Advanced Datastructures in Perl

Abigail

Day 1, at 14:00, room L-219 — 2.25 hours

In this tutorial, we will discuss datastructures that go beyond scalars, arrays and hashes. We will focus on search trees, balanced trees, multi-dimensional trees, priority queues. We will discuss exact match queries, nearest neighbour queries, counting queries, half planar queries and range queries. We will discuss both static and dynamic trees. We will discuss the asymptotic efficiency of the various datastructures and algorithms.

This is an advanced tutorial. Participants are expected to have a working knowledge of Perl, basic datastructures and algorithms, big-Oh.

The Return Of The JAPHi

Abigail

Day 2, at 14:45, room L-132 — 45 minutes

Following last year's success, we dive again in the wonderful world of JAPHs and other obscure signatures. We'll see new exploits of 5.6-isms, undocumented features, and we learn how bugs can be turned into JAPHS. We also will answer last year's homework problem.

Making a Module: And All the Crap That CPAN Wants Too

R. Geoffrey Avery

Day 2, at 09:00, room L-26 — 45 minutes

While writing `Apache::ParseLog` 2.0 I had to search high and low to learn how to wrap all the goodies around the module to share it on CPAN. This talk contains most of what I learned in that search. Including documentation, testing, distribution, installation, and uploading to CPAN of the module. Some examples will come from `Apache::ParseLog` 2.0.

Outline of Topics to be Covered:

`h2xs`

- How to create a template Perl module with the supporting files CPAN wants
- Discuss some other files often included in a distributed module, like `README`

`Test::Harness`

- How to build test cases to be used by make to verify the module does what the programmer claims it does

`ExtUtils::MakeMaker`

- How to build Makefile.PL and generate a makefile

`make`

- How to use make to build, package, and test the module with all the supporting files

CPAN (Comprehensive Perl Archive Network)

- The steps necessary to upload a module to CPAN

`AUTOLOAD`

- How to deal with methods that don't have code
- And how to actually create the functions for repeat use

How to get a Date with Perl

Rich Bowen

The Creative Group

Day 2, at 11:00, room L-219 — 45 minutes

This talk is about date, time and calendar calculations with Perl. I'll talk about the modules available on CPAN and talk about easy solutions to common date-related problems. And I'll talk about some rather strange and esoteric calendars, like the Discordian calendar, the ISO calendar, and the French Revolutionary calendar, among others.

The Perl Monger World Map

Leon Brocard

London Perl Mongers

Day 2, at 15:10, room L-26 — 20 minutes

This will explain how the Perl Monger World Map (<http://astray.com/Bath.pm/>) is generated, and explain how to generate your own. It will cover:

- A short intro on longitude/latitude and map projection
- Use of `XML::Simple` to fetch the data
- Use of `Image::Imlib` to plot the points
- The label overlap problem, which is NP-hard, and an explanation of label positioning solutions, including random placement and simulated annealing
- Use of `Inline` module for speed increase in the inner loops
- Use of `CGI` and Template Toolkit modules with small map tiles for fast cached display on the web

Creating an optimising compiler and interpreter for a toy language

Leon Brocard

London Perl Mongers

Day 3, at 11:00, room L-26 — 20 minutes

This will explain how to use either the `Parse::Yapp` or `Parse::RecDescent` modules to construct an optimising interpreter and then an optimising compiler for a simple Perl-like toy language, much like the Template Toolkit does. This is simplified by using Perl as a runtime environment. It will cover:

- A short intro on grammars
- Constructing a parser in `Parse::Yapp` or `Parse::RecDescent`, creating an object parse tree
- A simple pretty printer for such a tree
- Some simple constant optimisations
- A simple evaluator for such a tree
- A simple pretty printer which outputs Perl code, and uses for such a thing
- Some less simple optimisations
- Simple pretty printers which output Java code, including some Discussion of why this is hard
- Some talk about how this might help Perl 6

Graphing Perl

Leon Brocard

London Perl Mongers

Day 3, at 14:20, room L-26 — 20 minutes

This talk will explore explore visualisation of various parts of Perl using directed graphs and the `GraphViz` module. It will cover:

- Usage of the `GraphViz` module
- Graphs of Perl data structures instead of Data::Dumper
- Use of the `B::Xref` module to show potential subroutine call graphs
- Use of the `Devel::SmallProf` module to show line-level call graphs
- Use of the `Devel::DProf` module to show subroutine-level call graphs
- Graphing Perl opcodes with the `B::Graph` module
- Graphing the Perl grammar

Locale::Maketext

Sean M. Burke

Columnist for The Perl Journal

Day 2, at 14:00, room L-219 — 45 minutes

`Locale::Maketext` is a framework for software localization, allowing programs (whether as part of applications or utilities, or as part of web sites) to produce interfaces in the user's preferred language.

`Locale::Maketext` is object-oriented: you subclass it in order to make classes for each of the languages that your project is localized for. Each class represents one language, and (besides any useful methods) contains a lexicon of phrases that it knows how to say.

Each lexicon entry maps from an internal key (like `"Searching..."` or `"[_1]filesfound."`) to either a fixed string:

```
"Cherchant...",
```

a coderef:

```
sub {  ($_[1] == 0) ? "Aucun fichier trouvé."
     : ($_[1] == 1) ? "1 fichier trouvé."
     :                ($_[0]->numf($_[1]) ." fichiers trouvés.")
},
```

or a concise format for expressing most common kinds of interpolation:

```
"Message seen by [quant,_1,person,people,nobody].",
```

The OOPiness of `Locale::Maketext`, combined with Perl's ability to load classes and `eval(\$new_code)` as needed at runtime, makes `Locale::Maketext` able to fix problems in the "message catalog" approach to localization, where messages are essentially limited to being sprintf format strings.

Using GD for charting time-series data.

Michael John Burns

Reliance Technology Consultants Inc.

Day 3, at 15:10, room L-26 — 20 minutes

A project in Perl we undertook to chart time-series data using `GD` and `GD::Graph`. This was executed by by Michael John Burns (Junior, Wakefield (MA) High) with guidance from Ranga Nathan.

This project was the result of a need to extract data from an IBM mainframe report containing attributes of a mainframe database called ADABAS. We needed to capture time-series data and chart space usage by data, index and number of records in tables. Stacked bar and pie charts are supported. The captured data is held in a MySQL database and charting is done by a Perl CGI script.

Life, the Universe, and Everything

Damian Conway

Perl

Day 2, at 16:00, room L-132 — 45 minutes

Watch in horrified fascination as three utterly unrelated trains of thought…

- quantum finite state automata,

- the philosophical conundrum of Maxwell's demon, and
- programming in Klingon

. . . collide with Perl at high speed in Damian's brain.

The Conway Channel

Damian Conway

Perl

Day 3, at 16:00, room L-132 — 20 minutes

See `<http://www.yetanother.org/damian/>`

Building Perl Projects with MakeMaker

Sean Dague

IBM Linux Technology Center

Day 2, at 09:45, room L-26 — 45 minutes

The presentation will walk through the basic steps in building a Perl project using the `MakeMaker` infrastructure. Attention will be paid to many of the key configuration options of `Makefile.PL`, how to write effective test scripts against project modules, and tricks to make `Makefile.PL` do things beyond its normal scope (e.g. the creation of RPM packages).

Stolen Secrets of the Wizards of the Ivory Tower

Mark-Jason Dominus

Plover Systems

Day 1, at 14:00, room L-132 — 2.25 hours

Who should attend Intermediate to advanced Perl programmers with experience working on large projects.

What you will learn It's well-known that Perl resembles languages used in the Unix sysadmin community, such as C, Bourne shell, awk, and sed. People programmers tend to come from that community, or to have been trained by people who did.

Perhaps you've heard the joke that a dedicated Fortran programmer can write Fortran programs in any language. But Perl programmers have been writing C programs in Perl for years without realizing it. This class will show you how to use some of Perl's most powerful features to do organize your programs more powerfully and effectively.

I've raided the techniques of the Lisp and functional programming communities and come back with all the loot I can carry, techniques for increasing modularity and interoperability on large projects.

We'll take recursion to the next level and see how to build powerful recursive functions for managing hierarchical data. We'll see how callbacks, usually relegated to GUI programming, can be used to increase modularity in your library functions, so that they can be used in ways you didn't expect, by programmers years later, without anyone having to change or even look at your code. We'll look in detail at how to provide 'function factories' that manufacture, on demand, the library functions that the user needs. We'll see how objects based on anonymous functions can be used to get around some of the limitations of Perl's built in OOP system. We'll see how to make your own filehandle-like objects that generate data on demand.

Recursion • Using Callbacks to increase modularity Anonymous Functions User Parameters – Return Values – Emulating 'du' **Caching: Making functions faster** What is Caching – Caching Example – Caching Fixes Recursion **Memoization** Adding Caching Code is Too Much Work – Memoization: Getting Perl to rewrite your function – Closures – Memoization Caveats – More uses of memoization – Caching for benchmarking and profiling **Iterators** What and why are Iterators? – Iterator Example – Iterative File Tree Walker – Iterator Operations – More Generic Iterators – More Applications of Iterators **Objects** Anonymous Subroutines as Objects – Benefits of subroutine-based objects – Syntactic sugar – Iterator objects for numerical computations **Infinite Lists** Streams – Stream Implementation – Promises – Simple Example – map and grep for infinite lists – More applications – Merge Factory **Dispatch Tables • State Machines** Implementing State Machines in Perl – State Machine For NNTP – State Machines Are Very Easy to Read!

Caution This tutorial is substantially the same as last year's "Advanced Programming Techniques" class. People who took that class last year whould not take this one.

The Identity Function

Mark-Jason Dominus

Plover Systems

Day 3, at 09:00, room L-132 — 20 minutes

```
sub identity { $_[0] }
```

and its variations:

```
sub identity { $_[1] }
sub identity { @_ }
```

are more useful than they appear. In this cut-down, 20 minute version of my talk from TPC this year, I will

show surprising and unexpected uses for the identity function, in the style of my “Perl Hardware Store” and “Tricks of the Wizards” talks of the past.

The identity functions I’ll discuss may include:

- The identity function to emulate the C address-of operator
- The identity function to perform method calls in double-quoted strings
- The identity function to accumulate a list of all the broken symlinks under a directory or its subdirectories
- The identity function to reconnect a tied variable with its object
- Other surprising appearances of the identity function

Dirty Stories about the Regex Engine

Mark-Jason Dominus

Plover Systems

Day 3, at 09:25, room L-132 — 20 minutes

This talk is too short to tell you all about how the Perl regex engine works—that would take years, and cost millions of lives. Instead, I’ll do three very brief case studies of how it handles certain specific features. You’ve probably always suspected that the Perl regex engine was concealed filthy secrets and nasty surprises; these three features are noteworthy for being particularly disgusting.

Barf bags will be provided.

Outline:

- How `use re 'eval'` works
- What you wish you didn’t know about `/o`
- Unpleasant Surprise Feature

Lightning Talks

Mark-Jason Dominus

Plover Systems

Day 3, at 11:00, room L-132 — 1.5 hour

Lightning talks are sixteen five-minute talks in ninety minutes. Regrettably, nobody submitted a proposal for a talk to be given in French.

How OOP is like Japanese food
Sean M. Burke

Aspects in Perl
Marcel Grunauer

Who talks (to whom) - and how much
Gene Boggs

perl users unite: using our power for freedom in a less than freeware world
guinevere liberty

Examining Fossilized Code
Uri Guttman

A Do-It-Yourself Archive for Mailing Lists
Jim Keenan

Creepy Crawlies
Robert Spier

Inline::Files, a LightnIng(y) and Thunder (from down under) Talk
Brian Ingerson and Damian Conway

My modules: mine, mine, all mine
Dave Rolsky

The Everything Engine, a `mod_perl` web development environment
Nathan Oostendorp

CGI::Application, A Framework for Web Applications
William R. Rico

Parsing with regexes and Parse::RecDescent – which one to use
Joe Davidson

Object concurrency in Perl with Concurrent::Object
Vipul Ved Prakash

Lies We Tell Ourselves About Perl
Walt Mankowski

Improving Perl Modularity with ./
Todd Olson

The 80/20 Irony
Nathan Torkington

Perl on .NET and .NET on Perl

Gurusamy Sarathy

ActiveState Corporation

Day 2, at 09:00, room L-132 — 1.5 hour

ActiveState has been involved in a project to port Perl to the Microsoft .NET Framework. As part this effort, we have conducted various experiments to better understand: what the framework offers languages such as Perl, how a Perl implementation could be designed to take advantage of all that the framework offers, and how to cope with what the framework may not offer.

This talk will describe the results of some of these experiments and what we learnt from them that might be useful in the design for Perl 6.

Issues to be covered may include, but may not be limited to:

- Viable Object models in a unified type system universe
- Component model vs. the module approach
- Garbage collection as a system service
- Language interoperability in a unified runtime environment
- The Portable Executable (PE) format for runtime object distribution
- Versioning of executable content
- Perl in the context of standardized runtime infrastructure (see `<http://msdn.microsoft.com/net/ECMA/>`)

How to configure and use Stem

Uri Guttman

Stem Systems, Inc.

Day 2, at 11:45, room L-219 — 45 minutes

Stem is a network application development toolkit and suite of applications. It transforms common network programming to simple configurations and makes uncommon network programming much easier. This talk will show you how to configure Stem modules to create network applications.

- Overview of Stem architecture, including Cells and Messages
- Creation of Stem Cells and Objects including argument parsing
- Stem Configuration including examples covering Logging, Sockets, Processes
- Using StemLog: a complete suite that manages logs across a network.
- Using StemInetd: a flexible emulation of the standard inetd daemon
- Chat server examples

- Details of various Stem modules including `Cron`, `SockMsg`, `Proc`, `Log` and `LogTail`

Introduction to SpeedyCGI

Sam Horrocks

ESM Services Inc.

Day 2, at 11:25, room L-26 — 20 minutes

This talk will introduce SpeedyCGI, a persistent perl interpreter, and will answer the following questions:

- What is SpeedyCGI?
- How do I use it?
- How does it work?
- How does it compare to other persistent perl interpreters like `mod_perl` and fastcgi?
- What is the future direction of SpeedyCGI?

Building speech applications with Perl and free software

David Huggins-Daines

Cepstral LLC

Day 2, at 14:45, room L-219 — 45 minutes

Computerized speech technology has now reached the point where speech interfaces are becoming practical and accessible to most computer users. However, until recently, sophisticated speech technology has only been available as proprietary software with limited APIs. Therefore, the integration of speech into free software systems has been limited.

With the release of the CMU Sphinx speech recognition system and the Festival speech synthesis system as source code under unrestrictive licenses, this situation has begun to improve. The bad news is that, as these have primarily been research systems until now, a great deal of work remains to be done in order to make them suitable for widespread deployment in desktop or server systems. The good news is that it is

already possible to write Perl language bindings for them, which is, after all, the most important step in the evolution of any software system!

In this talk, I will present the case for using Perl as the primary language for developing speech applications, followed by a short overview of the Perl speech extensions available at this time, and finally a few suggestions and demonstrations on how to put them to use in real-world applications.

Programming with Perl and C using Inline.pm

Brian Ingerson

ActiveState Corporation

Day 1, at 14:00, room L-232 — 2.25 hours

This tutorial is intended for anyone who wants to use C or C++ in their Perl programming. The easy way! If you have an intermediate understanding of Perl, and a thorough knowledge of the program `hello_world.c`, you are fully qualified for this class. If you need to speed up your favorite Perl script, make use of some existing C library, or write an extension module for the CPAN, this tutorial is for you. Even if you are a seasoned XS hack, Inline is a great way to make your code fathomable to the rest of us.

Inline.pm is a new module that glues other programming languages to Perl. It allows you to write C and C++ code directly inside your Perl scripts and modules. The Inline experience is similar to that of Perl itself. You simply write your code and run it. Inline will silently take care of all the messy implementation details and "do the right thing".

Attendees of this tutorial will learn: Syntax, usage, and configuration of Inline.pm • How to write Perl subroutines in C • How to expose external C functions to Perl • How Inline works. Including a detailed look at its Parse::RecDescent grammar • Basic Perl internals • How to use typemapping to modify/extend Inline's functionality • How to write complete extension modules for CPAN distribution • How to create precompiled binary distributions • How to use and create Inline-enabled modules like Event.pm • Object Oriented and CGI issues • How to bind Perl to C++ • Discussion of many concise yet functional examples.

Tutorial outline:

- Introduction

What – Where – Why

- Inline in Action

Simple Examples – Basic Syntax

- What about XS and SWIG

The Inline Philosophy – Pros and Cons

- Syntax & Configuration

The DATA Section – MakeMaker and XS Controls – Shortcuts

- Real Life Examples

5-6 Examples – Discussion – Q & A

- How Inline Works

The methods behind the magic – The Inline DIRECTORY – The Parse::RecDescent grammar

- More Advanced Examples

5-6 Examples – Introduce C++

- Extension Module

How write a complete extension – How to get it to CPAN – How to distribute it precompiled

- Conclusion

Summary – Future Plans – Resources

Pathologically Polluting Perl, with C, Java, and other Rubbish

Brian Ingerson

ActiveState Corporation

Day 2, at 11:00, room L-132 — 1.5 hour

No programming language is Perfect. Sometimes it just makes sense to use another language. Wouldn't it be great to use Perl most of the time, but be able to invoke something else when you had to? Inline.pm is a new module that glues other programming languages to Perl. It allows you to write C, C++, Python, Tcl, Java and Assembler code directly inside your Perl scripts and modules.

The Inline experience is similar to that of Perl itself. You simply write your code and run it. Inline will silently take care of all the messy implementation details and "do the right thing". It analyzes your code, compiles it if necessary, creates the correct Perl bindings, loads everything up, and runs the whole schmear. Now you can write functions, subroutines, classes, and methods in other languages and call them as if they were Perl. All in the privacy of your own script!

This talk is a whirlwind tour of the world of Inline. You'll see:

- Complete C extensions in one line of Perl!
- Perl glued to total strangers like Python and Java!
- How to run a plain C program without compiling it!
- How almost any programming language can be bound to Perl!

Data::Denter

Brian Ingerson

ActiveState Corporation

Day 2, at 14:20, room L-132 — 20 minutes

An overview of Data::Denter by its author.

CPAN, PPM & the Future

Brian Ingerson and Neil Kandalgaonkar

ActiveState Corporation

Day 1, at 11:00, room L-132 — 45 minutes

ActiveState's Programmer's Package Manager is familiar to ActivePerl users as a convenient tool for module installation and management.

Lately, there have been big changes in PPM; it is now backed by a semi-automated testing suite that tries to build every package on CPAN on various operating systems. It has also been extended to other languages such as Python, and much of the package information can be gleaned from ActiveState's Programmer Network website (ASPN).

In this talk we'll detail the current state of PPM, its relation to CPAN, and our plans for the future. Among our goals are to make it even easier to use and make repositories, and give the user hard metrics that help them trust repository code.

By the time YAPC rolls around we will have some interesting demos and we'd like to get feedback from both CPAN authors and PPM users.

Run-time Generation of JavaScript Code by Perl CGI Programs

Stephen B. Jenkins

Institute for Aerospace Research, National Research Council of Canada

Day 3, at 14:45, room L-26 — 20 minutes

Run-time generation of JavaScript code by Perl CGI programs is a technique that has been used with considerable success at the Aerodynamics Laboratory of the Institute for Aerospace Research, National Research Council of Canada.

After a brief introduction to the basic concept of dynamic code generation and its role within the data acquisition software of the 2m by 3m wind tunnel, four specific examples will be presented each highlighting a different application of the technique.

The first and simplest example will show the generation of trivial one-line JavaScript programs.

The second example will show the run-time generation of JavaScript subroutines that modify groups of HTML checkbox elements.

The third example will cover form validation in the browser, complete with pop-up alert boxes and dynamic images to prompt the user.

The final, and most complex case, will show the creation of JavaScript code that produces dynamic behavior in HTML forms. These examples not only demonstrate the primary benefit of the technique - a more interactive user interface - but also show the secondary benefits: reduced load on both the web-server and web-client computers.

XML-RPC: Web Services Made Easy

Joe Johnston

Day 3, at 12:10, room L-26 — 20 minutes

Web services allow disparate systems to communicate with each other. Before .NET was a glimmer in a Microsoft marketer's eye, there was Dave Winer's XML-RPC. Speaking alphabetically, it is simply RPC in XML over HTTP. What that means to Perl programmers is that with XML-RPC, any perl program can access resources written in any other language on any other platform. Not only that, but Perl programmers can make their libraries accessible to users of sub-Perl languages. This talk will spill the beans on Perl's XML-RPC library, `Frontier::RPC`, in all its client/server glory and present a simple web service application.

Programming Parrot or, Integrating Perl with Python

Neil Kandalgaonkar

ActiveState Corporation

Day 3, at 09:45, room L-132 — 20 minutes

This will be a survey of various techniques for integrating code written in these two similar but fundamentally different scripting languages.

Most people only know ActiveState as the "Perl for Windows" company. So it may come as a surprise that the presenter faces daily battles of tying together as much Perl as Python code, on both Linux and Windows platforms. Real-world examples will include: use of Perl in the Python-based Zope web application server, and Web Services as an answer to system integration.

If time permits, the talk will conclude with some speculative ideas on how useful Python language concepts can be "borrowed" into Perl.

“=~?” (or: How We Learned to Stop Worrying and Love the Binding Operator)

Jason Lee

Legal Information Institute, Cornell University (Ithaca, NY USA)

Day 2, at 14:20, room L-26 — 20 minutes

For centuries, the measure of a lawyer was the size of his bookshelf. The entire body of law was kept in nice, solid leatherbound books that generally sat on a shelf and looked suitably pretentious.

With the rise of the Internet, electronic documents have joined their ink-and-paper counterparts. However, the legal profession has been slow to adopt the electronic medium, relying on third-party sources to make selected legal information available online–often through proprietary interfaces and fee-based services. The Legal Information Institute at Cornell Law School seeks to provide an alternative by expanding the body of legal information publicly available online.

The goal of the Legal Data Markup Software (LDMS) student project at Cornell University was to create a tool for the Legal Information Institute that could markup the United States Code provided by the House of Representatives. This would have gone in the books as Yet Another Student Project, had it not been for two factors: first, the project took advantage of Perl’s extensive regular expression syntax to markup structurally inconsistent legal text as XML. Second, half the developers didn’t know Perl or XML to begin with.

This combination made for an interesting adventure.

This talk traces the history of this project, covering both the problematic structure of the U.S. Code and the team’s introduction to Perl before plunging into the murky depths of the CVS repository to track the evolution of the project. The result? A discussion about using Perl to mark up hard-to-parse text, comments about teaching Perl on-the-fly, lessons learned from using Perl as a platform for iterative software engineering, and perhaps just a few instances of infinite recursion.

PHP for Perl Programmers

Urban LeJeune

America’s Town Square

Day 3, at 09:00, room L-219 — 1.5 hour

Should you learn yet another scripting language? Like most computer related questions the answer is, “it depends.”

If you’re creating casual or non-production programs, by all mean employ the language having your highest level of proficiency. When creating production or computationally intensive programs, use the most

powerful language, even if it requires learning a new skill set.

PHP is the only language that was designed specifically for Internet related applications. Perl is an adaptation, albeit a good one. In addition, for most Web related applications, PHP is faster and in some case substantially faster than Perl.

The good news is that Perl programmers can obtain PHP programming parity with a relatively flat learning curve. Both Perl and PHP are procedurally oriented languages sharing a common heritage. In many ways PHP and C are closer relatives than Perl and C. Can't remember PHP syntax? Try the C or Perl equivalent and it works more times than not.

Transitioning from one procedurally oriented language to another is essentially a four-part process. This tutorial will focus on these four sections:

- What does PHP have that Perl does not?
- What does Perl have that PHP does not?
- What does PHP do differently than Perl?
- What common traps do Perl programmers face when switching to PHP?

YAPC Town Meeting

Kevin Lenzo

Yet Another Society

Day 3, at 16:25, room L-132 — 20 minutes

A discussion of YAPC and YAS for one and all.

Any2XML - Extract gold from Ore!

Michael MacHenry and Michael John Burns

Reliance Technology Consultants Inc.

Day 3, at 14:00, room L-26 — 20 minutes

A tool being developed in Perl to assist conversion of any text file to an XML file given a template containing transformation rules using regexes. The topic will be presented by Michael Machenry (Northeastern (MA) sophomore) and Michael John Burns (Wakefield (MA) High junior) with an introduction by Ranga Nathan.

For some time Ranga Nathan has been dreaming about a generic routine (Parser is too big a name for this) to help extract useful data from reports produced by IBM mainframe applications. Rather than write one routine for each type of report, would it not be nice to have one routine that can accept a regex based template? `Any2XML` was born. Eventually it will have a user-friendly front-end to enable generation of the template using mouse operations over a text file.

Orchard - A simple alternative to XS

Ken MacLeod

Day 2, at 14:00, room L-132 — 20 minutes

Orchard is the technology underlying a new family of high performance modules for the Perl-XML and Python-XML scene. However it is not limited to use for XML modules (despite having a built-in XML parser). Orchard uses pre-processed C, called Mostly-C (MOC), that provides a simple to use object attribute accessor syntax for programming object oriented XS modules in Perl. It features dynamic method dispatch, garbage collection using Boehm GC, transparent interface to the host language (Perl and Python currently, with Ruby planned), XML-style namespaced attributes and method names. Orchard still retains all the features of C for those who need them. As a bonus, MOC files written for Perl work with the Python interface too.

How to submit a documentation patch

Walt Mankowski

Myxa Corporation

Day 3, at 11:00, room L-219 — 20 minutes

Think Perl's documentation stinks? In this talk, you'll learn how to make it better. We'll go step by step through the process of creating and submitting a Perl documentation patch. We'll discuss how to find documentation that needs patching, issues to consider when making the patch, and how to submit it.

Outline of topics to be covered:

- Finding documentation that needs updating
  perlbug
  USENET
  perlmonks
  #perl
  Perl beginner's mailing list
  coworkers
- Is the patch really necessary?

- Updating the documentation

Which version of Perl should I patch?
Perl's documentation standards
Updating test cases (if necessary)

- Preparing the patch

Context vs unified patches
makepatch

- Testing the patch

Applying the patch locally

- Submitting the patch

Introduction to p5p
What a patch email should contain
Mailer issues (word wrap and mime)

- Followup

What happens if it gets accepted?
What happens if it gets rejected?
The pumpking is always right

- Related topics

Patching core modules
Patching CPAN modules

- More information

perldoc perlhack
perldoc patching
perldoc perlpod

Building Web Sites with PageKit

T.J. Mather

AnIdea Corporation

Day 2, at 11:00, room L-26 — 20 minutes

`Apache::PageKit` is a `mod_perl` based application framework that uses `HTML::Template` and XML to separate the programming, content, and presentation. It follows a MVC (Model/View/Controller) approach to design.

This talk will demonstrate how it solves many of the common problems of a web programmer, and allows for rapid development of dynamic web applications with reusable components. Many of the built-in features of Pagekit will be covered, including language localization, authentication, form validation, co-branding, and session management.

Learning Perl

Tad McClellan

Stonehenge Consulting Services, Inc.

Day 1, at 11:00, room L-26 — 4 hours

An extended introductory Perl tutorial given by Tad McClellan from Stonehenge Consulting.

Application programming with Mason

Michael McClennen

Day 3, at 09:00, room L-232 — 3 hours

Mason is a perl-based, open-source package for building web-based applications. It provides a framework in which one can define a hierarchy of components that call each other to produce interactive web pages. Each component can contain a mixture of Perl code with HTML (or other text) that is automatically compiled into Perl print statements. Mason is superficially similar to EmbPerl, PHP, and other such packages. However, it provides considerably more functionality. Several high-visibility, high-traffic websites are powered by Mason, including `<http://www.techweb.com>`, `<http://www.salon.com>` and `<http://www.bizland.com>`. At the same time, Mason is surprisingly easy to use. It makes easy jobs easy, and hard jobs possible! (now where have I heard that one before?)

The tutorial will cover the following topics:

1. structure and syntax of a Mason component
2. parameter passing
3. organization and configuration of a Mason application
4. debugging
5. session handling
6. web server interface

Each of these topics will be discussed in the context of a demonstration application, the source code for which will be included in the course notes.

More information about Mason can be found at: `<http://www.masonhq.com/>`.

The presenter is not affiliated with the Mason project, but is a very satisfied user. `:-)`

Slash: Taming the Beast

Chris Nandor

Day 3, at 14:00, room L-132 — 45 minutes

Slash, the code that runs the popular “News for Nerds” site, Slashdot, started as a small bit of Perl code running on a shared web server. Slashdot quickly grew, and the code did, too. It was not long before the code became hard to manage. It needed to be rewritten. First, the code had to be cleaned up; global variables, hardcoded HTML, SQL interspersed with Perl, lack of coherent API, and other problems needed to be straightened out. Further, there was a lot of room for optimization, taking advantage of many of the features `mod_perl` and Apache have to offer.

This session will go over the architecture of the system (LAMP: Linux, Apache, MySQL, Perl), including how we solved the issue of having multiple database backends, moved to a template system to make customization easier, and created a plugin structure to make the system’s functionality easily extensible.

This is the story of what was wrong with Slash, how its problems were fixed, and how it was made better.

COBOL Explorer - Perl’s gift to COBOL

Ranga Nathan

Reliance Technology Consultants Inc.

Day 3, at 11:25, room L-26 — 20 minutes

Mainframe world, why Fortune 1000 companies run on COBOL, the most pervasive and ancient language. COBOL programs are compiled to produce a huge listings which are printed or viewed via 3270 terminal for debugging. Why not render the listings as web document for easy navigation, asked Ranga Nathan. Looking for a good language to implement this, he accidentally discovered the beauty of Perl.

State & County QuickFacts – Bringing Census Data to Everyone

Lisa Nyman

US Census Bureau

Day 2, at 14:00, room L-26 — 20 minutes

The Census Bureau is a large agency devoted to collecting and publishing statistical data about the people and economy of the United States. Last year, the Census Bureau used Perl to collect Census 2000 forms online. Today, in the reporting phase of Census 2000, Perl plays a critical role in distributing Census data online, helping the Census Bureau to meet the expectations of visitors to the Census web site, the primary source of Census 2000 data.

The many different statistical programs at the Census Bureau publish data sets online, each focusing on specific surveys. Prior to the State & County QuickFacts Program, no single web site offered timely, multiple Census data items by geographic area instead of by survey program. With QuickFacts, the most commonly requested statistics are now available in a simple, usable application. Now, novice users interested in community profiles can find the information they need quickly and easily, without prior knowledge of Census data, formats or programs. Seasoned users are satisfied with the detailed metadata and easy access to more complete data sets.

Started as an unfunded project within the Census Bureau, State & County QuickFacts and has become most heavily used single-purpose application on the Census website. Using Perl and Open Source software, the small development team was able to focus on creating a responsive, innovative web site quickly, instead of wrestling with technology. Lessons learned from building QuickFacts can help Perl and Open Source advocates to promote Perl for building a solid technology infrastructure, allowing developers to concentrate on the less tractable aspects of web site development.

Writing a Perl Book

Clinton A. Pierce

Decision Consultants, Inc.

Day 3, at 11:45, room L-26 — 20 minutes

A brief tour of what goes on writing a book for a large publisher. Some topics covered are generating ideas, finding a publisher, writing the text, finding help, technical editing, timelines involved, the kind of money to be made (or not made) and so on.

Demystifying Regular Expressions

Jeff Pinyan

RiskMetrics Group, Inc.

Day 3, at 14:45, room L-232 — 45 minutes

I'll pick apart the workings of the `YAPE::Regex` and `YAPE::Regex::Explain` modules, showing how to parse a regular expression, and how this could possibly be useful. Then I'll show the `YAPE::Regex::Reverse` module, an attempt at making regexes efficient for matching towards the end of a string.

e-smith server and gateway

Kirrily Robert

e-smith

Day 2, at 09:00, room L-219 — 45 minutes

The e-smith server and gateway is a specialised Linux distribution suitable for small-to-medium enterprises or home networks. One of its great strengths is its modular, extensible architecture, which uses Perl to trigger events and actions, expand templated configuration files (all of `/etc` handled by `Text::Template`!), generate a sophisticated web based administration system, and more.

This talk will discuss the use of Perl in this product, and showcase some of the cooler parts of the code.

Reefknot

Kirrily Robert and Shane Landrum

e-smith

Day 2, at 11:45, room L-26 — 20 minutes

The Reefknot project is an attempt to create a toolkit for developing RFC-compliant calendaring and scheduling applications in Perl.

Currently, the open source calendar field is a mishmash of mutually incompatible applications. Many web-based calendar apps exist, but none of them comply with RFC2445 (the iCalendar specification). On the desktop, the only RFC-compliant calendar application is Ximian's Evolution, which is not quite ready for prime time.

In theory, any RFC-compliant calendar app should be able to interoperate with any other. In practice, this will probably require a certain amount of glue. And what better glue language than Perl?

The Reefknot project's first task has been to reimpliment `Net::ICal` (previously a SWIG-generated port of libical, and not very friendly to use) in pure, idiomatic Perl. This is progressing well, and already people are beginning to develop small applications for demonstration purposes, including `Gtk` todo-list managers, web calendars, and a script to refuse any meeting invitations received by email.

The next step will be to allow programmers to avoid most of the tedium of calendar-parsing by creating more abstract interfaces. This may involve abstracting the way we access a calendar store (which might be on the filesystem, or in a database, or out on the 'net somewhere), how we query it, and how we display it (as a web page, or as text, or in a GUI).

Ideally, the resulting Perl modules will allow programmers to easily create calendar applications with code no more complex than this completely made-up example:

```
use Reefknot;
my $cal = new Reefknot(store => $store_url, who => $email);
print $cal->week_as_html;
```

It is our hope that people will be able to use Reefknot to easily create calendar and scheduling systems to match the functionality (if not the marketing budget) of the current market leader – Microsoft Outlook/Exchange.

CGI::FormMagick

Kirrily Robert and Shane Landrum

e-smith

Day 2, at 12:10, room L-26 — 20 minutes

`CGI::FormMagick` is a toolkit for generating complex web form applications with minimal actual code on the part of the FormMagick user. It parses an XML description of the application, and can generate a multi-page "wizard" form complete with session handling, data validation, internationalisation, and more, in as little as three lines of Perl:

```
use CGI::FormMagick;
my $fm = new CGI::FormMagick(TYPE => 'FILE', SOURCE => 'myfile.xml');
$fm->display();
```

FormMagick relies heavily on various CPAN modules for much of its functionality, including `CGI::Persistent` (CGI session handling), `XML::Parser` (parsing XML input files), `Locale::Maketext` (internationalisation/localisation) and `Text::Template` (templated headers and footers to modify the look of the generated webpages).

FormMagick's input consists of `xml` describing a `FORM` which can contain one or more `PAGE`s. Each page can contain one or more `FIELD` elements. These elements have XML attributes to provide them with unique IDs or names, descriptions, types and validation routines for the fields, and so on.

A variety of validation routines are built into FormMagick. The FormMagick user need only provide a `VALIDATION` attribute on a `FIELD` element to specify which routine should be used. Examples include `nonblank`, `ip_address` and `credit_card_number`. Programmers can also add their own validation routines or override FormMagick's built-in ones.

Localisation is the last major feature that needs to be described. FormMagick programmers can easily add translations into other languages by calling the `add_lexicon()` method, which takes a two-letter abbreviation of a language and a reference to a hash of translated phrases. The end-user's preferred language is picked up automatically from the browser.

Perl and XML

Michel Rodriguez

IEEE Standards Activities

Day 1, at 11:00, room L-219 — 1.5 hour

This tutorial starts with an introduction to XML, its syntax, what it is good for, examples of XML applications and a quick overview of related standards. Then it goes over some of the most common XML modules for generating XML, from scratch or from a data base. The most important section comes last: how to process XML with Perl, with an introduction to generic transformation models and a review of the most common modules: `XML::PYX`, `XML::Simple`, `XML::Parser`, `XML::DOM` and `XML::Twig`, all with code examples.

Alzabo, a Data Modeller and RDBMS-OO Mapper

Dave Rolsky

Day 2, at 09:45, room L-219 — 45 minutes

Alzabo is a tool designed to ease the pains of dealing with a database. It serves several functions. First, it is a data modelling tool. You describe your schema (tables, columns, indexes, relations, etc.) via a data modelling interface and then Alzabo can generate the SQL necessary to instantiate your schema. It can also reverse engineer an existing schema from a database. Finally, it is capable of generating the SQL necessary to transform an existing schema based on changes made in the data modeller (an SQL diff, if you will).

As an RDBMS-OO mapper, Alzabo is capable of abstracting most types of SQL queries. It has several efficiency features such as lazy column loading, cursors, and row caching. It is also capable of auto-generating various convenience methods to simplify programming.

The presentation will cover:

- Alzabo's web-based schema creation interface.
- How to perform basic data fetching/alteration (insert/select/update/delete). How to construct joins and conditionals.
- The row caching system and how it can be used to keep objects in sync across multiple processes (particularly applicable under `mod_perl`).
- Lazy column loading.
- Method auto-generation.

Disciplined Programming or How To Be Lazy Without Really Trying

Michael G Schwern

Day 2, at 14:00, room L-232 — 1.5 hour

Target audience Programmers who are comfortable with a language and project managers. For those who are having difficulty maintaining quality and flexibility in larger projects, meeting deadlines and staying happy. A working knowledge of any modern programming language will be very helpful (espcially Perl).

What they will learn How to think about programming as an engineering discipline A set of software engineering heuristics and techniques will be shown to reduce bugs, risk and stress as well as improve maintainability. Above all, it will be made clear that there is a better way.

First half: **Defensive Programming** The first half of the tutorial will be about improving code construction, the actual act of putting code on the pavement. Its primarily for the programmers in the audience, but managers will find it enlightening to know what their charges are up to. Perl will be used for the examples, but the principles and techniques will be applicable to all languages. Topics include (in no particular order): Thinking about the next guy–writing for maintainability • Name calling–good variable and routine names • Version control • Bug tracking • Take a typing class!–stopwritingcompressedcode • Build your personal tool library • Clear interfaces • Personal style vs good style–reconciling the two • Writing testable code • Separation of form and functionality • Building up from small blocks • Writing good subroutines • Encapsulation • Writing good tests • Writing good docs • Assert yourself!–executable comments • Testing GUIs • strict, warn & taint • Rules of optimization • Fight Entropy!–refactoring and other ways to recover bad code • Ineffective Idioms–common mistakes and how to correct • Books and references

Second half: **Defensive Management** The second half is about rethinking management of your software projects, both on a personal level and on a project level. This is for both the programmer and the manager. Topics include (in no particular order): Build and smoke–test every day • Pairing–don't walk alone • Software engineering statistics and metrics • Quality must be a design decision • Handcuffing–getting the programmer and the expert talking • Spiral Development–rapid releasing every week • Skunk Works–lessons from Lockheed • Morale and Motivation–keeping your programmers happy • Force Multipliers–making one programmer worth ten • Eye On The Ball–adjusting your goal mid-project • Selling it all–convincing your programmers and boss that this is all worth the trouble • Code reviews • Hiring Practices–getting quality programmers • Out of the ballpark–schedule estimation and negociation • History Lesson–learning from failure and success • Mini-milestones/iteration–estimates you can count on • Change control–managing changes and feature creep • Death March–common causes of project failure • Risk Management • Recovery–what to do when you're late • Books and references

Pod::Tests–Embedded Testing

Michael G Schwern

Day 3, at 11:25, room L-219 — 20 minutes

Its long been known that the closer your documentation is to the code it documents, the more likely it will be kept up to date and the more accurate it will be. Perl has POD for this purpose, allowing us to intermingle docs and code as we please. But what about tests? Shouldn't a test be close to the code its testing?

`Pod::Tests` and the accompanying `pod2test` squeeze one more feature out of POD by allowing tests to be embedded right next to the code its testing and the documentation its validating! As an added bonus, its also possible to test example code in your documentation. And with no performance penalties!

We'll show how this was pulled off, how it is used in concert with traditional tests, how it is made to work *without* `Pod::Tests` installed, and how it makes the unglamorous job of writing tests a bit more palatable than licking all the bathrooms in Grand Central Station.

Special Secret Surprise Session

(It's a secret!)

Day 1, at 11:00, room L-232 — 1.5 hour

To be announced.

Open Source DB for Enterprise

Indy Singh

NuSphere Corporation

Day 2, at 14:45, room L-26 — 20 minutes

The costs of doing e-business at "Internet speed" have contributed to several high profile dot bombs. Stakeholders and senior executives are demanding efficient, prudent spending on IT initiatives more than ever before.

The battle lines within an e-business are often drawn between the CTO and CFO. Can open source applications cut costs while still delivering reliable, secure daily operations? This session, presented by the lead developer that established MySQL as a transactional database, will explore viable open source options for executives and developers who work within fiscal constraints.

Attendees will learn how enterprises are embracing the open source movement for database infrastructures as a natural offshoot of the well-known open source OS arena. The presentation, given by D. Britton Johnston, will introduce case studies including NASA, OpenAir.com, University of Arkansas and other current implementations. Participants will learn how to prepare a cost benefit analysis for a possible switch to open source database infrastructure. In addition, the seminar will discuss realistic expectations for open

source within mainstream companies and what professional support is now available to IT departments.

Perl 5 Internals

Nathan Torkington

O'Reilly and Associates

Day 2, at 09:00, room L-232 — 3 hours

Learn how `/usr/bin/perl` compiles and interprets your programs. At the end you'll be able to follow most `perl5-porters` discussion, and will have all the groundwork you need to start hacking the Perl interpreter.

What's the big deal about P2P and Web Services?

Nathan Torkington

O'Reilly and Associates

Day 3, at 10:10, room L-132 — 20 minutes

"Web services" and "P2P" are the latest buzzwords out of the Silicon Valley PR machine. But underneath the layers of marketing bullshit, there are actually some useful ideas and directions. I'll separate the interesting stuff from the marketese, and then show the opportunities for Perl.

Perl 6 Meeting

Nathan Torkington

Day 3, at 14:00, room L-219 — 1.5 hour

Perl 6: What's happened so far (RFCs, Apocalypses, Exegeses), Some sample programs. Why are we making these changes?

Perl Apprenticeship Hour

Adam Turoff

Day 1, at 11:45, room L-132 — 45 minutes

Format: A series of brief presentations, much like Mark-Jason Dominus's Lightning Talks **Goal:** Improve Perl by spreading the knowledge and the workload

Many members of the Perl community have large numbers ideas for Perl modules, tools and services that they personally do not have time to design, implement or build. Sadly, many of these ideas sit ignored on a TODO list awaiting for a neverending supply of tuits to be found. The Perl Apprenticeship Hour is one attempt to remedy this situation. Each presenter will describe an interesting project s/he does not have time to start or complete. Projects may include things like:

- documentation
- tutorials
- modules
- websites
- test code
- tools
- bugfixes
- enhancements to existing code
- collaborative efforts
- program suites

For each project described, a presenter may offer that project to a willing apprentice one of the following terms:

1. You've heard my idea, do something with it; it's yours.
2. You've heard my idea, you have enough clue to understand what needs to be done, now go do something interesting with it.
3. Here's my idea, I don't have time to implement it; if you understand the fundementals, I'll be available to answer your questions.
4. Here's my idea, I don't have time to implement it. If you have enough time, energy and clue to do something with it, then do something great; I can't answer your questions.

Proposals will be solicited from the Perl community. Each presenter will be given a brief amount of time (about 5 minutes) to present a project, some issues associated with that project, and briefly answer questions if desired. Members of the audience interested on working on a project presented during the Apprenticeship Hour are strongly encourage to discuss their commitments with the presenter offline.

Web Services and Perl

Adam Turoff

Day 3, at 09:00, room L-26 — 45 minutes

As the web applications move away from browser-based dynamic web page generation and towards interoperable peer-to-peer services, new technologies are needed to keep the web moving forward. Standards such as XML-RPC, SOAP, and WSDL fill these needs.

This talk introduces the metaphor of peer-to-peer computing by revisiting IPC and RPC, while highlighting the new twists provided by modern web-based RPC protocols. The major protocols, XML-RPC and SOAP, are contrasted highlighting the strengths and weaknesses of each. Practical web service applications are demonstrated, including simple Perl-based SOAP clients and servers using `SOAP::Lite`.

Shunning XML

Adam Turoff

Day 3, at 11:45, room L-219 — 45 minutes

Since its introduction in the late 1990's, XML has added both excitement and hot air to the technology arena. Proponents of XML intend it to be be the one last data format for everything, including online purchases, print and online publications, system configuration files and quite possibly your birth certificate. Unfortunately, with all the hype about interoperability, large questions remain unanswered, including, "Is this XML appropriate?", "Is XML really interoperable?" and "Is XML meeting our expectations, or just inflating our budgets?"

This talk will discuss some of the failures of XML, addressing some of the inherent problems with XML, and discuss some workable alternatives for those problems, specifically focusing on using Perl and CPAN. This talk will further show that in some common cases, using Perl programs and modules in conjunction with a simpler data format can solve problems much more naturally than XML-centric approaches.

A Talk by Larry Wall

Larry Wall

Perl

Day 1, at 09:00, room L-132 — 1.5 hour

Yes, a talk by Larry Wall.

Using AI::Categorize to classify documents

Ken Williams

Day 3, at 14:45, room L-132 — 45 minutes

I will discuss my new modules in the `AI::Categorize` namespace, which provides mechanisms for automatically categorizing text into categories. The categories are learned from a set of existing categorized documents. This can be useful for ask-an-expert services (the motivation for my work), information retrieval, or perhaps even mail filtering.

The `AI::Categorize` modules are intentionally simple and were written as a means of learning and showcasing categorization techniques. Anyone who needs to do a really good job categorizing documents will probably need to pay lots of money for a professional categorization system, or a team of human experts in the subject matter of the documents, or a categorization expert. Probably all three. But anyone curious about the machine learning algorithms employed may be interested in `AI::Categorize`.

So far two algorithms are implemented, Naive Bayes Probability and k-Nearest Neighbor. Both implementations currently use MySQL to store and manipulate the training data, which is probably slower than using custom flat files but makes it easier to write new algorithms and allows for easy inspection of the data.

Also included is an `AI::Categorize::Evaluate` module, which helps compare the accuracy of various categorization algorithms.

OpenInteract, An Extensible Web Application Server

Chris Winters

Day 3, at 09:45, room L-26 — 45 minutes

OpenInteract is an extensible application server using Apache and `mod_perl`. It is built as a framework for developers but also to be manageable almost entirely via the web browser. It includes:

- database-independent, object-oriented data access,
- security at both the application and individual object levels,
- user and group management,
- template-based presentation, and
- a package management system that makes it simple to distribute new modules, table structures and data.

OpenInteract is a stable but growing package and is currently available on CPAN. In addition to `mod_perl`, it also makes good use of other CPAN modules like `SPOPS`, Template Toolkit, `Apache::Session`,

`Apache::DBI` and a host of others.

The presentation, by the primary developer of OpenInteract, will cover system architecture and provide a quick but detailed look at what goes into a package and how it gets installed and distributed.

ProtoWrap: Using Wrappers to Protect Specific Network Services

Gunnar Eyal Wolf Iszaevich

Universidad Nacional Autónoma de México, Campus Iztacala

Day 3, at 14:00, room L-232 — 45 minutes

In this work I address some of the currently most dangerous security issues for Internet servers which are responsible for most of the attacks suffered by servers every day: Protocol misuse and buffer overflows. Both problems can be addressed with a single approach, which will be explained and built in the following pages: Protocol-specific TCP service wrappers.

Having worked since November 1999 in UNAM's Computer Security Department and after over three years of being a security-conscious UNIX system administrator, I found there was a great need for such a product. History has proved that most successful attacks manage to break into a system exploiting buffer overflows. This kind of attacks are usually not hard to patch at the source code level, but sometimes that is not enough. I saw the need for a proactive defense scheme to save the administrators from the disable-download-patch-enable-pray cycle, building a wrapper immune to buffer overflows, or at least a wrapper which would not compromise the entire system if successfully attacked.

When comparing the programming languages with which to do this project, using Perl was chosen over C and C++ because, albeit being a slower, non-compiled language, it has many desirable characteristics, notoriously the ability to manage the memory without exposing the programmer to make a critical mistake in such a critical part, and powerful pattern recognition techniques.

It should be noted that this work is a proof of concept; a working implementation of the suggested ideas, but it should be viewed more as an invitation for other programmers to continue working on this program or even reimplement it if a better way to do so is found than as a finished product.

This work is being done as a final paper for Computer Science B.Sc. at the Kennedy-Western University; the full text can be found at `http://www.gwolf.cx/wrap`.