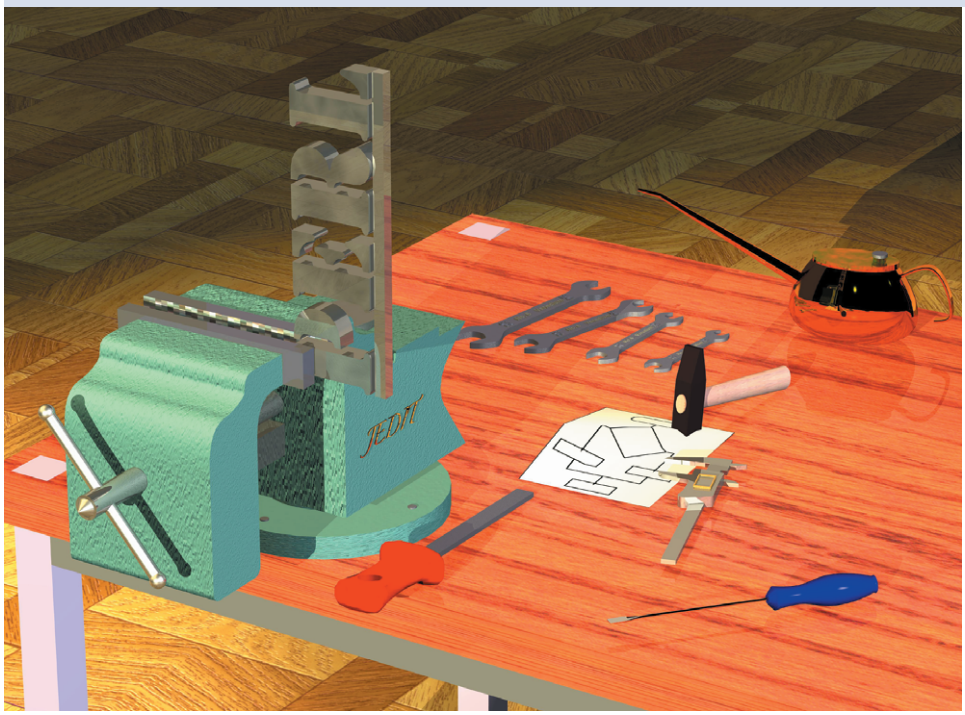
Perl-Entwicklungsumgebung als Eigenbau

Workshop für Werkzeugmacher

Linux-taugliche Entwicklungswerkzeuge für Perl sind rar, doch eine Eigenbau-IDE ist mit JEdits Hilfe kein Problem und zudem kostenlos, maßgeschneidert und plattformunabhängig. Jens-Christoph Brendel



Es ist paradox: Ausgerechnet unter Unix, also jenem Betriebssystem, auf dem Perl vor gut zehn Jahren entstand und zu dessen Standardinstallationen es bis heute gehört, gibt es kaum eine Handvoll Programmier-Umgebungen für diese Skriptsprache – die Windows-Welt bevölkern sie im Dutzend. Eine Erklärung mag sein, dass der Unix-User per se eine gut sortierte Werkzeugkiste mit mächtigen Editoren und Utilities zur Hand hat und womöglich eher zum Kommandozeilen-Puristen als zum Fenster-Fetischisten tendiert.

Dennoch ist der Nutzen mancher Features einer integrierten Entwicklungsumgebung (IDE) schwer von der Hand zu weisen – etwa die nahtlose Verbindung von Editor und Debugger, die automatische Quellcode-Formatierung oder die kontextsensitive Online-Referenz. Sicher ebenso nützlich sind CVS-Anbindung, Syntaxcheck oder Navigationshilfen für umfangreiche Programmtexte.

Wer sich dabei nicht für eine der seltenen Perl-IDEs unter Linux erwärmen mag – etwa das sehr gute, aber kommerzielle Kommodo (Activestate, [1]) oder das freie Perl-Plugin für das schwergewichtige Eclipse [2] – hat eine Alternative: Do it yourself.

Alleskönner

Als Fundament eines solchen Projekts dient hier der Open-Source-Editor JEdit, von dem erst kürzlich die lange erwartete Version 4.2 erschienen ist. In Java geschrieben ist er auf jeder Plattform zu Hause und in puncto Funktionsumfang, Konfigurationsmöglichkeiten und Erweiterbarkeit nimmt er es mit den Altvorderen seiner Gattung, Emacs und Vi nebst Nachfahren, leicht auf. JEdit [3] beherrscht das Syntax-Highlighting für mehr als 100 Sprachen von Actionscript bis XML, darunter natürlich auch für Perl. Für die Navigation durch das Dokument und die Textmanipulation bietet er zahlreiche ausgeklügelte Funktionen. Dabei können alle irrtümlichen Eingaben unbeschränkt zurückgenommen werden.

Das Arbeiten mit gleichzeitig geöffneten Fenstern dürfte kaum Wünsche offen lassen. Suchen und Ersetzen ist über mehrere Files hinweg und mit Hilfe regulärer Ausdrücke möglich. Textteile lassen sich in beliebig benennbare Clipboards kopieren und von dort aus an anderer Stelle einfügen. Eine leistungsstarke Makrosprache automatisiert wiederkehrende Aufgaben. Fast alle Funktionen sind auch über Menüs erreichbar – wer das wenig effizient findet, definiert sich seine eigenen Shortcuts oder freundet sich einfach mit den voreingestellten Tastaturbefehlen an.

Toolbox-Tuning

Schon die Grundausstattung ist also opulent. Um sie um jene Extras zu erweitern, die einen guten Editor in eine vollwertige Entwicklungsumgebung verwandeln, führt der kürzeste Weg über [http://plugins.jedit.org]. Dort finden sich mehr als 100 fertige, in Java programmierte Module, mit denen sich im Handumdrehen komplexe, häufig auch nach Gusto des Anwenders konfigurierbare Funktionen nachrüsten lassen.

JEdit bringt ein Hilfsmittel für die Verwaltung der Module bereits mit: den Plugin Manager, der eine Zusammenstellung verfügbarer Erweiterungen via Internet besorgt und in einer Auswahlliste präsentiert. Mit wenigen Klicks hat

man damit seine Favoriten markiert, heruntergeladen und – inklusive Dokumentation in der Onlinehilfe – automatisch installiert.

Grundausstattung

Für eine Entwicklungsumgebung unverzichtbar ist das Plugin »Console«, das nicht nur Shell-Fenster im Editor zu öffnen vermag, sondern später auch für eigene Makros benötigt wird, die Kommandozeilen-Utilities integrieren sollen. Daneben ist es beim Testen der Beanshell-Programme nützlich, aber dazu später mehr.

Das Plugin »Error List« spendiert JEdit ein eigenes Fenster für Fehlermeldungen, das sich wie die meisten Editorfenster auf Wunsch an einer bestimmten Position des Hauptfensters andocken lässt »Utilities | Global Options… | jEdit: Docking«. Jede Fehlernachricht des Editors, des Perl-Interpreters oder einer Shell präsentiert es als Verweis auf die jeweilige Problemstelle im Programm.

Das Ein- und Auschecken von Files aus einem CVS-Repository ist mit Hilfe des Plugins »Gruntspud« direkt aus dem Editor heraus möglich. An anderen Orten gespeicherte Files findet »OpenIt« besonders schnell: Sobald der Benutzer einen Filenamen einzutippen beginnt, fahndet es in einstellbaren Suchpfaden nach den Dateien, die gemeint sein könnten, und bietet die Treffer in einer Kurzwahlliste an.

Verwaltungshelfer

Der »Project Viewer« ergänzt den Editor um eine einfache Projektverwaltung auf der Grundlage von Verzeichnissen zusammengehöriger Files, die er in einer faltbaren Baumstruktur darstellt. Die Navigation in langen Quelltexten erleichtert JEdit bereits ohne Zusätze durch Lesezeichen und das so genannte Folding, also das Ein- und Ausblenden von Codeblöcken und Kommentaren, wahlweise automatisch oder anhand manuell platzierter Markierungen.

Einen Schritt weiter geht das Plugin »Code Browser«, das in einem separaten Fenster alle Perl-Packages und -Subroutinen (respektive Klassen und Methoden) in Form von Links in den Programmtext sammelt. Es wertet dazu ein Index-File aus, das es automatisch bei jedem Fensterwechsel und jedem Sichern mit Hilfe des Utility Exuberant Ctags erstellt [4]. Mit diesen nützlichen Funktionen aufgerüstet präsentiert sich JEdit wie in Abbildung 1 zu sehen.

Die erwähnte Plugin-Website lädt zum Stöbern im gut dokumentierten Fundus der JEdit-Extras ein, wo sich für zahlreiche Anwendungsfälle passende Bausteine finden: Sei es für die Anbindung an einen SQL-Server, die eigene Template-Bibliothek, Textvergleiche, die Bearbeitung von XML-Dokumenten (nebst Transformation mit XSLT-Stylesheets), das Testen komplizierter regulärer Ausdrücke und für vieles mehr.

Wenn der geplanten Perl-IDE dann noch Funktionen fehlen, für die es bis jetzt kein fertiges Plugin gibt, dann kann ein

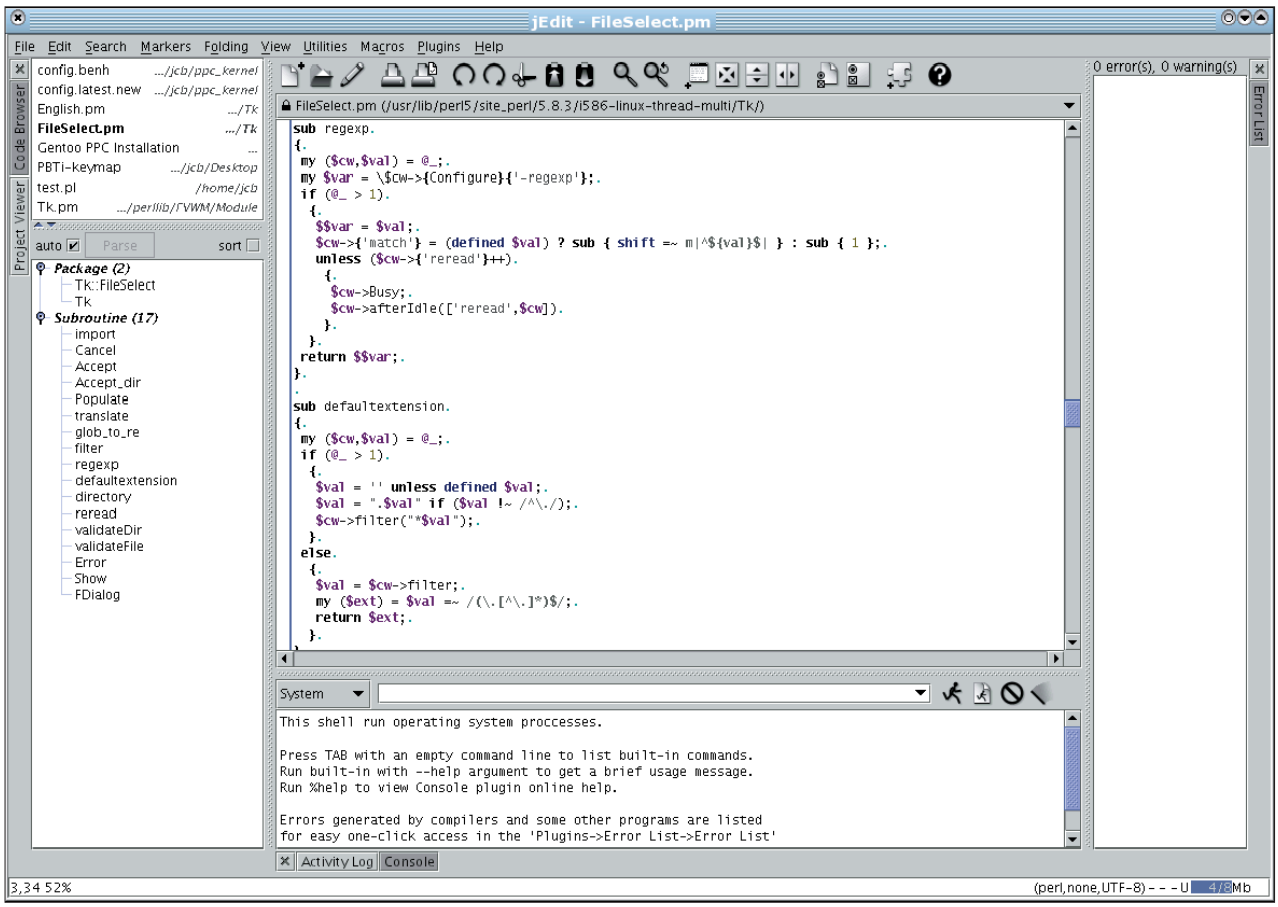


Abbildung 1: Das Hauptfenster des Editors. Rechts ist das Fenster für Fehlermeldungen angedockt, links das Code-Browser-Window. Den Platz am unteren Rand belegt das Console-Plugin mit seinem Shell-Fenster. Alle Unterfenster lassen sich mit einem Klick einklappen, wenn die Übersicht verloren zu gehen droht.

passendes Standalone-Tool einspringen – JEdit integriert externe Hilfsprogramme einfach und bereitwillig.
So ist beispielsweise das kleine Utility Perltidy [5] für die optische Verschönerung des Programmtextes prädestiniert. Es wird mit dem Namen eines Perl-File und einer Auswahl aus mehreren dutzend Optionen aufgerufen, die jeden erdenklichen Formatierungsparameter steuern können: Einzüge, Weißraum, Klammerausrichtung, Tabs, Zeilenumbrüche und vieles mehr. Um dafür nicht den Editor verlassen zu müssen und das Ergebnis nach dem Make-up gleich weiterbearbeiten zu können, bietet es sich an, den Aufruf einem Makro zu übertragen.

Makro fürs Make-up

JEdits Makrosprache heißt Beanshell. Dabei handelt es sich um einen Dialekt mit Java-Syntax und den typischen Eigenschaften einer Skriptsprache wie Interpreter statt Compiler, Dynamic statt Strong Typing oder globalen Variablen und Funktionen. Die Stärken des Konzepts kommen besonders zum Tragen, wenn die Beanshell wie im Fall von JEdit als eingebettete Skriptsprache einer Java-Applikationen eingesetzt wird. Denn damit können die Makros aus dem Vollen schöpfen: Alle Objekte und APIs ihrer Mutter-Applikation stehen ihnen zu Diensten. Sämtliche Methoden, die in JEdits Innenleben die eigentliche Arbeit verrichten, können Makros für eigene Zwecke einspannen. Die selbst programmierten Funktionen lassen sich bei Bedarf sogar mit einer grafischen Oberfläche ausstatten, der das komplette Repertoire von Javas AWT/Swing zur Verfügung steht.
Die hier gezeigten Beispiele sind nicht übermäßig kompliziert, trotzdem wird sicherlich deutlich, dass dem ambitionierten Makro-Programmierer damit ein sehr mächtiges Werkzeug in die Hand gegeben ist. Eine umfangreiche Dokumentation findet sich auf der Beanshell-Homepage [7], auch das JEdit-Manual enthält ein ausführliches Kapitel über Beanshell-Programmierung.
An die Grenzen stößt man beispielsweise dort, wo es sinnvoll wäre, eigene Klassen als Erben existierender Java-Klassen einzusetzen – das ist den Makros zurzeit nicht möglich – oder wo die Ausführungsgeschwindigkeit zum kritischen Faktor wird. In diesen Fällen wäre die Entwicklung eines Plugins mit Java sicherlich besser.

Listing 1: Code-Make-up

```
01 run_perltidy() {
02    String tidyOptions = "-ce -st -se";
03    String bufferPath = buffer.getPath();
04    String runTidy = "/usr/bin/perltidy"+"
   "+tidyOptions+" "+bufferPath;
05    runCommandToBuffer(view,"System", runTidy);
06 }
07
08 run_perltidy();
```

Eine geskriptete Shell

Zurück zu Perltidy und einem ersten, einfachen Beispiel. Ein passendes Makro für seinen Aufruf könnte etwa so aussehen wie in Listing 1. Die Zeilen zwei bis vier setzen die Kommandozeile zusammen und legen sie in der String-Variablen »runTidy« ab. Die Optionen für die Formatierung (»-ce«) lassen sich beliebig dem eigenen Stil anpassen und »-st« sowie »-se« sind erforderlich, damit Perltidy auf die Standardausgabe beziehungsweise den Standard-Error-Kanal schreibt. Sonst würde es stattdessen ein neues File mit der Endung ».tdy« erzeugen, was beispielsweise im Batch-Betrieb sinnvoll wäre.

Grafisch garniert

Kandidat für die Verschönerung ist das File im aktuellen Fenster. Dessen Pfad ist über das »buffer«-Objekt des zugehörigen Textpuffers zu erfahren. Danach wird Perltidy aus der Shell heraus gestartet. »RunCommandToBuffer()« – eine Funktion des schon erwähnten Console-Plugins – bewirkt, dass das geschminkte File gleich in einem neuen Puffer des Editors landet.
Damit liegen beide Versionen für einen Vorher-Nachher-Vergleich vor. Wer mit dem Ergebnis zufrieden ist, sichert das neue, noch unbenannte File unter dem Namen des Originals, JEdit schließt dabei automatisch das Fenster der überschriebenen Vorgängerversion. Andernfalls verwirft man einfach das Ergebnis des Makros.
Die Datei »perltidy.bsh« wird am besten in einem Verzeichnis namens »Perl« unterhalb des Makro-Ordners im JEdit-Installationsverzeichnis abgelegt. Nach einem Neustart des Editors findet sich im Menü »Macros | Perl« nun der neue Unterpunkt »perltidy«. Um sich den Weg dorthin zu ersparen, besteht die Möglichkeit, sich eine Tastenkombination für den Aufruf einzurichten (»Utilities | Global Options | Shortcuts«).

Listing 2: »runperl.xml«

```
01 <!DOCTYPE COMMANDO SYSTEM "commando.dtd">
02 <COMMANDO>
03  <UI>
04   <CAPTION LABEL="Perl">
05    <CHOICE LABEL="Action" VARNAME="action">
06     <OPTION LABEL="Run" VALUE="run"/>
07     <OPTION LABEL="Syntax Check"
   VALUE="check"/>
08     <OPTION LABEL="Debug" VALUE="debug" />
09    </CHOICE>
10    <ENTRY LABEL="Perl Options"
   VARNAME="compileOptions" />
11    <ENTRY LABEL="File name" VARNAME="file"
   EVAL="buffer.getPath()" />
12    <ENTRY LABEL="Arguments" VARNAME="args" />
13   </CAPTION>
14  </UI>
15  <COMMAND CONFIRM="FALSE" SHELL="System"
   TO_BUFFER="FALSE">
16   buf = new StringBuffer();
17   boolean run = false;
18   if(action.equals("check"))
19   {
20    buf.append("perl -c");
21   } else if (action.equals("debug")) {
22    buf.append("perl -d:ptkdb");
23    run = true;
24   } else {
25    buf.append("perl");
26    run = true;
27   }
28   if(!compileOptions.equals("") &amp;&amp;
   !run)
29    buf.append(" "+ compileOptions);
30    buf.append(" " +file);
31   if( !action.equals("compile") )
32    buf.append(" "+args);
33 // return value
34    buf;
35  </COMMAND>
36 </COMMANDO>
```

Für einen ganz ähnlichen Einsatzfall kennt JEdit noch eine spezielle Makro-Unterart, das so genannte Commando. Im Unterschied zu dem vorgestellten Beispiel ist es dessen Spezialität, automatisch eine Combo-Box zu generieren, in der der Benutzer sowohl vordefinierte Parameter auswählen als auch von Fall zu Fall frei definierbare Programmoptionen eintragen kann.

Schnellstarter

Das nächste Makro ermöglicht es, dass sich der Programmtext eines Editorfensters auf Syntaxfehler prüfen, in einen Debugger laden und probeweise vom Shell-Prompt aus starten lässt. Dafür reicht der Aufruf des Perl-Interpreters, jeweils mit speziellen Optionen. Zusätzlich soll das eigene Skript noch mit Argumenten versorgt werden können. Eine solche flexible Möglichkeit für den Programmstart aus dem Editor heraus ist auf elegante Weise einzurichten: mit Hilfe einer XML-Datei, nach deren Vorgaben das Console-Plugin ein entsprechendes Dialogfenster zusammenbaut (siehe Listing 2).

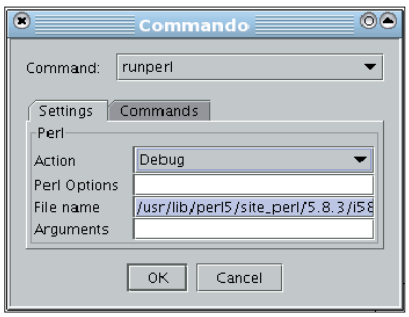


Abbildung 2: Das nach den Vorgaben einer XML-Datei automatisch genrierte Dialogfenster eines Commando-Makros.

Bauplan in XML

Das Root-Element »COMMANDO« enthält zwei Bereiche, die unterschiedliche Funktionen haben. Die zwischen die »<UI>«-Tags geschachtelten Elemente definieren die grafischen Zutaten des Dialogfensters: Eingabefelder (»ENTRY«) und ein Menü (»CHOICE«) mit seinen Optionen (»OPTION«). Die Attribute geben die Beschriftungen (»LABEL«), die Rückgabewerte (»VALUE«) beziehungsweise jene Variablen vor, die die getroffene Auswahl oder Eingaben speichern sollen (»VARNAME«).

Innerhalb des Elements »COMMAND« dagegen wird in Abhängigkeit davon, welche Aktion der Benutzer ausgewählt und welche Werte er eventuell in die Textfelder eingegeben hat, die Kommandozeile zusammengestellt und ausgeführt. Das Beispiel erzeugt ein Pull-Down-Menü »Action«, das die Auswahlmöglichkeiten »Run«, »Syntax Check« und »Debug« anbietet. Die Entscheidung des Benutzers merkt sich die Variable »action«.

Daneben gibt es Texteingabefelder: »Perl Options« nimmt Optionen des Perl-Interpreters entgegen und speichert sie in der Variablen »compileOptions«. Das Feld »File Name« ist automatisch mit dem Pfad zur Datei im aktuellen Editorfenster vorbelegt. Schließlich kommt als drittes Eingabefeld noch »Arguments« hinzu, das Kommandozeilen-Argumente für das Skript selbst aufnimmt.

Maskiert

XML verwendet übrigens eine Reihe von Zeichen (<, >, &, ', ") für eigene Zwecke. Will der Benutzer ein solches Zeichen verwenden, ohne dass der Parser es in XML-Manier interpretiert, muss er es als so genannte Character Entity kodieren. Im Beispiel betrifft dies »&« (in

Installationshinweise

JEdit in der Version 4.2final steht auf [3] zum Download bereit. Für Linux sind RPM- und Debian-Packages im Angebot, zusätzlich gibt es einen Java-basierten Installer, der auf allen Plattformen funktioniert. Empfohlen wird Java 1.4.x.

Das Utility Perldoc ist meist Bestandteil der Perl-Distribution. Falls es fehlt, ist es auf [9] zu haben. Installiert wird es genauso wie Perlindex [10] und Perltidy [5]: herunterladen, entpacken, »perl Makefile.PL«, »make«, »sudo make install«.

Perlindex – selbst ein Perl-Skript – ist danach einmalig durch den Root-User mit der Option »-index« aufzurufen. Perlindex weiß, wo es suchen muss, durchkämmt selbstständig die gesamte Perl-Verzeichnishierarchie und indiziert alle verwertbaren Files.

Zuvor ist jedoch ein minimaler Eingriff erforderlich: Zeile 60 des Skripts definiert eine Variable »$prefix« und füllt sie mit dem Directorynamen »/usr«. Die Verweise im Index werden dann relativ zu diesem Präfix gespeichert. Das Hilfe-Makro benötigt später allerdings absolute Pfadangaben. Also ist entweder im Perlindex-Skript die Zeile 60 in »$prefix=''« zu verändern oder das Präfix im Makro jedem Pfad voranzustellen. Ersteres ist bestimmt einfacher.

Für das Code-Browser-Plugin muss außerdem der Pfad zum Ctags-Utility [4] über »Plugins | Plugin Options... | Code Browser« eingetragen werden.

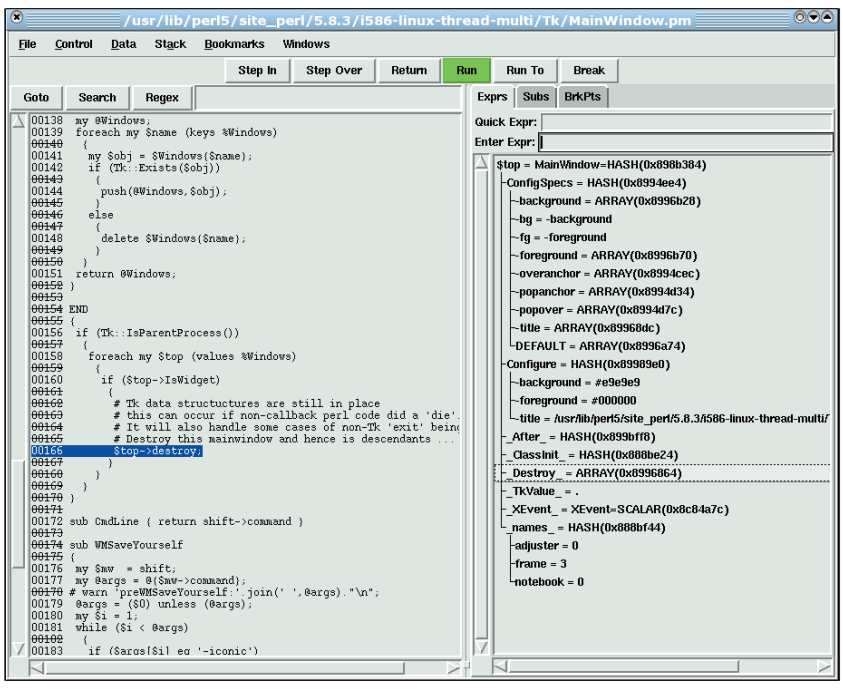


Abbildung 3: Ist alles richtig konfiguriert, erscheint auf Knopfdruck der grafische Perl-Debugger »ptkdb«.

XML dient es als Markierung für eine Entity-Referenz), das hier als logisches UND gelesen werden soll und deshalb als »&« zu schreiben ist. Den fertigen Dialog zeigt Abbildung 2.

Damit der Start des externen Debuggers auch wie im Beispiel klappt (Abbildung 3), muss das Perl-Modul Devel::ptkdb installiert sein. In ähnlicher Weise ließe sich alternativ der DDD [8] einbinden oder zum Beispiel zusätzlich der Profiler Devel::Dprof startklar machen. Und zu guter Letzt das kleine XML-File in »~/.jedit/console/commando« speichern, Editor neu starten – und schon findet sich das eigene Kommando unter »Plugins | Console | Commando« wieder.

Das folgende Projekt ist schon umfangreicher: Es geht um eine kontextsensitive, auf Tastendruck aufrufbare Onlinehilfe zu allen Perl-Funktionen, Operatoren, Datenstrukturen und dergleichen, inklusive Volltextsuche, und das alles integriert in die grafische Oberfläche des Editors.

Perlentaucher

Die eigentliche Arbeit verrichten wieder kleine Helfer im Hintergrund. Diesmal sind es zwei: Perldoc [9] für die Anzeige von Dokumenten im POD-Format (Plain Old Documentation, ein einfaches Text-Markup) – auf diese Weise erschließt sich die gesamte Perl-Referenz – und Perlindex [10] für die Suche nach relevanten Texten in der kompletten lokalen Perl-Installation, und zwar einschließlich der in Modulen, Packages und anderen Programmtexten eingebetteten Dokumentation.

Eine Voraussetzung dafür ist, dass die Perl-Dokumentation im POD-Format tatsächlich installiert ist, was meist, aber nicht immer zutrifft. Wer deswegen unsicher ist, verschafft sich etwa über ein versuchsweise ausgeführtes »perldoc perltoc« Klarheit. Fehlen die ».pod«-Files, kann man sie am einfachsten aus einem Perl-Pod-Package nachinstallieren, das mit einer einschlägigen Such-

Listing 3: »perlhelp.bsh«

```
001 import java.awt.event.*;
002 import java.awt.*;
003 import javax.swing.*;
004 import javax.swing.text.*;
005
006 getKeyword() {
007   noWordSep = "_/:";
008   lineNr    = textArea.getCaretLine();
009   lineStart = textArea.getLineStartOffset(lineNr);
010   offset    = textArea.getCaretPosition() - lineStart;
011   if(textArea.getLineLength(lineNr) == 0) return "";
012   String lineText = textArea.getLineText(lineNr);
013   if(offset == textArea.getLineLength(lineNr)) offset--;
014   wordStart  = TextUtilities.findWordStart(lineText, offset, noWordSep);
015   wordEnd  = TextUtilities.findWordEnd(lineText, offset + 1, noWordSep);
016   textArea.select(lineStart + wordStart, lineStart + wordEnd);
017   String keyword = lineText.substring(wordStart, wordEnd);
018   return keyword;
019 }
020
021 runQuery(keyword) {
022   functionSearchCmd = "perldoc -t -T -f ";
023   indexSearchCmd   = "perlindex -nomenu -nocbreak ";
024   keywordSearchCmd  = "perldoc -t -T ";
025   Vector resultVector= new Vector();
026   MainWin = new JFrame( "PerlDoc Results" );
027   SplitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
028   SplitPane.setResizeWeight(0.0);
029
030   doSearch(command, keyword) {
031     String s;
032     resultVector.removeAllElements();
033     p   = Runtime.getRuntime().exec(command + keyword);
034     out = new BufferedReader(new InputStreamReader(p.getInputStream()));
035     while ( (s = out.readLine()) != null ) {
036       resultVector.addElement(s);
037     }
038     p.waitFor();
039     if ( resultVector.size() == 0 ) {
040       return 1;
041     } else {
042       return p.exitValue();
043     }
044   }
045
046   prepareKeywordSearchResults() {
047     SimpleAttributeSet outputAttr  = new SimpleAttributeSet();
048     DefaultStyledDocument textDocument = new DefaultStyledDocument();
049     JTextPane docPane = new JTextPane(textDocument);
050     resElements = resultVector.elements();
051
052     while( resElements.hasMoreElements() ) {
053       textDocument.insertString(textDocument.getLength(),
054        resElements.nextElement() + "\n", outputAttr);
055     }
056
057     docPane.setEditable(false);
058     docPane.setCaretPosition(0);
059     JScrollPane docScrollPane = new JScrollPane( docPane );
060     docScrollPane.setMinimumSize(new Dimension(650,200));
061     return docScrollPane;
062   }
063
064   prepareIndexSearchResults() {
065     i = 0;
066     String[] columnNames = {"SCORE","DOCUMENT"};
067     String [][] resultArray = new String [15][2];
068     resElements = resultVector.elements();
069
070     while( resElements.hasMoreElements() ) {
071       resTokens = new StringTokenizer( resElements.nextElement() );
072         while ( resTokens.hasMoreElements() ) {
073           nextToken = resTokens.nextToken();
074           resultArray[i][0] = nextToken;
075           nextToken = resTokens.nextToken();
076           if ( ! nextToken.startsWith("/") ) nextToken = "/" + nextToken;
077             resultArray[i][1] = nextToken;
078             i++;
079         }
080     }
081
082     MenuTab = new JTable(resultArray, columnNames);
```

maschine wie Rpmfind.net schnell aufgespürt ist. Wenn alle Stricke reißen und für die eigene Kombination aus Plattform und Perl-Version kein fertiges Paket zu finden ist, bleibt als letzter Ausweg, sich die zur installierten Perl-Version passenden Sourcen zu besorgen [11], auszupacken und das enthaltene »pod«-Verzeichnis in den eigenen Perl-Pfad zu verschieben (meist »/usr/local/lib/perl5/*Version*/pod«).

Javaesk

Die Ausgaben von Perlindex sind in diesem Makro noch nicht das Endergebnis – die Textsuche produziert zuerst eine nach Relevanz sortierte Trefferliste, die zunächst zu einem Menü aufbereitet werden soll und erst darüber zu einem Dokument führt.

Wären für den Programmstart wie bisher die Fähigkeiten des Console-Plugins oder ein Commando-File zuständig, ließe sich eine solche Verarbeitungskette allerdings nur schwer realisieren. Das im Folgenden beschriebene Makro nutzt deshalb Javas Prozess-Mechanismen, um die externen Programme zur Arbeit zu rufen, fängt deren Output-Stream auf, werkelt ein wenig an den Daten herum und gibt das Endergebnis schließlich wieder aus. Da das Makro auch so merklich länger gerät als seine Vorgänger, ist es etwas strukturierter angelegt und in einzelne Funktionen unterteilt.

Hand in Hand

Da wäre zunächst »getKeyword()«, zuständig für die Ermittlung des Wortes, in dem sich gerade der Cursor befindet. Die Funktion extrahiert mit Hilfe diverser Methoden des Objekts »textArea« die aktuelle Zeile und daraus schließlich das Wort an der Cursorposition, das nun als Suchbegriff herhalten soll. Befand sich der Cursor beim Makrostart zwischen zwei Wörtern oder in einer leeren Zeile, erscheint ein Dialogfenster, das die freie Eingabe eines Suchworts erlaubt. ▶

Listing 3: »perlhelp.bsh« (Fortsetzung)

```
083   MenuTab.getColumnModel().getColumn(0).setMaxWidth(100);
084   MenuTab.setPreferredScrollableViewportSize(new Dimension(500, 100));
085   MenuTab.setSelectionBackground(new Color(255,255,155));
086   MenuTab.setRowSelectionAllowed(true);
087   MenuTab.setDefaultEditor(Object.class, null);
088   MenuTab.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
089   currentRow = 0;
090
091   ListSelectionModel rowSM = MenuTab.getSelectionModel();
092   rowSM.addListSelectionListener(new ListSelectionListener() {
093     void valueChanged(ListSelectionEvent e) {
094       if ( ! e.getValueIsAdjusting() ) {
095         currentRow = MenuTab.getSelectedRow();
096         keyword="";
097         keyword = MenuTab.getValueAt(currentRow,1);
098         doSearch(keywordSearchCmd, keyword);
099         showResultWin(prepareKeywordSearchResults(), "bottom");
100       }
101     }
102   });
103   JScrollPane docScrollPane = new JScrollPane( MenuTab );
104     docScrollPane.setMinimumSize(new Dimension(650,225));
105   return docScrollPane;
106 }
107
108 showIndexSearchDialog() {
109   String keyword = "";
110   while ( keyword.length() == 0 ) {
111     keyword = Macros.input(view, "Either you has not positioned the "
112     + "cursor within a \nword or the word was no perl keyword. However "
113     + "\nyou can search for any phrase in the complete \n"
114     + "perl documentation and the documentation of all \n"
115     + "installed modules using this dialog:" );
116   }
117 return keyword;
118 }
119
120 showResultWin(results, hint) {
121   MainWin.getContentPane().add(SplitPane);
122   if ( hint.equals("top")) {
123     MainWin.setSize(700,315);
124       SplitPane.setTopComponent(results);
125     } else {
126       MainWin.setSize(700,500);
127       SplitPane.setBottomComponent(results);
128     }
129   MainWin.getContentPane().setBorder(BorderFactory.createEmptyBorder
                                                        (10,10,10,10));
130   MainWin.setDefaultCloseOperation( javax.swing.JFrame.DISPOSE_
                                                          ON_CLOSE );
131   MainWin.show();
132 }
133
134 if ( keyword.length() != 0 ) {
135   if ( doSearch( functionSearchCmd, keyword ) == 0 )
      {
136     showResultWin(prepareKeywordSearchResults(), "top");
137   } else {
138     if ( doSearch( indexSearchCmd, keyword ) == 0 )
      {
139       showResultWin( prepareIndexSearchResults(), "top" );
140     } else {
141       if ( doSearch( indexSearchCmd, showIndexSearchDialog() )
                                                          == 0 )
      {
142           showResultWin( prepareIndexSearchResults(), "top" );
143       } else {
144           Macros.message(view, "Sorry, your search returned no results");
145       }
146     }
147   }
148 } else {
149   if ( doSearch( indexSearchCmd, showIndexSearchDialog() ) == 0 )
    {
150       showResultWin( prepareIndexSearchResults(), "top" );
151   } else {
152       Macros.message(view, "Sorry, your search returned no results");
153   }
154 }
155 }
156
157 runQuery( getKeyword() );
```

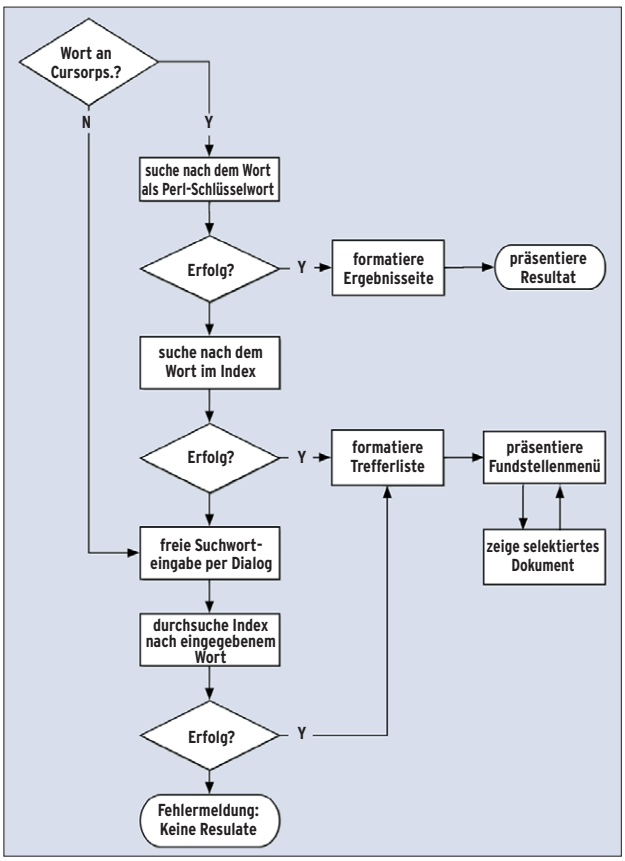


Abbildung 4: Der Arbeitsablauf des Makros für die Onlinehilfe.

Als nächster Akteur tritt in jedem Fall »keywordSearch()« auf den Plan, das seinerseits das schon vorgestellte Perldoc als Subunternehmer mit der eigentlichen Suche beauftragt: Wörter, die unter dem Cursor stehen, werden dabei als Perl-Schlüsselwörter behandelt. War die Suche erfolgreich, dann erscheint die Fundstelle in einem Fenster.

Verlief die Fahndung aber ohne Erfolg, geht es mit Plan B weiter: »indexSearch()« übernimmt den Suchbegriff und veranlasst Perlindex, seinen Index zu durchforsten (der natürlich zuvor einmalig mit »perlindex -index«, angelegt worden ist, siehe Kasten „Installationshinweise“).

Hat »indexSearch()« Einträge gefunden, erzeugt es aus den Fundstellen ein nach Trefferhäufigkeit sortiertes Menü in Form einer Liste (siehe Abbildung 5). Klickt der Benutzer auf eine Zeile der Liste, teilt sich das Fenster und in der unteren Hälfte erscheint der Text des oben selektierten Dokuments. Auf diese Weise lassen sich im Bedarfsfall alle gefundenen Hilfe-Texte leicht nacheinander durchsehen und auf ihre Eignung prüfen. Schlägt auch die Suche im Index fehl, kapituliert das Makro mit einer entsprechenden Fehlermeldung.

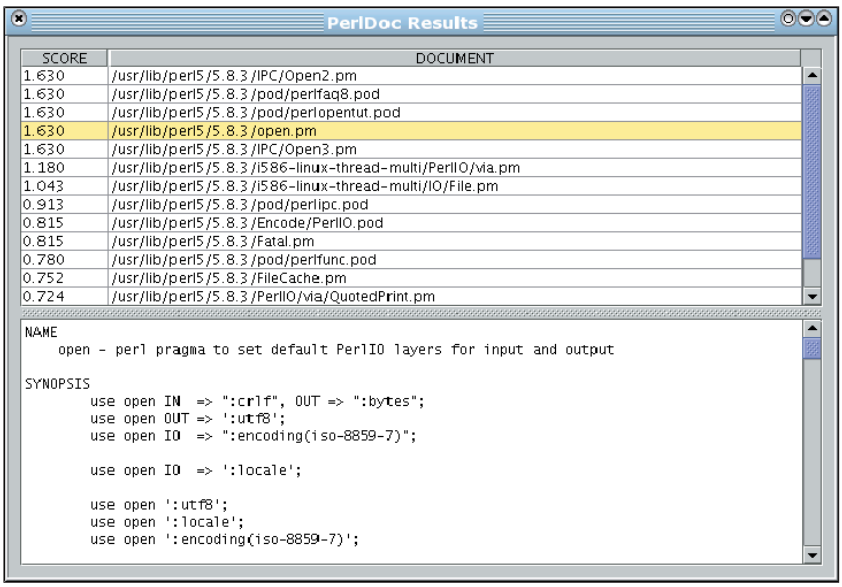


Abbildung 5: Ließ sich das Wort unter dem Cursor nicht als reserviertes Perl-Schlüsselwort identifizieren oder wurden Suchbegriffe frei eingegeben, produziert die Volltextsuche ein solches Menü aus der sortierten Trefferliste. Ein Klick auf die Zeile mit dem gewünschten Dokument blendet dessen Text unten im Fenster ein.

Fazit

Dieses Beanshell-Makro kann jeder Programmierer auf vielfältige Weise erweitern oder modifizieren. Der Beitrag weist lediglich einen möglichen Weg, um zu einer ebenso leistungsfähigen wie ganz am eigenen Bedarf orientierten Entwicklungsumgebung zu gelangen, die sich hinter vergleichbaren, vorgefertigten Werkzeugen bestimmt nicht zu verstecken braucht.

Mit JEdit ist dafür ein exzellenter Editor für Programmierer frei verfügbar, der dank seiner üppigen Ausstattung und sehr guten Dokumentation, seiner großen und ständig wachsenden Palette von Erweiterungsmodulen und seiner mächtigen Makrosprache ein tragfähiges Fundament bietet. ■

Infos

[1] Kommodo: [http://www.activestate.com]
[2] Eclipse Perl-Plugin: [http://e-p-i-c-.sourceforge.net]
[3] JEdit: [http://www.jedit.org]
[4] Ctags (ab Version 5.5): [http://ctags.sourceforge.net]
[5] Perl-Beautifier: [http://sourceforge.net/projects/perltidy/]
[6] Perl-Beautifier 2: Ein gepatchtes Perltidy, das außerdem automatisch POD-Kommentarsektionen einfügen kann, findet sich unter: [http://neuronenstern.de/proj_perltidy.html]
[7] Beanshell: [http://www.beanshell.org]
[8] Altenativer grafischer Debugger: [http://www.gnu.org/software/ddd/]
[9] Perl-Dokumentation: [http://search.cpan.org/~sburke/Pod-Perldoc-3.13/]
[10] Perlindex: [http://search.cpan.org/~ulpfr/perlindex-1.301/]
[11] Perl-Quellen, Version 5: [http://www.cpan.org/src/5.0/]
[12] Website der JEdit-Community: [http://community.jedit.org]
[13] Die Skripte zum Download: [ftp://ftp.linux-magazin.de/listings/magazin/2004/12/JEdit]