearbeiten. Auf die Eigenschaften von Elementen will ich hier weniger eingehen, da das Teil des WxPerl-Tutorials sein wird.

In der Anwendung muss nur noch das `XRC` verwendet werden (Listing 1) und schon steht die Oberfläche.

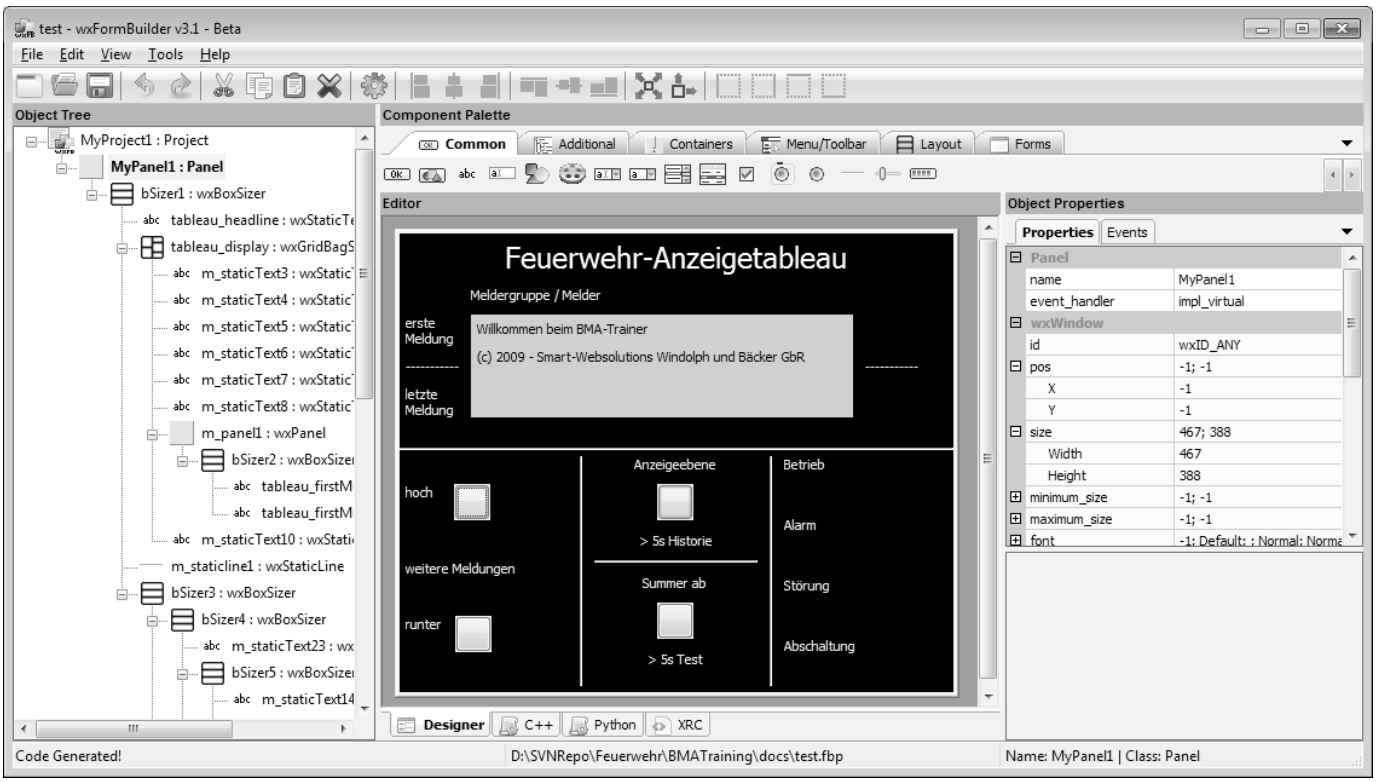


Abbildung 2: Entwickeln der GUI mit Hilfe von wxFormBuilder

```
# the XRC handler for the tableau
my $tableau_xrc = Wx::XmlResource->new;
$tableau_xrc->InitAllHandlers;
$tableau_xrc->Load( 'tableau.xrc' );

# insert main window here
my $main_sizer   = Wx::GridBagSizer->new(
                     0, 2 );
my $tableau      = $tableau_xrc->LoadPanel(
                    $frame, 'MyPanel1' );

$main_sizer->Add($tableau,
            Wx::GBPosition->new( 0, 0 ),
            Wx::GBSpan->new(1,1), wxLEFT |
            wxALIGN_CENTER_VERTICAL , 2);
```

Listing 1

Konfiguration des Übungsszenarios

Immer nur das gleiche Szenario zu üben ist sinnlos, denn auch die Einsätze sind immer unterschiedlich. Damit die Feuerwehr mit verschiedenen Situationen konfrontiert wird, werden nicht nur die Übungsteilnehmer immer wieder neu eingewiesen, auch das Programm sollte unterschiedliche Szenarien beherrschen. Da die gewünschten Szenarien auch auf örtliche Gegebenheiten angepasst werden sollen, muss das Programm konfiguriert werden können.
Aber nicht jeder Feuerwehrmann ist ein Programmierer. Auf dem Frankfurter Perl-Community Workshop 2009 hat Roland Schmitz einen interessanten Ansatz gezeigt, wie man Webseiten-Tests mit Excel bzw. OpenOffice Calc konfigurieren kann. Das Prinzip habe ich übernommen, so dass Meldungen auf dem Anzeigetableau beliebig geändert werden können - sowohl vom Text her als auch von der Anzahl her.

Eine Beispielkonfiguration ist in Abbildung 3 zu sehen. Hier werden in zwei Spalten alle notwendigen Angaben gemacht: In der ersten Spalte ist der Text festgelegt, der angezeigt werden soll und in der zweiten Spalte ist die "Wartezeit" von Anzeige des vorigen Texts bis zur Einblendung dieses Textes.

Zum Auslesen der Konfiguration verwende ich das Modul `Spreadsheet::ParseExcel`. Den Namen der Datei kann man über die Oberfläche bestimmen. Zurzeit ist es noch so, dass nur das erste Tabellenblatt in der Datei ausgewertet wird (Listing 2).

Neben den Aktionen, die die Feuerwehr an der Oberfläche machen kann, muss die Anzeige der Texte zeitgesteuert ablaufen. Dazu wird `Wx::Timer` verwendet. Da die Zeitspannen unterschiedlich sind, muss jedes Mal ein neuer Timer

```
sub _parse {
    my ($self) = @_;

    # file has to exist
    return [] unless -e $self->file and -f $self->file;

    # parse the szenario file
    my $parser = Spreadsheet::ParseExcel->new;
    my $book   = $parser->Parse( $self->file );

    my $sheet  = $book->worksheet(0);        # get first worksheet

    my ($min,$max) = $sheet->row_range;      # get start and end row

    # get all messages
    my @messages;
    ROW:
    for my $row ( $min .. $max ) {

        next ROW if $row == $min; # first line are the headers

        # text is in first column, timer in second column
        my $text = $sheet->get_cell( $row, 0 );
        my $time = $sheet->get_cell( $row, 1 );

        push @messages, { text => $text->value, time => $time->value }; # save messages
    }

    return \@messages; # return the messages
}
```

Listing 2

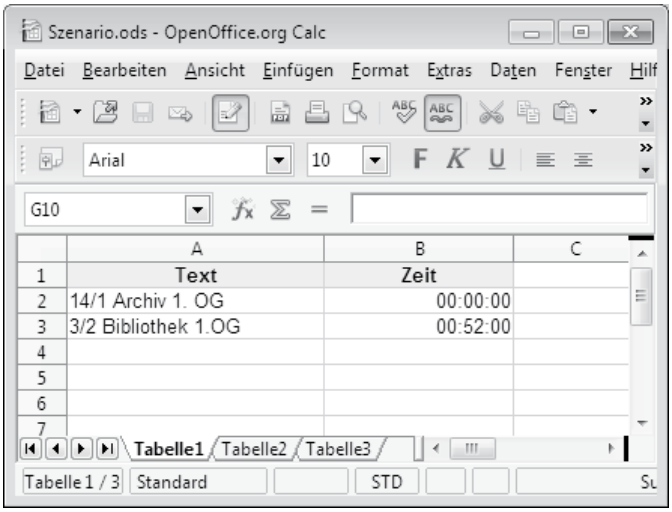


Abbildung 3: Tabelle zur Konfiguration eines Einsatzszenarios

```
sub start_alarm {
   my ($self) = @_;

   my $szenario = BMA::Szenario->new(
                     $self->szenario_file );
   my $sum      = 0;
   for my $message ( $szenario->messages ) {

     my $timer = Wx::Timer->new(
              $self->{frame}, wxID_ANY );
     $sum     += $message->{time};

     $timer->Start( ( $sum * 1000 ) + 1, 1 );
     EVT_TIMER( $self->{frame}, $timer,
          sub{ $self->display_message(
                 $message, $timer )
          } );
     }
 }
```

Listing 3

erzeugt werden (Listing 3).

Das `+ 1` bei `$timer->Start(...)` ist notwendig, damit der erste Text angezeigt wird. Der Timer kann nicht mit "0 Millisekunden" umgehen.

Die Anzeige des Textes passiert dann in der Methode `display_message`. Hier ist dann darauf zu achten, dass sowohl die "erste Meldung" als auch die "letzte Meldung" zu ändern ist, falls noch keine Meldung auf dem Display erschienen ist. Alle weiteren Meldungen werden nur unter "letzte Meldung" angezeigt.

Loggen der Aktionen

Für die Auswertung der Übung ist es natürlich auch ganz praktisch, wenn man weiß, wann welche Aktion durchgeführt wurde. Wann also die Feuerwehr das erste Mal an den Bedienelementen war, wann der Melder wieder aktiviert wurde und wann die Feuerwehr den Übungsort wieder verlassen hat. Zusätzlich ist momentan ein Programm in Planung, mit dem der Übungsleiter schon während der Übung diese Auswertung lesen und darauf reagieren kann, um z.B. einen weiteren Melder oder Nebelmaschinen zu aktivieren.

Logging-Module gibt es viele auf CPAN. Ich bevorzuge `Log::Log4perl` (siehe auch $foo Nr. 8) für solche Aufgaben. In der Konfiguration habe ich dann über ein angepasstes Layout den Statuscode von der eigentlichen Meldung getrennt (Listing 4), so dass die Aktionen sehr leicht identifiziert werden können (siehe Listing 5).

Noch ein Hinweis für diejenigen, die das Programm ausprobieren wollen: Die Oberfläche hat kein ständig sichtbares Menü, damit dieses bei der Übung nicht ablenkt. Um das Menü einzublenden muss `F4` gedrückt werden.

```
log4perl.logger = WARN, FILER
log4perl.appender.FILER = Log::Log4perl::Appender::Screen
log4perl.appender.FILER.name = file1
log4perl.appender.FILER.filename = bma.log
log4perl.appender.FILER.layout = PatternLayout
log4perl.appender.FILER.layout.cspec.S = \
      sub { my ($code) = $_[1] =~ /^(\d{3}):\s+/; $code ||= '000'; $code }
log4perl.appender.FILER.layout.cspec.W = \
      sub { (my $without_status = $_[1]) =~ s/^(\d{3}):\s+//; $without_status }
log4perl.appender.FILER.layout.ConversionPattern = [%S] %W%n
```

Renée Bäcker

Listing 4

```
D:\>perl bma.pl
[100] Start Alarm
[110] 14/1 Archiv 1. OG
[110] 3/2 Bibliothek 1.OG
[110] 3/3 Test EG
```

Listing 5

Perl Scopes Tutorial - Teil 3

Im vorigen Teil wurde Scoping in Funktionen und Schleifen betrachtet und Probleme, die beim Verschachteln von Funktionen auftreten können, wurden erläutert und Lösungen aufgezeigt. Heute werden noch Closures vorgestellt, bevor wir uns den Sprachelementen, die Scoping betreffen, zuwenden und dabei mit `local` *beginnen.*

Closures

Verbindet man das Konzept persistenter lexikalischer Variable mit der Zuweisung von Codereferenzen, erhält man ein Konstrukt namens *Closure*, d.h. eine zur Laufzeit angelegte Funktion, die auf lexikalische Variable außerhalb ihres eigenen Scopes zugreift:

```
sub gen_multi {
  my $multi = shift;
  return sub {
    $multi * shift;
  }
}
```

Die Funktion `gen_multi()` ist ein *Funktionsgenerator*, d.h. ihr Rückgabewert ist selbst eine Funktion (genauer gesagt, eine Referenz darauf), die an einen Typeglob oder einen Skalar zugewiesen werden kann:

```
my $mult20 = gen_multi(20);
my $mult30 = gen_multi(30);
print $mult20->(2);      # Gibt 40 aus
print $mult30->(3);      # Gibt 90 aus
```

Wie bereits erkennbar, kann es beliebig viele solche Funktionen, jede mit einem eigenen Multiplikator (einer eigenen *Instanz* von `$multi`) geben. Die lexikalische Variable wird daher mit dem Wert, den sie beim Anlegen der Closure hatte, in dieser *eingeschlossen* (Name!) und später, wenn die Closure aufgerufen wird, wird dieser Wert wieder verwendet.

Für das Verständnis über die Wirkungsweise von Closures ist es hilfreich, sich vorzustellen, dass ein Konstrukt wie

```
sub {
  $multi * shift;
}
```

letztlich auch nur einen Wert - ein *Codeliteral* (genauer gesagt: eine Referenz auf ein Codeliteral) - darstellt. Und ähnlich, wie in einem Stringliteral bei einer Zuweisung

```
my $str = "Guten Morgen, $name!\n";
```

alle Variablen durch deren momentanen Werte ersetzt (*interpoliert*) werden, werden auch beim Anlegen einer Closure die Werte angesprochener lexikalischer Variable, die außerhalb ihres Scopes definiert wurden, in den Code übernommen [1]. Ein Befehl wie

```
my $coderef = sub {...};
```

ist daher eine zur Laufzeit ausgeführte Zuweisung eines Ausdrucks an eine Variable, genauso wie im obigen Beispiel die Zuweisung des Stringliterals.

Closures kann man sich als *Funktions-Templates* vorstellen, die zur Laufzeit mit den Werten der jeweiligen lexikalischen Variablen befüllt werden. Und genauso, wie in der obigen Zuweisung jede folgende Änderung von `$name` am Wert von `$str` nichts mehr ändert, bleibt auch der Wert der lexikalischen Variable in der Closure unverändert (außer die Closure ändert ihn selbst), unabhängig davon, was mit der Variable nach dem Anlegen der Closure weiter passiert.

Das ist das Typische an Closures - sie werden innerhalb einer bestimmten lexikalischen Umgebung definiert und schließen diese Umgebung in sich ein. Werden sie später aufgerufen, wird diese Umgebung während ihrer Ausführung wieder hergestellt, wann und von wo auch immer sie loslaufen.

Beachte, dass nicht jede zur Laufzeit angelegte Funktion eine Closure darstellt:

```
sub gen_multi {
  return sub {
    shift * shift;
  };
}

my $mult = gen_multi();  # Kein Argument
print $mult->(2, 3);     # Gibt 6 aus
```

Hier liefert `gen_multi()` zwar wiederum eine Funktion zurück, diese beinhaltet aber keine lexikalische Umgebung, da an den Funktionsgenerator keine Argumente übergeben werden, die diese Umgebung einrichten könnten. Anders gesagt: ruft man hier `gen_multi()` zehnmal auf, erhält man zehnmal dieselbe, vollkommen identische Funktion zurück. Closures hingegen beinhalten zumindest eine lexikalische Variable, die für jede Closure unterschiedliche Werte annehmen kann. Daher kann man aus der Funktions-Template durch Aufruf des Funktionsgenerators mit unterschiedlichen Argumenten (theoretisch) beliebig viele verschiedene Funktionen erzeugen - vom Code her identisch, aber jede mit einer anderen lexikalischen Umgebung.

Aber auch die folgenden Funktionen gelten nicht als Closures, obwohl sie ebenfalls gemeinsam auf eine lexikalische Variable zugreifen:

```
{
   my $hugo = 0;

   sub inc {
     $hugo++;
   }

   sub dec {
     $hugo--;
   }

   sub show {
     print "Hugo = $hugo\n";
   }
}
```

Diese Funktionen werden *definiert*, d.h. sie werden bereits beim Kompilieren eingerichtet. Für Closures ist aber der Zugriff auf außerhalb des Funktionsscopes deklarierte lexikalische Variable *zur Laufzeit* wesentlich - nur so wird der Mechanismus des *Einschließens* der Variablen mit dem gerade aktuellen Wert (das Bilden eigener Variableninstanzen) möglich. Und natürlich funktioniert das auch nur mit lexikalischen Variablen, da von einer globalen Variable ja immer nur eine Instanz existiert.

Im Unterschied zu statisch definierten Funktionen sind Closures *vergänglich* - und damit auch die in ihnen enthaltenen, persistenten Variablen. Im obigen Beispiel wurde eine Closure angelegt:

```
my $mult20 = gen_multi(20);
```

Der an den Funktionsgenerator übergebene Wert `20` ist nun in der persistenten Variable innerhalb der Closure gespeichert und bleibt bestehen, solange die Closure bestehen bleibt. Wird jedoch die Variable, die die Codereferenz auf die Closure enthält, wieder gelöscht oder überschrieben, z.B.:

```
undef $mult20;
```

so geht es ans Aufräumen - da die Closure nun nicht mehr ansprechbar ist, wird Perl den Platz, den ihr Code belegt hat, freigeben. Dabei wird es auf die persistente Variable stoßen und auch diese, da niemand mehr darauf zugreift, entfernen.

Abschließend sei noch darauf hingewiesen, dass die in `perlref` getroffene Aussage

```
In the general case, then, named subroutines
do not nest properly,
although anonymous ones do.
```

ungenau formuliert ist. Wie im vorigen Teil dieses Tutorials ($foo #12) gezeigt wurde, geht es nicht um die Anonymität, sondern um den Zeitpunkt und die Art des Einrichtens. Mit

```
sub myfunc {...}
```

werden stets nur benannte Funktionen beim Kompilieren definiert; hier kann es beim Verschachteln von Scopes Probleme geben. Man kann aber mit

```
local *myfunc = sub {...};
```

zur Laufzeit ebenfalls eine benannte Funktion (die natürlich auch eine Closure sein kann) einrichten. Durch die Verwendung von `local` kann man sogar deren Benutzung auf den aktuellen Scope einschränken und damit *lokale Funktionen* erzeugen - etwas, das in Perl ansonsten nicht möglich ist. Die Funktion kann sodann mit

```
myfunc(...);
```

wie jede andere benannte Funktion aufgerufen werden. Es lassen sich also auch benannte Funktionen ohne Probleme verschachteln, *solange sie nicht statisch definiert werden*. Eine mögliche Anonymität spielt dabei keine Rolle.

Sprachelemente für Scoping

Perl kennt vier Sprachelemente (*Schlüsselworte*), die das Scoping beeinflussen: die Deklarationen `my`, `our` und `state` sowie die Anweisung `local`. In diesem Abschnitt werden diese Elemente näher betrachtet, und zwar in der Reihenfolge, in der sie nach und nach zum Befehlssatz von Perl hinzugefügt wurden.

Globale Werte lokalisieren mit `local`

`local` gab es bereits in Perl 4 und es war dort die einzige Möglichkeit, Scoping (nämlich dynamisches Scoping) durchzuführen.

`local` ist eine Laufzeitanweisung [2], keine Deklaration. Daher wird folgender Code nicht funktionieren:

```
use strict 'vars';
local $hugo = 'Test';
```

da `strict 'vars'` die Deklaration aller Variablen bereits beim Kompilieren erwartet, `local` aber nichts deklariert, sondern erst zur Laufzeit wirksam wird.

`local` implementiert *dynamisches Scoping*, d.h. die Beeinflussung von Scopes zur Laufzeit. `local` nimmt den augenblicklichen Wert der angegebenen Variable und legt diesen in einem eigenen Bereich (*Scopestack*) ab. Vor dem Verlassen des aktuellen Scopes wird die Variable wieder auf diesen Wert zurückgesetzt; der Wert wird also nur temporär - innerhalb des Scopes - durch einen neuen Wert ersetzt. Diesen Vorgang bezeichnet man auch als **lokalisieren**.

Es ist wichtig zu verstehen, dass `local` nur die *Werte* globaler Variable (Paketvariable) sichert, die Variable selbst - und ihr allenfalls anhaftende besondere Eigenschaften - bleiben unverändert und wirken auch mit dem neuen Wert weiter. Von dieser Tatsache macht man z.B. beim Arbeiten mit `$/` (der Variable, die das Trennzeichen beim Einlesen von Daten enthält) Gebrauch:

```
{ # Neuer Scope
  # Jetzt kein Trennzeichen mehr (undef)
  local $/;
  # Alle Saetze auf einmal einlesen
  $alldata = <IN>;
} # Scopeende, vorheriger Wert von $/
  # wieder aktiv
```

Ohne den `local` Befehl müsste man den Wert von `$/` vor dem Ändern wegsichern und nach dem Verlassen des Blocks wieder herstellen, andernfalls würde auch bei allen folgenden Leseoperationen der gesamte, zur Verfügung stehende Dateiinhalt auf einmal eingelesen werden.

Hier hätte die Deklaration einer lexikalischen Variable nicht den gewünschten Erfolg, da Perl beim Einlesen von Daten in der *globalen* Variable `$/` nach dem Trennzeichen Ausschau hält, nicht in einer Lexikalischen [3].

Der lokalisierte Wert bleibt nicht erhalten, sondern die Variable enthält nach Ausführung des Befehls den Wert `undef` (das ist eine häufige Fehlerquelle!). Im vorigen Beispiel bewirkt daher

```
local $/;
```

dass `$/` nun auf `undef` gesetzt wird, was hier durchaus erwünscht ist. Möchte man einer Variable beim Lokalisieren gleichzeitig einen neuen Wert zuweisen, kann man die Zuweisung zum Befehl hinzufügen:

```
# Nun koennen wir DOS-Dateien lesen!
local $/ = "\n\r";
```

Mehrere Variable können gleichzeitig lokalisiert werden, wenn sie als Liste - in Klammern gesetzt - angegeben werden:

```
local ($x, $y, $z) = (2, 3, 4);
```

Im Falle von Arrays und Hashes ist es auch möglich, nur einzelne Elemente zu lokalisieren:

```
local $hugo[4];
local $alpha{Xanadu};
```

was mit `my`, `our` oder `state` nicht erlaubt ist.

Ähnlich, wie beim Deklarieren einer lexikalischen Variable in einem Scope eine gleichnamige, außerhalb des Scopes deklarierte Variable nicht mehr ansprechbar ist, kann auch der ursprüngliche Wert einer Variable nach dem Lokalisieren innerhalb des aktuellen Scopes nicht mehr angesprochen werden, auch nicht aus anderen Funktionen heraus:

```
sub prt_x {
  print $x;
}
$x = 4;
prt_x();            # Gibt 4 aus
{                   # Innerer Scope
  local $x = 5;
  prt_x();          # Gibt 5 aus
}
prt_x();            # Gibt 4 aus
```

Beachte, dass `prt_x()` außerhalb des inneren Scopes deklariert wurde, aber im Scope dennoch auf den neuen Wert von `$x` zugreift. Im Unterschied zu statischem Scoping geht es hier darum, wann `prt_x()` *aufgerufen wird*; der Scope, in dem es deklariert wurde, ist nebensächlich.

Lokalisiert kann *jede* globale Variable werden, auch solche aus anderen Packages. Verwendet ein Modul eine globale Variable zur Konfiguration, kann man damit innerhalb von Scopes die Konfiguration vorübergehend ändern:

```
# Debugging ausschalten
$MyModule::debug = 0;
...
{
  # Aber fuer diesen Block einschalten
  local $MyModule::debug = 1;
  ...
} # Jetzt wieder ausgeschaltet
```

Eine abschließende Bemerkung zum allgemeinen Gebrauch von `local`: In folgendem Codestück

```
while (my $var = <IN>) {
  local $x = substr($var, 1, 2);
  ...
}
```

fällt zunächst nichts Böses auf - dennoch steckt hier ein kleiner Wurm drin: da `local` eine Anweisung ist, wird es bei *jedem* Schleifendurchlauf ausgeführt. Anders gesagt, wird der Wert von `$x` in der Schleife jedesmal aufs Neue lokalisiert und nach dem Schleifenende wieder hergestellt. Das ist natürlich unnötig und sollte durch ein `local` außerhalb der Schleife ersetzt werden.

Das folgende Beispiel

```
$x = 1;
$y = \$x;
{
  local $x = 2;
  print $$y;              # Gibt 1 aus
}
```

zeigt noch einmal, dass *Werte* und keine Variablen lokalisiert werden: im Scope wird der Wert von `$x` lokalisiert. Dadurch wird die Bindung des Wertes 1 an die Variable `$x` innerhalb des Scopes unterbrochen. Die vorher an `$y` zugewiesene Referenz auf diesen Wert ist jedoch von dieser Bindung unabhängig und ändert sich durch das Lokalisieren von `$x` nicht. Durch das Dereferenzieren wird daher innerhalb des Scopes der ursprüngliche Wert von `$x` zurückgegeben. Tatsächlich ist das ein einfacher Trick, auch nach dem Lokalisieren einer Variable auf deren alten Wert zuzugreifen.

Anwendungsmöglichkeiten von `local`

`local` hat durch die Einführung von `my` und lexikalischen Variablen einiges an Bedeutung eingebüßt. Dennoch gibt es auch heute noch Fälle, wo man nicht darauf verzichten kann.

• Lokalisieren spezieller Variable

Neben den bereits erwähnten Interpunktionsvariablen, die nicht als lexikalisch deklariert werden können und daher immer global angenommen werden (und für die vielfach besondere Regeln für das Scoping gelten, siehe *perlvar*), kennt Perl noch weitere Variable mit spezieller Bedeutung (z.B. `@ARGV`, `%ENV`, `@INC`, `%SIG`). Es ist zwar nicht verboten, gleichnamige lexikalische Variable zu verwenden, doch nur die globalen Versionen dieser Variable besitzen die in `perlvar` dokumentierten Eigenschaften. Hingegen macht das Lokalisieren dieser Variablen durchaus Sinn, womit man häufig interessante Effekte erzielt:

```
my $pattern = shift;
my %found;
{
  local @ARGV = </mydir/*>;
  while (<>) {
     $found{$ARGV}++ if /$pattern/o;
  }
}
print "$_: $found{$_}\n"
   foreach keys %found;
```

Dieses Beispiel zeigt, wie eine oder mehrere Dateien in einem Verzeichnis schnell nach einem Muster durchsucht werden können, indem das spezielle Array `@ARGV` lokalisiert wird. `@ARGV` enthält normalerweise alle an ein Skript übergebenen Argumente; häufig sind das Namen einzulesender Dateien. Dann kann man mittels des `<>` Operators (eine Kurzform für `readline *ARGV`) diese Dateien Zeile für Zeile einlesen; Perl übernimmt das implizite Öffnen und Schließen der Dateien und hält den Namen der gerade geöffneten Datei im Skalar `$ARGV` bereit.

Dieser Mechanismus ist aber nicht an die Argumente der Befehlszeile gebunden, sondern wann immer mittels <> eine Zeile gelesen werden soll, wird diese aus der nächsten in `@ARGV` aufscheinenden Datei geholt (und die Datei ggf. vorher geöffnet). Durch das Lokalisieren wird für `@ARGV` ein neuer Wert angelegt und daher bei der nächsten Leseoperation die erste Datei des aktualisierten `@ARGV` verarbeitet.

Der alte Wert von `@ARGV` wird durch das Lokalisieren gesichert und am Scopeende wieder hergestellt. Weitere Leseoperationen beziehen sich dann wieder auf die Dateien des vorigen Wertes, wo exakt an jenem Punkt weitergemacht wird, an dem vor der Ausführung der Schleife angehalten wurde. Man könnte also den obigen Code auch rekursiv mit jeweils anderen Werten für `@ARGV` aufrufen, ohne dass es zu Problemen kommen würde.

Beachte, dass es nicht verboten ist,

```
my @ARGV = </mydir/*>;
```

zu schreiben. Allerdings wird das nicht wie erwartet funktionieren, denn der <>-Operator untersucht nur das *globale* `@ARGV`, nicht eine lexikalische Version.

• Lokalisieren von Array- oder Hashelementen oder -slices
Hash- und Arrayelemente enthalten Werte, die genau wie die Werte jeder anderen Variable lokalisiert und temporär durch andere Werte ersetzt werden können. Obwohl man selten davon Gebrauch machen wird, kann es doch in Einzelfällen sinnvoll sein:

```
{
  local $SIG{INT} = 'IGNORE';
  ...
}
```

Hier wird ein Element des speziellen Hashes `%SIG`, welches die Namen aller Signalhandler enthält, auf einen neuen Wert gesetzt, wodurch innerhalb des Scopes das Interrupt-Signal anders verarbeitet wird. Natürlich könnte man auch den ganzen Hash lokalisieren, aber dies hätte das Löschen aller anderen bereits gesetzten Signalhandler (bzw. das Setzen auf deren Standardwerte) zur Folge, was meistens nicht beabsichtigt ist. Durch das Lokalisieren eines Elements hingegen belässt man die übrigen Elemente unverändert.

Auch hier muss `local` verwendet werden, weil Perl nur im *globalen* Hash `%SIG` nach Handlern Ausschau hält. Das Verwenden von `my` in diesem Konstrukt würde sich außerdem der Parser nicht gefallen lassen.

Auch Slices (Elementgruppen) lassen sich lokalisieren:

```
local @x[1, 2, 3];
# local ($x[1], $x[2], $x[3]);
local @y{a, b, c};
# local ($y{a}, $y{b}, $y{c});
```

Beim Lokalisieren von Array- und Hashelementen ist zu beachten, dass diese nach Scopeende auch dann wieder hergestellt werden, wenn dies eine Änderung des Arrays oder Hashes nach sich zieht:

```
@x = qw(1 2 3 4 5);
{
  local $x[4] = 11;
  pop @x; pop @x; pop @x;
  # @x enthaelt nun (1 2)
}
# @x enthaelt jetzt (1 2 undef undef 5)
```

Das Lokalisieren des fünften Elements zwingt Perl dazu, das Array, das vor dem Verlassen des Scopes nur mehr aus zwei Elementen bestanden hat (und selbst nicht lokalisiert wurde), wieder auf die ursprüngliche Größe von fünf Elementen zu erweitern, wobei gelöschte, aber nicht lokalisierte Elemente mit `undef` initialisiert werden.

Ähnlich verhält es sich mit Hashes - das Lokalisieren eines Hashelements stellt sicher, dass dieses beim Verlassen des Scopes wieder hergestellt wird, auch wenn es (oder der ganze Hash) zwischenzeitlich gelöscht worden sein sollte:

```
%y = (a => 1, b => 2, c => 3);
{
  local $y{c};        # Element wegsichern
  ...
  delete $y{c};       # oder: undef %y;
  print exists $y{c};
  # Element existiert nicht
}
print exists $y{c};      # Gibt 1 aus,
# Element existiert wieder
```

• Lokalisieren von Typeglobs
Durch das Lokalisieren eines Typeglobs wird *alles* mit diesem Namen lokalisiert:

```
local *x;
# entspricht local ($x, @x, %x, &x, <x>);
```

(Beachte, dass hier <x> als Platzhalter für Handles steht, das ist aber keine gültige Syntax.)

Obwohl es selten vorkommen wird, dass man alles Gleichnamige gleichzeitig lokalisieren muss, ist vor allem die Tatsache von Bedeutung, dass damit auch ein Lokalisieren jener Wertetypen von Perl möglich ist, die ansonsten nicht direkt angesprochen werden können. Besonders im Umgang mit Filehandles wurde früher (vor Perl 5.6) hiervon oft Gebrauch gemacht:

```
open FILE, '>x.x';
...
{
  local *STDOUT = *FILE; # Typeglob-Aliasing
  print "Halli";         # Geht nach x.x
}
print "Hallo";     # wieder nach STDOUT
```

Hier wird mittels der Typeglob-Zuweisung der Filehandle `STDOUT` temporär auf den Filehandle `FILE` gesetzt, der folgende `print` Befehl schreibt dadurch in die Datei `x.x`.`print` selbst weiß davon nichts, die Ausgabe wird wie immer nach `STDOUT` geschickt, aber durch das Aliasing der Typeglobs (`STDOUT` ist innerhalb des Scopes gleichbedeutend mit `FILE`) landet die Ausgabe wie durch Zauberhand dennoch in `x.x`.

Ein anderes Beispiel, das ein *rekursives* Lokalisieren veranschaulicht, wird mit folgender Funktion, die einen `include` Befehl implementiert, gezeigt: die angegebene Datei wird geöffnet und eingelesen; wird dabei die Anweisung `#include` am Zeilenbeginn entdeckt, ruft sich die Funktion mit dem der Anweisung folgenden Dateinamen selber auf:

```
my $depth = 0;

sub include {
  $depth++ < 10 or
      die "Function nested too deeply";
  my $name = shift;
  local *FH;
  open FH, $name or
      die "Failed to open $name: $!\n";
  while (<FH>) {
    include($1),
      next if /^#\s*include\s+(.+)/;
    ...
  }
}
```

Bei jedem Aufruf wird der Typeglob und damit der verwendete Dateihandle lokalisiert. Innerhalb der Funktion wird dann die lokalisierte Version zum Öffnen und Einlesen der als Argument übergebenen Datei verwendet. Nach dem Verlassen der Funktion wird in der vorigen Datei mit der dem `#include` folgenden Zeile Datei weitergemacht. Der Zähler stellt sicher, dass die Rekursion nicht zu tief wird, wenn z.B. zwei Dateien einander inkludieren.

Ein Seiteneffekt davon ist, dass Perl die über den lokalisierten Typeglob geöffnete Datei automatisch schließt, bevor der jeweilige Scope verlassen und der vorige Wert des Typeglobs wieder hergestellt wird.

Neben Dateihandles gibt es auch die Möglichkeit, mittels Typeglobs *Funktionen* zu lokalisieren:

```
sub myfunc {
  print "Old: ", shift;
}
...
# Gibt "Old: 1" aus
myfunc(1);
{
  # Kein "Subroutine redefined"
  no warnings 'redefine';
  local *myfunc = sub {
    print "New: ", shift;
  };
  myfunc(2);      # Gibt "New: 2" aus
}
myfunc(1);        # Gibt "Old: 1" aus
```

Hier wird dem lokalisierten Typeglob - wie beim Einrichten benannter Funktionen zur Laufzeit - eine Codereferenz zugewiesen, die den vorigen Wert überschreibt. Das `no warnings` Pragma unterdrückt die Ausgabe der `Subroutine redefined` Meldung.

Beachte: die Syntax

```
local &myfunc;
```

ist zwar in Verbindung mit `lvalue`-Funktionen zulässig, hat aber derzeit keine Bedeutung (der Parser ignoriert die `local`-Anweisung).. Zum Lokalisieren von Funktionen sind immer Typeglobs zu verwenden.

Beachte, dass beim Lokalisieren der Typeglobs von Perls Interpunktionsvariablen Vorsicht geboten ist - viele ihrer speziellen Eigenschaften sind im Typeglob abgespeichert und gehen durch dessen Lokalisierung verloren:

```
local $/;
# Lokalisiert den Skalarwert - OK
$/ = "\n\r";
# Spezielles $/: funktioniert
local */;
# Lokalisiert den Typeglob - nicht OK!
$/ = "\n\r";
# $/ nicht mehr speziell:
# funktioniert nicht!
```

In diesem Beispiel wird die gewünschte Wirkung ausbleiben, da Perl beim Einlesen (`readline`) nur im ursprünglichen,

jetzt lokalisierten Typeglob (und damit auf das alte `$/` nach dem Wert von `$/` sucht und nicht in der neuen Version [4]. Durch das Lokalisieren hat die Zuweisung auf den alten Typeglob jedoch keinen Einfluss mehr, sein **Wert** - und dadurch das Verhalten von `readline` - ändert sich nicht.

Fazit: das Lokalisieren einer **Variable** lokalisiert ihren Wert, das Lokalisieren eines Typeglobs hingegen alle zugehörigen Variable, einschließlich deren besondere Eigenschaften. Der neu angelegte Typeglob (und damit auch im obigen Beispiel die neue Variable `$/`) besitzt keine besonderen Eigenschaften mehr.

Im nächsten Teil wird noch ein kurzer Blick auf lexikalische Pragmas geworfen, bevor die Direktiven, die auf Scoping Einfluss haben, beschrieben werden: `my`, `our` *und* `state`.

Ferry Bolhár-Nordenkampf

[1] Mit "Übernehmen" ist hier gemeint, dass die Variable innerhalb der Closure mit dem übernommenen Wert initialisiert wird, aber dennoch - im Unterschied zur Interpolation in einem String - eine Variable bleibt, die vom Code der Closure später auch verändert werden kann. Ruft man also

```
my $mult20 = gen_mult(20);
```

auf, so lautet der erzeugte Code (in etwa):

```
sub {
  my $mult = 20;
  $mult * shift;
}
```

und nicht :

```
sub {
  20 * shift;
}
```

Für den praktischen Gebrauch ist der Unterschied meist gering; im zweiten Fall allerdings könnte die Closure die Variable nicht mehr modifizieren.

[2] Der Name `local` ist nicht besonders glücklich gewählt; `localize` oder `save` wären wohl passender gewesen, da sie den Befehl als solchen besser erkennbar gemacht hätten. Dennoch gibt es auch in Perl 6 ein `local`. Alte Gewohnheiten behält man eben gerne bei, egal ob gut oder schlecht.

[3] Außerdem dürfen die Namen lexikalischer Variable nur alphanumerische Zeichen enthalten, Interpunktionsvariable wie `$/` werden *immer* als global (im Package `main`) angenommen. Der Parser weist daher eine Deklaration wie

```
my $/;
```

mit

```
Can't use global $/ in "my"
```

zurück. Lediglich `$_` darf seit Perl 5.10 auch als lexikalische Variable deklariert und verwendet werden.

[4] Beachte in diesem Zusammenhang, dass das *Read-Only* (Schreibschutz)-Attribut einer Variable in ihrem *Wert* kodiert ist, nicht im Typeglob. Daher funktioniert folgender Code nicht:

```
$] = 'Hugo';
# Modification of a read-only value attempted
```

Dieser hingegen aber schon:

```
local $] = 'Hugo';        # OK
```

Vor der Zuweisung wird der schreibgeschützte Wert weggesichert, anschließend wird der Variable ganz normal ein neuer Wert zugewiesen. Ob eine Perl-Version `Hugo` sinnvoll ist, ist eine andere Frage. Aber dieser Code wird fehlerfrei ablaufen.

Perl Upload Form

Eigentlich wollte ich nur mal eben im Internet nachsehen, wie das mit dem Hochladen von Bildern noch mal ging. Dass `CGI.pm` dazu bereits recht viel Dokumentation bietet, hatte ich vergessen. Bei den Recherchen stieß ich auf eine Unmenge unbrauchbarer oder uralter Codes, die den schlechten Ruf von Perl wohl erheblich stützen. Daher beschloss ich es einmal besser zu machen - oder zumindest zu versuchen, es besser zu machen.

Funktionsumfang

Das "besser machen wollen" resultierte eigentlich aus dem Modellversuch, den ich anstellte, um zu prüfen, ob das Upload-Formular auch genau das macht, was ich will. Solch kleinere Aufgabenstellungen vom großen Ganzen abgekapselt zu betrachten hilft mir, eine solche Teilaufgabenstellung gezielt und gut umsetzen zu können. Auf Dauer wird es allerdings langweilig. Deshalb trägt die Klasse in der Vorlage, die ich für meine `CGI::Application`-Vorabtests verwende, in Erinnerung an meinen Englischunterricht den beschaulichen Namen CatcherInTheRye.

Was soll nun das Upload-Formular leisten? Offensichtlich ist das Ziel des Uploads einer Datei. Impliziert wurde bereits, dass die Dateien nur Bilder sein sollen. Außerdem würde ich gerne auch direkt im Live-System sehen, ob die Dateien korrekt ankommen und auch ausgeliefert werden können. Das ist nützlich, um dort, sozusagen als Vorab-Test, z.B. fehlende technische Voraussetzungen zu identifizieren (fehlende Module, zu alte Perl-Version, etc.). Um etwas Spannung vorweg zu nehmen: es hat sich im Laufe der Entwicklung des Programms als außerordentlich nützlich erwiesen, eine Datei, die hochgeladen wurde, auch einmal schnell wieder löschen zu können. Das muss also auch noch rein. Schlussendlich impliziere ich bei der Programmierung von Web-Formularen auch immer die Anzeige von Fehlern, wenn der Benutzer ungültige Eingaben abschickt. Eingaben die der Benutzer richtig geraten hat, soll er natürlich behalten dürfen und nicht neu eingeben müssen. Das alles impliziere ich, weil das Modul, welches dazu verwendet wird, einem genau solche Arbeiten fast vollständig abnimmt.

Zusammengefasst gilt es folgendes zu implementieren:

- Upload von Bildern
- Evaluation der Eingaben und Anzeige von Fehlern
- Anzeigen hochgeladener Bilder
- Öffnen & Löschen von Bildern

CGI::Application::Plugin::ValidateRM

Das Projekt, in welches das Upload-Form eingebaut werden sollte, ist mit `CGI::Application` programmiert. Dafür gibt es einige tolle Module, die den Umgang mit Formularen vereinfachen, oder zumindest angenehmer gestalten, als eine gänzlich selbst programmierte Prüfung. Im speziellen meine ich damit `Data::FormValidator` und das entsprechende `CGI::Application::Plugin::ValidateRM`. ValidateRM wird im Folgenden vorgestellt.

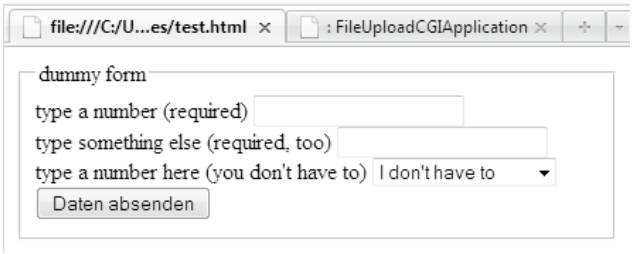
Das Plugin ist wirklich sehr einfach zu benutzen, den Großteil der Arbeit machen das Erstellen der Profile und das Durchblicken des komplexen `Data::FormValidator` aus. Hier ein Codebeispiel für das Plugin ValidateRM:

```
use CGI::Application::Plugin::ValidateRM;

my $profile = {
    required => [qw/required_field_number
                    another_required_field/],
    optional => [qw/optional_number
                    submit/],
    constraints => {
        required_field_number => qr/^\d+$/,
        optional_number => qr/^\d+$/,
    },
};

my $results = $self->check_rm(
     'form_display', $form_profile
   )
   || return $self->check_rm_error_page();
```

Der Code zeigt ein Profil für ein Formular, welches (im Idealfall) 3 Felder und einen Submit-Button enthält. Es könnte so aussehen:



Wenn das Formular abgesandt wird, werden die Benutzereingaben anhand des Profils geprüft. Trifft eine der Regeln nicht zu, landet der Benutzer wieder im Formular. Das Schöne hierbei ist, dass die Feldwerte - auch die von select-Boxen - wieder in das Formular eingefügt werden. Man kann dann auch noch Fehlermeldungen spezifizieren. Ich für meinen Teil habe davon abgesehen. Mehr dazu im Abschnitt "Fehlermeldungen visualisieren".

Data::FormValidator::Constraints::Upload

`Data::FormValidator` bietet in der Grundversion nur Methoden zur Prüfung textueller Felder. Für Uploads gibt es `Data::FormValidator::Constraints::Upload`. Dieses geniale Modul stellt Methoden zur Prüfung hochgeladener Dateien bereit, so z.B. die Methode `file_format()` der man eine Liste mit MIME-Types geben kann, die hochgeladen werden dürfen. Wird eine Datei eines anderen MIME-Types hochgeladen, gilt das als ungültige Formulareingabe. So als hätte man in ein Feld für Zahlen einen Buchstaben eingegeben.

Das Profil zur Prüfung des Upload Form wird also im Gegensatz zu dem eines normalen Formulars um die Methoden von `Data::FormValidator::Constraints::Upload` erweitert:

```
my $form_profile = {
    required => [qw/new_file/],
    optional => [qw/rm submit/],
    constraint_methods => {
        new_file => [
            file_format(mime_types =>
                [qw!image/jpeg
                    image/gif image/png!]),
            file_max_bytes(1024000),
            image_max_dimensions(700,500),
            image_min_dimensions(100,100),
        ],
    }
};

my $results = $self->check_rm(
                'start', $form_profile
                )
    || return $self->check_rm_error_page();
```

Das Formular soll einen Fehler anzeigen, wenn das Bild nicht einen der angegeben MIME-Types hat, oder es größer als 1MB ist, oder seine Abmessungen größer als 700x500px oder kleiner als 100x100px sind. Und das Beste hieran: es funktioniert!

Formular

Da ich, wie eingangs erwähnt, nur mal eben testen wollte, fällt das Formular spartanisch aus:

Der Runmode zur Anzeige des Formulars, hier besonders mnemonisch benannt (start), gestaltet sich gleichermaßen:

```
=head2 start( $errs? )

Display the upload form.

=cut

sub start {
    my $self = shift;
    my $errs = shift; # may be undef

    my $t = $self->load_tmpl(
                        'upload_form.tmpl');
    $t->param($errs) if $errs;
    return $t->output();
} # /start
```

Fehlermeldungen visualisieren

$errs (eine Hashreferenz) ist das, was von ValidateRM an den Fallback-Runmode gegeben wird, wenn ein Fehler auftritt. Sie enthält - wer hätte es gedacht - die Fehlermeldungen. Man könnte diese Fehlermeldungen nun mit einem einfachen `<TMPL_VAR err_fieldname>` anzeigen, jedoch sind die Meldungen bei oben genanntem Fehlerprofil wenig aussagekräftig. Außerdem stellt sich im Falle der Mehrsprachigkeit einer Software das Problem, dass die Meldungen nur in einer bestimmten Sprache vorhanden sind. Zur Lösung des Problems der Mehrsprachigkeit gibt es zwar bereits einen Ansatz, jedoch interessiert der uns hier gerade nicht. Das Vorgehen geht exzellent mit fast jeder anderen Art, die Fehler zu vertextlichen, einher.

Jedes Formularfeld besitzt einen Namen. Tritt ein Fehler auf, wird `$errs` ein Eintrag mit prefix_feldname hinzugefügt. Das Präfix kann arbiträr festgelegt werden. Es empfiehlt sich jedoch, einfach einmalig in der Anwendung Defaults zu definieren (das geschieht normalerweise im `cgiapp_init`) und diese dann durchgehend zu verwenden. So kann man Codestücke / Templates einfacher wieder verwenden.

Die `Data::FormValidator`-Defaults sehen bei mir wie folgt aus:

```
=head2 cgiapp_init()

Open database connection, setup config
files, etc.

=cut

sub cgiapp_init {
    my $self = shift;

    # -- Set some defaults for DFV unless
    # -- they already exist.
    $self->param('dfv_defaults') ||
        $self->param('dfv_defaults', {
            missing_optional_valid => 1,
            filters => 'trim',
            msgs => {
                any_errors => 'some_errors',
                prefix     => 'err_',
                invalid    => 'Invalid',
                missing    => 'Missing',
                format     =>
       '<span class="dfv-errors">%s</span>',
                },
        });

} # /cgiapp_init
```

Der Schlüssel, der in `$errs` enthalten ist, wird vom Formular ausgewertet. Wird das Formular zum ersten Mal aufgerufen, ist der Schlüssel nicht vorhanden und es passiert nichts. Wird mit Fehlermeldungen auf das Formular umgeleitet, sind die Schlüssel vorhanden und werden im Template eingefügt: `$t->param($errs) if $errs;`

Ist ein Schlüssel für ein gegebenes Formularfeld vorhanden, wird dieses mit einer CSS-Klasse versehen und farblich markiert. Dann reicht eine Meldung, dass die markierten Felder überprüft werden müssen.

Man kann dann wahlweise zusätzlich die Fehlermeldungen selbst anzeigen, oder Text im Template in einem `<TMPL_IF err_feldname>` einschließen und nur beim entsprechenden Fehler anzeigen. Es bietet sich manchmal an, Hinweise zur Feldbefüllung immer anzuzeigen - auch wenn kein Fehler auftritt - und darauf zu vertrauen, dass der Nutzer diese Hinweise wenigstens bei Fehlern liest.

Hier das Beispiel aus dem Upload Form zur CSS-Markierung:

```
<input type="file" name="new_file"
       id="new_file"
       <TMPL_IF err_new_file>
         class="error"
       </TMPL_IF>
/>
```

Beispiel für spezielle Fehlermeldungen:

```
<div <TMPL_IF err_new_file>
     class="nxErrorBox"</TMPL_IF>>
    <label for="new_file">new file</label>
    <input type="file"
           name="new_file"
           id="new_file"
          <TMPL_IF err_new_file>
             class="error"</TMPL_IF> />
    <TMPL_IF err_new_file>
        <p>File must be an image with
           max dimensions 700x500 and min
           dimensions 100x100.</p>
    </TMPL_IF>
</div>
```

Ich für meinen Teil werte gerne den Schlüssel aus, der bei jedem Fehler gesetzt wird (some_errors), und umschließe eventuell hilfreiche Fehlermeldungen im Template mit TMPL_IF's.

Beispiel für eine Meldung, die bei jedem Fehler in unmittelbarer Nähe zum Formular angezeigt wird:

```
<!-- TMPL_IF some_errors -->
    <div class="nxErrorBox">
        <img alt="Fehler!"
             src="/gfx/important.png"
             class="errorlogo" />
        <h1>ACHTUNG</h1>
        <p>
            Ein Fehler ist aufgetreten.
Bitte überprüfe die rot markierten Felder.
            <br style="clear: both;" />
        </p>
    </div>
<!-- /TMPL_IF -->
```

`nxErrorBox` als Attribut zum div sorgt dafür, dass der ganze Absatz des Feldes (also inkl. dem Label und der Fehlermeldung) als Fehler markiert wird. Man kann ja bekanntlich viel mit CSS spielen.

Das Schöne an der Verwendung der CSS-basierten Markierung ist, dass Fehlermeldungen seitenweit stets identisch aussehen. Ich habe einmal gelesen, dass es benutzerfreundlich wäre, Anzeigen konsistent zu halten, um die Benutzung der Webseite zu erleichtern.

Sicherheit

Eines der Tutorials war nicht ganz so schlecht wie der Rest, den ich zum Thema Upload-Form gefunden habe. Es behandelte ausführlich den Punkt CGI-Sicherheit. Daran habe ich mich orientiert und solche Dinge wie `$CGI::POST_MAX = 1024 * 5000; # = 5MB` berücksichtigt. Das hätte ich fast vergessen, weil ich sonst immer `CGI::Simple` ohne Upload-Funktionen verwende.

Weiterhin gilt es, böse Zeichen aus den Dateinamen zu filtern. Dazu gehören auch solche Zeichen, die das Betriebssystem des Servers z.B. einfach nicht unterstützt, oder die im Web gerne einmal zu Anzeigeproblemen führen (z.B. das Leerzeichen). Eine Reduktion auf `a-zA-Z0-9_.-` ist da schon ganz hilfreich.

Sowohl beim Hochladen als auch beim Löschen oder Ausliefern von Dateien sollte man solche Tricks, wie die Angabe relativer übergeordneter Ordner, verhindern. Dateinamen wie '../../../etc/hosts' werden weder gerne hochgeladen, noch ausgeliefert.

Da solche Sicherheitschecks an verschiedenen Stellen im Code verwendet werden, wird die Funktionalität natürlich ordentlich als Methode gekapselt:

```
=head2 get_save_filename( $filename )

 Catch characters we don't want. Catch hacky
 filenames.

 =cut

 sub get_save_filename {
     my $self        = shift;
     my $filename    = shift or
         die("Missing filename.");
     my $upload_dir  = $self->cfg(
         'upload_dir'
     ) or die("Missing upload dir param.");

     # -- Check filename for forbidden
     # -- characters and other hacks.
     my ( $name, $path, $extension ) =
             fileparse ( $filename, '\..*' );
     $filename = $name . $extension;

     my $safe_filename_characters =
                              "a-zA-Z0-9_.-";

     $filename =~ tr/ /_/;
     $filename =~
           s/[^$safe_filename_characters]//g;

     return $filename;
 } # /get_save_filename
```

Die Methode `get_save_filename()` wandelt einen gegebenen Dateinamen in einen sicheren Namen um. Dabei werden die ungültigen Zeichen ersetzt und Hacks abgefangen. Die Verwendung dieser Methode zum Upload leuchtet ein. Beim Ausliefern oder Löschen kommt diese Methode aber auch zum Einsatz. Es werden nur Dateien gelöscht und ausgeliefert, deren Dateiname identisch zum Resultat von `get_save_filename()` ist.

Beispiel zur Datei-Auslieferung:

```
=head2 file_download()

Provide a file for download. This simply
sets the content type of the CGI header
to the mime type of the file and returns the
file content without any other
data (e.g. the layout html in
cgiapp_postrun).

=cut

sub file_download {
    my $self = shift;
    my $file = $self->param('filename')
             or return("Missing filename.");
    my $upload_dir = $self->cfg(
        'upload_dir'
    ) or die("Missing upload dir param.");

    return "Will not provide this file:
        [$file]. Go away."
        if $file ne
            $self->get_save_filename($file);

    # [snip]

    binmode( STDOUT );
    return $data;
} # /file_download
```

CGI::Application::Plugin::ConfigAuto

Eher so aus Gewohnheit habe ich eine Konfigurationsdatei angelegt, die das Verzeichnis enthält, in das die Dateien hochgeladen werden sollen. Das ist ganz angenehm, wenn man sehr viel Code in den Klassen ändert und beim Upload nur sicherstellen muss, dass die Konfigurationsdatei auf dem Server nicht überschrieben wird. Außerdem kann man die Konfiguration als Perl-Datenstruktur schreiben, die man mit jedem Perl-Editor direkt auf Syntax-Tippfehler prüfen lassen kann.

Konfigurationsdatei in Perl:

```
#!/usr/bin/perl

use strict;
use warnings;

my %CFG = (
    upload_dir =>
                '/your/path/to/upload/files',
);

\%CFG;
```

CGI::Application::Plugin::MessageStack

Wenn ein Bild erfolgreich hochgeladen wurde, kann man wahlweise eine extra Seite anzeigen, auf der der Benutzer beglückwünscht wird, oder - was mir persönlich besser gefällt, da man mehrere Uploads mit weniger Geklicke durchführen kann - wieder zum Upload-Form zurückkehren und eine Meldung über dem Formular anzeigen, die den Upload bestätigt. Für so etwas bietet sich `CGI::Application::Plugin::MessageStack` an:

```
=head2 form_validate()

Validate user input.
If valid, upload the file, redirect to form,
display success message,
Unless valid, forward to form, show errors.

=cut

sub form_validate {
    my $self = shift;
    my $upload_dir =
        $self->cfg('upload_dir')
        or die("Missing upload dir param.");

    # [BIG SNIP]

    # --Set up a small confirmation message.
    $self->push_message(
        -scope   => 'start',
        -message =>
             'Your file has been uploaded.',
        -classification => 'INFO',
    );

    return $self->redirect(
        $self->query()->url()
    );
} # /form_validate
```

Aufmerksamen Lesern, die meine Ausführungen zur Mehrsprachigkeit von Anwendungen noch im Hinterkopf haben, seien auf das Modul `CGI::Application::Plugin::I18N` verwiesen.

Vermeiden hässlicher Quellcodes

Natürlich ist es stets Ansichtssache, was lesbar ist und was nicht. Wer das en détail wissen möchte, darf gerne in einem Python-Forum fragen, ob lieber Perl oder Python gelernt werden sollte. Aber so mancher Code im Internet ist böse veraltet, fehleranfällig oder einfach nur schlecht konzipiert.

Hier ein Beispiel für potentiell als hässlich empfundenen Code, der aber erstaunlich oft im Internet zu finden ist:

```
open ( FILE, ">$filename" ) or die "$!";
binmode FILE;

while ( <$source> )
{
 print FILE;
}

close FILE;
```

Immerhin werden Fehler beim Öffnen der Datei abgefangen. Kritisch betrachte ich jedoch, dass innerhalb der while-Schleife die Default-Variable `$_` verwendet wird, ohne dass man sie irgendwo sieht. Man kann wissen, dass das die Voreinstellung von Perl ist (dessen Robustheit sei Dank); aber auch dieses Wissen muss man sich erst aneignen. Ein erster Schritt zur Besserung wäre also, `$_` explizit zu benennen.

Und warum wird bei `open()` eine Klammer verwendet, bei `binmode()`, `print()` und `close()` aber nicht? Es mag einen Grund dafür geben, aber jeder Laie der das sieht, kennt ihn wohl nicht. Die konsistente Verwendung von Klammern wäre schöner.

Es ist schon eine ganze Weile so, dass in der Perl-Core-Distribution das Modul `FileHandle` beiliegt. Es ist ein OO-Wrapper um die Funktionen, die Perl zur Dateiverarbeitung bereitstellt. Mein persönlicher Code-Beautifier für die Dateibehandlung stellt sich mittels `FileHandle` so dar:

```
# -- Upload the file.
my $fh = FileHandle->new();
if( $fh->open($picture_uri, '>') ) {

    $fh->binmode();
    while( <$upload_filehandle> ) {
        $fh->print($_);
    }

    $fh->close();

}else{
    die "Error opening [$picture_uri]: $!.";
}
```

So ist ersichtlich, dass `binmode()` eine Funktion auf das FileHandle ist, genauso wie `print()` und `close()` auch. Außerdem wurde die Default-Variable im `print`-Statement explizit aufgeführt.

So kann ein Neuling wenigstens fragen, was `$_` ist. Auch der obige Code enthält noch Spielraum für Verschönerungen. Der geneigte Leser sei ermuntert, auch in dieser Hinsicht selbst tätig zu werden.

Anzeige der Bilder

Zu Beginn wurde festgelegt, dass die Bilder, die hochgeladen wurden, auch mal eben schnell angezeigt, geöffnet und gelöscht werden können sollen. Dazu muss man das Verzeichnis auslesen, welches die Bilder beheimatet. Anstatt da jetzt mit `opendir` oder `DirHandle` herumzuhantieren, wird `File::Find::Rule` zur Satisfaktion der Faulheit (bzw. Schreibeffizienz) verwendet:

```
=head2 show_uploaded_files()

Display a list of all uploaded files. Add
some information (e.g. what kind of
file it is). Add a download button.

=cut

sub show_uploaded_files {
    my $self = shift;
    my $upload_dir = $self->cfg(
        'upload_dir'
    ) or die("Missing upload dir param.");

    # Find all files in the upload dir.
    my @files_on_disk =
        File::Find::Rule->file()
                        ->in( $upload_dir );

    # [SNIP]
} # /show_uploaded_files
```

Das macht gleich viel mehr Spaß - vorausgesetzt, man mag diese Perl-Einzeiler.

Zusammenfassung

Die Anwendung soll mit `CGI::Application` programmiert werden. Die Prüfung und Wiederanzeige bei Fehlern in den Benutzereingaben macht `Data::FormValidator`, Status-

mitteilungen (ist hochgeladen / gelöscht) macht `CGI::Application::Plugin::MessageStack`, `Config::Auto` ist auch mit an Bord und die Anzeige der Dateien im Upload-Ordner wird von `File::Find::Rule` gestützt.

Das muss dann ja nur noch alles zusammengesteckt werden. Die eigentliche Programmlogik reduziert sich auf ein Minimum. Das Betrifft auch die Instanz, die "CatcherInTheRye" instanziiert und ausführt.

Sie verwendet `CGI::Application::Dispatch`:

```
#!/Perl/bin/perl

use strict;
use warnings;
use FindBin qw/$Bin/;
use lib ($Bin);
use CGI::Application::Dispatch;

CGI::Application::Dispatch->dispatch(
    args_to_new => {
        TMPL_PATH => [$Bin . '/templates'],
        PARAMS => {
            cfg_file => $Bin .
               '/config/upload_form.config',
        },
    },
    table       => [
        '' => {
            app => 'CatcherInTheRye',
            rm => 'start',
        },
        ':app/:rm'                => {},
        ':app/:rm/file/:filename'  => {},
    ],
);
```

Die Verzeichnisstruktur des Programms orientiert sich am verwendeten Webserver und dem OS (WinXP) - siehe Listing 1.

So bleibt schlussendlich nur noch der (hoffentlich moderne, nicht hässliche) Quellcode. Er ist in vollständiger Form auf der Website *http://foo-magazin.de* herunterzuladen.

\# Alexander Becker

```
| Verzeichnis                          | Inhalt
| c:/apache/cgi-bin/upload             | Instanz (upload_form.cgi), Klasse (CatcherInTheRye.pm)|
| c:/apache/cgi-bin/upload/config      | Konfigurationsdatei (upload_form.config)              |
| c:/apache/cgi-bin/upload/templates | Templates                                             |
| /your/path/to/upload/files           | Ort für die Dateien                                   |
```

Listing 1

WxPerl Tutorial - Teil 2: Dialoge und Events

Inhalt dieser Folge

Nach dem ersten, eher theoretischen Teil, der Entstehung, Alternativen, Architektur, Installation und API erläuterte, soll dieser Teil wesentlich praktischer werden. Deshalb werden Dialoge und Events Thema, um erste sinnvolle Programme zu erzeugen. Dialoge sind oft die einfachste Möglichkeit für Benutzereingaben und erst wenn Programme auf Ereignisse (englisch „Events“) der Eingabe reagieren, können sie eine Funktion erfüllen. Die bewegliche Anordnung der Bedienelemente mithilfe von Sizern wird Thema der nächsten Folge sein. Für dieses Mal werden die Widgets mit absoluten Koordinaten platziert und eine Änderung der Fenstergröße durch das Setzen einer Konstante verhindert. Für einfache Programme kann das eine effektive Technik sein.

Wo der erste Teil aufhörte

Beginnen wir mit dem Beispiel aus der letzten Folge in Listing 1.

Einzige Änderung ist: anstatt eines Schriftzugs (`Wx::StaticText`) habe ich einen Knopf (`Wx::Button`) eingesetzt, um damit ein erstes Ereignis auszulösen. Da die Parameterpositionen immer dem gleichen Grundmuster folgen, hat auch der Knopf die gleiche Beschriftung, Position und Größe, wie vorher der Schriftzug.

Beim Lauf dieses Beispiel fällt schnell der Nachteil dieser Oberflächengestaltung auf: verengt man die Fenstergröße, bleibt der Knopf unbewegt stehen und ist nur noch teilweise zu sehen. Bei einer Vergrößerung hängt er unbewegt in der linken oberen Ecke des Fensters. Um das zu vermeiden, könnte man folgende Zeile einfügen:

```
$frame->SetWindowStyle(
   $frame->GetWindowStyleFlag()
        ^ &Wx::wxRESIZE_BORDER
);
```

Aussehen und Verhalten eines Widgets wird meistens von dem Parameter mit dem Namen *Style* bestimmt, welcher der Widgetgröße folgt (falls beide vorhanden). Da es hier aber nur um das optische Verhalten geht und die Basisklasse aller sichtbaren Widgets `Wx::Window` ist, tragen die Methoden `WindowStyle` im Namen.

```
use strict;
use warnings;

package ServusWelt;
use Wx;
use base qw(Wx::App);

sub OnInit {
    my $app   = shift;
    my $frame = Wx::Frame->new(
       undef, -1, 'Mein erstes Programm',
       [-1,-1],[250,100]
    );
    my $panel = Wx::Panel->new(
       $frame, -1
    );
    my $knopf = Wx::Button->new(
       $panel, -1,
       'Grreeeting Earthlings!',
       [65,20]
    );

    # Fenster zeichnen
    $frame->Show(1);
    # Fenster als oberstes bestimmen
    $app->SetTopWindow($frame);
    # nicht vergessen
    1;
}
package main;
# Programminstanz erzeugen und starten
ServusWelt->new->MainLoop;
```

Listing 1

Style enthält eine Bitmaske, in der bestimmte Bits bestimmte Merkmale aktivieren. Mit einem bitweisen, exklusiven Oder (*XOR*) lassen sich dort einzelne Bits, die man besser in der Form einer benannten Konstante benutzt, ein und ausschalten. Logisch gleichwertig ist eine bitweise Und-Verknüpfung (*AND*) mit der Negation des gewünschten Bits (einer Bitmaske in der alle Bits gesetzt sind außer dem einen):

```
$frame->SetWindowStyle(
    $frame->GetWindowStyleFlag()
        & ~&Wx::wxRESIZE_BORDER
);
```

Es ließe sich auch wesentlich einfacher schreiben:

```
$frame->ToggleWindowStyle(
    &Wx::wxRESIZE_BORDER
);
```

Diese Variante wird allerdings nicht mit einem älteren Wx < 2.8 funktionieren. Gleichzeitig kam auch die Methode `HasFlag` hinzu, womit einzelne Flags (Bits) geprüft werden können.

Manch einer wunderte sich, warum es im Beispiel `&Wx::wxRESIZE_BORDER` und nicht, wie in der ersten Folge vorgeschlagen, `wxRESIZE_BORDER` hieß. Die erste Schreibweise hat den praktischen Vorteil, das dann Konstanten keine Anmeldung mehr benötigen. Dies ermöglicht auch Code innerhalb eines Projektes ohne Aufwand zu bewegen.

Ein letzter Einwand mag lauten: "Die Konstante kann doch auch gleich bei Erzeugung des *Frame* angegeben werden?". Ja, aber nicht ganz so einfach. Wir machten keine Styleangaben was einem -1 und in diesem Fall einem `wxDEFAULT_FRAME_STYLE` entspricht. Es ginge also auch so:

```
my $frame = Wx::Frame->new(
    undef, -1, 'Mein erstes Programm',
    [-1 , -1] , [250 , 100],
    wxDEFAULT_FRAME_STYLE
        ^ wxRESIZE_BORDER
);
```

Das erste Ereignis

In Wx können manche Widgets über ein Dutzend Ereignisse auslösen. Welche Ereignisse das sind, steht am Anfang der Beschreibung der Widgetklasse in der Dokumentation. Um überlange Parameterlisten zu vermeiden, wird jede Rückruffunktion (englisch "Callback" genannt) mit einem gesonderten Befehl an ein Widget gebunden. Dies kann direkt eine anonyme `sub` sein oder eine beliebige andere Codereferenz, die bei dem Ereignis ausgeführt wird. Das Ereignis selbst ist, wie fast alles in Wx, ein Objekt, hier einer abgeleiteten Klasse von `Wx::Event`. Um einen Callback für das Drücken unseres einzigen Knopfes zu definieren, braucht es die Zeile:

```
Wx::Event::EVT_BUTTON(
    $knopf, $knopf, sub {...}
);
```

Dies ist wieder die "bewegliche" Version, ohne Anmeldung. Mit einem vorangehenden

```
use Wx::Event qw(EVT_BUTTON);
```

oder

```
use Wx::Event qw( :everything );
```

reicht ein

```
EVT_BUTTON( $knopf, $knopf, sub {
    $frame->SetTitle('geändert');
} );
```

Und das Programm beginnt ein Leben zu entwickeln. Dass `SetTitle` den Textinhalt der Titelleiste des Hauptfensters verändert ist selbsterklärend, aber warum besitzt `EVT_BUTTON` die gleiche Schreibweise wie Konstanten, wo es doch deutlich eine Routine ist? Tja, hier emuliert WxPerl die WxWidgets-Schreibweise. In C++ sind Konstanten, Menü und Eventdefinition mit Macros gelöst, die per Konvention so geschrieben werden. Eine andere offene Frage wäre auch: Warum wird das Widget, welches auf das Event reagiert, zweimal angegeben? Der zweite Parameter gibt das Widget an, auf dessen Ereignis reagiert werden soll. Der erste dient einem ganz anderen Zweck. Das lässt sich einfach überprüfen, denn auch mit der Schreibweise:

```
EVT_BUTTON( $frame, $knopf, sub { ... } );
```

funktioniert unser Beispiel noch genauso. Mit

```
Wx::Event::EVT_BUTTON(
    $knopf, $frame, sub { ... }
);
```

gibt es zwar keinen Kompilierungsfehler, aber den Knopf drückt man nun ohne eine Reaktion zu bekommen. Bei längeren Rückruffunktionen klärt sich das auf.

Fang mich

Folgende Routine platziert ein Fenster an einem zufälligen Ort am Bildschirm.

```
sub spring_weg {
   my $frame = shift;
   my $screen = Wx::GetDisplaySize();
   my ($x_size, $y_size) =
       $frame->GetSizeWH();
   my $x_pos = int rand $screen->GetWidth
       - $x_size;
   my $y_pos = int rand $screen->GetHeight
       - $y_size;
   $frame->SetSize(
       $x_pos, $y_pos, $x_size, $y_size
   );
}
```

Sie enthält keine komplizierten Funktionen, lediglich mit `GetSizeWH` eine Sondermethode, die nur WxPerl kennt, was aber in der WxWidgets-Dokumentation ausgewiesen ist. Es erspart den Arbeitschritt, aus dem sonst erhaltenen Size-Objekt die Länge und Breite herauszupulen.

Würde man nun in das Beispielprogramm folgendes einsetzen:

```
Wx::Event::EVT_BUTTON(
    $knopf, $knopf, \&spring_weg
);
```

würde es nicht funktionieren. Aus einfachem Grund: Die meisten Callback-Routinen erhalten 2 Parameter. Das Auslöser-Widget und das Ereignisobjekt. Zeilen wie:

```
my ($widget, $event) = @_;
```

sind darum Standard-WxPerl-Code. Ah, in der Variable `$frame` in *spring_weg* ist die Referenz auf `$knopf` gelandet. Klar dann geht's nicht. Nicht ganz. `EVT_BUTTON` ist eine Ausnahme, die es erlaubt eine Referenz anzugeben, die als erster Parameter übergeben wird. Da auf Knopfdruck meist woanders etwas passiert, ist dies auch sinnvoll. Erst die Zeile

```
Wx::Event::EVT_BUTTON(
    $frame, $knopf, \&spring_weg
);
```

verwandelt unser Beispiel in ein kurzweiliges Kinderspiel (für das 19. Jh.).

Einfache Dialoge

Die angesprochene alphabetische Objektübersicht der WxWidget-Dokumentation ist meist ein guter Ort, um Antworten zu suchen. Außer man braucht einfache Dialoge. Die haben eine Kategorie im Abschnitt "functions", denn es sind einfache Aufrufe. Die allereinfachsten unter ihnen sind die wohlbekannten *Messageboxen*, die oft als leidliche Kästen mit einer Botschaft wie "Wollen sie das wirklich tun?" und mit Knöpfen wie "Ja", "Nein" und "Vielleicht" den Benutzer zu einer Antwort nötigen, da sie sich in den Vordergrund schieben und eine weitere Benutzung des Programms unterbrechen. In Wx heißt das – naheliegend – `Wx::MessageBox` und ist vielfältig einsetzbar. Der erste und einzig notwendige Parameter ist die Botschaft, gefolgt von der Überschrift (Titelleiste) und dem *Style*. Es können auch noch Elternwidget und Koordinaten angegeben werden, was aber kaum benutzt wird, da ohne Angaben die Box zentriert wird. Entscheidend ist der Stil, da er bestimmt, welches Icon im Dialog erscheint und welche Knöpfe er hat. Es gibt nämlich in Wx keine Schachtel für Fehlermeldungen, sondern nur einen Nachrichtenkasten mit `wxICON_ERROR`. Für Warnungen empfiehlt sich `wxICON_EXCLAMATION`, bei Abfragen `wxICON_QUESTION`, bei Bestätigungen `wxICON_INFORMATION`. Als Knöpfe stehen zur Verfügung: `wxYES`, `wxNO`, `wxCANCEL` und `wxOK`, wobei es für `wxYES|wxNO` noch die Schreibweise `wxYES_NO` gibt. `wxICON_INFORMATION|wxOK` ist default. Mit diesen Konstanten kann auch der Rückgabewert verglichen werden, wenn es interessiert, welcher Knopf gedrückt wurde.

```
my $ret = Wx::MessageBox(
    'Wirklich beenden?',
    'Es ist doch grad so schön.',
    wxYES_NO | wxICON_QUESTION
);
app_herunterfahren() if $ret == wxYES;
```

Die Dialogfunktionen sind meist für den Programmierer der einfachste Weg, diverse Arten von Informationen vom Benutzer zu bekommen. Bei Entscheidungen zu einer Frage können Messageboxen verwendet werden, bei Dialogen wie `Wx::GetColourFromUser`, `Wx::GetFontFromUser`, `Wx::GetNumberFromUser`, `Wx::GetPasswordFromUser` und `Wx::GetTextFromUser` sind es die bekannten Standarddialoge, deren Name bereits den Ergebnistyp verrät. Dennoch sollte man immer den Rückgabewert prüfen, denn er könnte auch leer oder -1 sein. Bei Entscheidungen mit mehreren, genau vorgegebenen Antworten reicht eine *MessageBox* nicht mehr. Dann benötigt man:

```
Wx::GetSingleChoice(
    'Frage','Überschrift', [1..5]
);
```

Dem Benutzer stellt sich eine Frage mit 5 Antworten (1..5), die bei Linksklick markiert werden und 2 Knöpfen zum Bestätigen oder Abbrechen. Neben 2 Varianten, die nur den Index oder komplexere Daten zurückgeben können, gibt es auch noch:

```
@a = Wx::GetMultipleChoices(
    '?','!', [1..5]
);
```

Hier sieht man eine Liste von Checkboxen (kleine Quadrate in die man einen Haken setzen kann) und der Dialog liefert eine Liste mit den angewählten Werten. Es ist also von der Art her ein anderer Dialog, der zu Recht einen anderen Namen trägt und der nicht mit `Wx::GetSingleChoice` vereinheitlicht werden kann. Bei einem (`Wx::FileSelector`) ist das etwas ganz Anderes. Hier ist nur das Setzen von `wxFD_MULTIPLE` entscheidend ob nur eine oder mehrere Dateien ausgewählt dürfen. Das **FD** in der Konstante steht für *File Dialog* und ist eine erst kürzlich eingeführte Gruppe von Konstanten. Wenn ältere Wx-Versionen Kompiliermeldungen geben, reicht es meist, das **FD** wegzulassen, also z.B. `wxOPEN` anstatt `wxFD_OPEN`. Da im Gegensatz zum `Wx::DirSelector` die Dateiauswahl sehr beliebt ist und auch für kleine Konverterscripte oft gebraucht wird, möchte ich näher darauf eingehen.

```
$datei = Wx::FileSelector (
    'Datei auswählen',  # Überschrift
    '.',                # Startverzeichnis
    '',             # Text im Dateinamefeld
    '',             # ausgewählter Filter
    "*.*",          # alle Filter
    wxFD_OPEN       # Art des Dialogs
);
```

Besonders der fünfte (vorletzte) Parameter bedarf Erklärung, die nur unzureichend in der Dokumentation erfolgt. Sieht ein sattelfester Programmierer '*.*' und das Wort Dateifilter, ist ihm klar was gespielt wird und das wenn ich den Parameter auf '*.*' oder '*' setzte, alle Dateien angezeigt werden und bei z.B. '*.pl' nur Perlskripte mit der Endung '.pl' in dem Dialog erscheinen. Aber selber mehrere Filter zur Auswahl zu stellen ist nicht ganz trivial. Konfigurationsdateien in Kephra können zum Beispiel vom Typ '.yaml' oder '.conf' sein. Ein Filter dafür würde lauten:

```
'Config Dateien (*.conf;*.yaml)|' .
    '*.conf;*.yaml|'.
'Alle Dateien (*.*)|*.*"
```

wobei der erste der aktuell vom Dialog verwendete wäre. Senkrechte Striche trennen nicht nur die Filtergruppen, sondern den ersten und zweiten Teil des Filters. Der erste Teil ist der sichtbare Text im Dialog, ein Label. Die Definition des Filters geschieht im zweiten Teil durch eine Gruppe semikolongetrennter Glob-Filter.

Dialogobjekte

Wem das zu einfach war, der kann es gerne auch komplizierter haben. Die bisherigen Dialoge werden sofort erstellt, zur Anwendung gebracht und nach egal welchem Knopfdruck abgebaut. Das ist für einfache Abfragen ideal, für höhere Anforderungen sollte man die Dialoge aus der alphabetischen Objektliste nehmen. Damit sind nicht jene Klassen gemeint, deren Name auf `'PickerCtrl'` endet. Das sind meist Knöpfe, die bei Betätigung einen Dialog öffnen, der sich nach Auswahl wieder schließt und über den der Programmierer keine Kontrolle hat, da die gewonnenen Daten über die `'PickerCtrl'` abzufragen sind. Zur Verdeutlichung des Umgangs mit Dialogobjekten, hier inhaltlich das letzte Beispiel nochmal:

```
my $dialog = Wx::FileDialog->new(
    $frame, 'Datei auswählen', '.', '',
     "*.*", wxFD_OPEN | wxFD_MULTIPLE
);
if ($dialog->ShowModal != wxID_CANCEL) {
    my @dateien = $dialog->GetPaths;
    ...
}
```

`ShowModal` ist der Befehl, der den bereits erstellten Dialog sichtbar werden lässt. Mit `Show` wird jedes Widget oder Fenster sichtbar oder unsichtbar. Hier heißt es jedoch `ShowModal`, da der Dialog alle Aufmerksamkeit auf sich zieht, in dem das Elternwidget eingefroren wird. Deswegen verlangten selbst die *Messageboxen* das Elternwidget als Parameter. Meist funktioniert es auch wenn der Parameter ausgelassen wird, aber besonders unter *MacOS* empfiehlt es sich den `$parent` trotzdem anzugeben, da *MacOS* im Zweifel gerne auch alle Programme einfriert um auf dieses Fenster zu warten.

Dialogobjekte haben den Vorteil, dass `ShowModal` beliebig oft aufgerufen werden kann. Dieses Prinzip ist überall gleich (`Wx::ColourDialog`, `Wx::FontDialog` ...), deshalb auf zum Meistergrad der Dialogprogrammierung.

Eigene Dialoge

Um eigene, modale Dialoge zu bauen, gibt es die Klasse `Wx::Dialog`. Wie `Wx::Frame` ist sie von `Wx::TopLevelWindow` abgeleitet und besitzt daher die Methoden für Titelleiste, einfache Zentrierung, Vollbild, Icon, Transparenz, geschwungene Fensterformen usw.. Es gibt aber auch wesentliche Unterschiede zu einem *Frame*, der um einiges mächtiger ist und dessen Fähigkeiten wohl in der übernächsten Folge vorgestellt werden. Ein wichtiger Unterschied ist: Dialoge haben eine fixe Größe, was alle Bemühungen des zweiten Kapitels überflüssig machen würde. Dialoge haben auch bereits das *Panel* inklusive. Würden wir das anfängliche Beispiel als Dialog konstruieren, bräuchten wir diese Zeile nicht:

```
my $panel = Wx::Panel->new( $frame, -1);
```

Allerdings würden unvermutet Probleme auftauchen, denn ein weiterer Unterschied liegt darin, dass Dialoge keinen Einfluß die Applikation `Wx::App` haben. Kaum jemand möchte mit einem Dialog auch gleich das ganze Programm beenden. Würden wir also unser Haupfenster, das ein Dialog ist, schließen, würde das Programm unsichtbar weiterlaufen. Nur folgender kleiner Hack würde die Situation retten:

```
EVT_CLOSE ($frame, sub {
    $app->ExitMainLoop
});
```

`EVT_CLOSE` ist das Ereignis, das beim Drücken des kleinen roten Knopfes in der rechten, oberen Ecke eintritt. In diesem Fall wird der Hauptprozess, der Anfangs mit `ServusWelt->new->MainLoop;` gestartet wurde, beendet. Normal reicht ein `$frame->Close()`, um das Hauptprogramm zu beenden, da der *Frame* die *App* gleich mitbeendet.

Im Detail entspricht dann der Bau des eigenen Dialoges einer beliebigen Oberfläche, was Stoff der nächsten Folge ist. Er wird auch benutzt wie andere Dialoge auch, da er die Methoden wie `ShowModal` - wie andere Dialoge auch - von `Wx::Dialog` erbt. Nur Zugriffsmethoden auf die Daten müssen selbst geschrieben werden und dazu kann ein OOP-Rahmenwerk der Wahl benutzt werden. Bei kleinen Lösungen kann man auch die Referenzen der interessanten Widget wie eine Texteingabefenster, direkt im Objekthash des Dialoges speichern,

```
my $textfeld = Wx::TextCtrl->new( ...
$dialog->{text} = $textfeld;
```

was später einen einfachen Zugriff erlaubt. Etwa:

```
$dialog->{text}->GetValue();
```

Nur ein Problem wäre da vielleicht noch. Wenn man standardmäßig Dialoge mit

```
if ( $dialog->ShowModal != wxID_CANCEL ) {
    ...
}
```

aufruft, so sollte das auch mit dem eignen Dialog gehen. Dieser hat sicher eigene Buttons mit selbstgewählten Schriftzügen. Aber mit folgenden Befehlen:

```
$dialog->SetAffirmativeId(
    $knopf_ja->GetId
);

$dialog->SetEscapeId(
    $knopf_nein->GetId
);
```

Wird beim Drücken eines von diesen beiden Knöpfen der Dialog geschlossen und als Rückgabewert `wxID_OK` bzw. `wxID_CANCEL` gesendet. Alle weiteren Details in der nächsten Ausgabe.

Herbert Breunung

INTERVIEW

Richard Dice und Karen Pauley "State of TPF"

Renée Bäcker (RB): Hallo Karen, hallo Richard! Danke, dass Ihr Euch Zeit für unser drittes Interview über den Stand bei der Perl Foundation (TPF) genommen habt. 2009 sind etliche Dinge in der Perl-Welt passiert. Über diese Dinge möchte ich gerne sprechen und darüber, was in 2010 zu erwarten ist. Unsere Leser kennen Richard, Karen würdest Du Dich vorstellen?

Karen Pauley (KP): Hi Renée. Ich werde versuchen, mich vorzustellen.

Ich bin seit 2000 in der Perl Community dabei, auch wenn ich nie als Perl-Programmiererin gearbeitet habe: Bis 1999 war ich eine C- und SQL-Entwicklerin. Dann änderte sich meine Karriere und ich begann ein Team von Perl-Programmierern zu leiten. Danach hatte ich ein paar kleine Technologie-Firmen.

Im März 2008 machte die TPF einen offenen Aufruf für Freiwillige für den Posten des "Steering Committee" Vorsitzenden. 16 Monate zuvor bin ich nach Japan gezogen und hatte die Zeit reduziert, in der ich Unternehmen führte. Das bedeutete, dass ich mehr Zeit einem Community Projekt widmen konnte. Ich fühlte, dass die TPF-Position - die das Managen von einer Vielzahl verschiedener Projekte beinhaltete - genau das richtige war. Ich bewarb mich und bekam den Posten. Ich habe immer noch den Posten der "Steering Committee" Vorsitzenden, weil ich mir vorstellen kann, dass meine Rolle als Vize-Präsidentin nur vorübergehend ist.

RB: Richard, Karen ist die neue Vize-Präsidentin der TPF seit Dein TPF-Grant angenommen wurde und Jim Brandt - der frühere Vize-Präsident - den Posten des Präsidenten angenommen hat. Kannst Du etwas über diesen Grant und was Du bisher getan hast erzählen? Wie können unsere Leser Dir dabei helfen, Deine Ziele zu erreichen?

Richard Dice (RD): Hi Renée. Ich bin froh, dass ich wieder die Gelegenheit habe, ein paar Fragen zu beantworten.

Deine Frage nach dem Grant überspringt einiges der dahinterliegenden Geschichte. Also werde ich diese jetzt beschreiben. Die Spende von Ian Hague (*http://www.perlfoundation.org/ian_hague_perl_6_development_grants*), die die TPF im Mai 2008 bekommen hat, war dafür gedacht, zwei große Dinge zu unterstützen: Direkte Unterstützung der Aufwände für Perl 6 und Ressourcen für die TPF zu bieten, um sich selbst zu verbessern. Bezüglich des zweiten Punkte habe ich Anfang 2009 ein Bewerbung an die TPF geschickt, um meine Zeit zu unterstützen, in der ich an Verbesserungen der TPF arbeite. Damit begann ein längerer Prozess, der sowohl interne Diskussionen des "TPF Board of Directors" als auch eine Komponente für Feedback aus der Community beinhaltete. Dieser Prozess wurde im Juli 2009 durch das "Board" mit dem folgenden Ergebnis beendet: Eine Aufgabenliste mit zu erbringenden Leistungen, die die TPF als Verbesserungen für die Organisation anerkannte, wurde erstellt - ich würde an dieser Liste auf der Grant-Basis arbeiten und ich würde aus dem "Board" ausscheiden und die Position des Präsidenten für die Dauer des Grants aufgeben.

Die Arbeitspakete sind fast ausschließlich interne TPF-Punkte, die nahezu keinen von außen sichtbaren Effekt haben, wenn sie abgeschlossen sind. Dennoch werden sie die TPF zu einer fähigeren Organisation machen und so wird das, was die Perl Community hauptsächlich merken wird, das folgende sein: Die TPF ist einer fähigere, aktivere Organisation, die Perl und die Community besser unterstützen kann. Die Punkte sind: Verbesserungen in den Beziehungen zu unseren Spendern und Verbesserungen im Spenden-System, Verbesserungen bei den internen Haushaltsverfahren und Haushaltsprojekten, erstellen eines "verbundenen" Spenden-Modells, bessere Organisation der TPF-internen

Dokumente, erstellen einer Pressemappe und erstellen eines Prozesses für Pressemitteilungen. Was bei der Betrachtung dieser Liste offensichtlich ist, ist, dass diese Dinge ziemlich un-sexy sind. Sie sind langweilig, mühsam, zeitaufwändige Verwaltungsaufgaben, die die meisten Programmierer höchst unangenehm finden. Und sie sind schwierig zu erreichen in freiwilligen Organisationen wie der TPF. Grundsätzlich ist diese Arbeit, die Sorte von Dreck, die sich in den Ecken von Organisationen über die Zeit ansammelt wenn keiner danach schaut. Und wenn es bemerkt wird, ist es recht schwierig irgendwas dagegen zu tun.

Bezüglich der Hilfe von den $foo-Lesern -- Ich schätze Dein Angebot, aber die meisten Sachen können nicht von außerhalb der TPF gemacht werden. Es ist eine Liste von interner Arbeit. Es gibt aber zwei Ausnahmen, eine kleine und eine etwas größere. Die erste Ausnahme ist die Strategie für Pressemitteilungen. Ich werde eine öffentliche Anfrage an die Mitglieder überall in der Perl Community machen müssen, um sie zu fragen, welche Medien in der IT Industrie in ihrer Umgebung (Stadt, Land, etc.) existieren, so dass sie kontaktiert werden können, wann immer eine Pressemitteilung von Bedeutung ansteht. Wir werden auch einen ständigen Pool von freiwilligen Übersetzern in die lokalen Sprachen für Pressemitteilungen benötigen.

Der größere Stück der Hilfe ist das verbundene Spendenmodell. Das heißt, wir glauben, dass es ein hohes Maß an lokaler Information über Perl-verwendende Unternehmen gibt, die TPF selbst nicht erreichen kann. Sei es wegen Sprachbarrieren oder (wahrscheinlicher) aus Personalmangel innerhalb der TPF. Wir müssen Prozesse schaffen, die auf "local champions" aufbauen, die die Reichweite der TPF und Aktivitäten zur Spenden-Sammlung in Städten oder Ländern überall auf der Welt organisieren. In den letzten Jahren meiner Beteiligung in der TPF sind einige Male Leute auf mich zugekommen und haben gefragt, wie sie TPF bei der Spendensammlung helfen können. Und jedes Mal fehlten mir die grundlegenden Materialien, die ich gebraucht hätte, um sie in die richtige Richtung zu bringen. Dieser Punkt des Grants geht genau das an. Der erste Teil wird sein, mit Community-Mitgliedern überall auf der Welt zu sprechen, um zu sehen, wer Interesse daran hat, ein "local champion" zu sein und um Input und Ideen von ihnen zu bekommen, wie sie helfen können. Sobald dieses Feedback in einem Reihe von Prozessen und Regeln verankert ist und das unterstützende Material für sie da ist, dann werden wir die Hilfe der Community-Mitglieder brauchen, um das alles in die Praxis umzusetzen. Hilfe der $foo-Leser könnte bei diesem Punkt des Grants definitiv hilfreich sein.

Der Status der Arbeit bis jetzt ist immer noch in den Anfängen. Ich litt unter meinen üblichen Nach-Konferenz-Saison Ablenkungen bis in den späten September. Die Konferenz-Saison war für mich in diesem Jahr ungewöhnlich lang, weil es auch eine Woche in Tokio für die YAPC::Asia in diesem Jahr beinhaltete. Seitdem ging die Zeit für einige andere Projekte drauf, die die sofortige Aufmerksamkeit verlangen. Ich hoffe, dass ich bald mit der Arbeit am TPF Grant anfangen kann, so dass große Teile davon ungefähr zu der gleichen Zeit Online gehen können wie die Rakudo Star Veröffentlichung.

RB: Karen, wie wichtig ist dieser Grant für die TPF selbst und für Perl?

KP: Der Grant ist für die TPF wichtig, weil das Ziel die Verbesserung der Prozesse und internen Abläufe der Organisation sind. Das sollte uns effizienter machen. Es ist auch wichtig, weil es das erste Mal war, dass wir einen Grant an jemanden vergeben haben, um TPF zu verbessern. Der Grant berührt nicht direkt Perl, aber alle Verbesserungen der TPF wird indirekt einen Nutzen für Perl haben.

RB: Jetzt lasst uns nach vorne schauen. Was wird 2010 Perl bringen? Patrick Michaud hat auf der YAPC::EU 2009 in Lissabon angekündigt, dass "Rakudo Star" (Perl 6 auf Parrot) im Frühjahr 2010 veröffentlicht wird. Wie verändert das die Priorisierung der TPF in Bezug auf Perl 5 vs. Perl 6?

KP: Ich bin mir nicht wirklich sicher, ob das die Prioritäten ändern, auch wenn es sicherlich einige Dinge beeinflussen wird. Ich hoffe, mit Patrick Michaud und Jonathan Worthington daran arbeiten zu können, eine Spendenaktion für Rakudo * zu starten, weil ich sehr gerne einen Rakudo * Hackathon mit allen wichtigen Akteuren sehen würde. Zur gleichen Zeit jedoch, glaube ich, wird ein anderes TPF Vorstandsmitglied an einem Projekt arbeiten, um Perl 5 in irgendeiner Art und Weise zu unterstützen. Die Diskussion hat aber gerade erst begonnen, so dass ich nicht wirklich weitere Details bieten kann. Aber es zeigt, dass wir weiterhin an neuen Projekten arbeiten - für Perl 5 und Perl 6.

RB: Apropros YAPC::EU: Es gab ein Programm, das "send-a-newbie" hieß, das drei Personen zur YAPC::EU 2009 gebracht

hat, denen es sonst nicht möglich gewesen wäre, an der Konferenz teilzunehmen. Das Programm wurde von vielen Privatleuten aus der Perl-Community unterstützt. Wäre es möglich, so ein Programm auch für andere Konferenzen zu haben?

KP: Ich denke, es war ein großartiges Programm und ich glaube, dass es möglich wäre, es auf andere Konferenzen auszuweiten. Ich habe mit Edmund von der Burg gesprochen, der das erste "send-a-newbie" Programm organisiert hat, und er wäre froh, wenn das Programm ausgeweitet und für andere Konferenzen verwendet würde. Nächstes Jahr würde ich es gerne für die YAPC::NA machen.

RB: Es hat sich etwas bei den Releases von Perl geändert. Nachdem Perl 5.10.1 erschienen war - für das Dave Mitchell einen TPF-Grant bekommen hat - begann Jesse Vincent, monatliche Releases von Perl 5.11 (einem Entwickler-Zweig) zu machen. Was denkst Du über diese regelmäßigen Releases?

KP: Wenn es zu dieser Art von Änderungen oder Entscheidungen kommt, vertraue ich der Community und Leuten wie Jesse Vincent, dass sie das Richtige tun. I denke hauptsächlich darüber nach, wie TPF Jesse bei seinen Zielen helfen kann und nicht darüber ob richtige technische Entscheidungen getroffen wurden oder nicht.

RD: Eine Sache, die ich an den monatlichen 5.11-Zweig-Releases mag, ist dass es ein Experiment ist, dass zum ersten Mal im Kontext des Perl-Core gemacht wird. Es gab verschiedene Meinungen darüber, ob die monatlichen Releases des Perl-Cores funktionieren, die im vergangen Jahr laut debattiert wurden. Aber wir werden den wirklichen Nutzen oder die wirklichen Nachteile dieses Ansatzes erst durch Erfahrungen im wirklichen Leben erfahren. Sicherlich ist es am besten, dies am 5.11-Zweig zu versuchen und nicht bei 5.10.

Es gibt ein Element der neuen Strategie der monatlichen Releases, das nahezu unabhängig der tatsächlichen monatlichen Releases ist: Die Technik parat zu haben, ein "Knopfdruck"-Release zu erstellen. Das heißt, Du rufst ein einfaches Skript auf, das absolut minimalen manuellen Eingriff erforderlich macht und ein komplettes Core-Release wird erstellt. I weiß, dass das Erstellen dieser Technik das ist, mit dem Jesse viel Zeit verbracht hat, als er den Prozess begonnen hat. Diese Technik ist hilfreich, egal wie häufig Du einen Release-Plan ausführst, besonders in der Art von verteilter Versionierung und Entwicklungsumgebungen wie es das github-Hosting bietet.

RB: Wir haben jetzt drei größere Organisationen in der Perl-Welt: Die Enlightened Perl Organisation (EPO), TPF und die Japan Perl Association (JPA). Arbeiten sie zusammen oder verfolgen sie unterschiedliche Ziele?

KP: Ich denke, dass sie hauptsächlich unterschiedliche Ziele haben. Zum Beispiel ist es eines der Ziele der JPA, qualitatives Perl-Training in Japanischer Sprache anzubieten. Es gibt einige Dinge bei der JPA, die ähnlich denen der TPF sind, so wie JPA jetzt die YAPC::Asia durchführt und TPF in der Durchführung der YAPC::NA involviert ist. Aber in Wirklichkeit sind beide Organisationen sehr unterschiedlich, auch in ihren Strukturen.

Natürlich sind andere Organisationen in der Werbung von Perl eingebunden, so wie die Associação Portuguesa de Programadores Perl und die YAPC Europe Foundation (YEF). Einzelne Perlmonger-Gruppen waren auch in diesem Bereich aktiv. Die Birmingham Perlmongers haben die Birmingham Perl Mongers Limited gegründet, die in diesem Jahr den QA Hackathon in Birmingham unterstützt haben und auch CPAN Testes sponsorn. Ich weiß nicht, ob Vienna.pm eine Organisation gegründet haben, aber sie haben auch einige Perl-Projekte unterstützt.

Es gibt also viele Organisationen, aber ich glaube, dass locker Raum für alle ist.

RB: Was denkst Du über den "Iron Man Blogging Contest" der EPO?

KP: Ich denke, dass es fantastisch ist. Ich habe es sogar einen Abschnitt in meiner Keynote bei der OSDC Australia im November gewidmet. Es ist ein praktisches und effektives Programm und sie haben sichergestellt, dass es unglaublich einfach für jeden ist, daran teilzunehmen. Ich selbst habe mich vor einiger Zeit angeschlossen, obwohl es mir peinlich ist zuzugeben, dass ich ständig auf den "Paper Man" Status zurückfalle. Aber ich glaube, dass ich jetzt mehr über Perl blogge, als ich es eine zeitlang getan habe. Es hat auch andere dazu ermutigt, mehr zu schreiben, und einige der Blogs sind ausgezeichnet. Yuval Kogmans Blog fällt mir dazu insbesondere ein.

RB: Ende 2009 wurde ein neues Komitee in der TPF gegründet - das Marketing Komitee. Kannst Du uns etwas über deren Ziele erzählen? Können unsere Leser helfen? Welche Aktivitäten sind geplant?

KP: In der Vergangenheit hatte TPF eine Person, die für PR und Marketing zuständig war. Wir haben realisiert, dass es für einen Freiwilligen nicht wirklich möglich war, so eine große Rolle zu spielen. Also haben wir das Marketing Komitee gegründet, das es uns erlaubt, viel mehr Leute einzubinden, mit dem Plan, die einzelnen Aufgaben und Zuständigkeiten zwischen den Komitee-Mitgliedern aufzuteilen. Hoffentlich wird es das verbessern, was wir zur Zeit haben.

Wir haben es geschafft, einige dieser Rollen zu besetzen. Zum Beispiel wurde Dave Cross kürzlich für die Social-Networking-Rolle ernannt. Aber nicht alle Stellen wurden besetzt und wir werden sicherlich nach mehr Freiwilligen suchen. Wir möchten Freiwillige haben, die an Blogs, dem Design der Webseiten, Marktforschung und Pressemitteilungen arbeiten.

Als weiteren Teil der Marketing-Bewegung haben wir eine TPF Marketing Mailingliste gestartet, die es uns erlaubt direkter mit der Perl-Community zu arbeiten. Eine Idee, die daraus geboren wurde, war, dass wir Perl auf nicht-Perl-Konferenzen bewerben sollten. Gerade hat Gabor Szabo Stände auf der FOSDEM und der CeBIT Open Source 2010 organisiert. Er sucht nach Freiwilligen, die bei diesen Veranstaltungen helfen und wir hoffen dass diese beiden Konferenzen nur die ersten von vielen nicht-Perl-Konferenzen sind, auf die wir abzielen können.

RB: Eine letzte Frage an euch beide: Was erwartest Du für 2010? Was könnte passieren und was sollte der Perl und/oder der TPF-Welt passieren?

KP: Eine großartige Sache an der Perl-Community ist, dass unsere Mitglieder wundervolle Dinge oft unerwartet machen. Eine dieser Überraschungen 2009 war Plack. Für 2010 erwarte ich, dass die Nutzung von Plack wachsen wird und mehr System PSGI unterstützen. Ich habe immer noch eine Vorliebe für Perl-Webapplikationen und ich glaube, Plack und PSGI sind wichtige Teile einer willkommenen Belebung.

Natürlich, wie ich bereits gesagt habe, freue ich mich sehr auf die Veröffentlichung von Rakudo * im Frühjahr. Ich glaube, dass das offizielle Release mehr Leute ermutigen wird, Perl 6 auszuprobieren und sie geben uns dann interessantes Feedback über die Sprache und die Implementierung geben. Aber mehr als das, hoffe ich, dass das Release die Kern-Rakudo-Entwickler ermutigt, die viel ihrer Zeit und ihres Talents in dieses Projekt gesteckt haben. Und diesbezüglich wird die TPF versuchen Wege zu finden, unsere Unterstützung für die Rakudo-Entwickler in 2010 zu verbessern.

RD: Ich denke nicht, dass es 2010 Überraschungen geben wird, nur die stete Entwicklung einiger exzellenter Projekte, die es schon seit vielen Jahren in der Perl-Community gibt.

Worauf ich am meisten gespannt bin, ist die Veröffentlichung von Rakudo Star, das (nach bester Schätzung) im April 2010 passieren wird. Das kommt schnell, aber auch einiges der unterstützenden Infrastruktur der Perl 6 Umgebung passiert jetzt schnell, so wie einige Perl 6 Kernmodule, eine Versuche ein CPAN-ähnliches System für Perl 6 aufzubauen und die ganz wichtige Perl 6 Spezifikation-Tests.

Ich freue mich auf die Entwicklung von Strawberry Perl und schließlich Chocolate Perl.

Ich glaube, es gibt großartige Arbeiten, die in der CPAN-Testers-Welt passieren. Einschließlich großartiger Arbeit vieler Leute an den Perl-Werkzeugen.

Und wie du vorhin gefragt hast, bin ich sehr daran interessiert zu sehen, wie die Welt von Perl 5 Core auf die Fortschritte im 5.11-Zweig reagiert und wie viel davon zurück in 5.10 fließt.

RB: Vielen Dank und viel Glück.

Neues von TPF

Mailingliste für alle, die Perl promoten wollen

Das Marketing Komitee der Perl Foundation möchte auch auf Veranstaltungen präsent sein, die nicht nur auf Perl ausgerichtet sind, wie zum Beispiel die FOSDEM.

Um diese Anstrengungen zu koordinieren und auch Außenstehenden die Möglichkeit zu bieten, sich zu informieren und zu helfen, wurde eine neue Mailingliste eingerichtet: events@lists.perlfoundation.org.

Die Liste hat auch ein öffentliches Archiv.

Jeder ist eingeladen, mitzuhelfen. Als erstes steht z.B. der Stand auf der FOSDEM 2010 an.

Pod-Tools-Grant

Ricardo Signes hat nach einer längeren Unterbrechung - unter anderem wegen einige Konferenzen - an seinem TPF-Grant weitergearbeitet.

Er hat einige Tests für Pod::Elemental geschrieben, das jetzt ausreichend spezifiziert, dokumentiert und getestet ist. PodPurler nutzt jetzt diese Modul, was ein guter Schritt in Richtung Dist::Zilla ist.

Mit den Pod Tools Pod::Elemental und Pod::Weaver schafft Ricardo Signes eine Modulsammlung, mit der man Pod-Dokumente sehr einfach in andere Formate transformieren kann.

Mittlerweile hat Signes den Grant abgeschlossen.

Grant: Tcl::Tk für Parrot/Rakudo

Vadim Konovalov hat nach längerer Auszeit an seinem Grant weitergearbeitet, der die Verwendung von Tcl/Tk auf Parrot/Rakudo ermöglichen soll.

Er hat die Funktion 'call' implementiert, mit dem Aufrufe auf Tk-Widget-Methoden ausgeführt werden können.

learn.perl.org-Grant:

Eric Wilhelm hat seinen Grant "improving learn.perl.org" abgeschlossen. Während des Grants hat er sechs Tutorials geschrieben:

- Einstieg in Perl
- Kontrolle, Strukturen, Gültigkeitsbereiche und Subroutinen
- Verwenden und Erstellen von Modulen
- Eine Einführung in Objekte
- Perl/CPAN Konfiguration Howto
- Deine erste GUI Anwendung

Ursprünglich sollte Eric auch das Layout von http://learn.perl.org ändern, aber in der Zwischenzeit wurden die ganzen *.perl.org-Seiten überarbeitet.

Auf der Seite http://learnperl.scratchcomputing.com sind diese Tutorials zu finden.

Perl-Umfrage-Grant

Die Inhalte hat Kieren Diment fertig gestellt. Jetzt wartet die Umfrage nur auf die letzten Tests und die Freischaltung.

Google Summer of Code 2009 Zusammenfassung

Jonathan Letho, in diesem Jahr für die Vertretung der Perl Foundation beim Google Summer of Code (GSoC), hat eine Zusammenfassung der Perl-Projekte gepostet:

Die Perl-Projekte wurden sehr erfolgreich abgeschlossen, nur Kevin Tew konnte aus verschiedenen Gründen nicht an seinem Projekt arbeiten. Er bleibt aber weiterhin Parrot-Entwickler und hoffentlich wird er an dem Projekt zu gegebener Zeit weiterarbeiten. Viele Projekte waren Parrot und Perl 6 bezogen, aber auch für Catalyst, Mojo und WebKit gab es Projekte. Insgesamt waren es neun Projekte, die für die Perl Foundation durchgeführt wurden.

Mehr Infos gibt es unter http://google-opensource.blogspot.com/2009/10/perls-of-wisdom-perl-foundation-parrots.html

Wer beim nächsten GSoC für die Perl Foundation mithelfen möchte, kann sich jetzt schon im IRC-Channel #soc-help auf irc.perl.org oder auf der Mailingliste melden.

Marketing Komitee

Die Perl Foundation hat ein Marketing Komitee eingerichtet. Die Mitglieder des Komitees sollen sich um folgende Sachen kümmern:

1. Blog der TPF
2. Design der Webseite
3. Marktforschung
4. Pressemitteilungen
5. News-Artikel
6. Social networking

Vorsitzender ist Dan Magnuszewski.

"Merkwürdigkeiten" - return

Nicht selten soll vorzeitig aus einer Subroutine herausgesprungen werden, wenn "fehlerhafte" Daten vorliegen. In unserem Beispiel gehen wir davon aus, dass wir eine Subroutine haben, die einen Integer-Wert erwartet. Ist der übergebene Wert keine Ganzzahl, soll die Subroutine vorzeitig beendet werden. Damit man schon am Rückgabewert erkennt, dass ein Fehler vorliegt, soll es `undef` sein. Wie die Subroutine aussieht, ist jetzt doch ziemlich klar, oder?

```
#!/usr/bin/perl

use strict;
use warnings;

my $retval = test(1.1);
print $retval ? $retval : '<undef>';

sub test {
    my ($value) = @_;
    return undef unless int( $value )
                        eq $value;
    return $value;
}
```

Da ich in dem Beispiel eine Fließkommazahl übergebe, ist das Ergebnis eindeutig: Es wird "<undef>" ausgegeben. Schaut jemand mit mehr Perl-Erfahrung auf diesen Code, bekommt man schnell gesagt, dass das `return undef unless ...` nicht optimal ist und man lieber `return unless ...` schreiben sollte. Warum?

Hier kommt wieder eine Eigenheit von Perl ins Spiel: Perl ist kontextsensitiv. D.h. Subroutinen können sich anders verhalten, je nach dem, wie ich die Subroutine aufrufe. In dem obigen Beispiel speichere ich das Ergebnis in einem Skalar, die Subroutine wird also in skalarem Kontext aufgerufen.

Ändere ich das Beispiel so ab, dass der Aufruf wie folgt aussieht

```
my @retval = test(1.1);
print @retval ? 'wahr' : '<undef>';
```

dann speichere ich das Ergebnis in einem Array. Die Subroutine wird jetzt im sogenannten Listenkontext aufgerufen. Und in diesem Listenkontext verhält sich das `return undef` nicht so, wie es viele Leute erwarten. Wenn ich den geänderten Code laufen lasse, erhalte ich als Ausgabe "wahr". Das ist aber kein Integer, den ich der Subroutine da übergeben habe.

Jetzt sollte man sich anschauen, was Perl da im Listenkontext zurückgibt.

```
use Data::Dumper;
my @retval = test(1.1);
print Dumper \@retval;
```

Als Ausgabe bekommt man dann

```
$VAR1 = [
          undef
        ];
```

Perl hat jetzt also ein Array mit einem einzigen Element zurückgeliefert, das `undef` ist. Viele erwarten jedoch eine leere Liste, was man an den typischen `if( @retval ){ # mach was }` erkennt.

Schreibt man `return unless ...`, macht Perl das richtige: Im skalaren Kontext liefert es `undef` und im Listenkontext eine leere Liste.

```
my @retval = test(1.1);
print @retval ? 'wahr' : '<undef>';

my $test = test(1.1);
print defined $test ? 'wahr' : '<undef>';

sub test {
    my ($value) = @_;
    return unless int( $value ) eq $value;
    return $value;
}
```

Jetzt wird "<undef><undef>" ausgegeben. Das erwünschte Ergebnis.

Jetzt gibt es aber eine Falle: Wenn ich folgenden Code habe, wie sieht dann der Hash aus?

```
my %hash = (
    1 => test(1.1),
    2 => 3,
);
```

viele werden hier falsch raten und erwarten, dass zum Schlüssel "1" der Wert `undef` gespeichert wird, dass der Code also im Prinzip dem hier entspricht:

```
my %hash = (
    1 => undef,
    2 => 3,
);
```

Das ist aber nicht der Fall. Vielleicht wird es deutlicher was hier passiert, wenn man das etwas anders hinschreibt:

```
my %hash = ( 1, test(1.1), 2, 3 );
```

Und das ganze ist im Listenkontext also entspricht das einem

```
my %hash = ( 1, (), 2, 3 );
```

Was als Ergnis folgenden Hash hat:

```
my %hash = ( 1 => 2, 3 => undef );
```

Das letzte `undef` wird eingefügt, weil das eigentliche Ergebnis eine ungerade Anzahl von Elementen hat (1,2,3), ein Hash aber immer eine gerade Anzahl haben muss.

Als Lösung muss man dafür sorgen, dass der Funktionsaufruf in skalarem Kontext stattfindet:

```
# Loesung 1
my $result = test(1.1);
my %hash   = ( 1 => $result, 2 => 3 );

# Loesung 2
my %hash   = ( 1 => scalar( test(1.1) ),
               2 => 3 );
```

Renée Bäcker

Leserbriefe

Perl-Scopes Tutorial

Hallo,
ich beziehe mich auf den Artikel "Perl Scopes Tutorial - Teil 2" in der Ausgabe 4/2009. Am Schluss beschreibt der Autor folgende 2 Funktionen:

```
{
  my $x;
  sub inc_x {
    $x++;
  }
}
```

oder in 5.10 besser:

```
sub inc_x {
  state $x++;
}
```

Es gibt eine weitere Möglichkeit, eine entsprechend gekapselte Variable zu erzeugen, nämlich:

```
sub inc_x {
  my $x if 0;
  $x++;
}
```

Ich bin weit davon entfernt, diesen Code zu empfehlen. Doch wenn man fremde Module liest, findet man diese Konstruktion manchmal. Zum Verständnis sollte man sie daher kennen.

Perl selbst mag sie auch nicht richtig. Sie ist "deprecated" und Perl erklärt auch warum, aber sie funktioniert auch mit 5.10 - siehe Listing 1.

Viele Grüße,
Torsten Förtsch

```
$ perl -Mstrict -Mdiagnostics -wle 'sub x {my $x if 0; print $x++} x;x;x;'
Deprecated use of my() in false conditional at -e line 1 (#1)
  (D deprecated) You used a declaration similar to my $x if 0.
  There has been a long-standing bug in Perl that causes a lexical variable
  not to be cleared at scope exit when its declaration includes a false
  conditional. Some people have exploited this bug to achieve a kind of
  static variable. Since we intend to fix this bug, we don't want people
  relying on this behavior. You can achieve a similar static effect by
  declaring the variable in a separate block outside the function, eg

      sub f { my $x if 0; return $x++ }

  becomes

      { my $x; sub f { return $x++ } }

  Beginning with perl 5.9.4, you can also use state variables to
  have lexicals that are initialized only once (see feature):

      sub f { state $x; return $x++ }

0
1
2
```

Listing 1

WxPerl Tutorial

In der 12. Ausgabe (4/2009) beim Artikel Wx Perl Tutorial - Teil 1 gibt es 2 Aussagen die nicht passend sind.

FLTK lebt, wie unter *http://search.cpan.org/~sanko/FLTK-0.531/ bzw. http://github.com/sanko/fltk-perl* zu sehen ist, als auch QT4 Perl die Arbeit daran wurde begonnen, siehe *http://code.google.com/p/perlqt4/ was wohl auf http://socghop.appspot.com/student_project/show/google/gsoc2009/debian/t124022199116* zurückzuführen ist, davon abgesehen funktioniert QT3 bei mir im Test weiterhin.

Bei Tk fehlt mir die Erwähnung von Tkx. Sicherlich ist Wx Perl ein wunderbares Toolkit aber die Alternativen einfach mangels ordentlicher Recherche aus dem Rennen zu nehmen finde ich nicht toll.

Gruss Alexander

Anmerkung von Herbert Breunung

Hallo Alexander.

Es stimmt, dass ich mehr Zeit für Rechereche hätte aufbringen können, dennoch kannte ich bereits Tkx, weil es vom Active State ppm4-Client benutzt wird, den ich mal untersuchte. Beim Ausformulieren hatte ich mich entschieden es nicht zu erwähnen, da es dann noch andere Details geben würde, die ich behandeln müsste.

Das FLTK-Revival kam erst nach Abgabe des Artikel, sodass ich es nicht erwähnen konnte. Und auch QT besitzt derzeit eine gelbe bis rote Testmatrix mit einem letzten release von 1996 (CPAN). Ich bin dankbar, dass du mich auf das googolprojekt aufmerksam gemacht hast, aber selbst jetzt, einige Monate später, sehe ich dort nichts das man produktiv benutzten könnte. Ich war selbst einmal Mentor dort und weiß, dass nur ein Drittel dieser Programme etwas langfristig Lebendiges produzieren. Deswegen finde ich es etwas vorschnell, zu schreiben, ich hätte zu wenig geprüft.

Alles Gute Herbert Breunung

CPAN News XIII

XUL::Gui

XUL ist die XML User Interface Language, mit der auch Firefox und Thunderbird umgesetzt sind. Mit `XUL::Gui` ist es möglich, eine GUI zu erstellen, die dann im Firefox angezeigt werden kann, in deren Backend man aber das bekannte Perl nutzen kann. Das Dokument kommt mit einer größeren Dokumentation, die sehr hilfreich ist.

```
use XUL::Gui;

# start http server, start firefox,
# wait for events

display Window title =>
  '$foo - Perl- Magazin - XUL Test',
   minwidth=>300,
 GroupBox(

   # a small 'headline'
   Caption('XUL Test'),

   # id is necessary for interactions
   TextBox( id => 'show' ),

   # when button is clicked, show URL
   # in textfield
   Button(
       label=>'show URL for $foo website',
           oncommand=> sub {
               $ID{show}->value =
               'http://perl-magazin.de';
   }),
 );
```

HTML::Filter::Callbacks

Aufgabe: Umzug der Webseiten auf einen neuen Server mit einer neuen Domain. Alle HTML-Dateien müssen entsprechend angepasst werden. Um sich die Arbeit hierbei zu vereinfachen, kann man das Modul `HTML::Filter::Callbacks` verwenden. Hier lässt sich für bestimmte "Ereignisse" (bzw. Tags) eine Aktion bestimmen. So können zum Beispiel die *hrefs* in Links angepasst werden:

```
use HTML::Filter::Callbacks;

my $html = do { local (@ARGV,$/) =
                    'test.html'; <> };

my $filter = HTML::Filter::Callbacks->new;
$filter->add_callbacks(
  'a' => {
    start => sub {
      shift->replace_attr(href => sub {
          my ($url) = @_;
          $url =~ s/alte-domain\.de/
                   neue-domain.de/;
          $url;
      })
    },
  },
);
my $new_html = $filter->process($html);
print $new_html;
```

Acme::Daily::Fail

Reporter sein ist ganz schön schwer. Immer muss man sich Überschriften für die Artikel überlegen. Für englischsprachige Boulevard-Journalisten gibt es jetzt eine Hilfe auf CPAN - `Acme::Daily::Fail`.

```
use Acme::Daily::Fail qw(get_headline);

print get_headline(), "\n";
```

Path::Executable

Möchte man den Pfad zu einer ausführbaren Datei herausfinden, kann man sich `Path::Executable` zu Hilfe nehmen. Damit werden die Pfade, die in der `PATH`-Umgebungsvariablen gesetzt sind, nach der Datei durchsucht.

```
use Path::Executable;

my ($perl) = path_exe( 'perl' );
print $perl;
```

File::Find::Upwards

Mit `File::Find` kann man Dateien suchen, die in einem bestimmten Verzeichnis oder darunter liegen. Mit `File::Find::Upwards` kann man nach einer Datei suchen, die im aktuellen Verzeichnis oder darüber liegt.

```
use File::Find::Upwards
                  qw(file_find_upwards);
my $file =
      file_find_upwards( 'article.pod' );
print $file;
```

XML::Compare

`XML::Compare` bietet die Möglichkeit, zwei XML-Dateien zu vergleichen und damit zu prüfen, ob diese semantisch identisch sind. Dabei werden Namensräume nicht berücksichtigt. Also sind

```
<magazin xmlns="urn:message">
  <i id="13"><art id="1">CPANNews</art></i>
</magazin>
```

semantisch identisch mit

```
<foo:magazin  xmlns:foo="urn:message">
  <foo:i id="13">
    <foo:art id="1">CPANNews</foo:art>
  </foo:i>
</foo:magazin>
```

```
use XML::Compare tests => 2;

my $xml1 = do{ local (@ARGV,$/) =
               'plain.xml'; <> };
my $xml2 = do{ local (@ARGV,$/) =
               'namespace.xml'; <> };

my $error;
eval {
       XML::Compare::same($xml1, $xml2); 1;
     }
     or $error = $@;
if ( !$error ) {
    print "same\n";
}
else {
     print "different: $@\n";
}
```

Termine

Februar 2010

02. Treffen Vienna.pm
Treffen Frankfurt.pm
04. Treffen Dresden.pm
06./07. FOSDEM
08. Treffen Ruhr.pm
09. Treffen Stuttgart.pm
12. Treffen Cologne.pm
15. Treffen Erlangen.pm
17. Treffen Darmstadt.pm
20. Perl 6 Discovery Workshop
23. Treffen Bielefeld.pm
24. Treffen Berlin.pm

März 2010

02. Treffen Frankfurt.pm
Treffen Vienna.pm
02.-06. CeBIT mit Stand der TPF
04. Treffen Dresden.pm
07. 7. Niederländischer Perl-Workshop
08. Treffen Ruhr.pm
09. Treffen Stuttgar.pm
15. Treffen Erlangen.pm
17. Treffen München.pm
Treffen Darmstadt.pm

April 2010

01. Treffen Dresden.pm
06. Treffen Frankfurt.pm
Treffen Vienna.pm
12. Treffen Ruhr.pm
13. Treffen Stuttgart.pm
19. Treffen Erlangen.pm
21. Treffen Darmstadt.pm
24./25. OSDC Taiwan
27. Treffen Bielefeld.pm
28. 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

LINKS

http://www.perl-nachrichten.de

Unter Perl-Nachrichten.de sind deutschsprachige News rund um die Programmiersprache Perl zu finden. Jeder ist dazu eingeladen, solche Nachrichten auf der Webseite einzureichen.

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.perlmongers.de/
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.

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.

Smart-Websolutions

Windolph und Bäcker GbR

Perl-Programmierung

info@smart-websolutions.de

// SEIBERT / MEDIA

BESSERE ATMOSPHÄRE? MEHR FREIRAUM?

Wir suchen erfahrene Perl-Programmierer/innen (Vollzeit)

//SEIBERT/MEDIA besteht aus den vier Kompetenzfeldern Consulting, Design, Technologies und Systems und gehört zu den erfahrenen und professionellen Multimedia-Agenturen in Deutschland. Wir entwickeln seit 1996 mit heute knapp 60 Mitarbeitern Intranets, Extranet-Systeme, Web-Portale aber auch klassische Internet-Seiten. Seit 2005 konzipiert unsere Designabteilung hochwertige Unternehmensauftritte. Beratungen 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 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

„Statt mit blumigen Worten umschreiben unsere Programmierer den Job so:

Apache, Catalyst, CGI, DBI, JSON, Log::Log4Perl, mod_perl, SOAP::Lite, XML::LibXML, YAML“