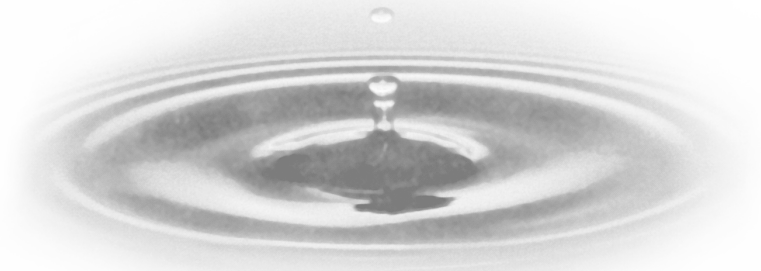


Debugger?

Real Programmers don't need debuggers, they can read core dumps.

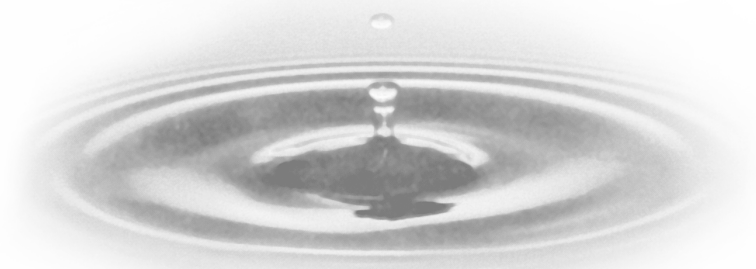
Larry Wall



Debugging?

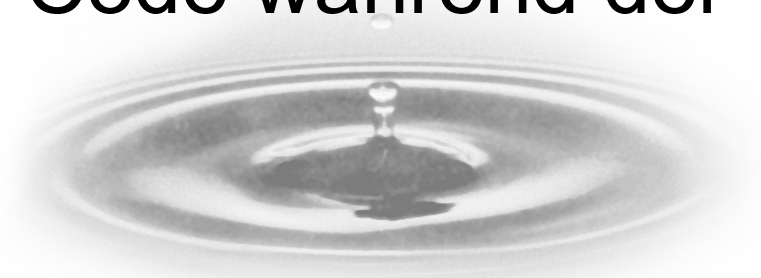
If debugging is the process of removing bugs, then programming must be the process of putting them in.

Edsger W. Dijkstra



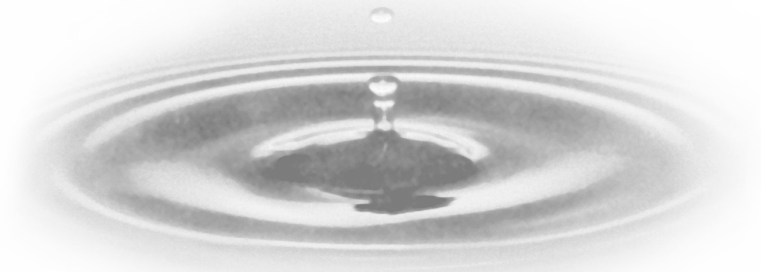
Perl Debugger Features I

- Überprüfung von Variablen, Methoden, Code und Klassenhierarchien
- Verfolgung (Trace) der Code-Ausführung und Subroutinen-Argumente
- Ausführung eigener Befehle (Perl-Code) vor und nach jeder Programmcode-Zeile
- Änderung von Variablen und Code während der Ausführung



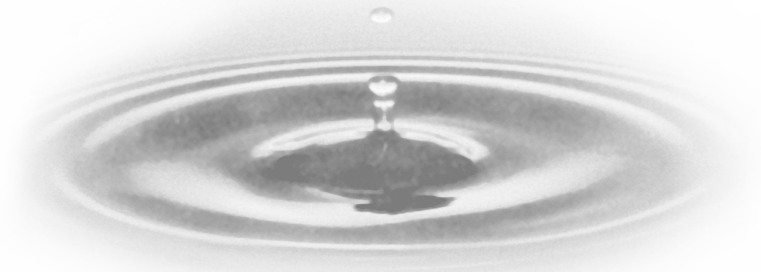
Perl Debugger Features II

- Haltepunkte
 - an jeder beliebigen ausführbaren Programmcodezeile
 - am Einstieg in ein Unterprogramm
 - bei Variablenmodifikation
 - bei selbst bestimmten Bedingungen



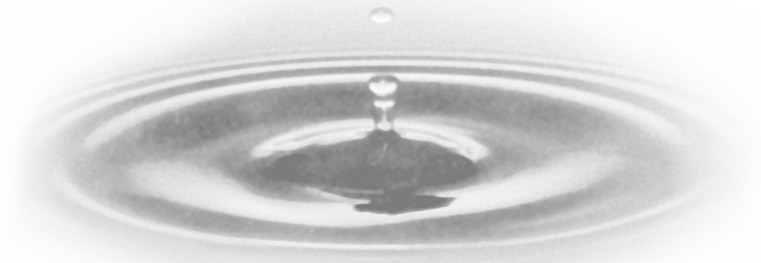
Perl Debugger Features III

- Verfolgung geforkter Programme - (abhängig vom Betriebssystem)
- Rudimentärer Support für Threads
- Integration mit Apache Web Server (mod_perl)



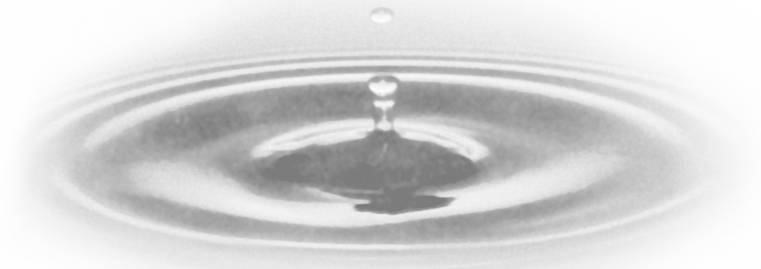
Perl Debugger Features IV

- Speichern und erneutes Ausführen der Debugger-Sessions
- History
- Suche mit regulären Ausdrücken
- Flexibel konfigurierbar (Customizable)
- Graphical User Interfaces



Risiken und Nebenwirkungen I

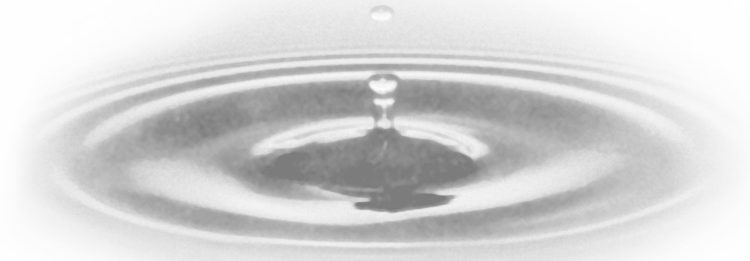
Es gibt subtile Änderungen, die der Debugger bei jedem Code vornimmt. Diese Änderungen gehen üblicherweise unbemerkt durch. Allerdings gibt es ganz seltene Fälle, in denen die Änderungen durch den Debugger das Debuggen sogar erschweren.



Risiken und Nebenwirkungen II

- **Heisenbugs:** Fehler, die verschwinden, wenn man versucht, sie zu debuggen.
- **Schrödingbugs:** Fehler, die nur auftauchen, wenn man versucht, etwas anderes zu debuggen.
- **Mandelbugs:** komplexe Fehler, die sich mehr und mehr ins Chaotische steigern, je genauer man hinsieht.

Quelle: Damian Conway: Perl - Best Practices

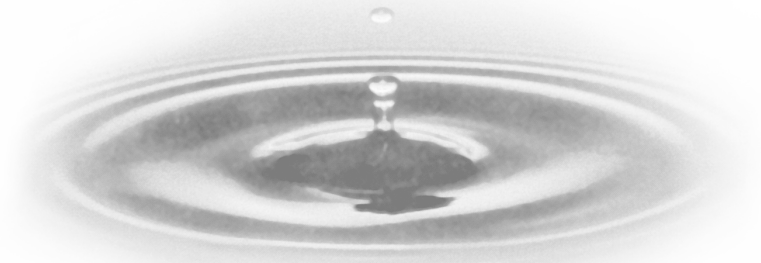


Risiken und Nebenwirkungen III

Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.

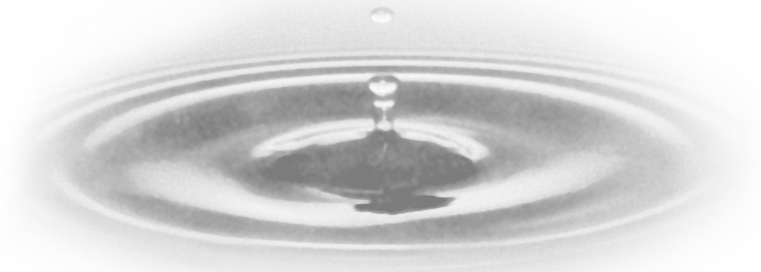
Brian W. Kernighan

<http://en.wikiquote.org/wiki/Programming>



Methoden, die helfen den Einsatz des Debuggers zu vermeiden

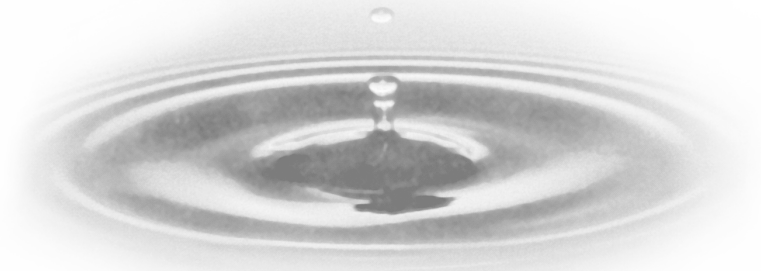
- use strict
- use warnings
- use diagnostics
- print oder (besser) warn if \$debug
- **Logging** (Log4perl)
- Code Review
- Test::* Suite
- Perl::Critic
- Data::Dumper



Other People's Code

Apologies to Sartre, "Hell is other people's code."

Daniel Allen: Unraveling Code with the Debugger
<http://www.perl.com/lpt/a/2006/04/06/debugger.html>



Debugger benutzen

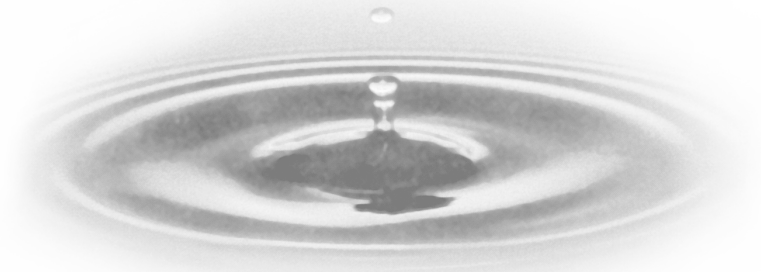
perl -d programm [argumente]

```
perl -d programm.pl arg0 arg1 arg2
```

```
perl -d programm.cgi name='X' param2='Y'
```

oder

Interaktiver Modus: `perl -d -e 0`



Interaktiver Modus

```
perl -d -e 0
```

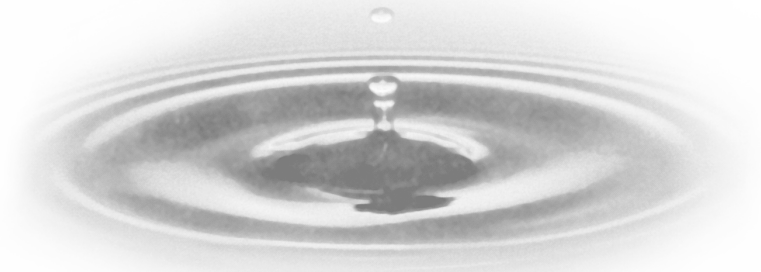
```
Loading DB routines from perl5db.pl  
version 1.3
```

```
Editor support available.
```

```
Enter h or `h h' for help, or `perldoc  
perldebug' for more help.
```

```
main::(-e:1):    0
```

```
DB<1>
```



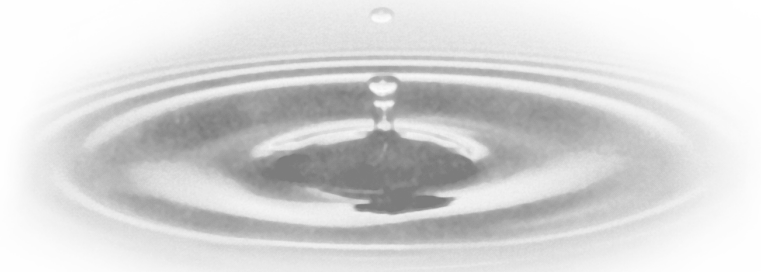
Hilfe

```
DB<1> h
```

```
# Zeigt alle verfügbaren Kommandos an
```

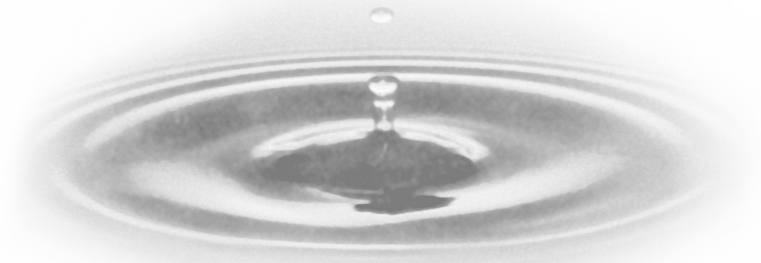
```
DB<1> h Kommando
```

```
# Zeigt eine kurze Hilfe zum Kommando an
```



Debugger als Perl-Shell

```
DB<1> print "Hallo Welt"
Hallo Welt
DB<2>print $^O
MSWin32
DB<3>print 6 * 7
42
DB<4>print join("\n", @INC)
C:/Perl/site/lib
C:/Perl/lib
.
DB<5>q
```

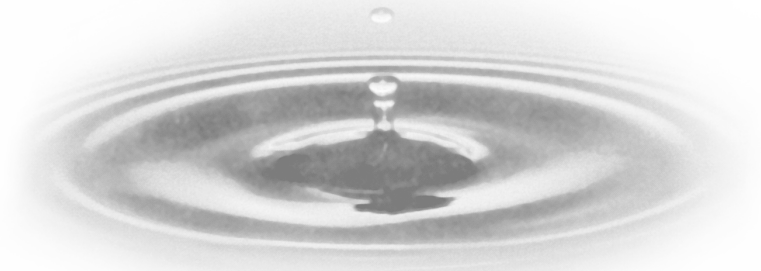


Continuation Lines \

```
DB<1> foreach my $counter ( 1..4 ) { \  
cont: print "$counter\n" \  
cont: }
```

1
2
3
4

```
DB<2>q
```



Variableninhalte anzeigen (p)

```
DB<1> $scalar = 'text';
```

```
DB<2> @array = qw(eins zwei drei)
```

```
DB<3> %hash = ( 'a' => '1', b=> '2' )
```

```
DB<4> p $scalar
```

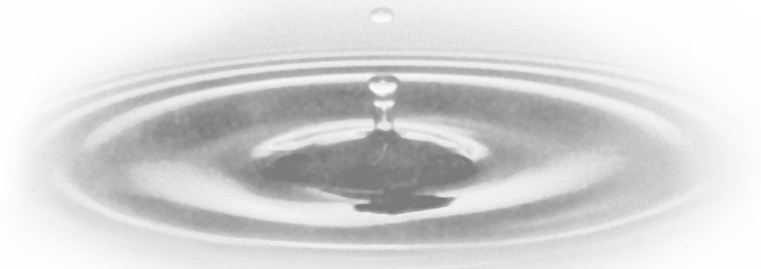
```
text
```

```
DB<5> p @array
```

```
einszweidrei
```

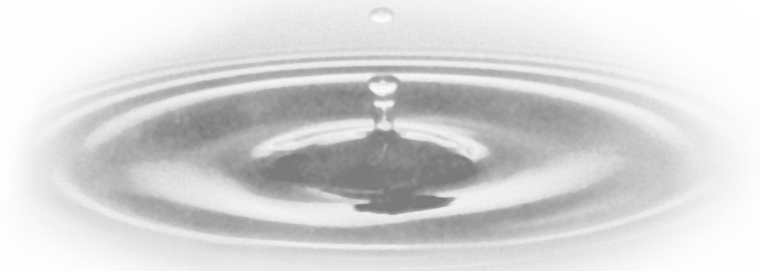
```
DB<6> p %hash
```

```
a1b2
```



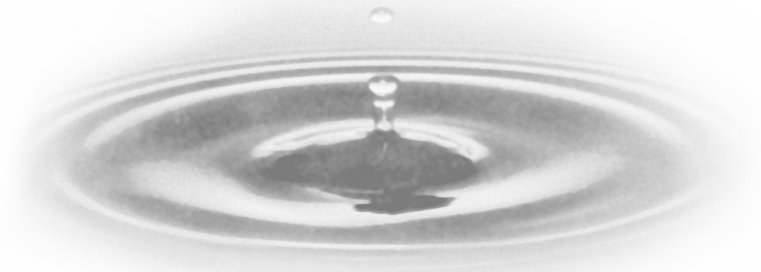
dump (x Variable)

```
DB<7> x $scalar
0  'text'
DB<8> x @array
0  'eins'
1  'zwei'
2  'drei'
DB<9> x %hash
0  'a'
1  1
2  'b'
3  2
```



dump (x \Variable)

```
DB<10> x \@array
0  ARRAY(0x1e22c68)
    0  'eins'
    1  'zwei'
    2  'drei'
DB<11> x \%hash
0  HASH(0x1e22d40)
    'a' => 1
    'b' => 2
```

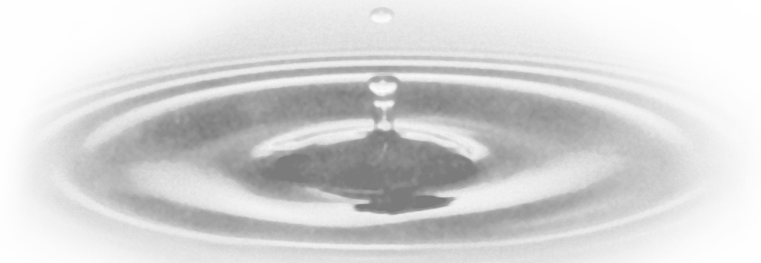


Datenstrukturen: Array of Arrays

```
DB<1> @AoA = ( [2, 3], [4, 5, 7], [0] );
```

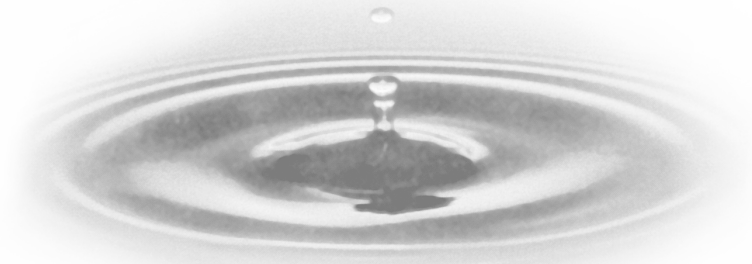
```
DB<2> x \@AoA
```

```
0  ARRAY(0x1b4de2c)
   0  ARRAY(0x182ec00)
     0  2
     1  3
   1  ARRAY(0x1e22cb0)
     0  4
     1  5
     2  7
   2  ARRAY(0x1e22d4c)
     0  0
```



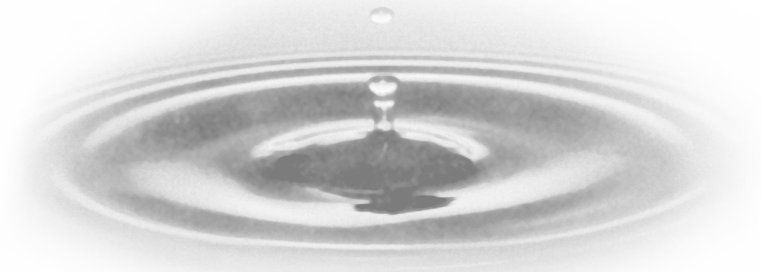
Datenstrukturen: Hash of Arrays

```
DB<1> %HoA = ( \
cont: flintstones => [ "fred", "barney" ], \
cont: jetsons => [ "george", "jane",
"elroy" ], )
DB<2> x \%HoA
0  HASH(0x1b4de2c)
   'flintstones' => ARRAY(0x1e22b48)
       0  'fred'
       1  'barney'
   'jetsons' => ARRAY(0x1e22bb4)
       0  'george'
       1  'jane'
       2  'elroy'
```



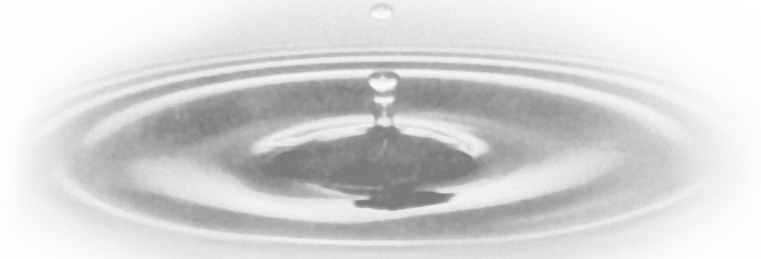
Datenstrukturen: Hash of Hashes I

```
DB<1> %HoH = ( \
cont:  flintstones => { \
cont:  lead        => "fred", \
cont:  pal         => "barney", }, \
cont:  jetsons     => { \
cont:  lead        => "george", \
cont:  wife        => "jane", \
cont:  "his boy"   => "elroy", }, )
```



Datenstrukturen: Hash of Hashes II

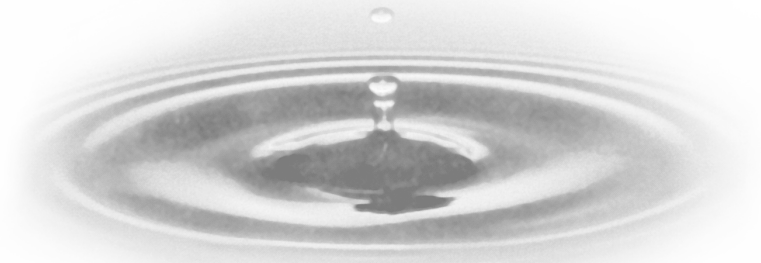
```
DB<2> x \%HoH
0  HASH(0x1df2cb0)
    'flintstones' => HASH(0x18fec00)
        'lead' => 'fred'
        'pal' => 'barney'
    'jetsons' => HASH(0x1df2db8)
        'his boy' => 'elroy'
        'lead' => 'george'
        'wife' => 'jane'
```



Vererbungsbaum (i)

i Class

zeigt den Vererbungsbaum (Inheritance Tree) für die Klasse oder das Paket Class an.

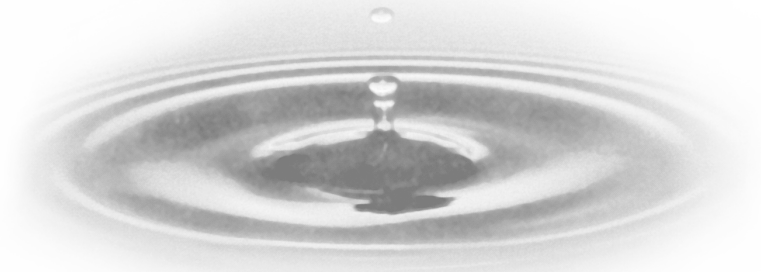


Beispiel inheritance

```
DB<1> use FileHandle
```

```
DB<2> i FileHandle
```

```
FileHandle 2.01, IO::File 1.14, IO::Handle  
1.27, IO::Seekable 1.1, Exporter 5.60
```



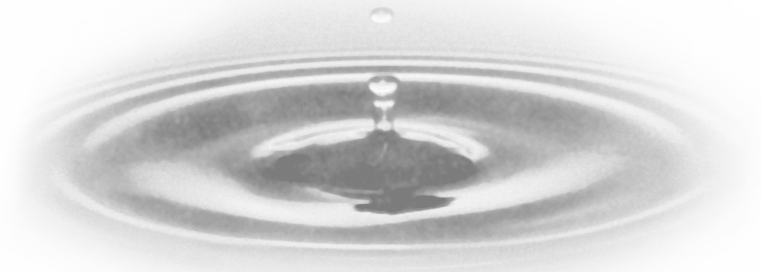
Sessions speichern und laden

save filename

speichert die aktuelle Session in der Datei filename

source filename

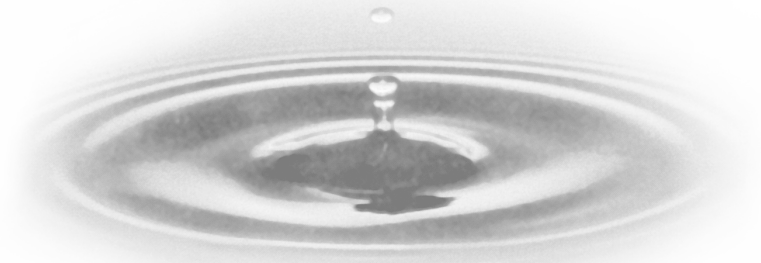
lädt eine (gespeicherte) Session aus der Datei filename und führt die Befehle aus



Restart (R)

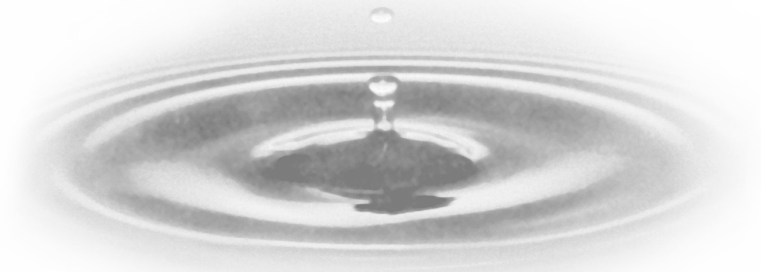
Sobald das Programm einmal vollständig durchgelaufen ist, ist ein Neustart des Programms im Debugger notwendig. Intern verwendet der Perl Debugger (`perl5db.pl`) dazu die Funktion `exec()`.

Mit ActiveState-Perl 5.8.x unter Windows funktioniert das leider nicht (http://bugs.activestate.com/show_bug.cgi?id=43344).



Beispielprogramm

```
#!/usr/bin/perl
use warnings;
use strict;
foreach my $number (1..2) {
    &display($number);
}
sub display {
    my $number = shift;
    $number = sprintf("*** %02d ***\n", $number);
    print $number;
}
```



Beispielprogramm verwenden

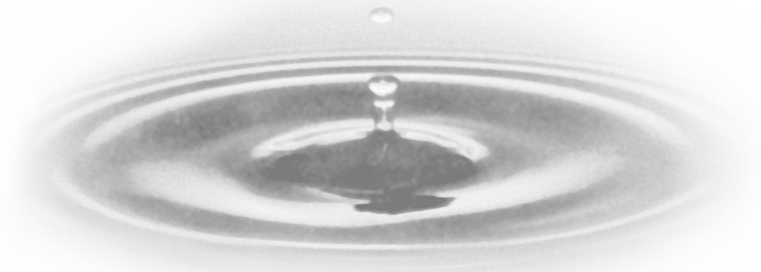
```
perl -d sample.pl
```

```
Loading DB routines from perl5db.pl  
version 1.3
```

```
Editor support available.
```

```
Enter h or `h h' for help, or `perldoc  
perldebug' for more help.
```

```
main::(sample.pl:4) :      foreach my $number  
    (1..2) {  
        DB<1>
```



Blättern - List Code

l (kleines el)

l [min+incr|min-max|line|subname|\$var]

Vorwärtsblättern mit 'l' (ohne Parameter)

Zurückblättern mit '-'

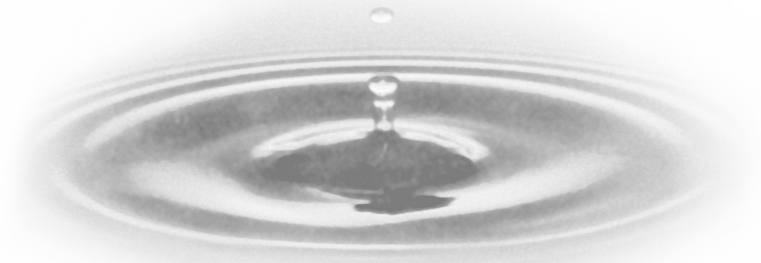
Doppelpunkte (:) zeigen an, wo ein Breakpoint oder eine Aktion gesetzt werden kann

==> zeigt die aktuelle Position im Programm an



Beispielprogramm auflisten

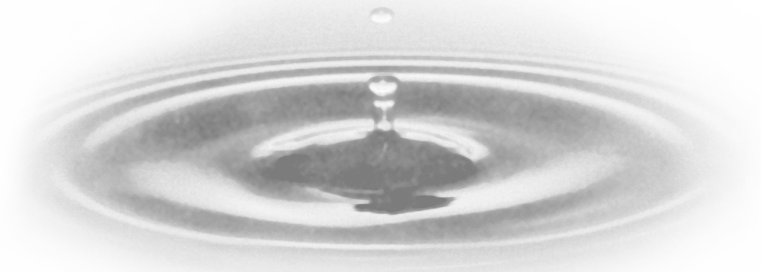
```
DB<1> 1 1-15
1      #!/usr/bin/perl
2:      use warnings;
3:      use strict;
4==>   foreach my $number (1..2) {
5:       &display($number);
6       }
7       sub display {
8:         my $number = shift;
9:         $number = sprintf("*** %02d
***\n", $number);
10:        print $number;
11      }
```



Subroutinen anzeigen (S)

S [[!] ~pattern]

S listet alle Unterprogramme auf, die dem regulären Ausdruck pattern (nicht !) entsprechen.



Beispiel Subroutinen anzeigen (S)

```
DB<1> S
```

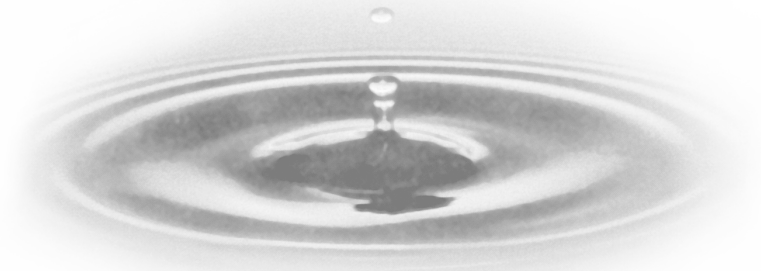
Alle verwendeten Unterprogramme werden angezeigt. Eine ziemlich lange Liste.

```
DB<1> S main
```

```
main::BEGIN
```

```
main::display
```

```
DB<2>
```



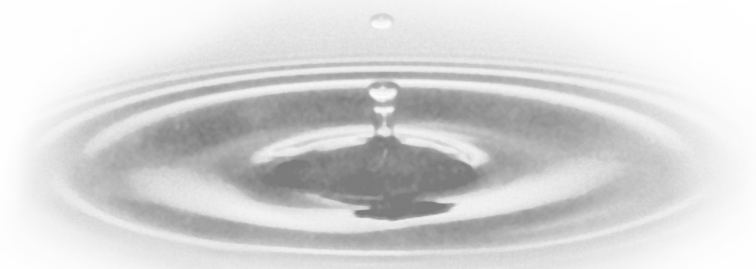
Step Into (s) und Step Over (n)

s [expr]

s führt die nächste Programmzeile aus und springt in die Unterprogramme (Step Into)

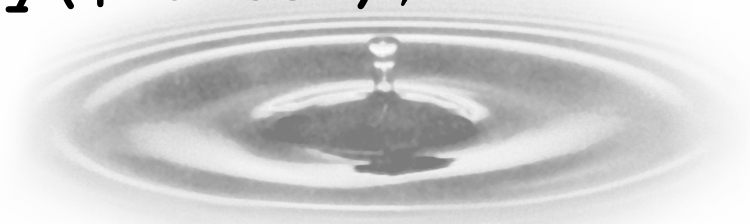
n [expr]

n führt die nächste Programmzeile aus und springt über die Unterprogramme (Step Over)



Beispiel Step Into

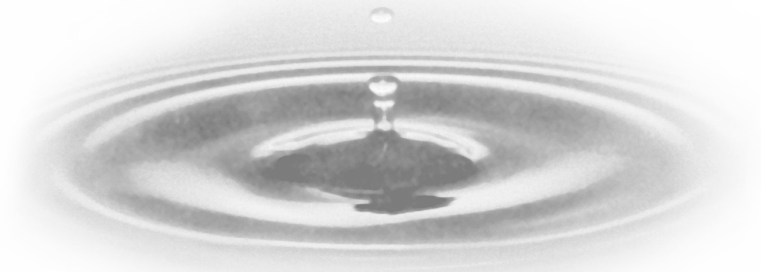
```
main::(sample.pl:4):  foreach my $number (1..2) {  
    DB<1> s  
main::(sample.pl:5):      &display($number);  
    DB<1> s  
main::display(sample.pl:8):  my $number = shift;  
    DB<1> s  
main::display(sample.pl:9):      $number =  
sprintf("*** %02d ***\n", $number);  
    DB<1> s  
main::display(sample.pl:10):    print $number;  
    DB<1> s  
*** 01 ***  
main::(sample.pl:5):      &display($number);  
    DB<1>
```



Beispielprogramm Step Over

```
main::(sample.pl:4) :      foreach my $number
(1..2) {
    DB<1> n
main::(sample.pl:5) :      &display($number) ;
    DB<1> n
*** 01 ***
main::(sample.pl:5) :      &display($number) ;
    DB<1> n
*** 02 ***

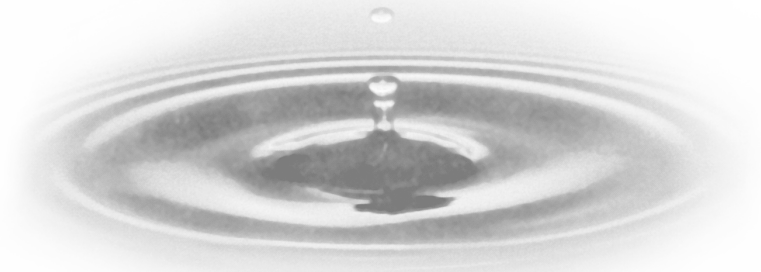
DB<1>
```



Aus Unterprogrammen aussteigen (r)

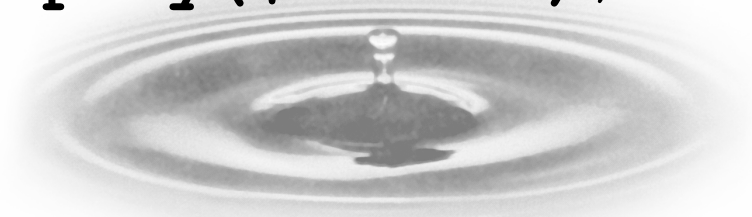
r

return kehrt aus Unterprogrammen zurück und zeigt den Rückgabewert an. Das ist beispielsweise in Verbindung mit s (Step Into) recht nützlich.



Beispiel return

```
main::(sample.pl:4) :      foreach my $number
(1..2) {
    DB<1> s
main::(sample.pl:5) :      &display($number) ;
    DB<1> s
main::display(sample.pl:8) :      my $number =
shift;
    DB<1> r
*** 01 ***
void context return from main::display
main::(sample.pl:5) :      &display($number) ;
    DB<1>
```

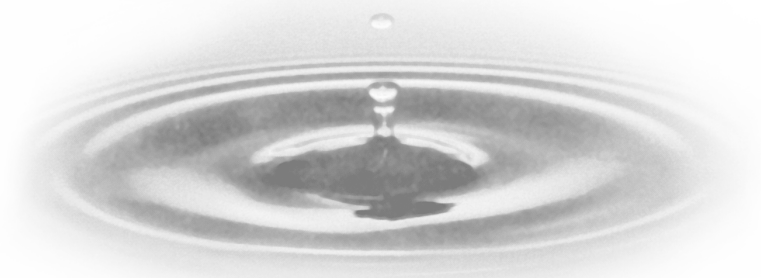


Continue

c [line|sub]

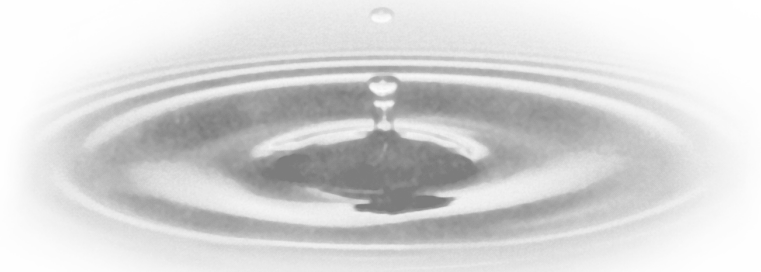
continue führt den Code bis zur angegebenen Zeile oder bis zum angegebenen Unterprogramm aus.

Ohne Parameter wird der Code von der aktuellen Zeile bis zum nächsten Breakpoint oder bis zum Ende des Programms ausgeführt.



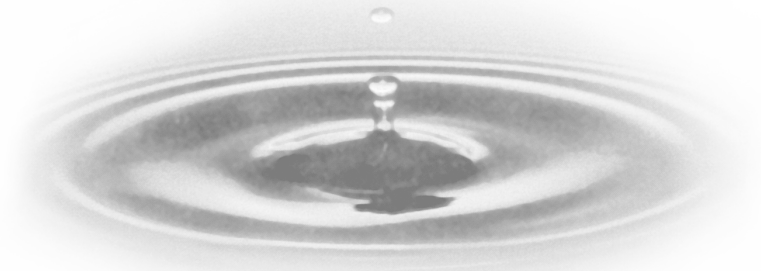
Beispiel continue

```
main::(sample.pl:4) :      foreach my $number
(1..2) {
    DB<1> c display
main::display(sample.pl:8) :      my $number =
shift;
    DB<2> c
*** 01 ***
*** 02 ***
    DB<3>
```



Aktiv werden

Actions
Breakpoints
Watches

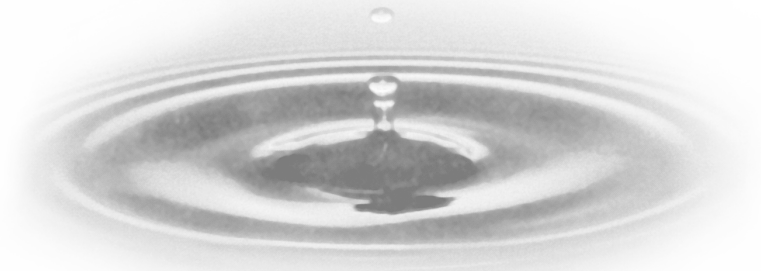


Breakpoints, Watchpoints und Actions anzeigen (L)

L [a|b|w]

Listet die gesetzten Actions (a), Breakpoints (b) und Watches (w).

Ohne Parameter werden alle Aktionen, Breakpoints und Watches angezeigt.



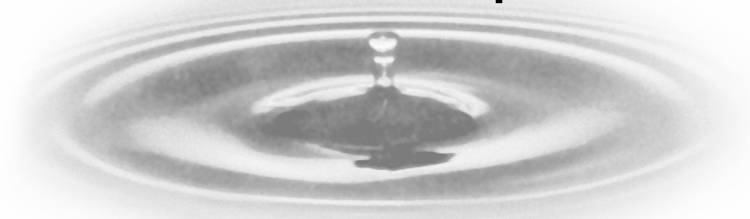
Breakpoints (Haltepunkte)

b [line|sub [condition]]

Setzt einen Haltepunkt in der angegebenen Zeile bzw. vor Ausführung des Unterprogramms.
Zusätzlich kann eine Bedingung (beliebiger Perl-Code) festgelegt werden.

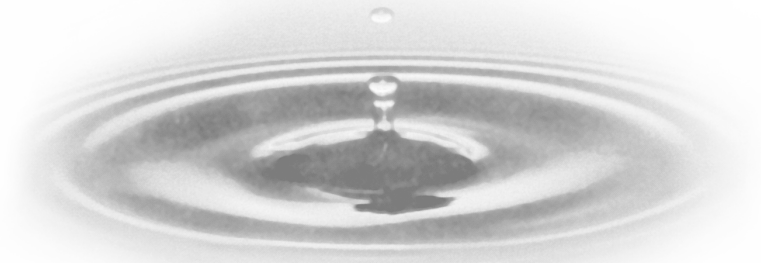
B (line|*)

Löscht den Haltepunkt in Zeile line oder alle Haltepunkte



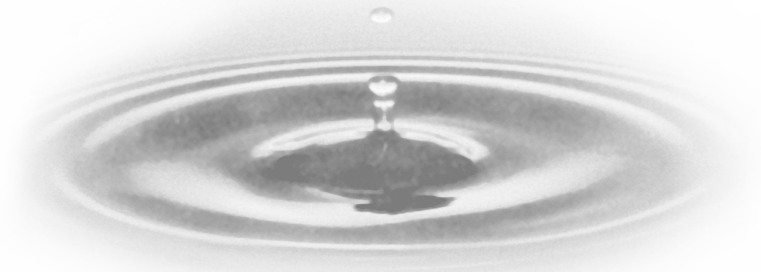
Beispiel Breakpoints I

```
DB<1> b display
DB<2> L
sample.pl:
8:      my $number = shift;
      break if (1)
DB<2> c
main::display(sample.pl:8) :      my $number =
shift;
DB<2> c
*** 01 ***
main::display(sample.pl:8) :      my $number =
shift;
DB<2> c
*** 02 ***
```



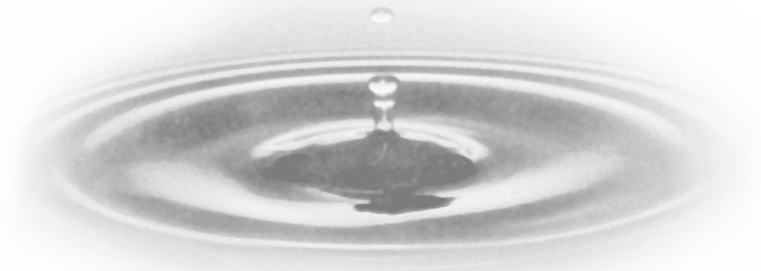
Code manipulieren an Breakpoints

Sobald ein Haltepunkt erreicht wird, stoppt der Debugger das Programm. Jetzt können die Variablen mit den bereits vorgestellten Methoden inspiziert und geändert werden.



Beispiel Breakpoints II

```
DB<2> b 5
DB<3> c
main::(sample.pl:5) :      &display($number) ;
    DB<3> x $number
0    1
    DB<4> $number = 42
    DB<5> c
*** 42 ***
main::(sample.pl:5) :      &display($number) ;
```



Actions

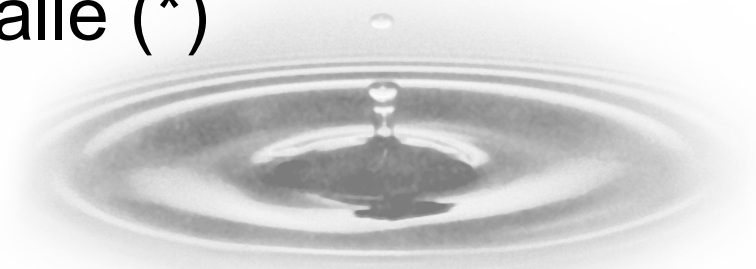
a [line] command [condition]

Setzt in Zeile line eine Aktion (beliebiger Perl-Code).
Zusätzlich kann eine Bedingung (beliebiger Perl-Code) festgelegt werden.

Achtung: Die Aktion wird **vor** der Ausführung der Zeile ausgeführt.

A [line|*]

Löscht die Action in Zeile line bzw. alle (*)



Beispiel Actions I

```
DB<1> a 5 print "A5a $number "; $number++;  
print "A5b $number\n"
```

```
DB<2> L
```

```
sample.pl:
```

```
5:      &display($number);
```

```
    action: print "A5a $number "; $number++;  
print "A5b $number\n"
```

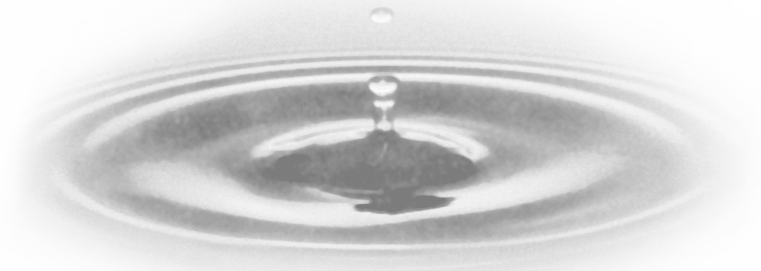
```
DB<2> c
```

```
A5a 1 A5b 2
```

```
*** 02 ***
```

```
A5a 2 A5b 3
```

```
*** 03 ***
```

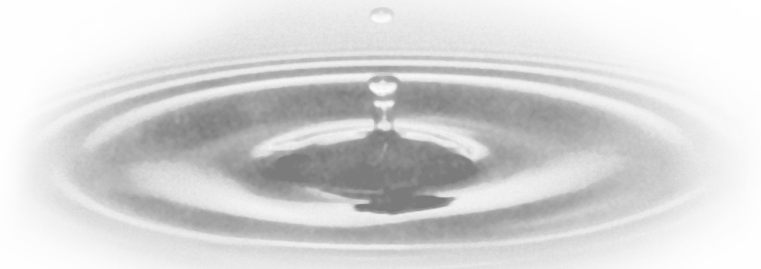


Beispiel Actions II

```
DB<1> 1 8
8:      my $number = shift;

DB<2> a 8 print "Parameter $number\n"

DB<3> c
Parameter
*** 01 ***
Parameter
*** 02 ***
```



Beispiel Actions III

```
DB<1> a 8 print "Parameter $_[0]\n"
```

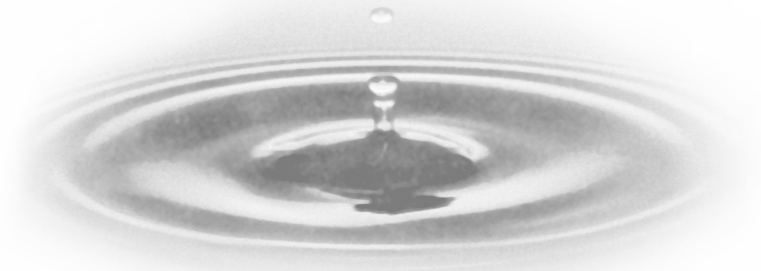
```
DB<2> c
```

```
Parameter 1
```

```
*** 01 ***
```

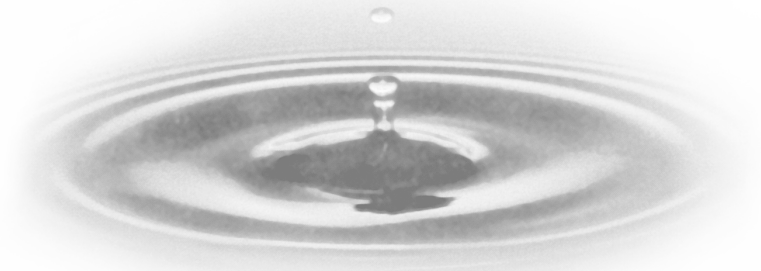
```
Parameter 2
```

```
*** 02 ***
```



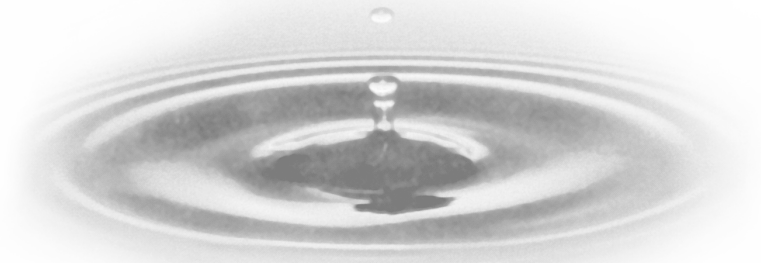
List Code revisited I

Sobald Haltepunkte oder Actions gesetzt sind, werden diese im Programmlisting durch ein b bzw. a nach der Zeilennummer gekennzeichnet.



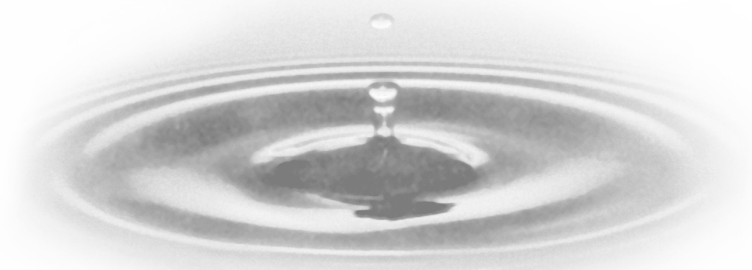
List Code revisited II

```
DB<1> b display
DB<2> l 2-10
2:      use warnings;
3:      use strict;
4==>   foreach my $number (1..2) {
5:       &display($number);
6       }
7       sub display {
8:b      my $number = shift;
9:      $number = sprintf("*** %02d ***\n",
$number);
10:     print $number;
```



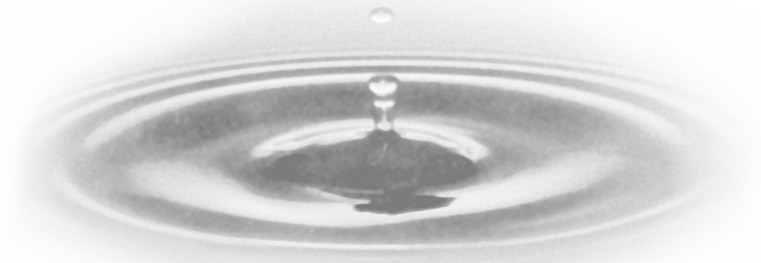
List Code revisited III

```
DB<3> a 5 print "A5a $number "; $number++;  
print "A5b $number\n"  
DB<4> 1 2-10  
2:      use warnings;  
3:      use strict;  
4==>   foreach my $number (1..2) {  
5:a      &display($number);  
6        }  
7      sub display {  
8:b      my $number = shift;  
9:      $number = sprintf("*** %02d ***\n",  
$number);  
10:     print $number;
```



List Code revisited IV

```
DB<5> a 8 print "A8a $number "; $number++;  
print "A8b $number\n"  
DB<6> 1 3-10  
3:      use strict;  
4==>    foreach my $number (1..2) {  
5:a      &display($number);  
6        }  
7        sub display {  
8:ba      my $number = shift;  
9:        $number = sprintf("*** %02d ***\n",  
$number);  
10:      print $number;
```



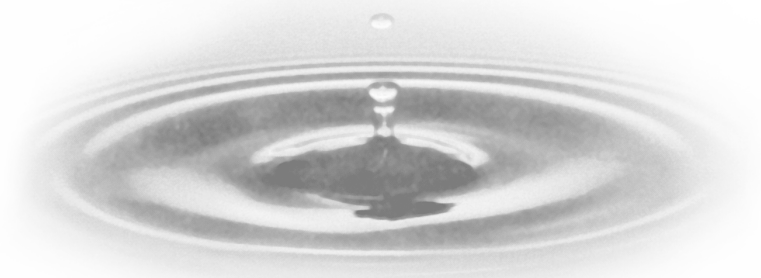
Variablen beobachten - watches

w [expr]

Setzt einen Beobachter für expr (beliebiger Perl-Code)

W (expr|*)

Löscht einen bzw. alle Beobachter.



Beispiel watches

```
DB<1> w $number
```

```
DB<2> c
```

```
Watchpoint 0:    $number changed:
```

```
    old value:    ''
```

```
    new value:    '1'
```

```
main::(sample.pl:5):    &display($number) ;
```

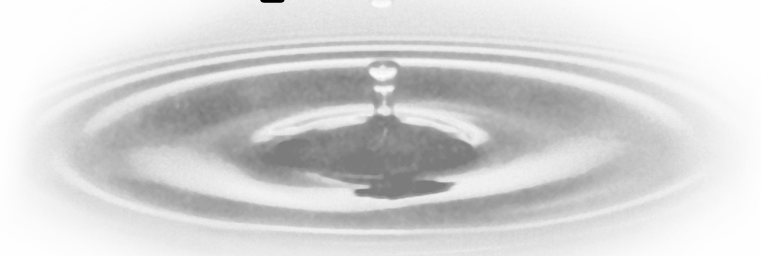
```
DB<2> c
```

```
Watchpoint 0:    $number changed:
```

```
    old value:    '1'
```

```
    new value:    ''
```

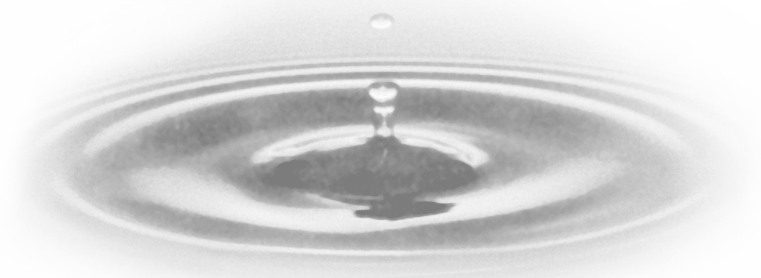
```
main::display(sample.pl:8):    my $number =  
shift;
```



pre-perl post-perl

pre-perl (<) wird vor Abarbeitung der aktuellen Zeile ausgeführt und wird **vor** dem Debugger-Prompt angezeigt.

post-perl (>) wird nach Abarbeitung der aktuellen Zeile ausgeführt und erscheint am Ende der Ausgabe des Debugger-Prompts.



Beispiel: pre-perl post-perl

```
DB<1> < print "PRE-Perl\n"
```

```
DB<2> > print "POST-Perl\n"
```

```
DB<3> >> print "N: $number\n" if $number
```

```
DB<4> n
```

POST-Perl

```
main::(sample.pl:5):          &display($number);
```

PRE-Perl

```
DB<4> n
```

POST-Perl

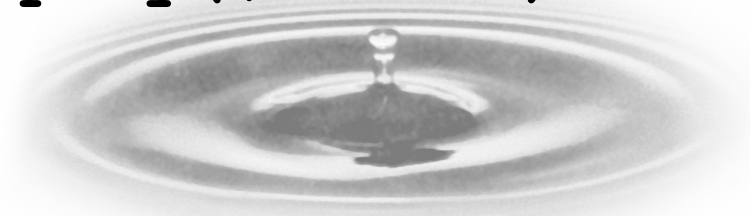
N: 1

*** 01 ***

```
main::(sample.pl:5):          &display($number);
```

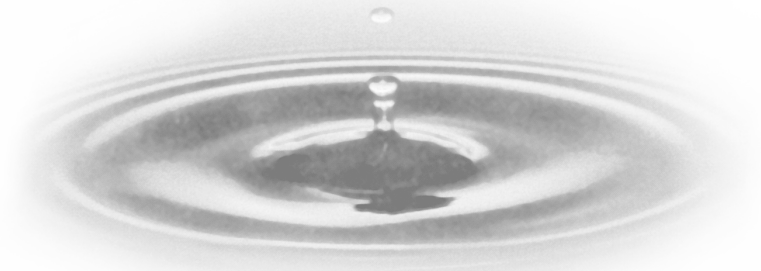
PRE-Perl

```
DB<4> n
```

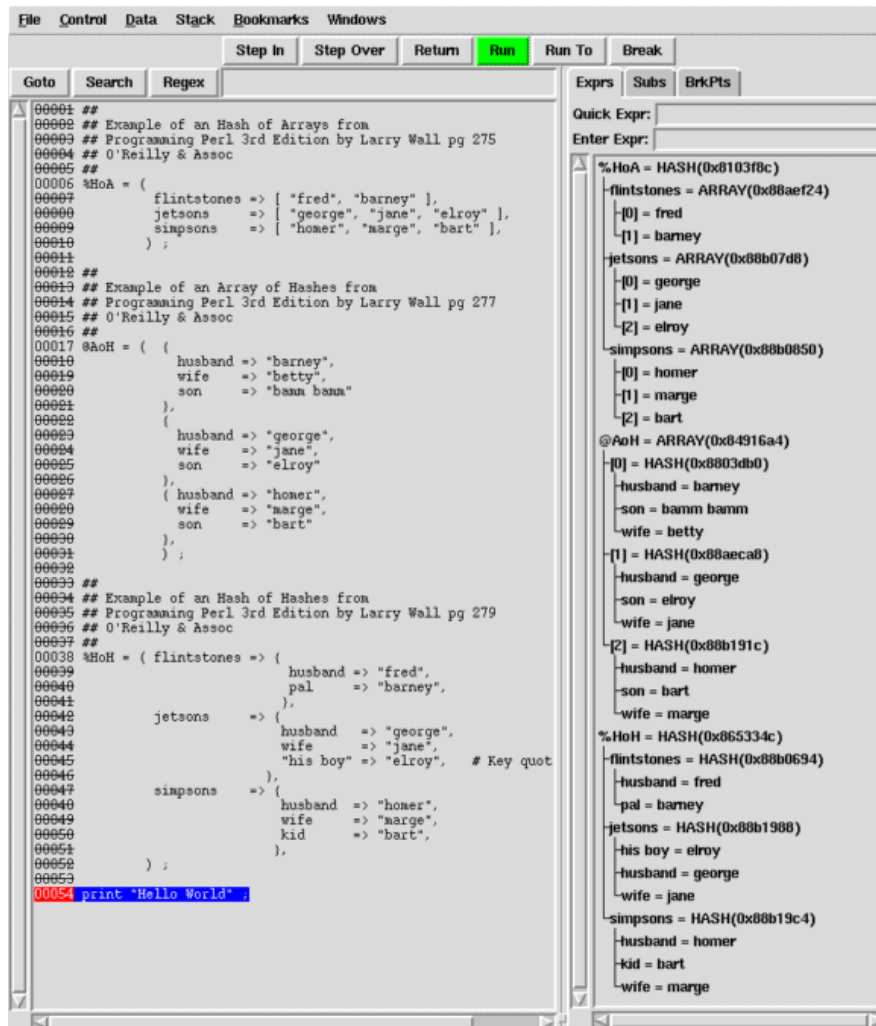


Graphical User Interfaces

Es geht auch graphisch!

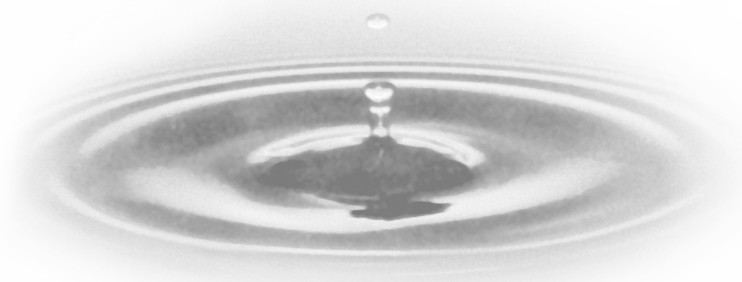


Graphical User Interfaces - ptkdb

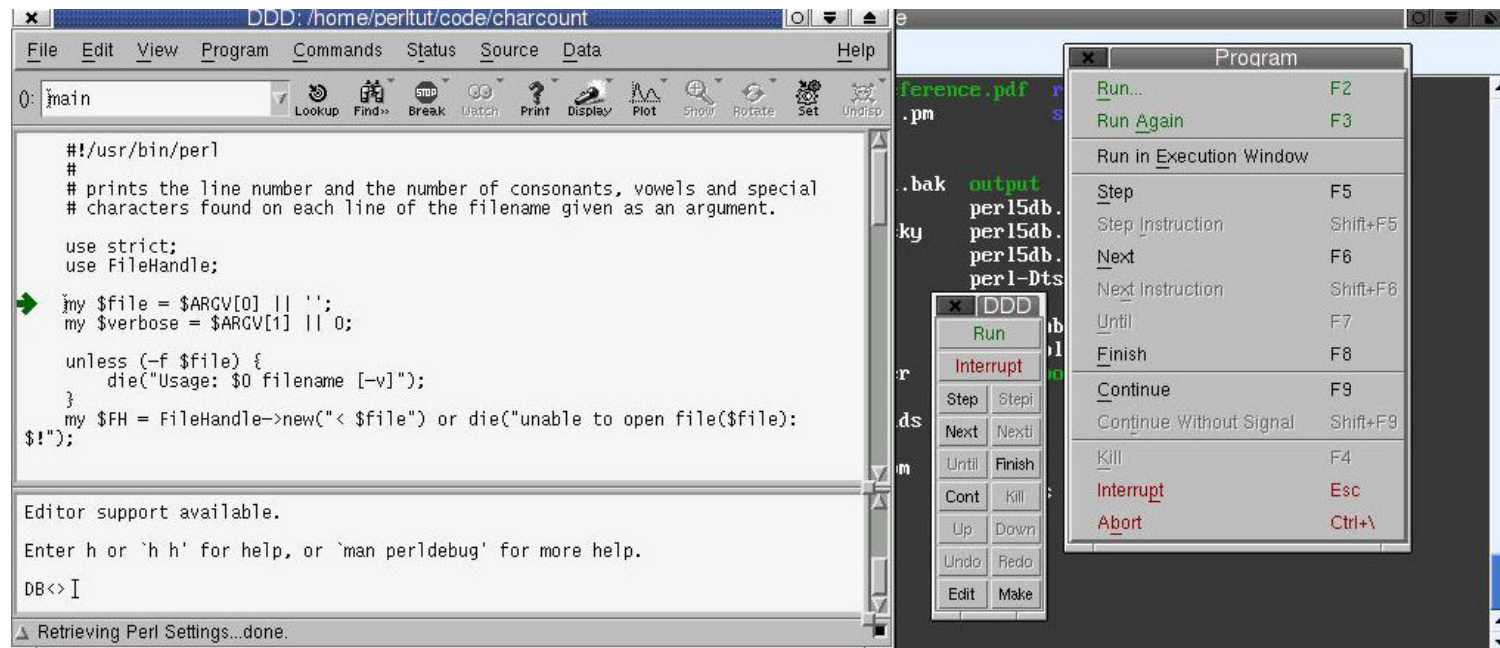


The screenshot shows the ptkdb Perl debugger interface. The main window displays a Perl script with line numbers 00001 to 00053. The script includes comments about examples from 'Programming Perl 3rd Edition' and defines several data structures: a Hash of Arrays (%HoA), an Array of Hashes (@AoH), and a Hash of Hashes (%HoH). The script ends with a print statement on line 00054: `print "Hello World";`. The right-hand pane shows the 'Exprs' tab, displaying the current state of variables in memory, such as `%HoA = HASH(0x8103f8c)` and its contents, `@AoH = ARRAY(0x84916a4)`, and `%HoH = HASH(0x865334c)` with its nested structure.

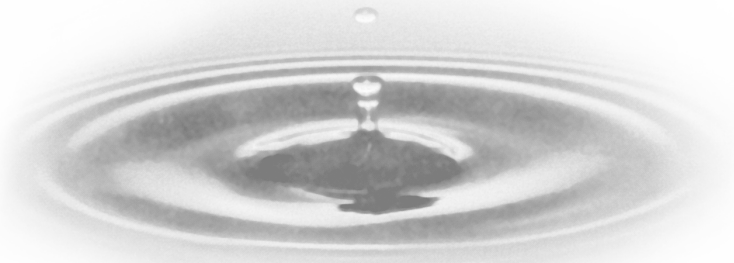
<http://ptkdb.sourceforge.net/>



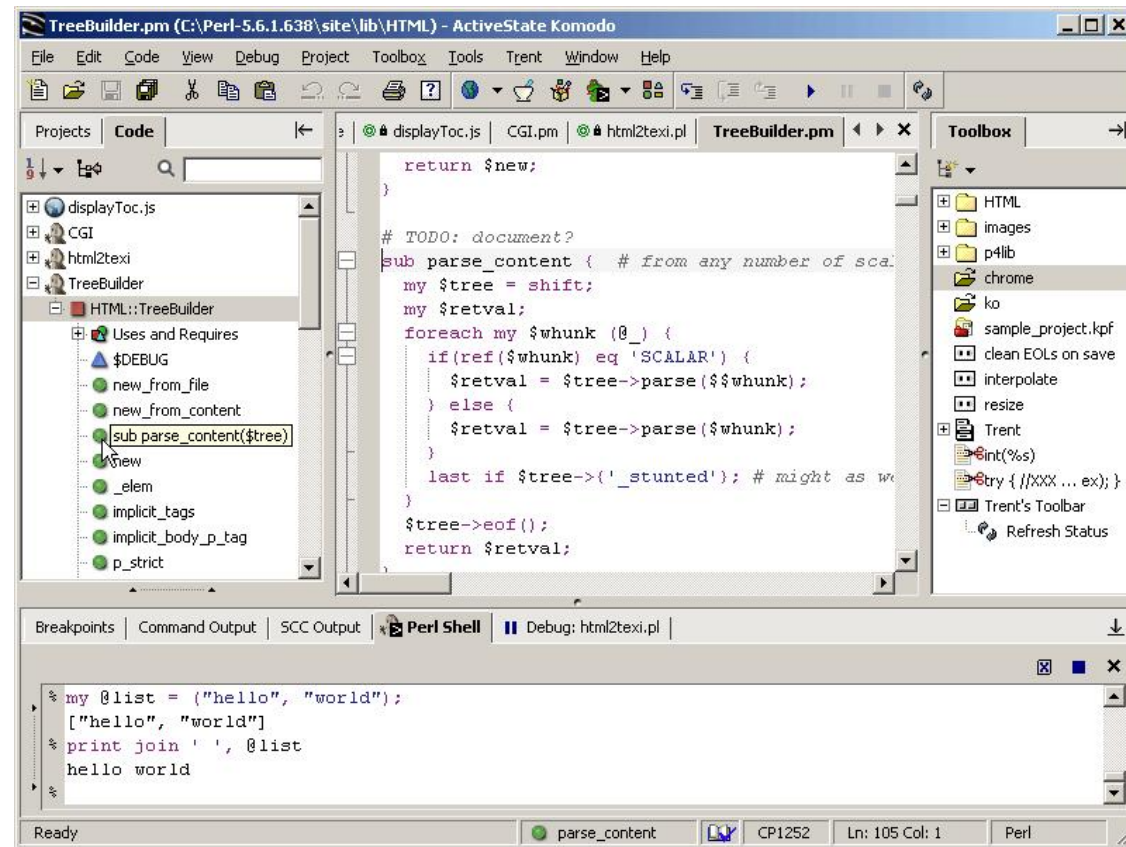
Graphical User Interfaces - ddd



<http://www.gnu.org/software/ddd/>

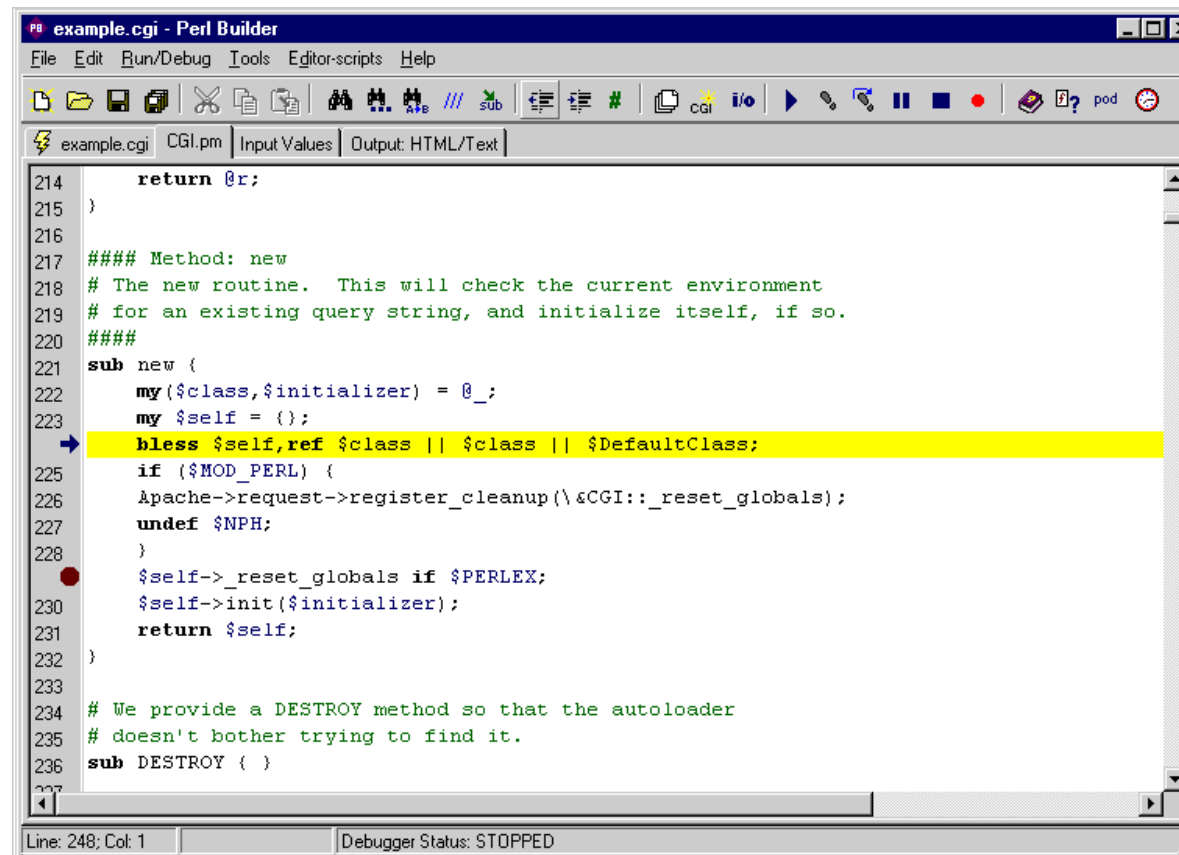


Graphical User Interfaces Active Perl Pro Studio



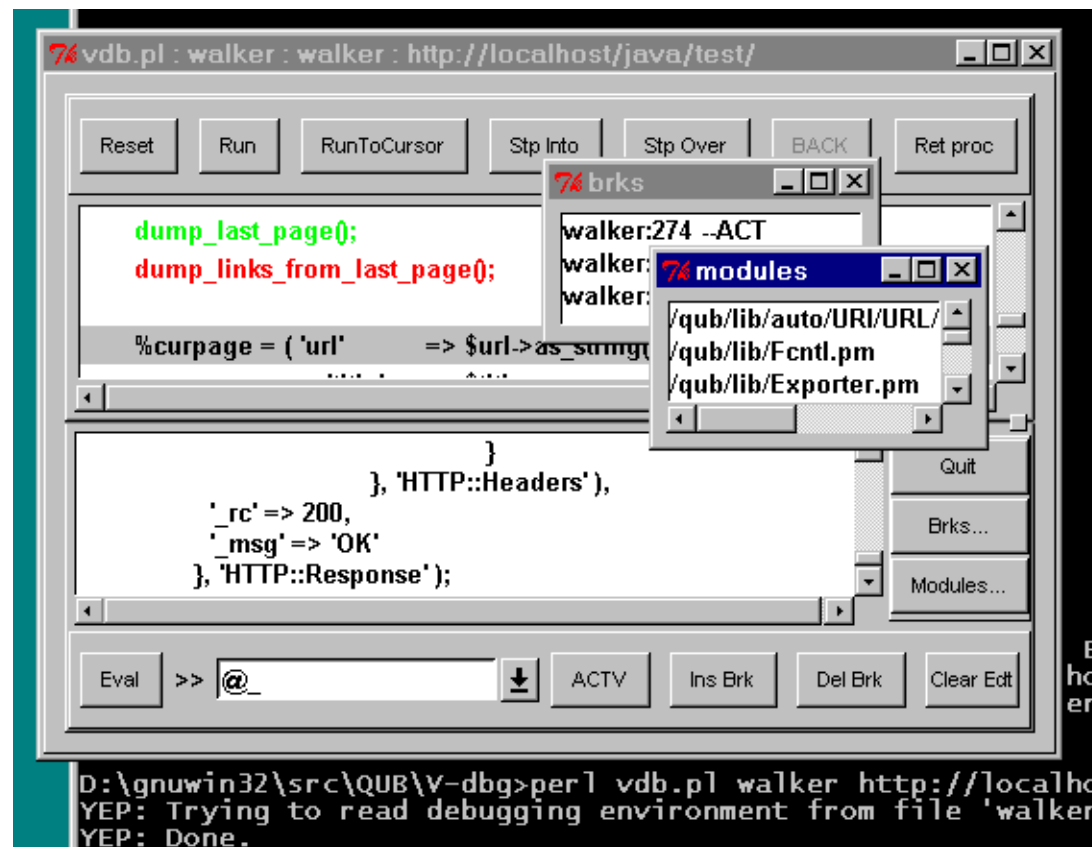
<http://www.activestate.com/Products/activeperlprostudio/>

Graphical User Interfaces Perl Builder



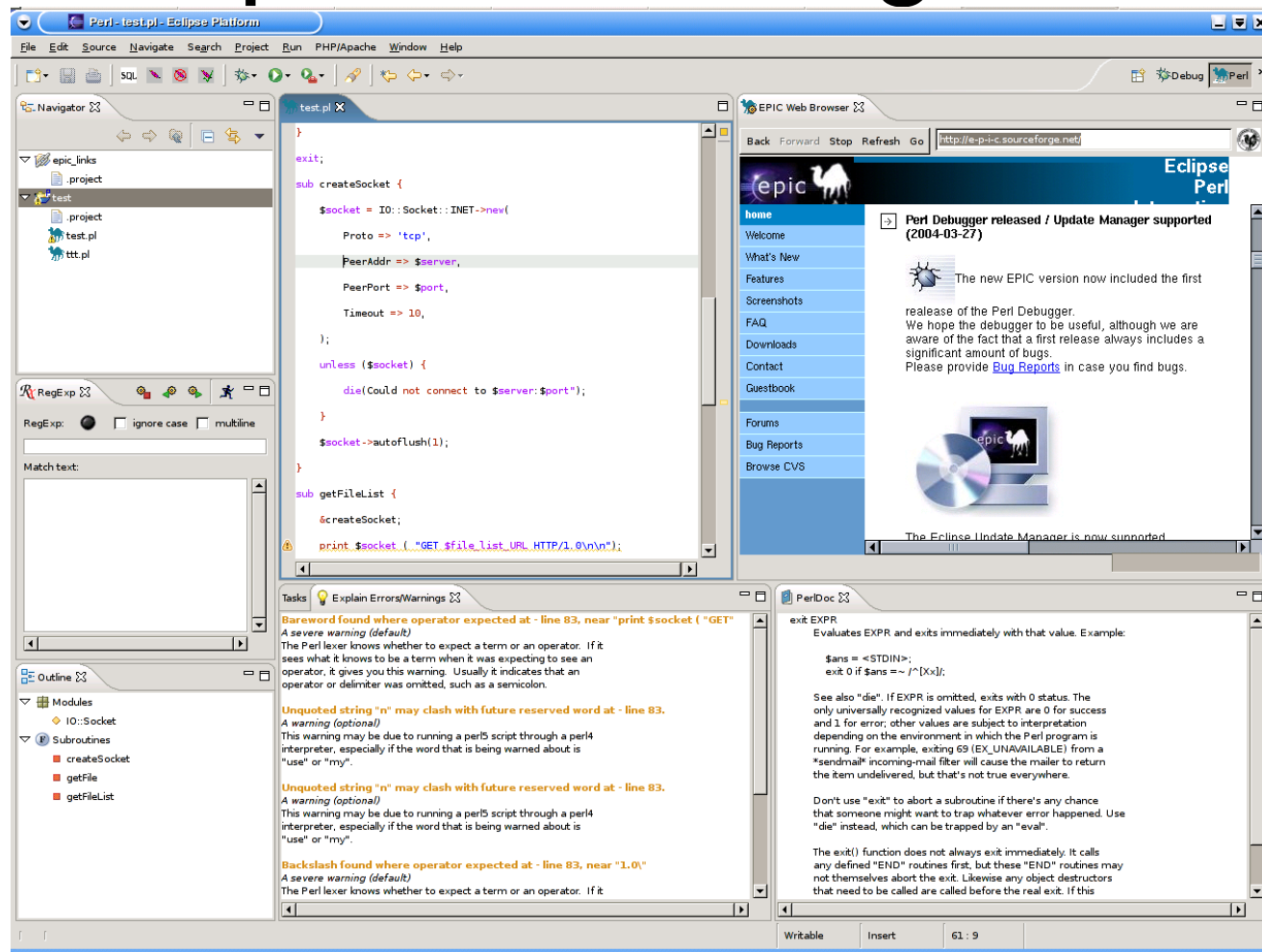
<http://www.solutionsoft.com/perl.htm>

Graphical User Interfaces vdb (Visual Perl Debugger)



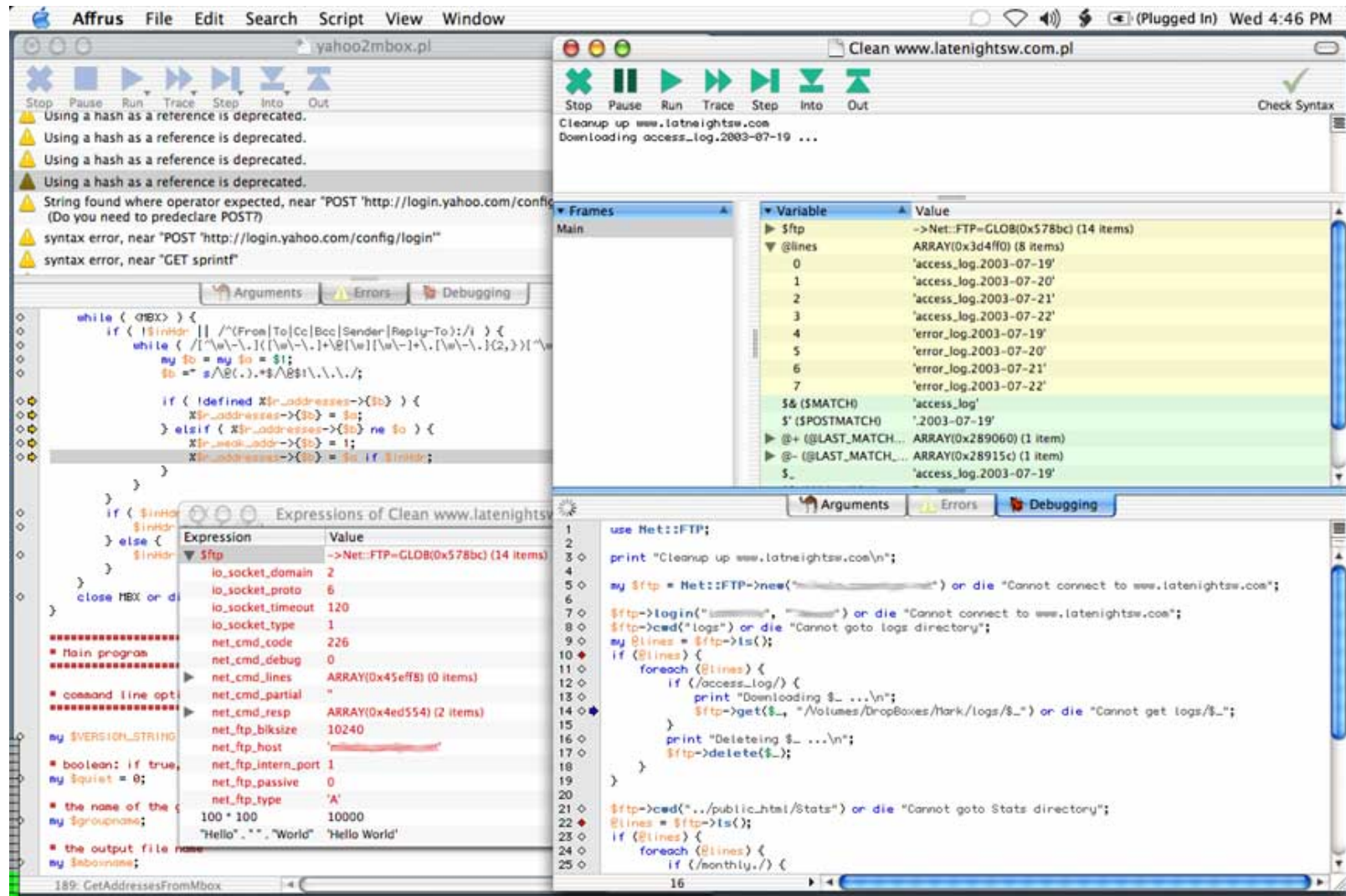
<http://www.qub.rinet.ru/group/our/vdb.htm>

Graphical User Interfaces Eclipse Perl Integration



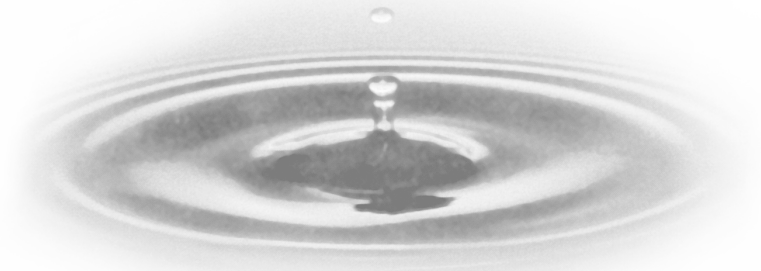
<http://e-p-i-c.sourceforge.net/>

Affrus

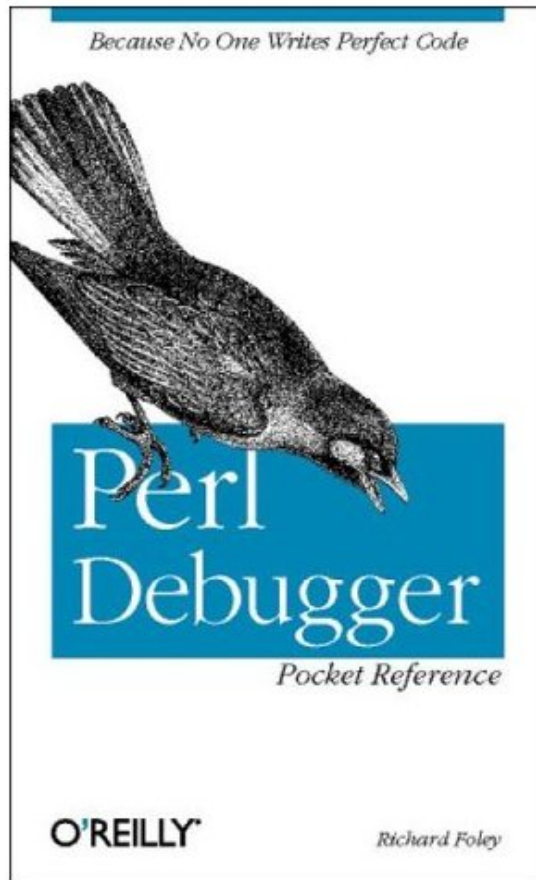


<http://www.latenightsw.com/affrus/>

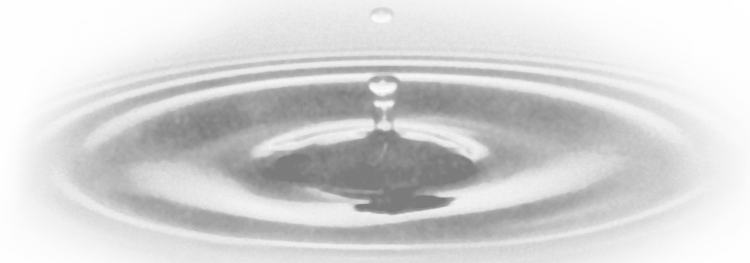
Literatur und Links



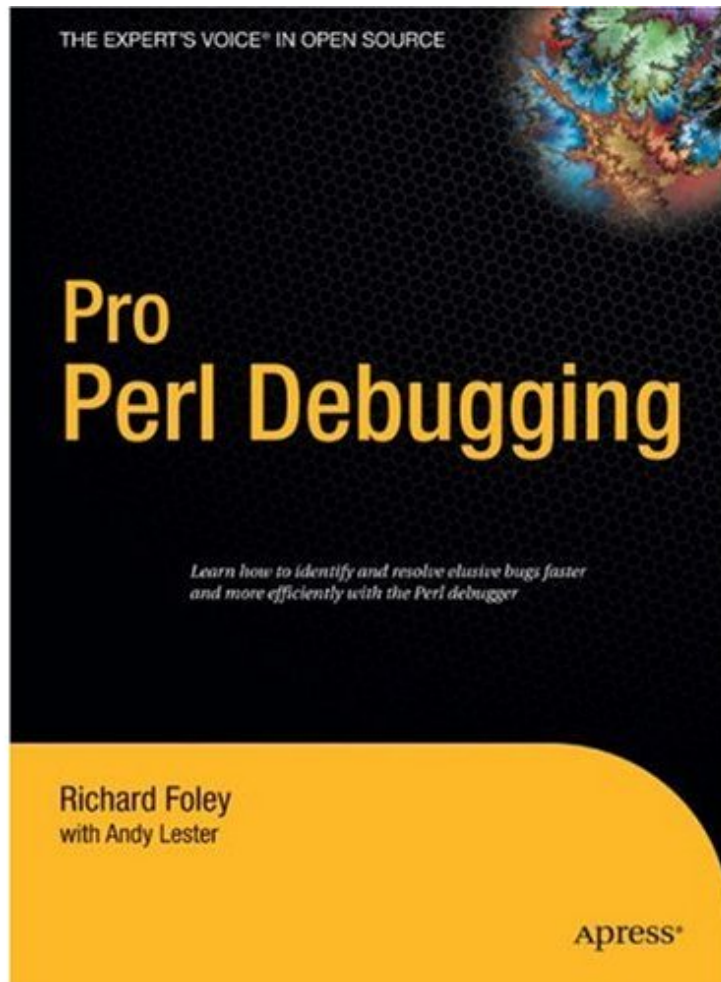
Literatur: Perl Debugger Pocket Reference



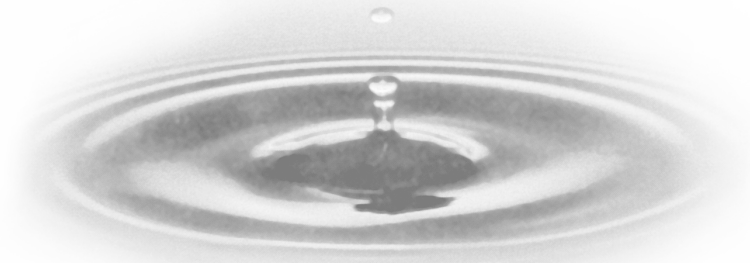
Foley, Richard:
Perl Debugger Pocket Reference.
(O'Reilly)
ISBN: 0596005032



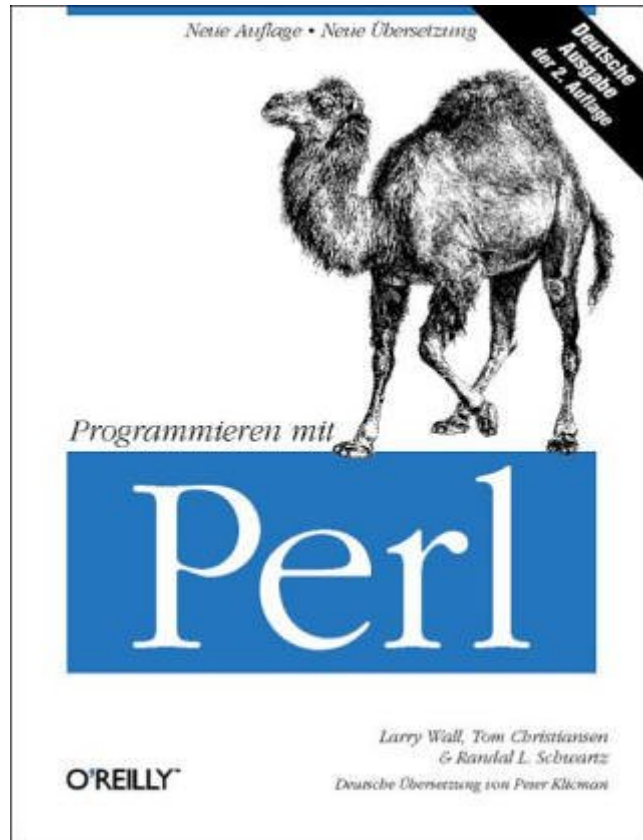
Literatur: Pro Perl Debugging



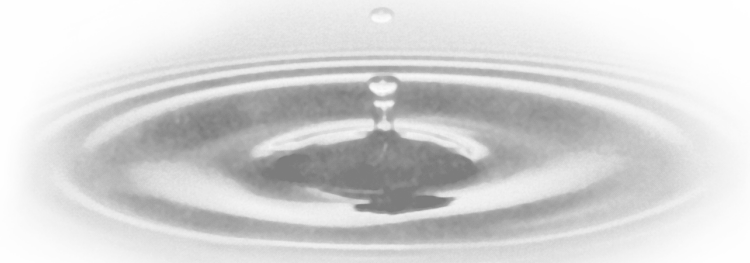
Foley, Richard,
Lester, Andy:
Pro Perl Debugging:
From Professional to Expert
(Apress)
ISBN: 1590594541



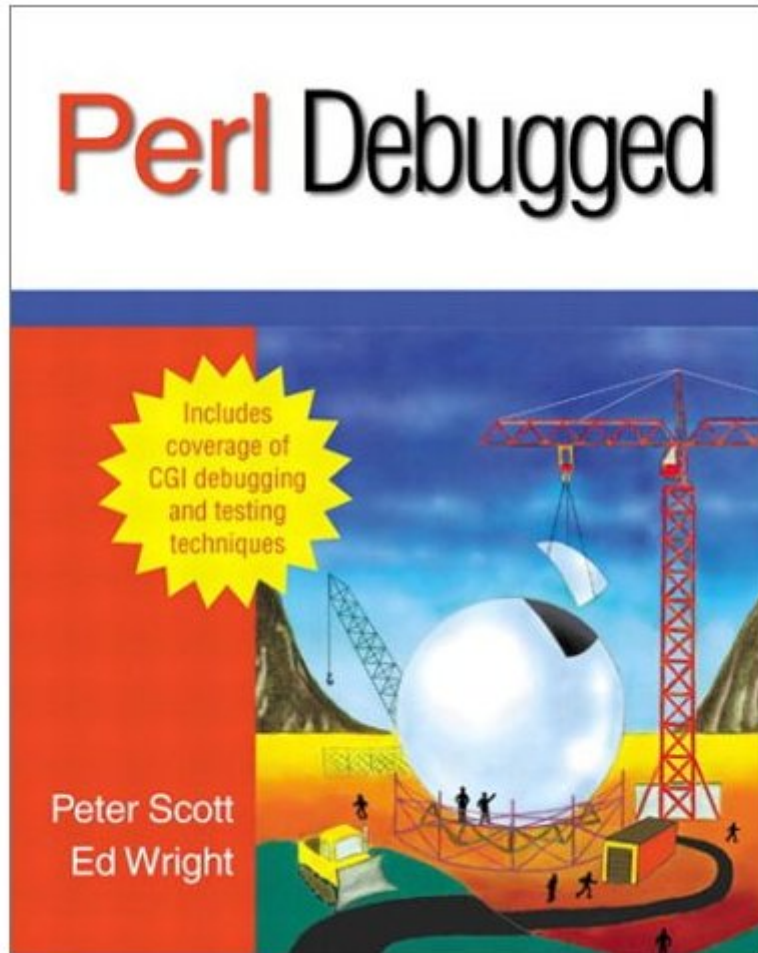
Literatur: Programmieren mit Perl



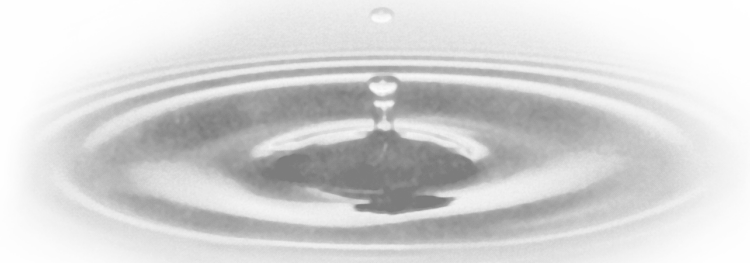
Larry Wall,
Tom Christiansen,
Jon Orwant,
Randal Schwartz:
Programmieren mit Perl
(O'Reilly)
ISBN: 3897211440
Kapitel 20.



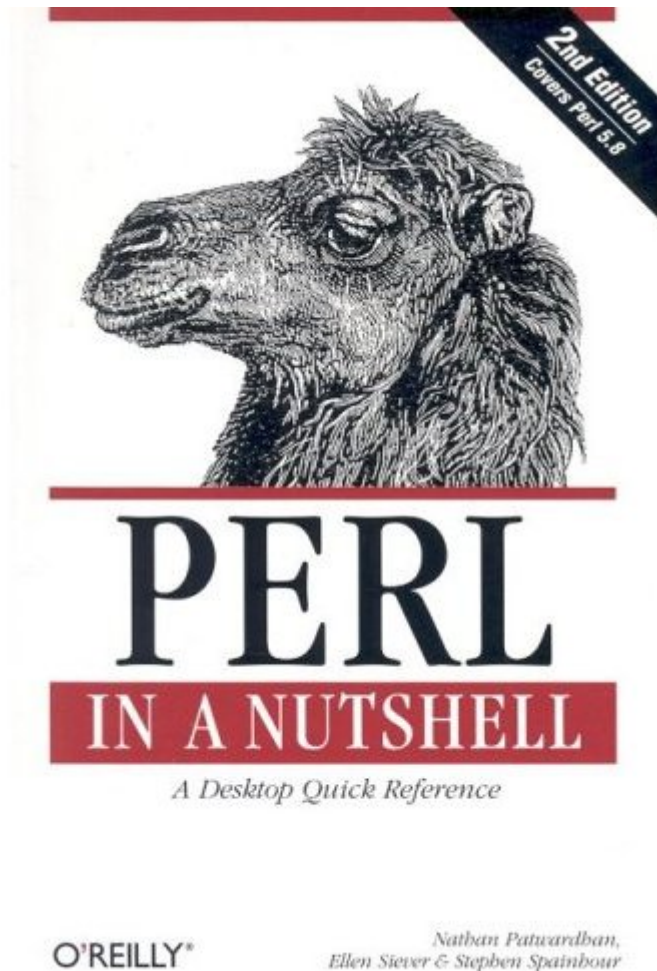
Literatur: Perl Debugged



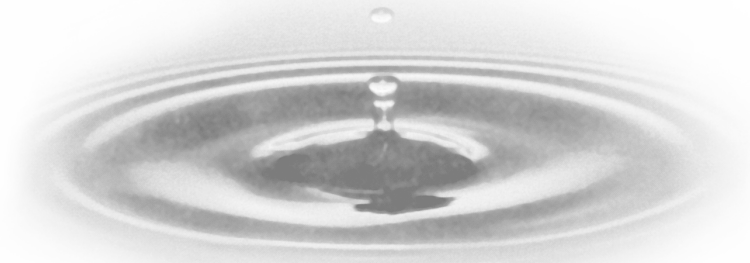
Peter Scott, Ed Wright:
Perl Debugged
ISBN: 02011700549
(Out of print)



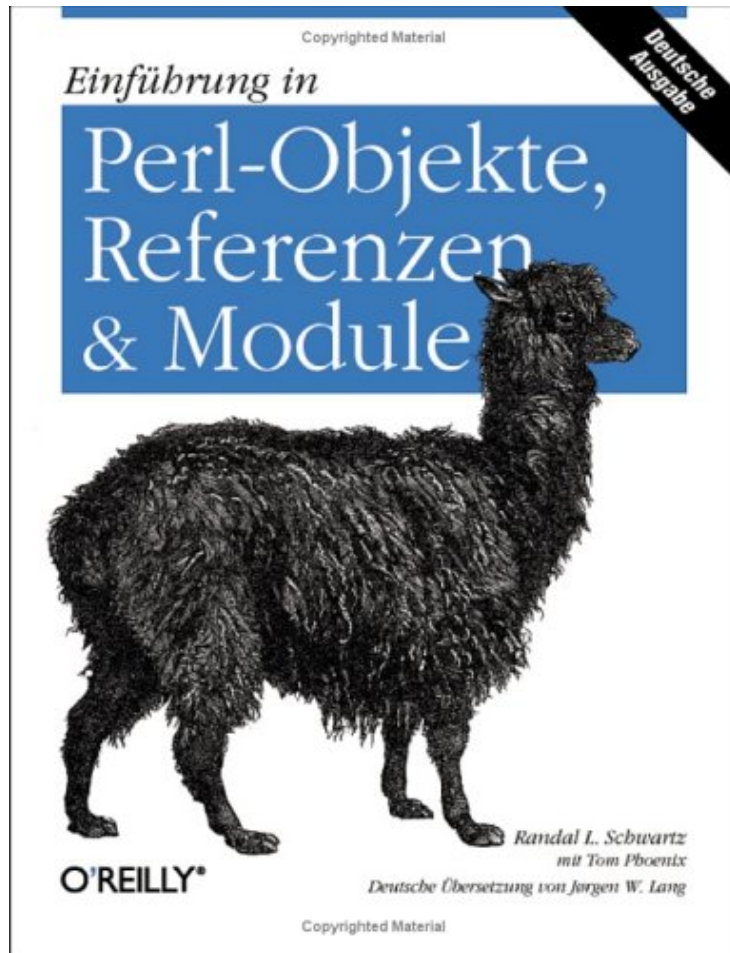
Literatur: Perl in a Nutshell



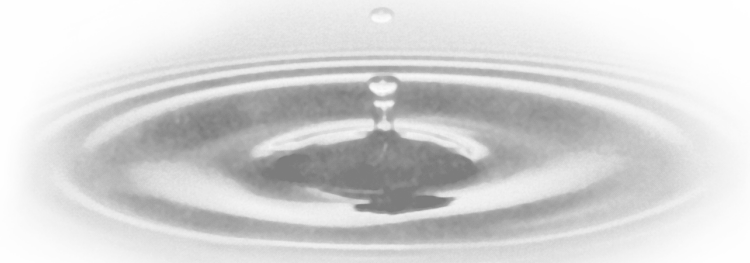
Nathan Patwardhan,
Ellen Siever,
Stephen Spainhour
Perl in a Nutshell
(O'Reilly)
ISBN: 0596002416
Kapitel 6.



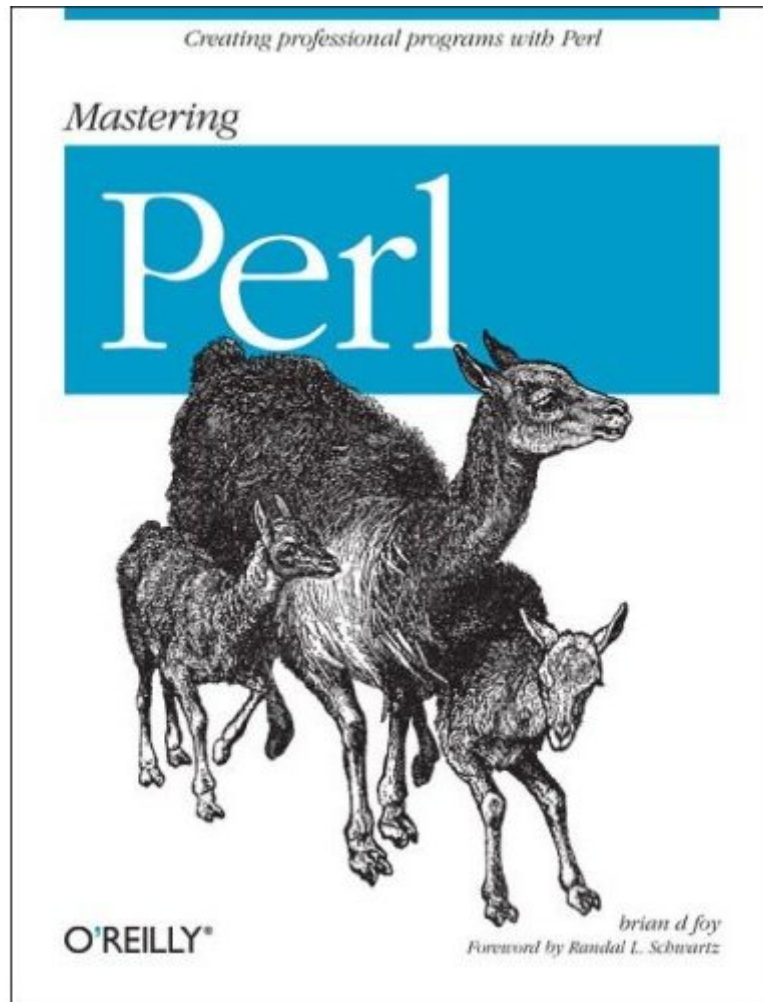
Literatur: Einführung in Perl-Objekte, Referenzen & Module



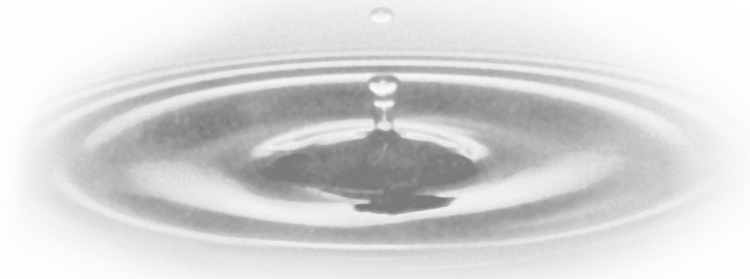
Randal L. Schwartz,
Tom Phoenix:
Einführung in Perl-Objekte,
Referenzen & Module
(O'Reilly)
ISBN: 3897211491
Kapitel 5.



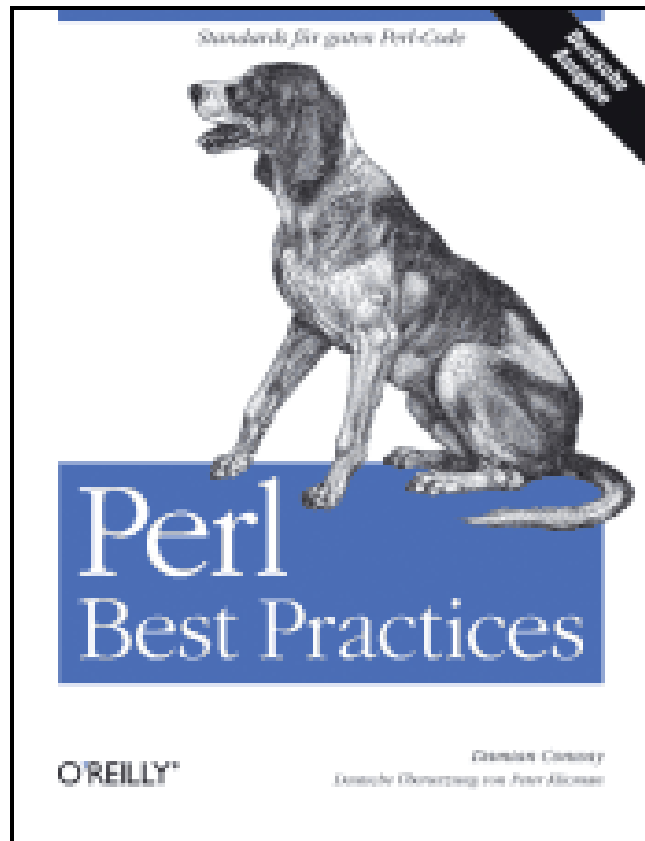
Literatur: Mastering Perl



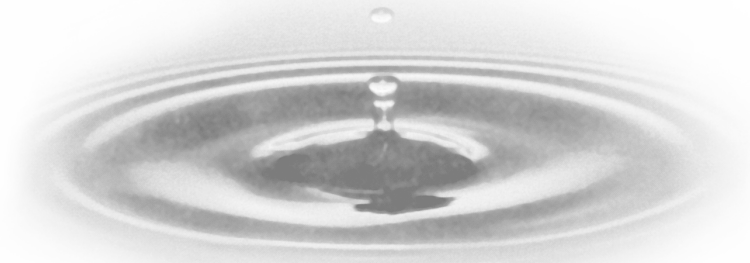
brian d foy:
Mastering Perl
(O'Reilly)
ISBN: 0596527241
Chapter 4.



Literatur: Perl - Best Practices

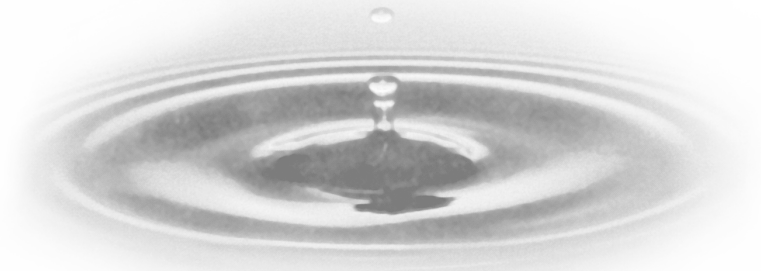


Damian Conway:
Perl - Best Practices
(O'Reilly),
ISBN: 3897214547
(deutsche Ausgabe)
Kapitel 18.



Links I

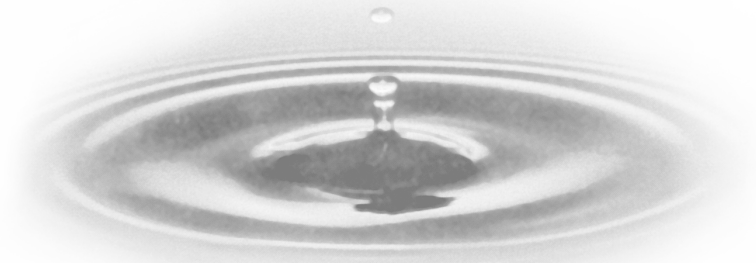
- debugger The Perl Debugger(s): <http://debugger.perl.org/>
- perldebug: <http://search.cpan.org/dist/perl/pod/perldebug.pod>
- perldebtut: <http://search.cpan.org/perldoc?perldebtut>



Links II

Referenzkarten

- Andrew Ford: Perl Debugger Reference Card
<http://refcards.com/refcard/perl-debugger-forda>
- The Perl Debugger Quick Reference Card, a PDF courtesy of O'Reilly <http://www.rfi.net/debugger-slides/reference-card.pdf>
- Perl Debugger Quick Reference
http://www.perl.com/2004/11/24/debugger_ref.pdf
- Einführung in Perl Debugger <http://www.ims.uni-stuttgart.de/lehre/teaching/2006-WS/Perl/debugger4up.pdf>

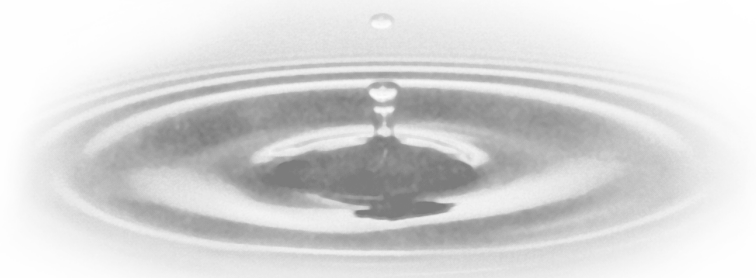


Links III

- YAPC::EU 2007 The Perl Debugger - What is it?
<http://www.rfi.net/debugger-slides/>
- Kapitel 4 aus Mastering Perl:
http://www252.pair.com/comdog/mastering_perl/Chapters/04.debugger.html
- Humpeln zur Diagnose von Michael Schilli (Linux-Magazin)
http://www.linux-magazin.de/heft_abo/ausgaben/2005/04/humpeln_zur_diagnose/
- Perl.com: Unraveling Code with the Debugger:
<http://www.perl.com/lpt/a/2006/04/06/debugger.html>
- YAPC::NA 2004 David Allen: The Perl Debugger
<http://www.coder.com/daniel/yapc::na::2004/debugger/slide001.html>

Links IV

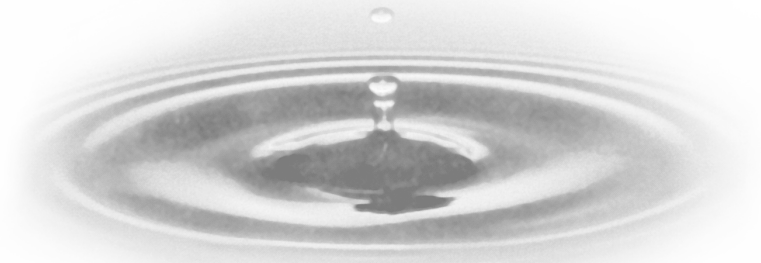
- The Perl Debugger | Linux Journal (2005-01-26):
<http://www.linuxjournal.com/article/7581/>
- Using The Perl Debugger (2003-06-25):
<http://www.devshed.com/c/a/Perl/Using-The-Perl-Debugger/>
- Dr. Dobb's | Using the Perl Debugger | August 9, 2001:
<http://www.ddj.com/184404744>
- Debugging and Devel:: - The Perl Journal, Summer 1998:
http://www.foo.be/docs/tpj/issues/vol3_2/tpj0302-0011.html



Links V

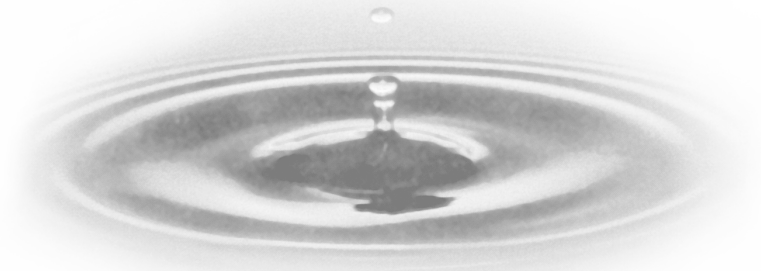
Perl Debugger unter mod_perl

- <http://cpan.uwinnipeg.ca/htdocs/Apache-DB/Apache/DB.html>
- <http://modperlbook.org/html/21-5-6-Introduction-to-the-Perl-Debugger.html>
- Perl.com: Debugging and Profiling mod_perl Applications:
<http://www.perl.com/lpt/a/974>

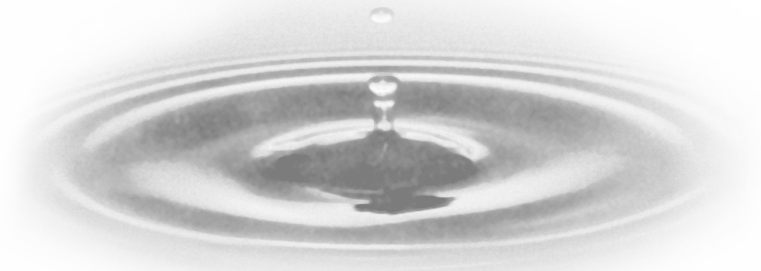


Weitere Debugger im CPAN

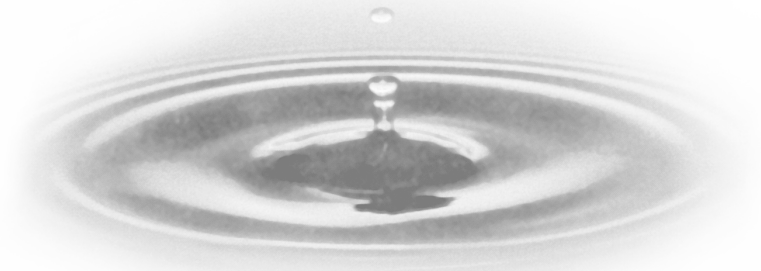
- `Devel::ebug`
- `Devel::TraceUse`
- `Devel::DProf`
- `Devel::Peek`
- `Devel::TraceFuncs`
- `Devel::WeakRef`
- `Devel::Symdump`
- `Devel::StackTrace`



Fragen?



Danke!



Über mich

<http://www.thomas-fahle.de>

info@thomas-fahle.de

<http://thomas-fahle.blogspot.com>

http://www.xing.com/profile/Thomas_Fahle

<http://www.linkedin.com/in/thomasfahle>

