Catalyst:
MVC Web Framework for Perl

Philadelphia Linux User's Group

- Catalyst
- MVC
- Modern Perl
- Moose
- Template Toolkit
- DBIx::Class

Web Application Development Framework

- The goal of web application frameworks is to simplify the building of web applications and automate the repetitive tasks.
- A key objective of application frameworks is the DRY principle: Don't Repeat Yourself.
- A popular architecture for Web Frameworks is MVC: Model View Controller.
- Ruby on Rails and Django (for Python) are popular MVC Frameworks.

So Why?

- Frameworks require an investment in learning.
- The return is that it is going to be easier to develop and maintain a web based application.
- This is significant versus writing from scratch or cut and paste from previous projects.
- A modern web application typically involves database access and collaboration with web-designers (graphic artists).
- Authentication and Authorization are typically absolute musts, and require quite a lot of work to implement.

So Why?

- A good template system allows the graphic designer to create the site independently of the programmers.
- Moving database operation away from your central logic, makes coding much easier.
- Plugins make it much easier to tackle difficult issues like Authentication and Authorization.

Other Perl Frameworks

- Titanium (formerly CGI Web Application) is the oldest active Perl Framework.
- It does not implement the MVC paradigm.
- When choosing a framework I got the impression that Titanium has its' following, but has a smaller community than Catalyst or Jifty.
- Although Titanium/ CGIWebApp is less complex than Jifty or Catalyst, I could not find a tutorial or guide for beginners, so I didn't take a serious look at it.

Jifty

- Jifty is another MVC framework.
- It imposes many choices on the user/developer.
- Jifty has the templating system hard coded in and has been shifting from Mason to Template Declare.
- Later on I will demonstrate how easy and powerful Template::Toolkit is, and this choice was a key point in my decision to work with Catalyst.
- Jifty is like Rails in that it limits developer options, this goes against the Perl Mentality of there being multiple ways to accomplish any given task.

Catalyst

- Catalyst is a mature solid framework with a community strong enough to have moved the entire project over to Modern Perl in a short period of time.
- Overall it is fairly well documented and there are even two published books on it.
- The best document (but not a terrific one) for starting is the Tutorial in the Catalyst Manual which you can download from CPAN.
- Starter Documentation is weak, and makes it harder to learn than it need be.

Pre-Requisites

- A Unix-like OS is recommended.
- You should have either Perl 5.8.8 or 5.10.1 installed.
- Perl 5.12.0 hasn't been tested enough.
  Perl 5.10.0 has known bugs.
- You should be comfortable with the material in Intermediate Perl (O'Reilly 978-0-596-10206-7).
- You should be familiar with Moose, Template::Toolkit and DBIx::Class.
- Have patience for a lengthy CPAN session.

Modern Perl

- Perl 6 is Vaporware, but for a long time waiting for Perl 6 slowed down Perl 5 development.
- Perl 6 features are now being ported to Perl 5, while development activity on Perl 5 has resumed.
- The Moose Module brings a modern OO option to Perl 5, and is a central pillar of Modern Perl.
- A new relationship is emerging where Perl 6 is becoming the idea lab, and Perl 5 the production line.

Moose

- Moose is implemented as a regular perl module, and has even spawned a couple of clones.
- With Moose, even I'm writing (or trying hard to write) object oriented code.
-

Moose Provides

- object declaration
- a new() method
- type checking
- the ability to extend other (even non-Mouse) modules.
- Type checking is currently limited to the new method, and does not provide a type constraint beyond the moment of creation.

Bonjour 1

```
package Bonjour ;

use Moose ;

has 'language' => ( isa => 'Str', is => 'rw', default =>
'Francais', ) ;

has 'hellostring' => ( isa => 'Str', is => 'rw', default =>
'Bonjour Monde!', ) ;

sub Hello {

  my $self = shift ;

  say "The Way to say Hello World in $self->{ 'language' } is:
  $self->{ 'hellostring' } " ;  }
```

Bonjour 2

```
Use Bonjour ;

my $bonjour = Bonjour->new() ;

$bonjour->Hello() ;

my $deutschehello = Bonjour->new( 'language' => 'Deutsche' ,
'hellostring' => 'Hallo Welt!' ) ;

$deutschehello->Hello() ;

$bonjour->{ 'language' } = 'English' ;

$bonjour->{ 'hellostring' } = 'Hello World!' ;

$bonjour->Hello() ;
```

Template Toolkit

- Catalyst supports all of the templating systems available for PERL. TT is the default
- I downloaded an html template, and changed the extension to .tt.
- Here are some lines snipped out of aqueous.tt.

```
<title>Hello World in:    [% language %]</title>
...
      <h2>[% language %]</h2>
      <h3>[% language %]</h3>
...
      <h2 id="Intro"><a href="http://www.free-css.com/">[%
language %]</a></h2>
      <p>[% message %] [% message %] [% message %] [% message %]
[% message %] [% message %] [% message %] </p>
```

aqueous.pl

```
use Template ;
use BonJour ;
my $bonjour = BonJour->new() ;
my $vars = { language => $bonjour->{ 'language' }, message =>
$bonjour->Hello() , } ;
my $tt = Template->new() ;
$tt->process( 'aqueous.tt', $vars) ;
```

- It's easy.
- TT has IF statements and Loops.
- For a new project there is no reason to choose another Templating system.

ORM: Object Relational Mapping

- Object Relational Mapping is trying to do for Database Access what Template::Toolkit did for outputting webpages.
- ORM abstracts away from the underlying database.
- ORM creates, in effect, a "virtual object database" that can be used from within the programming language.
- Most Catalyst Developers choose DBIx::Class, Rose::DB is the principal alternative.

DBIx::Class (DBIC)

- I don't like DBIC.
- I like DBIC so much I wrote IhasQuery. IhasQuery is not yet ready for CPAN, but you can find the development release on my website.
- Because of the popularity of DBIC, many plugins rely on it. This makes it unavoidable. You will have to use it at least some of the time.

DBIC

- DBIC requires a lot of effort to learn.
- DBIC syntax looks nothing like SQL.
- DBIC requires extensive Schema definition.
- DBIC has a lot of overhead.
- Every time any change is made to a database, the Schema needs to be rebuilt if you are auto generating it, or you need to manually edit the Schema to reflect any changes that impact your application.

Alternatives to DBIC: RoseDB

- Proponents of Rose claim better performance.
- Rose appears to have better documentation, but digesting the book sized tutorial for Rose::DB::Object, isn't all that appetizing.
- Rose is an ORM and provides a virtual object database like DBIC.
- Rose does not appear to require the kind of Schema definition that DBIC does.
- While I've looked at Rose Documentation I haven't tried to use it.

Alternatives to DBIC: IhasQuery

- I must be an incompetent CPAN searcher, because I find it hard to believe that there aren't already hundreds of similar modules.
- Hasn't been released to CPAN, yet.
- Each table is an object, that is the extent of Schema Mapping required.
- The current development release only provides select, insert, update, delete, count, and a simple where clause.

Alternatives to DBIC

- SQL::DB. Is more SQLike. I didn't get to anything functional in the time I gave it.
- Fey. Fey is also more SQLike than DBIC, I got farther with it than SQL::DB.
- Both Fey and SQL::DB require predefinition or schemas.
- Neither had a good quickstart or tutorial.
- You can use DBI to build your own Model.
- DBI Models violate the DRY principle.

Let's get to Catalyst

- Once you have Catalyst installed, it is easy to begin a new project.
- catalyst.pl TestApp.
- A new subdirectory is automatically created containing the skeleton of your application, in cpan format.
- In the new applications directory type: ./script/testapp_server.pl
- Connect to localhost:3000

Model View Controller

- Catalyst implements Model View Controller. If you browse TestApp/lib/TestApp you will see beneath it Model, View, and Controller.
- I create symbolic links to these in my approot to make navigation easier.
- The View renders Output.
- The Model accesses Data.
- The Controller does everything in between.

The View

- For HTML pages you should stick with TT.
- Catalyst allows multiple views, and any module that can render output can potentially be made into a view provider.
- A view can be an emailer plugin
- Other plugins allow a view to export to csv or xml.
- Their are even plugins to render a PDF view.

The Model

- DBIC is the standard way.
- DBI is the ugly way.
- You will write your own helper script to generate the DBIC schema, which will add a Schema directory under lib/MyApp alongside your Model, View, and Controller.
- Your Data Access Logic should be contained in the Model.

Controller

- If we eliminate Data and Output, what is left in between is the Controller.
- Most of your code will wind up in the Controller.
- The Controller will control the flow of your application.
- The Controller Matches subroutines to URIs. This URI matching allows clean URLs.

The Catalyst Object

- An important feature of Catalyst is the Catalyst object, $c.
- All controller methods will begin something like:

```
sub givemepage {
my ( $self, $c ) = @_ ; ..... more code ..... }
```

- The $c object is inherited by all methods, providing persistence of data through a chain of execution.
- The $c object provides catalyst methods.

Examining $c

- $c->request provides cgi and request information. If you use forms you will see $c->request->parameters() quite a lot.
- $c->response provides output methods. In the controller of a new untouched catalyst app $c->response is used to direct output even though there isn't a view.

The Stash

- One of the most important components of $c is the stash, which is just a Hash.
- You can put data in the stash.
- If a request matches a method in one controller which forwards the action to another method, perhaps in a different controller module, the data you put in the stash remains.
- Data in the stash dies with the request.
- Flash offers peristence between requests. But is susceptible to corruption.

Installation

- I left the worst for last.
- In honor of the DRY principle Catalyst Developers call widely on many modules from CPAN, creating a maze of dependencies.
- Pulling Catalyst down from CPAN is time consuming, and every module that asks some stupid question or wants configuration so it can test itself is just another delay.
- Meanwhile if you are lucky enough to have a bundle from the repository it is likely to be out of date.

Installation Continued

- Many linux distributions are currently shipping Perl 5.10.0 and for development you need 5.8.8 or 5.10.1.
- Upgrading Perl from either source or from the unstable repositories is necessary.
- Once you have a highly customized Perl you need to be wary of packages that install older packaged versions of modules.
- There will be modules that won't build and you must find a package that provides them.