Easy Logging
Einführung in Log::Log4perl qw/:easy/

Thomas Fahle

FrOSCon 2011, St. Augustin, 21. August 2011

Über diesen Vortrag

Einführung in Log::Log4perl

- Log::Log4perl Grundlagen
- Easy Mode - Stealth-Logger
  - Basics
  - Datenstrukturen
  - Appender
  - Packages
  - Performance
- Resourcen im Netz - Mehr Infos

Slides online

`http://www.slideshare.net/thomasfahle`

Log::Log4perl?

- Port von log4j
- Author: Michael Schilli
- sehr ausgereiftes Logging-Framework (Profilösung)
- Umfangreiche, komplette, aber oft verwirrende Dokumentation

Log::Log4perl

- liefert Information aus dem tatsächlichen Betrieb des Programms
- durch Konfigurationsdateien können Loggingstatements, die eigentlich zum Testen/Debuggen gedacht sind im ausgeliefertem Code verbleiben
- Ausgabeziele und Formate können über Konfigurationsdateien unterschiedlichen Umgebungen und Bedürfnissen angepasst werden.

Log::Log4perl Betriebsmodi

Log::Log4perl kennt zwei Betriebsmodi

- **Standardmodus**: Sehr sehr viele fein granulierte Optionen
- **Easy Mode oder Stealth Logger**: komfortables, einfaches Loggen, eben EASY.

Prioritäten

Prioritäten oder Loglevel

(lat. prior = der Vordere) legen die **Wichtigkeit** eines Loggingereignisses fest.
Ob ein Logger die ihm übergebenen Nachrichten tatsächlich weitergibt oder unterdrückt, wird in der Konfiguration festgelegt.

Vordefinierte Prioritäten

Log::Log4perl stellt sechs vordefinierte Loglevel zur Verfügung

Die Level in **absteigender** Reihenfolge:

- FATAL
- ERROR
- WARN
- INFO
- DEBUG
- TRACE

Zusätzlich wird auch das Pseudolevel ALWAYS bereitgestellt.

Priorität TRACE

TRACE

Das Level TRACE kennzeichnet sehr, sehr feinkörnige Ereignisse - feinkörniger als das nächste Level DEBUG.

Beispiel:

```
TRACE("$bytes bytes transferred");
```

Priorität DEBUG

DEBUG

Das Level DEBUG kennzeichnet feinkörnige Ereignisse, die nützlich sind, um eine Applikation zu debuggen.

Beispiel:

```
DEBUG("HTTP GET request OK");
```

Priorität INFO

INFO

Das Level INFO kennzeichnet grobkörnige Ereignisse, die über den Fortschritt des Programms informieren.

Beispiel:

```
INFO("Starte $0");
```

Priorität WARN

WARN

Das Level WARN kennzeichnet Ereignisse, die auf kleine Fehler hinweisen. Das Programm kann aber weiterlaufen.

Beispiel:

```
WARN("HTTP GET failed, retrying");
```

Priorität ERROR

ERROR

Das Level ERROR kennzeichnet Ereignisse, die auf größere Fehler hinweisen. Das Programm kann aber oft noch weiterlaufen.

Beispiel:

```
ERROR("Out of retries!");
```

Priorität FATAL

FATAL

Das Level FATAL kennzeichnet Ereignisse, die auf schwerwiegende Fehler hinweisen. Das Programm wird meist beendet werden müssen.

Beispiel:

```
FATAL("Can't connect to database.");
```

Priorität ALWAYS

ALWAYS

Das Level ALWAYS kennzeichnet Ereignisse, die immer geloggt werden sollen. Sie werden niemals geblockt.

Beispiel:

```
ALWAYS("This will be printed regardless
        of the log level.");
```

Konfiguration Priorität

Konfiguration

- entscheidet, wie sich ein Logger verhält.
- d.h. ob wie wohin geloggt wird.

Konfiguration und Priorität

Priorität	Konfiguration
FATAL	FATAL
ERROR	ERROR
WARN	WARN
INFO	INFO
DEBUG	DEBUG
TRACE	TRACE

Priorität und Konfiguration

Zu jeder Priorität im Programm gehört eine entsprechende Priorität in der Konfiguration.

Beispiel: Konfiguriertes Level WARN

Priorität	Konfiguriertes Level
FATAL	+
ERROR	+
WARN	WARN
INFO	-
DEBUG	-
TRACE	-

Konfiguriertes Level WARN

- Nachrichten unterhalb von WARN werden nicht ausgegeben.
- Nachrichten der Priorität WARN oder höher werden ausgegeben.

Beispiel: Konfiguriertes Level ERROR

Priorität	Konfiguriertes Level
FATAL	+
ERROR	ERROR
WARN	-
INFO	-
DEBUG	-
TRACE	-

Konfiguriertes Level ERROR

- Nachrichten unterhalb von ERROR werden nicht ausgegeben.
- Nachrichten der Priorität ERROR oder höher werden ausgegeben.

Beispiel: Konfiguriertes Level DEBUG

Priorität	Konfiguriertes Level
FATAL	+
ERROR	+
WARN	+
INFO	+
DEBUG	DEBUG
TRACE	-

Konfiguriertes Level DEBUG

- Nachrichten unterhalb von DEBUG werden nicht ausgegeben.
- Nachrichten der Priorität DEBUG oder höher werden ausgegeben.

Appender

Appender/Datensenken

Appender sind Datensenken, in die Log-Nachrichten geschrieben werden.
Als Datensenke lässt sich von der einfachen Bildschirmausgabe (STDERR) über Logdateien, Syslog, SMS- oder E-Mail-Benachrichtigung, bis hin zu Datenbanken, so ziemlich alles verwenden, was man sich als Ausgabe-Medium vorstellen kann.

Log::Log4perl Appender

Log::Log4perl Appender (Auswahl)

- Log::Log4perl::Appender::Screen
- Log::Log4perl::Appender::File
- Log::Log4perl::Appender::Socket
- Log::Log4perl::Appender::DBI
- Log::Log4perl::Appender::Synchronized
- Log::Log4perl::Appender::RRDs

Log::Dispatch Appender

Log::Dispatch Appender (Auswahl)

- Log::Dispatch::ApacheLog
- Log::Dispatch::DBI
- Log::Dispatch::Email
- Log::Dispatch::Email::MIMELite
- Log::Dispatch::File
- Log::Dispatch::FileRotate
- Log::Dispatch::Screen
- Log::Dispatch::Syslog
- u.v.a.m. ...

Layouts

Layouts - Informationsumfang

- Layouts formatieren Log-Nachrichten und legen den Informationsumfang fest.
- Log-Nachrichten können in einer printf()-ähnlichen Syntax einfach an die eigenen Bedürfnisse angepasst (formatiert) werden.

Layouts

Layouts - Formatierungsanweisungen 1

- %c Category of the logging event.
- %C Fully qualified package (or class) name of the caller
- %d Current date in yyyy/MM/dd hh:mm:ss format
- %F File where the logging event occurred
- %H Hostname (if Sys::Hostname is available)
- %l Fully qualified name of the calling method followed by the callers source the file name and line number between parentheses.

Layouts

Layouts - Formatierungsanweisungen 2

- %T A stack trace of functions called
- %L Line number within the file where the log statement was issued
- %m The message to be logged
- %M Method or function where the logging request was issued
- %n Newline (OS-independent)
- %p Priority of the logging event
- %P pid of the current process
- %r Number of milliseconds elapsed from program start to logging
- %% A literal percent (%) sign

Easy Mode (Stealth Logger)

qw/:easy/

```
1 #!/usr/bin/perl
2 use strict;
3 use warnings;
4
5 use Log::Log4perl qw/:easy/;
```

Automatisch importierte Funktionen qw/:easy/

Level/Priortitäten

- FATAL
- ERROR
- WARN
- INFO
- DEBUG
- TRACE

Automatisch importierte Funktionen qw/:easy/

Convenience: Log and ...

- LOGWARN
- LOGDIE
- LOGEXIT
- LOGCARP
- LOGCLUCK
- LOGCROAK
- LOGCONFESS

Automatisch importierte Konstanten qw/:easy/

Level/Priortitäten

- $FATAL
- $ERROR
- $WARN
- $INFO
- $DEBUG
- $TRACE

Easy Mode initialisieren 1

easy_init()

```
1 #!/usr/bin/perl
2 use strict;
3 use warnings;
4 use Log::Log4perl qw/:easy/;
5 Log::Log4perl->easy_init();
```

Easy Mode initialisieren 2

easy_init()

- initialisiert einen einfachen Logger.
- Der Appender ist vom Typ SCREEN, d.h. die Log-Nachrichten erscheinen auf STDERR.
- Das Layout der Nachrichten wird auf das Format "%d %m%n"gesetzt.

Hallo Welt im Easy Mode

Hallo Welt!

```
1 use Log::Log4perl qw/:easy/;
2 Log::Log4perl->easy_init();
3 INFO("Starte $0");
4 print "Hallo Welt!\n";
5 INFO("Beende $0");
```

Ausgabe

```
1 2011/08/20 15:31:21 Starte hallowelt.pl
2 Hallo Welt!
3 2011/08/20 15:31:21 Beende hallowelt.pl
```

Loglevel im Easy Mode konfigurieren

Priorität/Level DEBUG

```
1 use Log::Log4perl qw/:easy/;
2 Log::Log4perl->easy_init( {
3                     level => $DEBUG,
4                   } );
```

Layout im Easy Mode konfigurieren

Layout

```
1 use Log::Log4perl qw/:easy/;
2 Log::Log4perl->easy_init( {
3                     level => $DEBUG,
4                     layout =>
5                       '%d [%M] %F %L> %m%n', } );
```

Ausgabe

```
1 2011/08/20 16:27:36 [main::] hallowelt.pl \
2    10> Starte hallowelt.pl
3 Hallo Welt!
4 2011/08/20 16:27:36 [main::] hallowelt.pl \
5    12> Beende hallowelt.pl
```

Ausgabemedium im Easy Mode konfigurieren

Ausgabe in eine Datei

```
1  use Log::Log4perl qw/:easy/;
2  Log::Log4perl->easy_init( {
3                      level => $DEBUG,
4                      layout =>
5                        '%d [%M] %F %L> %m%n',
6                      file =>
7                        '>> ./test.log',
8   } );
```

Skalare Variablen

Skalare Variablen werden wie gewohnt expandiert

```
1 my $counter = 0;
2   # Double Quotes
3   DEBUG("Counter $counter");
```

Tipp: Begrenzer um Variablen setzen

```
1 my $counter = 0;
2   # Double Quotes
3   DEBUG("Counter <$counter>");
```

Konstanten

Bei Konstanten kommt der Verkettungsoperator . zum Einsatz.

```
1  use constant PI => 4 * atan2(1, 1);
2    DEBUG("Konstante PI " . PI . "..." );
```

Arrays, Hashes und komplexe Datenstrukturen

Data::Dumper to the rescue

- Arrays, Hashes und komplexe Datenstrukturen lassen sich meist nur sinnvoll mit Data::Dumper loggen.
- Data::Dumper in Version 2.09 ist seit Perl 5.005 ein Coremodule.

Kleine Arrays

Kleine Arrays - Expansion

```
1 sub foo {
2   DEBUG("Parameter: @_");
3 }
```

Kleine Arrays - foreach

```
1 my @array = 1 .. 10;
2 foreach my $element ( @array ) {
3   DEBUG("Arrayelement: <$element>");
4 }
```

Arrays

Arrays (langsam) dumpen

```
1 use Data::Dumper;
2 # turn off all pretty print
3 $Data::Dumper::Indent = 0;
4 my @array = 1 .. 10;
5 TRACE("Array: ", Dumper( \@array ) );
```

Kleine Hashes

Kleine Hashes - Expansion

```
1 my %hash;
2 DEBUG("Hash @{[ %hash ]}");
```

Kleine Hashes - while

```
1 while ( my ($key, $value) = each %hash ) {
2  DEBUG("Hash: Key <$key> Value <$value>");
3 }
```

Hashes

Hashes (langsam) dumpen

```
1 use Data::Dumper;
2 # turn off all pretty print
3 $Data::Dumper::Indent = 0;
4 my %hash;
5 TRACE("Hash: ", Dumper( \%hash ) );
```

Need for Speed - message output filter format

Arrays

```
1 TRACE("Array: ", {
2         filter => \&Data::Dumper::Dumper,
3         value  => \@array } );
```

Hashes

```
1 TRACE("Hash: ", {
2         filter => \&Data::Dumper::Dumper,
3         value  => \%hash } );
```

Message output filter format

Message output filter format

- Log::Log4perl wird die *filter* Funktion, erst dann aufrufen, wenn die Nachricht wirklich in einen Appender geschrieben wird.
- Die *filter* Funktion erwartet eine Referenz auf ein Unterprogramm, welches den Dump ausführt, und eine Referenz auf die zu loggende Datenstruktur.
- Spart spürbar Rechenzeit und Hauptspeicher.

Dump Profiling

Beispielprogramm

```
1  my %hash = ();
2  for ( 1 .. 10_000 ) {
3      $hash{$_} = 1;
4  }
5
6  while ( my ( $key, $value ) = each %hash ) {
7      TRACE("Hash: Key <$key> Value <$value>");
8  }
9  TRACE("Hash:", Dumper( \%hash ) );
10 TRACE("Hash: ", {
11         filter => \&Data::Dumper::Dumper,
12         value  => \%hash } );
```

Dump Profiling - Konfiguriertes Level TRACE

Konfiguriertes Level TRACE

d.h., es soll geloggt werden.

19	1	6µs			`my %hash = ();`
20					
21	1	17µs			`for ( 1 .. 10_000 ) {`
22	10000	131ms			`    $hash{$_} = 1;`
23					`}`
24					
25	1	337ms	10000	17.5s	`while ( my ( $key, $value ) = each %hash ) {` `# spent 17.5s making 10000 calls to Log::Log4perl::__ANON__[Log/Log4perl.pm:131], avg 1.75ms/call`
26					`    TRACE("Hash: Key <$key> Value <$value>");`
27					`}`
28					

29	1	35.8ms	2	228ms	`TRACE( "Hash:", Dumper( \%hash ) );`
					`# spent 222ms making 1 call to Data::Dumper::Dumper`
					`# spent 6.17ms making 1 call to Log::Log4perl::__ANON__[Log/Log4perl.pm:131]`

30					
31					`# Preferred`
32	1	65µs	1	275ms	`TRACE( "Hash: ", { filter => \&Data::Dumper::Dumper, value => \%hash } );` `# spent 275ms making 1 call to Log::Log4perl::__ANON__[Log/Log4perl.pm:131]`

Dump Profiling - Konfiguriertes Level INFO

Konfiguriertes Level INFO

d.h., es soll **nicht** geloggt werden.

18					
19	1	4µs			my %hash = ();
20					
21	1	12µs			for (1 .. 10_000) {
22	10000	74.9ms			$hash{$_} = 1;
23					}
24					
25	1	226ms	10000	398ms	while (my ($key, $value) = each %hash) { # spent 398ms making 10000 calls to Log::Log4perl::__ANON__[Log/Log4perl.pm:131], avg 40µs/call
26					TRACE("Hash: Key <$key> Value <$value>");
27					}
28					
29	1	32.2ms	2	119ms	TRACE("Hash:", Dumper(\%hash)); # spent 119ms making 1 call to Data::Dumper::Dumper # spent 60µs making 1 call to Log::Log4perl::__ANON__[Log/Log4perl.pm:131]
30					
31					# Preferred
32	1	64µs	1	122µs	TRACE("Hash: ", { filter => \&Data::Dumper::Dumper, value => \%hash }); # spent 122µs making 1 call to Log::Log4perl::__ANON__[Log/Log4perl.pm:131]

Konfiguration Appender

Konfiguration

- entscheidet, wie sich ein Logger verhält.
- d.h. ob wann wie wohin geloggt wird.

Easy Mode aus Konfigurationsdatei initialisieren

init()

```
1 #!/usr/bin/perl
2 use strict;
3 use warnings;
4 use Log::Log4perl qw/:easy/;
5 Log::Log4perl->init( 'l4p.conf' )
6     or die $!;
```

Konfigurationsdatei Beispiel

Log::Log4perl Konfigurationsdatei

```
1  log4perl.logger                                = TRACE, A1
2  log4perl.appender.A1 \
3      = Log::Dispatch::File
4  log4perl.appender.A1.filename       = ./test.log
5  log4perl.appender.A1.mode           = append
6  log4perl.appender.A1.layout \
7      = Log::Log4perl::Layout::PatternLayout
8  log4perl.appender.A1.layout.ConversionPattern\
9      = %d %p> %F{1}:%L %M - %m%n
```

Fehlende Konfigurationsdateien 1

Try

```
1 use Log::Log4perl qw/:easy/;
2 my $log4p_cfg_file = '/tmp/XXXXX.cfg';
3
4 # try to init with a config file
5 eval {
6     Log::Log4perl->init( $log4p_cfg_file )
7       or die "Error $log4p_cfg_file - $!";
8 };
```

Fehlende Konfigurationsdateien 2

Catch:

```
1  # catch errors from init and use a simple
2  # configuration, sending every message to
3  # STDERR
4  if ( $@ ) {
5      Log::Log4perl->easy_init( {
6          'level'  => $DEBUG,
7          'layout' => "%d %p> %F{1}: %L %M - %m%n",
8        } );
9      WARN("Falling back to easy configuration!");
10 }
```

init_once - nur ein einziges Mal initialisieren

init_once() - Nur initialisieren, falls noch nicht initialisiert

```
1 # Init or skip if already done
2 Log::Log4perl->init_once( $log4p_cfg_file );
```

mod_perl

Entweder in der *startup.pl* initialisieren oder init_once() verwenden.

init_and_watch - Konfiguration im laufendem Betrieb ändern.

init_and_watch() - zeitgesteuert.

```
1 my $delay = 300; # seconds
2 Log::Log4perl->init_and_watch(
3     $log4p_cfg_file, $delay);
```

init_and_watch() - signalgesteuert.

```
1 Log::Log4perl->init_and_watch(
2     $log4p_cfg_file, 'HUP');
```

Logdateien rotieren

Log::Log4perl Konfigurationsdatei

```
1  log4perl.logger                    = TRACE, A1
2  # Appender A1
3  log4perl.appender.A1
   = Log::Dispatch::FileRotate
4  log4perl.appender.A1.max           = 10
5  log4perl.appender.A1.size          = 20_000
6  log4perl.appender.A1.filename      = ./logs/logfile.log
7  log4perl.appender.A1.mode          = append
8  log4perl.appender.A1.layout        \
9      = Log::Log4perl::Layout::PatternLayout
10 log4perl.appender.A1.layout.ConversionPattern\
11     = %d %p> %F{1}:%L %M - %m%n
```

Logs per E-Mail versenden

Log::Log4perl Konfigurationsdatei

```
1 log4perl.logger                    = TRACE, Mailer
2 log4perl.appender.Mailer
  = Log::Dispatch::Email::MailSend
3 log4perl.appender.Mailer.to
  = your@email.address
4 log4perl.appender.Mailer.subject
  = My Program Notice
5 log4perl.appender.Mailer.buffered  = 1
6 log4perl.appender.Mailer.layout    \
7   = Log::Log4perl::Layout::PatternLayout
8 log4perl.appender.Mailer.layout.ConversionPattern\
9   = %d %p> %F{1}:%L %M - %m%n
```

Logdateien rotieren und FATALs per E-Mail versenden 1

Log::Log4perl Konfigurationsdatei 1

```
1  log4perl.logger                    = TRACE, A1, Mailer
2  # Appender A1
3  log4perl.appender.A1    \
4      = Log::Dispatch::FileRotate
5  log4perl.appender.A1.max       = 10
6  log4perl.appender.A1.size      = 20_000
7  log4perl.appender.A1.filename  = ./logs/logfile.log
8  log4perl.appender.A1.mode      = append
9  log4perl.appender.A1.layout    \
10     = Log::Log4perl::Layout::PatternLayout
11 log4perl.appender.A1.layout.ConversionPattern\
12     = %d %p> %F{1}:%L %M - %m%n
```

Logdateien rotieren und FATALs per E-Mail versenden 2

Log::Log4perl Konfigurationsdatei 2

```
1  # Mailer
2  log4perl.appender.Mailer \
3      = Log::Dispatch::Email::MailSend
4  log4perl.appender.Mailer.to \
5      = your@email.address
6  log4perl.appender.Mailer.subject \
7      = My Program Notice
8  log4perl.appender.Mailer.buffered  = 1
9  log4perl.appender.Mailer.min_level = fatal
10 log4perl.appender.Mailer.layout    \
11     = Log::Log4perl::Layout::PatternLayout
12 log4perl.appender.Mailer.layout.ConversionPattern\
13     = %d %p> %F{1}:%L %M - %m%n
```

Syslog

Log::Log4perl Konfigurationsdatei

```
1  log4perl.logger                                = TRACE, Syslog
2
3  log4perl.appender.Syslog
   = Log::Dispatch::Syslog
4  log4perl.appender.Syslog.ident     = MyIdent
5  log4perl.appender.Syslog.facility  = user
6  log4perl.appender.Syslog.layout    \
7      = Log::Log4perl::Layout::PatternLayout
8  log4perl.appender.Syslog.layout.ConversionPattern\
9      = %d %p> %F{1}:%L %M - %m%n
```

Loggen in eine Datenbank 1

Datenbank und Loggingtabelle erstellen

```
1  create database logs;
2  \u logs
3  create table L (
4   id BIGINT unsigned AUTO_INCREMENT NOT NULL,
5   date DATETIME,
6   loglevel VARCHAR(20),
7   category VARCHAR(255),
8   file VARCHAR(255),
9   line INT(10) UNSIGNED,
10  message TEXT,
11  PRIMARY KEY(id) );
```

Loggen in eine Datenbank 2

Log::Log4perl Konfigurationsdatei 1

```
1 log4perl.logger                          = TRACE, DB
2 log4perl.appender.DB                     = \
3   Log::Log4perl::Appender::DBI
4 log4perl.appender.DB.layout              = \
5  Log::Log4perl::Layout::PatternLayout
6 log4perl.appender.DB.datasource          = \
7   DBI:mysql:logs:localhost
8 log4perl.appender.DB.username            = loguser
9 log4perl.appender.DB.password            = geheym
```

Loggen in eine Datenbank 3

Log::Log4perl Konfigurationsdatei 2

```
1  log4perl.appender.DB.sql          = insert into L \
2   (date,loglevel,category,file,line,message) \
3    VALUES (?,?,?,?,?,?)
4  log4perl.appender.DB.params.1     = %d
5  log4perl.appender.DB.params.2     = %p
6  log4perl.appender.DB.params.3     = %c
7  log4perl.appender.DB.params.4     = %F
8  log4perl.appender.DB.params.5     = %L
9  log4perl.appender.DB.params.6     = %m
10 # For efficiency sake log only after n messages
11 #log4perl.appender.DB.bufferSize = 3
```

Loggen in eine XML-Datei

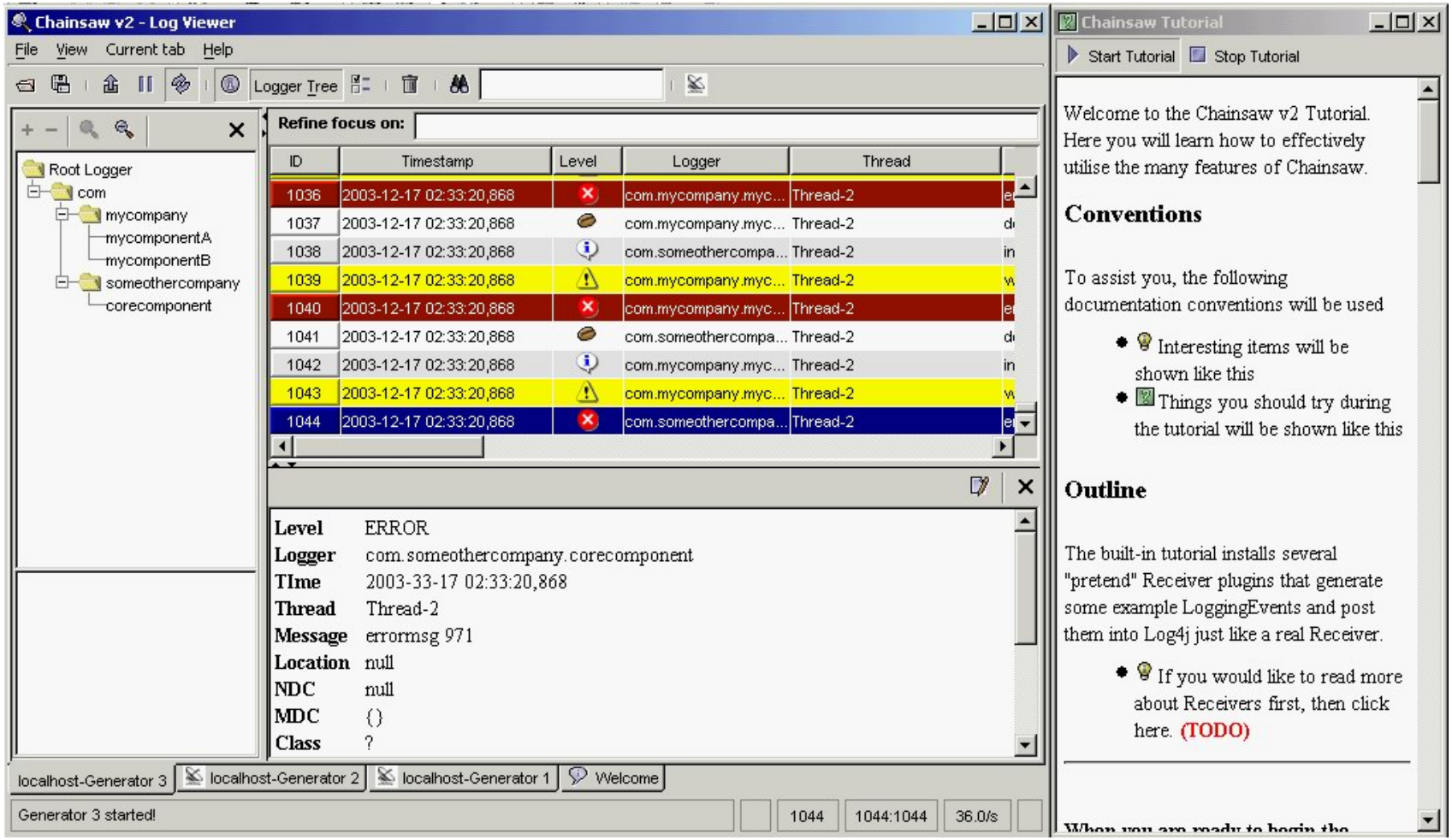
Log::Log4perl Konfigurationsdatei

```
1  log4perl.logger = WARN, A1
2  log4perl.appender.A1 = \
3    Log::Log4perl::Appender::File
4  log4perl.appender.A1.filename = log.xml
5  log4perl.appender.A1.layout \
6    = Log::Log4perl::Layout::XMLLayout
7  log4perl.appender.A1.layout.LocationInfo = TRUE
8  log4perl.appender.A1.layout.Encoding = iso8859-1
```

XML

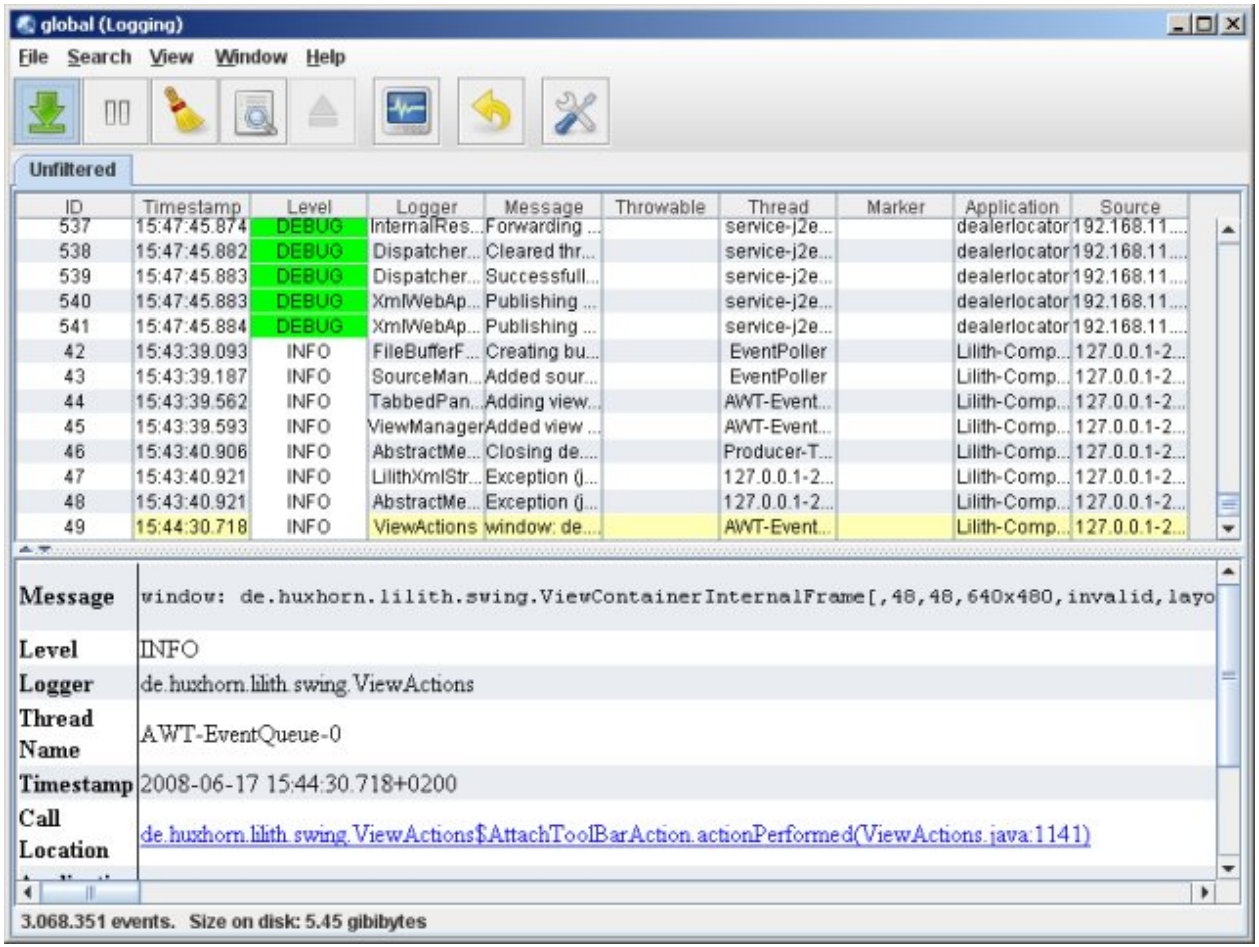
kann von Logviewern wie z.B. Chainsaw oder Lilith gelesen werden

Chainsaw



Bildnachweis: `http://logging.apache.org/chainsaw/index.html`

Lilith



Bildnachweis: `http://sourceforge.net/projects/lilith/`

Log::Log4perl in Packages

Packages

```
1  package Foo;
2  use strict;
3  use warnings;
4  use Log::Log4perl qw/:easy/;
5
6  sub foo {
7      my $bar = shift @_;
8      DEBUG("Fooing <$bar>");
9      # ... do something
10 }
11 1;
```

Log::Lop4perl in Packages

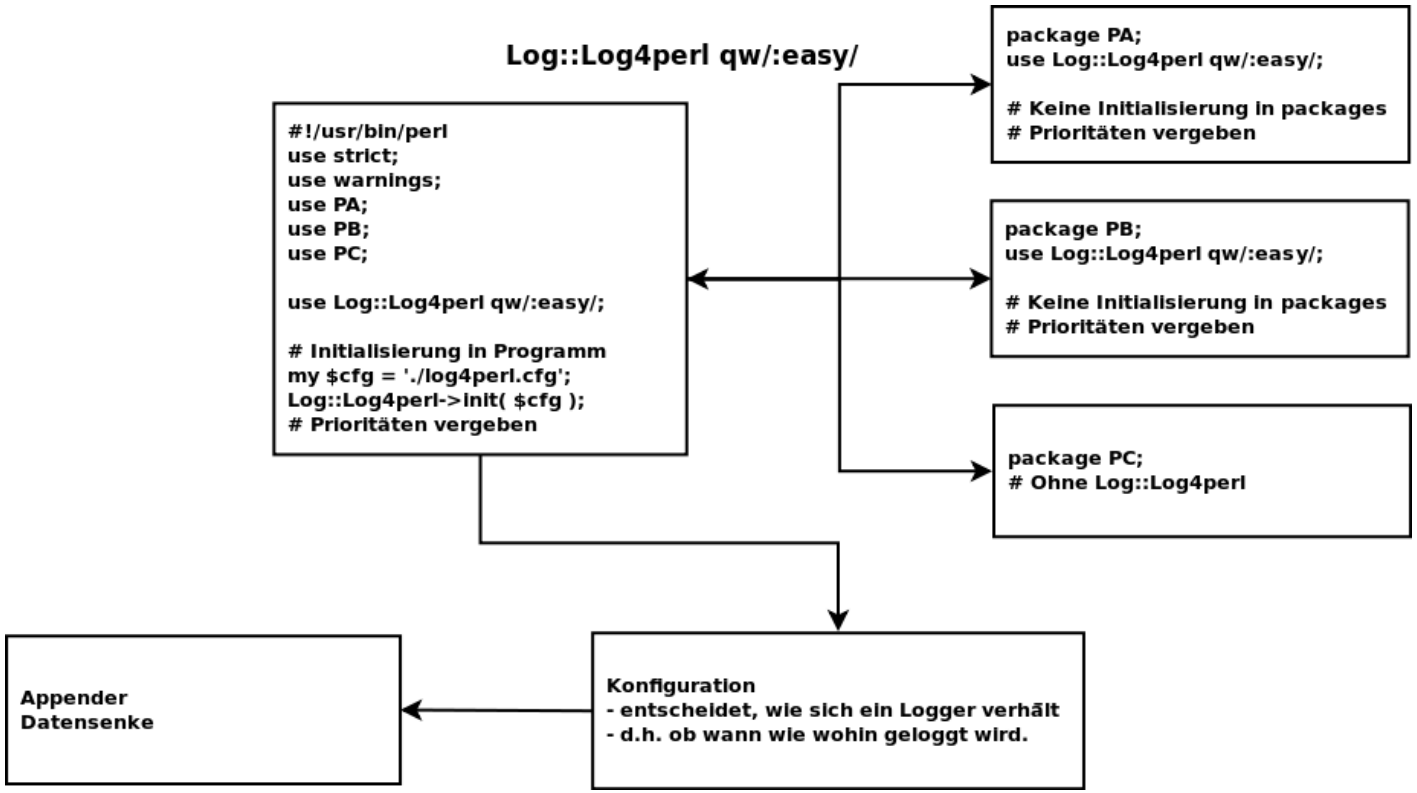
In Packages

- wird **nicht** initialisiert
- wird **nicht** konfiguriert
- werden Log Prioritäten vergeben

Stealth (engl. Heimlichkeit)

- Wenn ein Logger nicht durch ein Programm initialisiert wird, wird der Logging Code im Package **nicht** ausgeführt.
- Das hat den Charme, das man auch Pakete Log::Log4perl fähig machen kann, ohne treibende Programme zu änderen.

Workflow


Log::Log4perl qw/:easy/
#!/usr/bin/perl
use strict;
use warnings;
use PA;
use PB;
use PC;
use Log::Log4perl qw/:easy/;
Initialisierung in Programm
my $cfg = './log4perl.cfg';
Log::Log4perl->init($cfg);
Prioritäten vergeben
package PA;
use Log::Log4perl qw/:easy/;
Keine Initialisierung in packages
Prioritäten vergeben
package PB;
use Log::Log4perl qw/:easy/;
Keine Initialisierung in packages
Prioritäten vergeben
package PC;
Ohne Log::Log4perl
Konfiguration
- entscheidet, wie sich ein Logger verhält
- d.h. ob wann wie wohin geloggt wird.
Appender
Datensenke

Performance

Je mehr geloggt wird, desto schlechter die Performance

- Loglevel WARN für produktive Systeme
- Loglevel ERROR für produktive Systeme mit hoher Last
- Spezielle Konfigurationen nutzen, falls spezielle Probleme gelöst werden müssen
- Ein Profiler wie Devel::NYTProf ist hilfreich

RTFM

Log::Log4perl wird mit umfangreicher Dokumentation ausgeliefert.

- The log4perl project – log4j for Perl
  `http://mschilli.github.com/log4perl/`
- Manual
  `http://search.cpan.org/perldoc?Log::Log4perl`
- FAQ
  `http://search.cpan.org/perldoc?Log::Log4perl::FAQ`

Artikel

Michael Schilli

- Logging nach Bedarf mit Log::Log4perl - Wachsame Schläfer `http://www.linux-magazin.de/heft_abo/ausgaben/2003/01/wachsame_schlaefer`
- Retire your debugger, log smartly with Log::Log4perl! `http://www.perl.com/pub/a/2002/09/11/log4perl.html`
- Praktisch jeder Artikel im Linux-Magazin - Entwicklung - Perl `http://www.linux-magazin.de/Themengebiete/Entwicklung/Perl`

Artikel

Diverse Autoren

- brian d foy: Mastering Perl: Logging
  `http://www252.pair.com/comdog/mastering_perl/Chapters/13.logging.html`
- Horshack's Log4perl Page
  `http://lena.franken.de/perl_hier/log4perl.html`
- Kennedy Clark: Using Catalyst with Log4perl
  `http://www.catalystframework.org/calendar/2006/11`
- Patrick H. Piper: 8 Useful Log::Log4perl Recipes
  `http://netlinxinc.com/netlinx-blog/52-perl/126-eight-loglog4perl-recipes.html`

Vorträge

Michael Schilli

- Log4perl: the Only Logging System You'll Ever Need - OSCON 2008
  `http://mschilli.github.com/log4perl/l4p.ppt`
- Talk at YAPC 2007 (Houston, TX) - Smart Logging with Log4perl: `http://perlmeister.com/log4perl_yapc.html`

Vorträge

Diverse Autoren

- Madmongers - Logging
  `http://www.madmongers.org/uploads/Mc/oB/McoBWGNx-nYiEpqQfYz4kA/logging---madmongers.pdf`
- WebGUI TV - Logging
  `http://www.webgui.org/wgtv/logging`
- Renée Bäcker - Log::Log4perl (FrOSCon 2010)
  `http://www.renee-baecker.de/vortraege.html`
- Thomas Fahle - Log::Log4perl qw/:easy/
  `http://www.slideshare.net/thomasfahle/loglog4perl-qweasy-presentation`

log4j zum Vergleich

log4j

- Ceki Gülcü - Short introduction to log4j `http://logging.apache.org/log4j/1.2/manual.html`
- Apache log4j™ `http://logging.apache.org/log4j/`
- Vipan Singla - Don't Use System.out.println! Use Log4j `http://www.vipan.com/htdocs/log4jhelp.html`
- Logback Project - intended as a successor to the popular log4j project - `http://logback.qos.ch/`

Fragen?

Fragen?

- Fragen!

Vielen Dank!

Danke!

- Vielen Dank!

Perl.org

When you need Perl think Perl.org

- `http://www.Perl.org`
- `http://learn.Perl.org`
- `http://blogs.Perl.org`
- `http://jobs.Perl.org`

Flexible & Powerful

- That's why we love Perl!