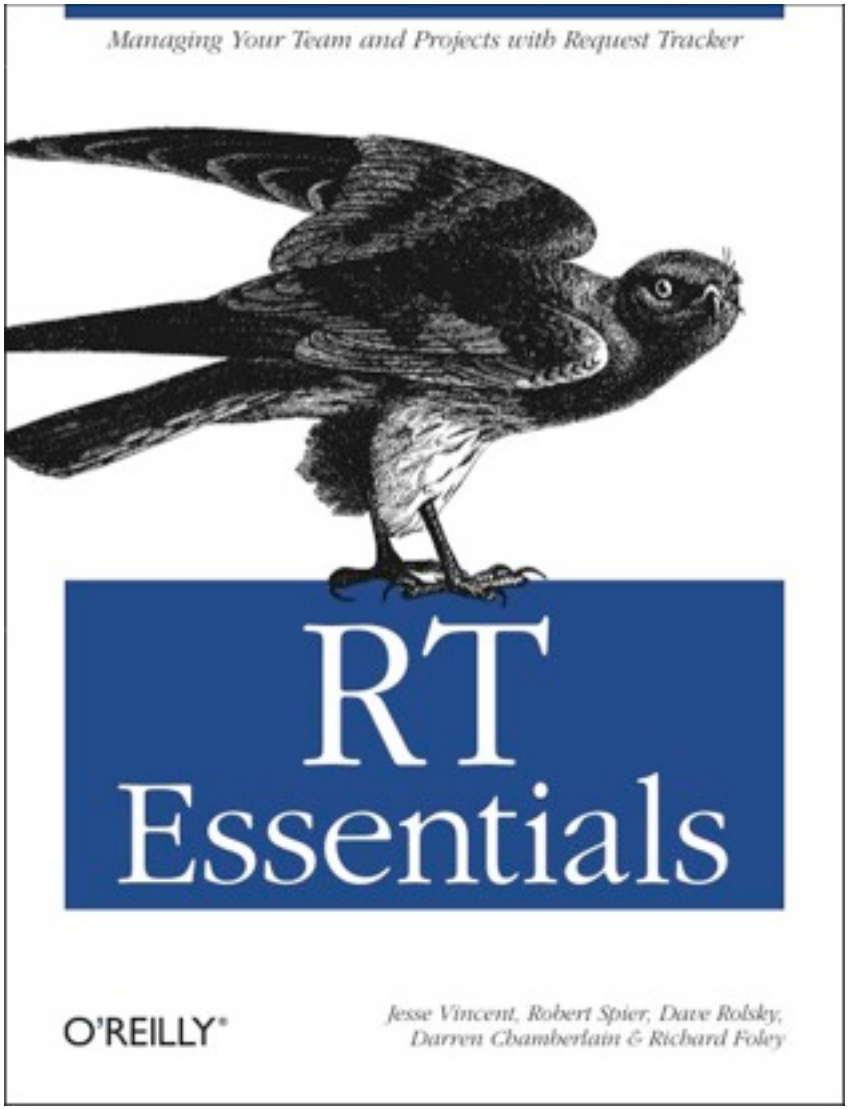
All About RT

Summer 2009

Jesse Vincent
jesse@bestpractical.com



BEST PRACTICAL ™

Today's agenda

- Introductions
- RT History
- Getting up and running
- Point-and-click configuration
- How to customize RT
- Customizing the web interface
- Data Model
- Internationalization
- Reporting
- Approvals and workflow
- Authentication
- Debugging

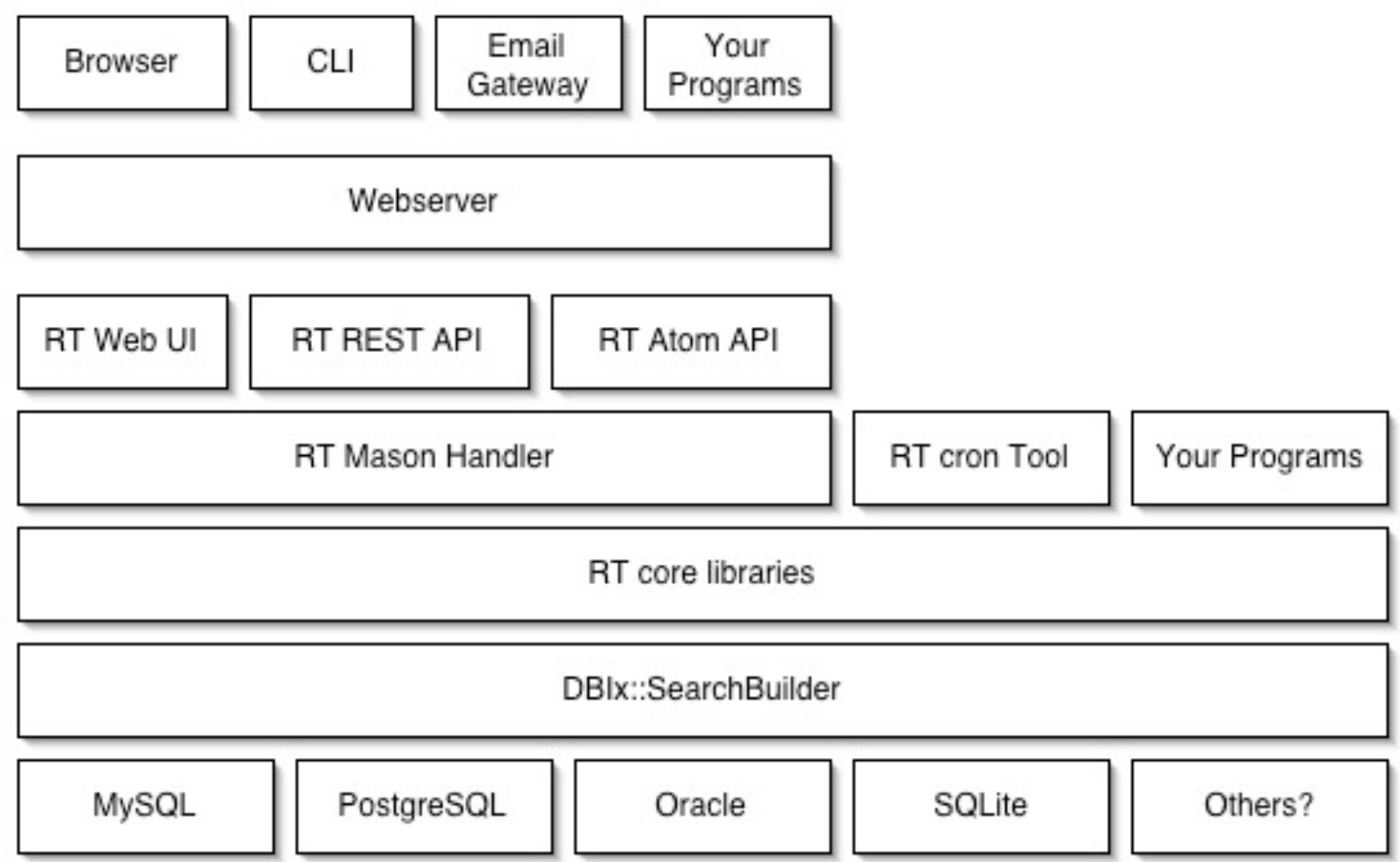
Hi, I’m Jesse Vincent.

Introduction

Ways RT Gets Used

- Helpdesk
- Network operations
- Bug tracking
- Customer service
- Project management

The RT Layer Cake



A Brief History of RT

RT 0.9 (1996)

- Designed for use at a single company
- 2 sysadmins
- 30 users

RT 1.0 (1999)

- Same as RT 0.9
- (+ a bit more courage)
- Used at hundeds of companies
- Dozens of CSRs
- Thouands of requests per day
- Intense guilt

RT 2.0 (2001)

- Total rewrite
- Just after Jesse escaped Microsoft
- DBIx::SearchBuilder
  - Abstraction
  - Multiple DBs supported
- Whole new UI
  - No more frames
- “Keywords”

RT 3.0 (2003)

- Overhauled web interface
- Extension mechanisms
- Internationalization
- Custom fields
- Cleaner internals
- Tests

RT 3.2 (2004)

- New search UI
- Spreadsheet / RSS output
- Outgoing mail preview and logging
- UI improvements
- Attributes for random metadata
- No major structural changes
- More tests

RT 3.4 (2005)

- Reimplemented Custom Fields
  - Custom fields on users, groups transactions
- Generalized Transaction system
- Faster, Faster, Faster
- Prettier
- Even more tests

RT 3.6 (2006)

- All CSS layout and styling
- Customizable homepage
- Built in charts and reports
- Ticket "reminders"
- Comprehensive test coverage
- Cleaner code

RT 3.8 (2008)

- More user preferences
  - Timezones
  - Theme
  - Ticket history order
- New configuration system
- Internals cleanup
- Even more tests
- PGP support

RT 3.8 (continued)

- “Favorite” tickets
- Ticket relationship graphs
- Branded queues
- iCal feeds
- Dashboards

RT 4.0!

The future

RT 4.0

- Never trust a software vendor who talks about the future...

Setting up RT

Download and Install

Getting up and running

Download and Install

- Latest Tarball
  - http://download.bestpractical.com/pub/rt/release/rt.tar.gz
- Subversion repository
  - svn co svn://svn.bestpractical.com/rt/3.8/trunk rt-3.8
- Dependencies from CPAN
- Instructions in README

The quick start guide

- wget http://download.bestpractical.com/pub/rt/release/rt.tar.gz
- tar xzvf rt.tar.gz
- cd rt-3.8.1
- ./configure
- make fixdeps install initdb

File System Layout (I)

- By default, RT installs everything in /opt/rt3/
- lib/
  - RT's perl libraries and message catalogs
- etc/
  - RT's configuration files and database schema
- bin/
  - Mail gateway, Mason handler and cron tool
- sbin/
  - Database setup and dependency checking tools

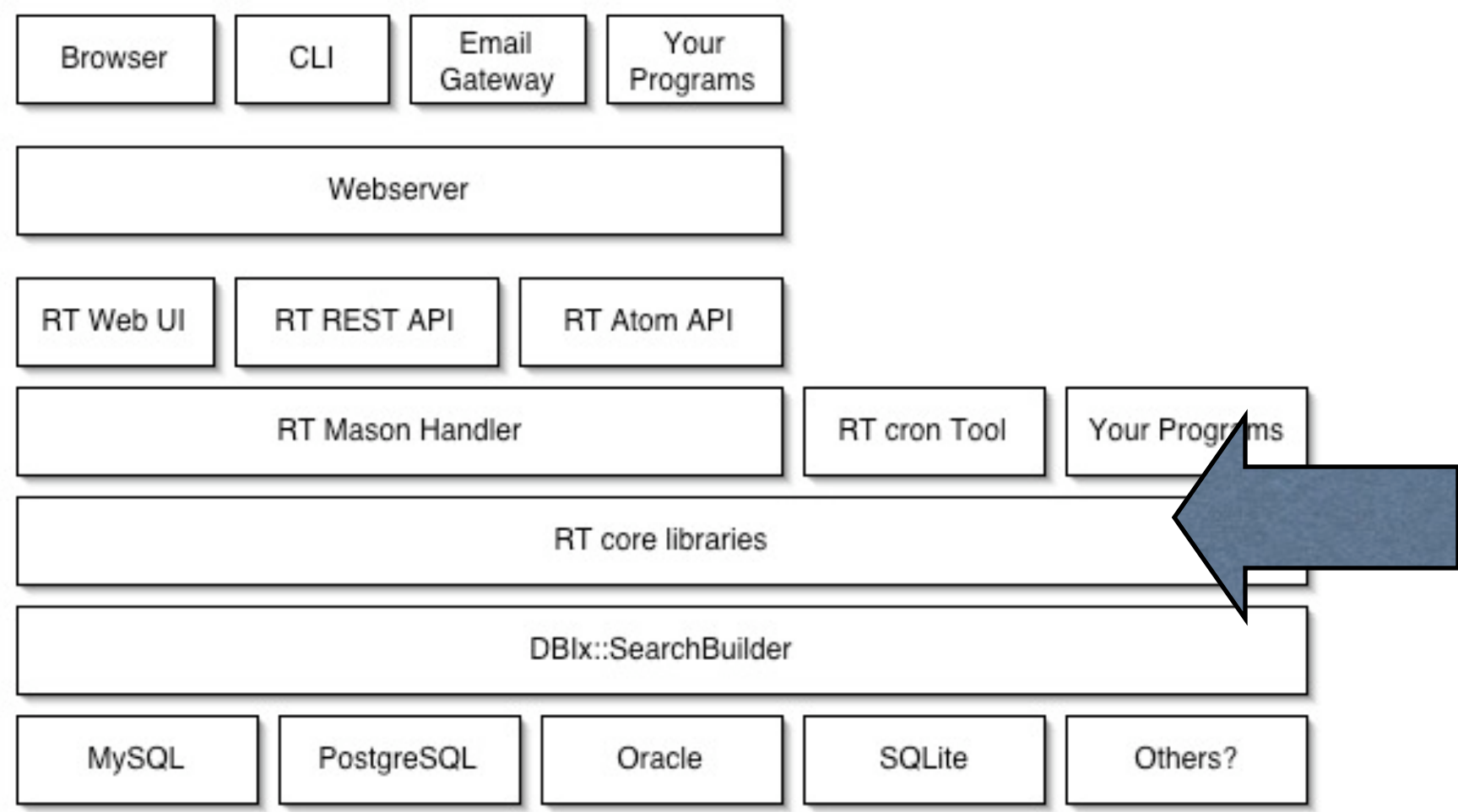
File System Layout (II)

- share/html/
  - RT's HTML::Mason frontend
- var/
  - Mason and web data cache
- local/{html,lib,po}/
  - Local components which override or enhance the core

Configuring RT

Point and Click

The RT Layer Cake



Custom Fields

Custom Fields

- Helpdesk
  - Hostname, OS, Browser, Site
- Bug Tracking
  - Severity, OS, Category, "Broken In", "Fixed In"
- Property Rental Requests
  - Smoking?, Beds, Pets, Special Needs

Custom Field Types

- Select from a List
- Enter Text
- Combobox
- Freeform
- WikiText
- Images
- File Attachments
- AJAX Source

Scrips

Scrips

- Condition / Action / Template
- Scrip Ideas:
  - OnCreate NotifyManager WithTemplate:ManagerNotify
  - OnResolve NotifyRequestor WithTemplate:HappynessSurvey
  - OnOwnerChange NotifyOwner WithTemplate:AssignmentNotifyWithHistory
  - OnCreate NotifyRequestor WithTemplate:SelfServiceNewPasswd

Conditions

- On Create
- On Transaction
- On Correspond
- On Comment
- On Status Change

On Owner Change
On Queue Change
On Resolve
User Defined

Actions

- AutoReply
- Notify Requestors
- Notify Owner
- Notify AdminCCs as Comment
- Notify Requestors and CCs
- Notify Requestors, CCs, and Admin CCs
- Notify Other Recipients
- Create Tickets
- Open Tickets
- … as Comment
- User Defined

Templates

- Blank
- AutoReply
- Transaction
- Correspondence
- Comment
- Status Change
- Resolved
- User Defined

Customized Scrips

- Custom Actions, Templates, and Conditions
  - In Database
  - On Disk
- Flexibility
  - Perl
  - RT's objects

Custom Templates

- Templates are Text::Template Objects
- They can be plain text
- They can contain Perl
- They are not sandboxed
- Customizing the AutoReply

Customizing the Autoreply

```
From: { $Transaction->CreatorObj->RealName } (via RT)
 <{ $Ticket->QueueObj->CorrespondAddress || $RT::CorrespondAddress }>
Precedence: normal
To: bugs-bitbucket@rt.perl.org
Mail-Followup-Tuo: perl5-porters@perl.org
Reply-To: perl5-porters@perl.org

# New Ticket Created by  { $Transaction->CreatorObj->RealName }
# Please include the string:  [{$rtname} #{$Ticket->id}]
# in the subject line of all future correspondence about this issue.
# <URL: {$RT::WebURL}Ticket/Display.html?id={$Ticket->id} >

{$Transaction->Content()}
```

Template Tweaks

- $Ticket, $Transaction
  - $Ticket->QueueObj, $Transaction->CreatorObj
- Message Headers
  - {$Transaction->Message->First->Headers}
- Real Code:

```
Thanks for contacting FooCorp Support!

It's { use LWP::Simple; get("http://foocorp/weatherword"); } here in
SomeCity today, but that won't stop us from answering your question within
48 business hours...
```

Custom Condition

- Code snippet that returns true or false
- Controls whether an action is run
- For example:
  - OnEmergency

```
if ($self->TransactionObj->Subject =~ /emergency/i
    && $self->TicketObj->Transactions->Count == 1) {
  1;
} else {
  0;
}
```

Custom Action

- PageSysadmins
  - Prepare

```
my $hour = (localtime)[2];
if ($hour > 8 && $hour < 18) {
  return 0;
} else {
  return 1;
}
```

  - Cleanup/Commit

```
use Net::SMS;
my $sms = Net::SMS->new();
$sms->msgPin("+1 555 123 1234");
$sms->msgFrom("RT");
$sms->msgText($self->TemplateObj->MIMEObj->body->as_string);
$sms->msgSend()
```

Example One

- EmergencyPage Template

```
Subject: 911

{$Ticket->Subject}
```

- Putting it All Together:
  - OnEmergency PageSysadmins WithTemplate:EmergencyPage

Example Two

- OnCreate EscalatePriorityIfBoss WithTemplate:Blank
- EscalatePriorityIfBoss Action:
  - Prepare

```
($self->TransactionObj->CreatorObj->EmailAddress
    =~ /president\@whitehouse.gov/i) ? 1 : 0;
```

  - Commit

```
$self->TicketObj->SetPriority(99);
```

Scrips limited to Queues

- Create a global ScripAction
  On Create where Queue is parrot or perl5
- Condition: User Defined
- Custom Condition Code:

```
my $self = shift;
if ($self->TransactionObj->Type eq "Create"
     && $self->TicketObj->QueueObj->Name =~ /^(?:parrot|perl5)$/)
{
     return 1;
} else {
     return 0;
}
```

Basic Custom Views

Saved Searches

- Any query
- Any output format
- Personal or Shared

Custom Homepage

- Users can customize their own
- Superusers can change the default
- Pre-written portlets
- Any saved search
- Graphs and Charts (from 3.6.3)

Dashboards

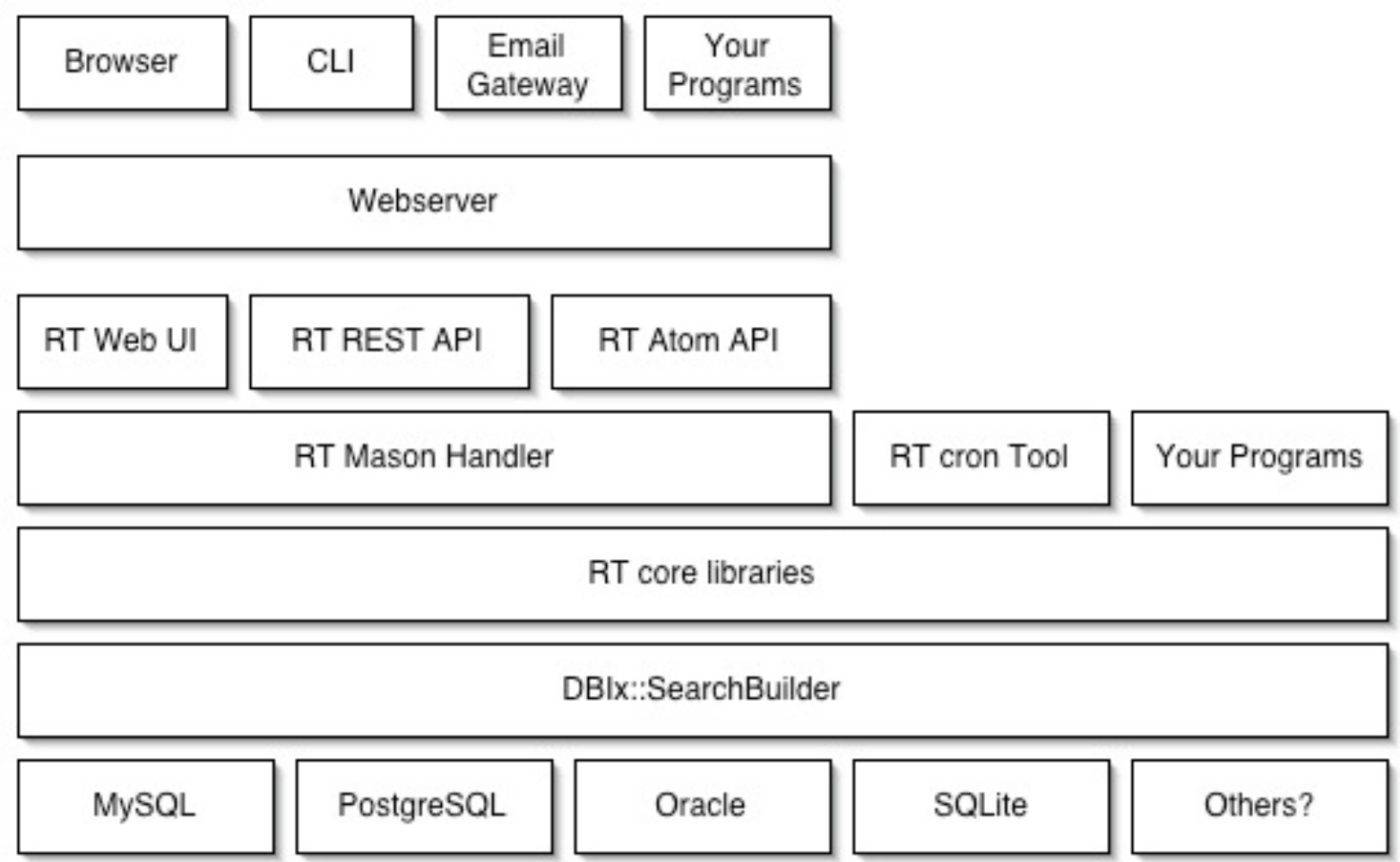
- Similar to custom homepages
- Shareable with groups
- Bookmarkable URLs
- Scheduled email delivery

The Mail Gateway

Letting your users open and update tickets from their email clients.

BEST PRACTICAL ™

The RT Layer Cake



The Mail Gateway

- Usually lives in /opt/rt3/bin
- Simple perl script
- Uses HTTP (or HTTPS) to talk to RT
- Uses REST interface to talk to RT
- Does no message processing
- Doesn't need to run on an RT server
- Doesn't need the rest of RT installed
- Doesn't need to be SetUID or SetGID

How It Works

- MTA pipes message to rt-mailgate
- On any system error, rt-mailgate returns a "Temporary Failure" error to the MTA
- rt-mailgate sends message to RT Server via HTTP
- Server processes message and creates or updates ticket
- Server returns a success or failure message

Setting Up the Mail Gateway

/etc/aliases

```
rt: "|/opt/rt3/bin/rt-mailgate
     --url http://localhost/
     --queue general
     --action correspond"
```

Debugging rt-mailgate

```
# /opt/rt3/bin/rt-mailgate \
      --url http://localhost/ \
      --queue general \
      --debug \
      --action correspond < /tmp/msg

/tmp/msg:

From: root@localhost
Subject: just testing

This is a message
```

Advanced Options

- MTAs support "+ notation" to pass information to MUAs by appending +data to an email address.
- --extension
  - 'queue'
    - rt+general@example.com
  - 'ticket'
    - rt+23@example.com
  - 'action'
    - rt+comment@example.com

Mail Extensions

- RT::Extension::
  - CommandByEmail
  - ExtractSubjectTagOnTransaction

Mail Handling Backend

- RT::Interface::Email
  - Routines for parsing and processing mail
  - Mail handling plugins
  - Filters
    - Plugins to modify messages before they're handed off for processing
  - Authentication handlers
    - Plugins to decide if the sender of a message is who they claim to be

Mail Handling Backend

- RT::EmailParser
  - Turns a MIME message into a MIME::Entity

Client Side Spam Filtering

- SpamAssassin + procmail
  - simple, you may already know how to do this
- SpamAssassin + Mail::Audit
  - more flexible
- Dspam
  - Web GUI for managing spam trap
- Whitelisting tools

Server Side Spam Filtering

- Using RT::Interface::Email::Filter plugins
  - Advantages
    - Runs inside the RT core
  - Disadvantages
    - Current API doesn't provide much functionality to change message processing
    - Runs inside the RT core

Self Service

Let your users help you do your job

Self Service

- RT provides a "SelfService" interface for end-users
  - Automatically displayed for unprivileged users
  - See new/open/stalled/resolved tickets
  - Create new tickets
  - Password needed to access it
    - Use external authentication for RT
    - Send users their passwords when they first create a ticket

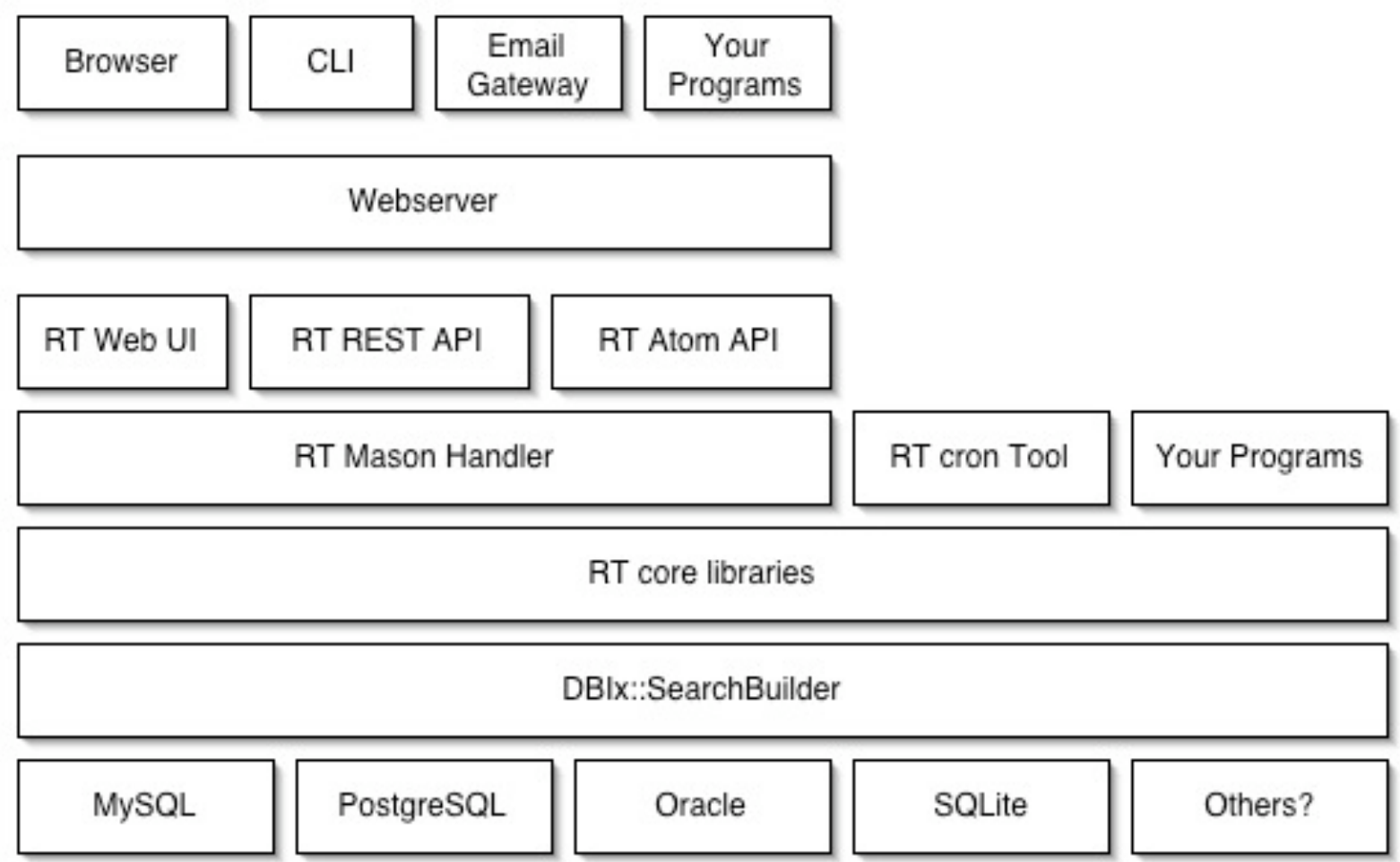
Autoreply with a password

```
{if (($Transaction->CreatorObj->id != $RT::Nobody->id) &&
   (!$Transaction->CreatorObj->Privileged) &&
    ($Transaction->CreatorObj->__Value('Password') eq '*NO-PASSWORD*')){

      my $user = RT::User->new($RT::SystemUser);
      $user->Load($Transaction->CreatorObj->Id);
      my ($stat, $pass) = $user->SetRandomPassword();

      if (!$stat) {

      $OUT .="An internal error has occurred. RT was not able to set a
password for you. Please contact your local RT administrator for
assistance.";

      }

    $OUT .= "You can check the current status and history of your requests
at: ".$RT::WebURL."When prompted, enter the following username and
password:  Username: ".$user->Name."   Password: ".$pass."";
}}
```

BEST PRACTICAL ™

The RT Commandline

The RT Layer Cake



The RT CLI

- Simple perl script
- Usually lives in /opt/rt3/bin
- Can run locally or remotely
- Uses HTTP (or HTTPS)
- Uses TicketSQL for Searching
- Talks to RT's "REST" interface
- Scriptable

bin/rt

rt ls "Priority > 5 and Status='new'"

rt ls "queue='perl5' and (Status='new' or Status='open')"

rt edit ticket/312 set queue=spam status=deleted

rt comment -m 'this is a comment' 151

bin/rt + your shell

```
rtresolve() {
rt edit ticket/$1 set status=resolved
}

$ rtresolve 551
```

Hacking the code

Not everything is point and click

I'll just make my changes on the live RT, right?

- Wrong!
- Use a development environment
- Isolate your changes
- Test your changes
- Give your users a break

Set up a development environment

- Download
- Configure
- Run
- Hack
- Run
- Hack
- Run

Setting up a development environment

```
$ ./configure --with-my-user-group
--enable-layout=inplace
--with-db-type=SQLite
--with-devel-mode

$ make install

$ ./bin/standalone_httpd 8888
```

What all that means

- Use the SQLite database engine
  - One file
  - Self-contained
- Install in the build directory
- Keep the permissions as me
- Enable RT's "Developer Mode"
- Run on port 8888

standalone_httpd

- RT Application Server
- Just like mod_perl or FastCGI
- Pure perl
- Can be easily profiled
- Can be run under perl's debugger

“Developer Mode”

- Primarily useful for web development
- Mason refreshes on the fly
  - Components (Web UI bits)
  - Perl Libraries

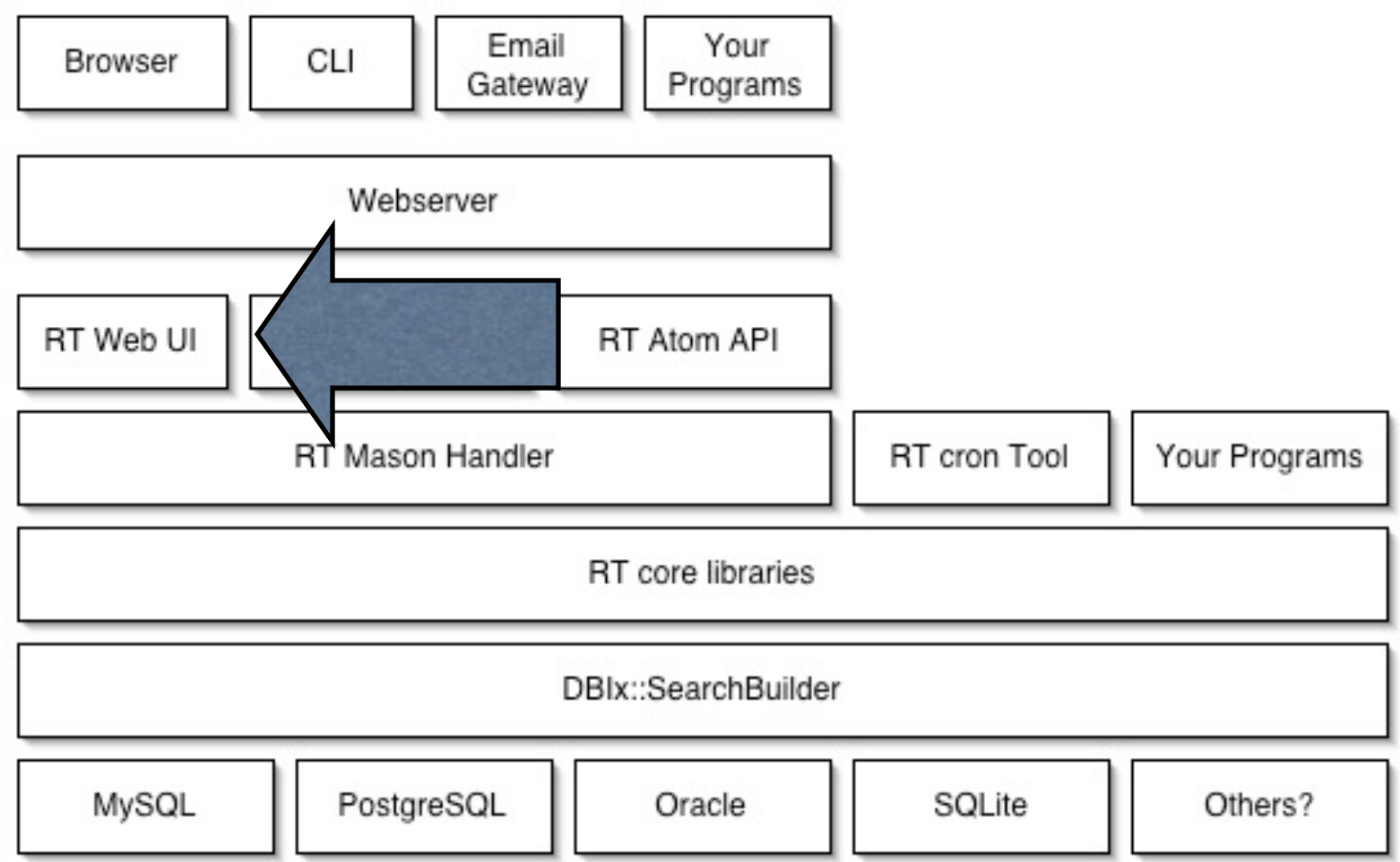
Basic development hints

- Use revision control
- Set up a development environment
- Document what you do
- Collaborate with others on rt-devel
- Use the wiki
  http://wiki.bestpractical.com/

Customizing the Web Interface

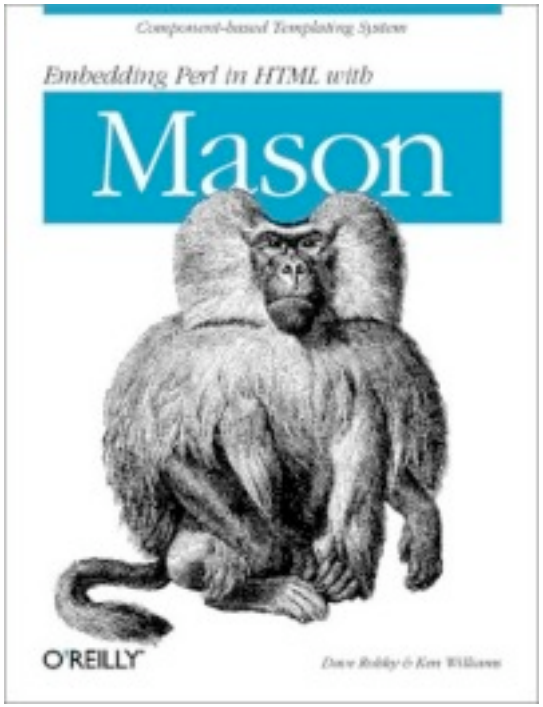
Something to do with that new development environment.

The RT Layer Cake



Mason application server

- RT's web interface is built in HTML::Mason
  - Easy to start hacking on, especially if you know perl
  - Fast
  - Flexible



A brief introduction to HTML::Mason

- % lines mean “Perl”
  - % my $user = RT::User->new(...);
- <% %> tags mean “Perl”
  - <% 1+ 1 %>
- <& &> is how Mason includes other templates
  - <& /Elements/Header,
  - Title => 'Hi!' &>

Mason Blocks

```
<%args>
$variable => 'Default value'
</%args>

<h1>This is a page!</h1>

<%init>
# This block runs before any page output
</%init>

<h1> This is page content, too</h1>
```

A Simple Web Tool

```
<%init>
my $tix = new RT::Tickets($session{'CurrentUser'});
$tix->Limit(FIELD => 'Owner',
 VALUE => $session{'CurrentUser'}->id);
</%init>
<h1><&|/l&>All my tickets</&></h1>
% while (my $ticket = $tix->Next) {
<%$ticket->id%>: <%$ticket->Subject%>
<%loc($ticket->Status)%><br>
% }
```

Custom Mason Templates

- RT puts templates in share/html/
- Find the template you want to change
- Copy it to local/html/
- Make your changes

Custom Mason Templates

- Replace entire Mason component
- Never clobber the core
- Easier to find your changes
- Easier to upgrade RT

Example Customization

/Ticket/Elements/ShowBasics

```
% if ($Ticket->TimeEstimated) {
  <tr>
    <td class="label time estimated"><&|/l&>Estimated</&>:</td>
    <td class="value time estimated"><& ShowTime, minutes => $Ticket-
>TimeEstimated &></td>
  </tr>
% }
% if ($Ticket->TimeWorked) {
  <tr>
    <td class="label time worked"><&|/l&>Worked</&>:</td>
    <td class="value time worked"><& ShowTime, minutes => $Ticket-
>TimeWorked &></td>
  </tr>
% }
```

Example Customization

/Ticket/Elements/ShowBasics

```
% # if ($Ticket->TimeEstimated) {
  <tr>
    <td class="label time estimated"><&|/l&>Estimated</&>:</td>
    <td class="value time estimated"><& ShowTime, minutes => $Ticket-
>TimeEstimated &></td>
  </tr>
% # }
% # if ($Ticket->TimeWorked) {
  <tr>
    <td class="label time worked"><&|/l&>Worked</&>:</td>
    <td class="value time worked"><& ShowTime, minutes => $Ticket-
>TimeWorked &></td>
  </tr>
% # }
```

Cascading Style Sheets

- /NoAuth/css/3.5-default/main.css
- /NoAuth/css/3.4-compat/main.css
  - Colors
  - Fonts
  - Alignment
  - Just a Mason component
- Make your own and @import a base one.
- $RT::WebDefaultStylesheet

RT Web Callbacks

- Insert Mason code inside at hook points
- Add things to pages and menus
- Set widget content
- ...all without changing core RT code

How Callbacks Work

- Hooks in the Mason templates that let you add custom components
  - In html/Ticket/Update.html:

```
<& /Elements/Callback,
_CallbackName => 'AfterTitle', %ARGS &>
```

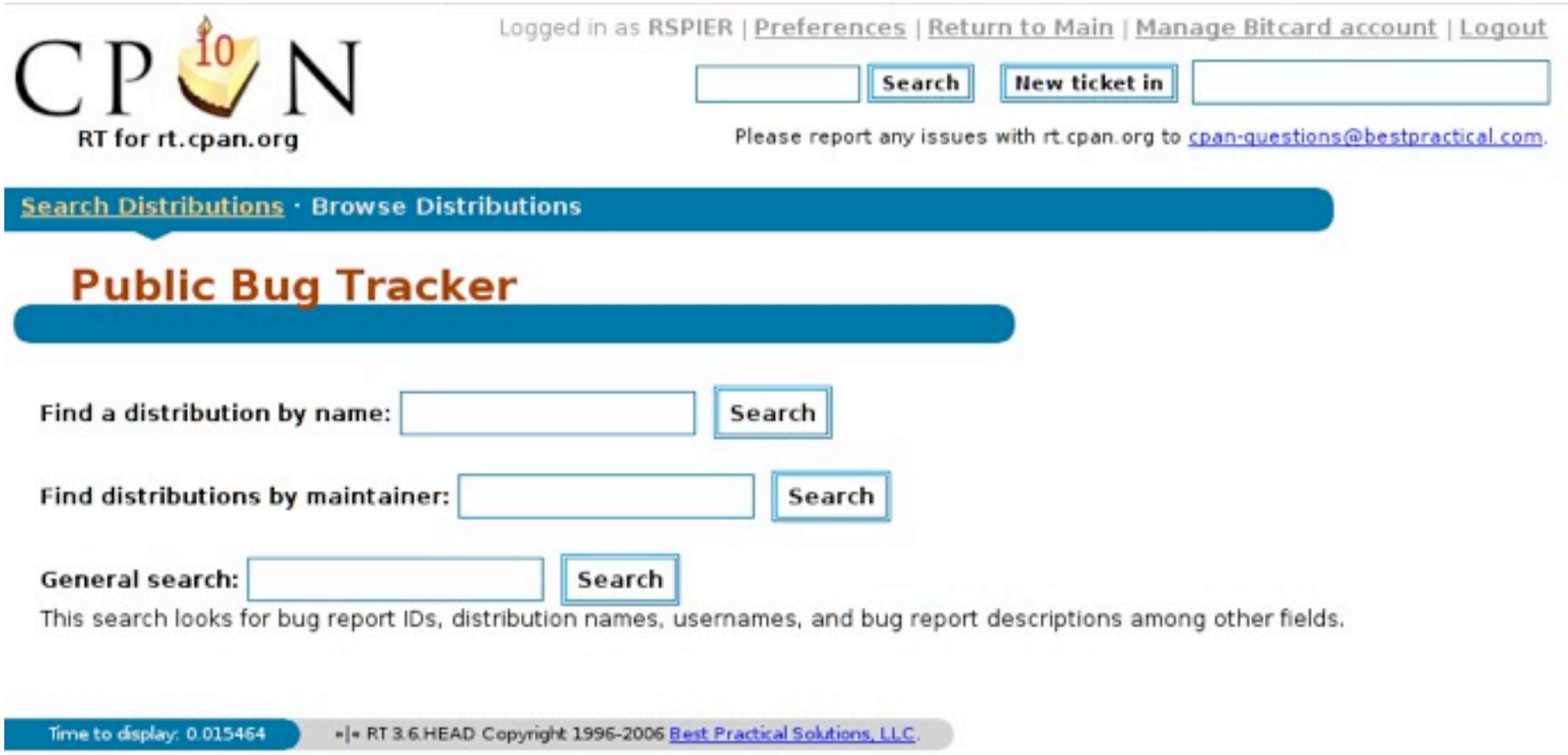
- Local code "registers" itself by living in the right place in the filesystem
- RT includes all components matching:

```
{share,local}/html/Callbacks/*/
  Ticket/Update.html/AfterTitle
```

RT::Extension::MenubarSearches

- Small example of callbacks
- Jump to active tickets in each queue
- Inserts a menu item with a callback

rt.cpan.org



BEST PRACTICAL ™

Extensions

- RT::BugTracker
- RT::BugTracker::Public
- RT::Extension::rt_cpan_org
- RT::Authen::PAUSE
- RT::Authen::Bitcard
- RT::Authen::OpenID

MasonX::Profiler

Figuring out how the Web UI fits together

MasonX::Profiler

```
Foundry/Home/MyRequests.html BEGINS
     /Elements/SetupSessionCookie 0.0610
         /Callbacks/Foundry/autohandler/Auth 0.0003
     /Elements/Callback 0.0242
     /Elements/Callback 0.0016
                 /Foundry/Elements/Top 0.0604
                 /Foundry/Elements/Tab 0.0194
             /Foundry/Elements/Header 0.1375
             /Foundry/Elements/Tabs 0.0037
         /Elements/Callback 0.0294
     /Elements/Footer 0.0308
 /autohandler 2.9179
 /Foundry/Home/MyRequests.html ENDS
```

BEST PRACTICAL ™

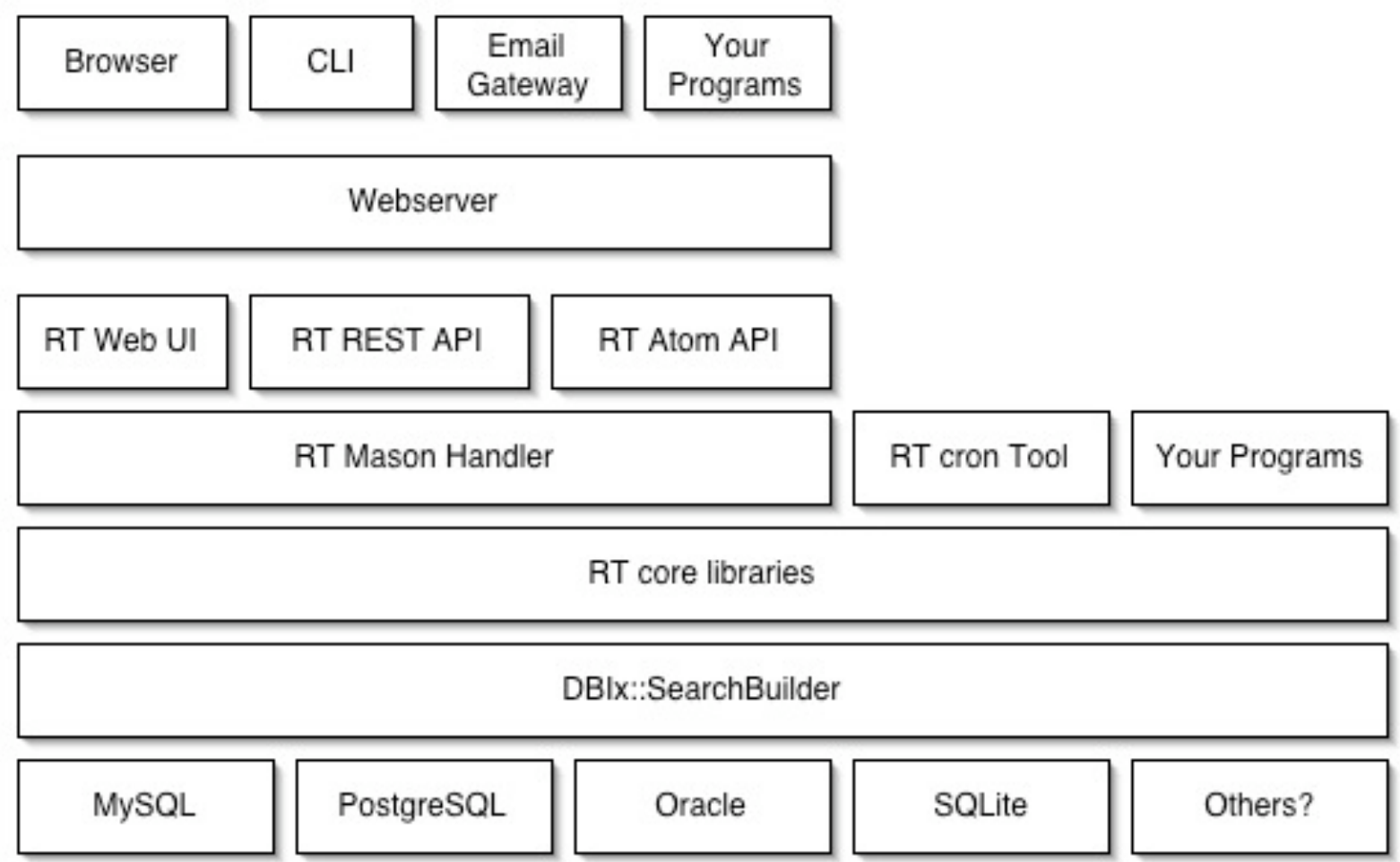
Activating MasonX::Profiler

RT_SiteConfig.pm

```
use MasonX::Profiler;
@MasonParameters = (
preamble => 'my $p =
 MasonX::Profiler->new($m,$r);'
);
```

The RT Data Model

The RT Layer Cake



Database

- RT is composed of over a dozen types of related objects
- Building your own relational database out of BDB or flat files on disks isn't our idea of fun
- Organizations want to be able to use their own tools to query RT
- RT connects to a SQL backend with an object-relational mapper

Schema– Core (I)

- Users
  - An individual who can perform actions within RT
- Groups
  - A collection of Users and other Groups
  - Can be assigned rights, made watchers of tickets, etc
- Principals
  - An abstraction of Users and Groups
  - Used internally so that anything that can apply to a user or group can apply to either

Schema– Core (II)

- GroupMembers
  - A listing of all the Users and Groups which are members of a Group
- CachedGroupMembers
  - An internal cache of all members of each group, fully unrolling the GroupMembers of each GroupMember
- ACL
  - A table detailing which rights each Principal has for any ACLed object in RT
  - ACLed Objects include: Ticket, Queue and Group

Schema– Core (III)

- Links
  - A mapping between RT internal objects and other RT internal objects
  - Can handle mapping between any two URIs
- Transactions
  - Records of object updates. Usually tickets.
- Attachments
  - Any message body or attachment for a Transaction
  - Hierarchical, so MIME email messages can be rebuilt

Schema– Core (IV)

- CustomFields
  - Single or multiple values
  - Select from list, freeform or file upload
- ObjectCustomFields
  - Map CustomFields to Tickets, Transactions, Users, Groups
- CustomFieldValues
  - Acceptable values for "select from list" custom fields
- ObjectsCustomFieldValues
  - Values of custom fields for specific records

Schema– Core (V)

- Attributes
  - Non searchable metadata
  - Can apply to any object (or row)

Schema– Ticketing (I)

- Tickets
  - Issues of one sort or another

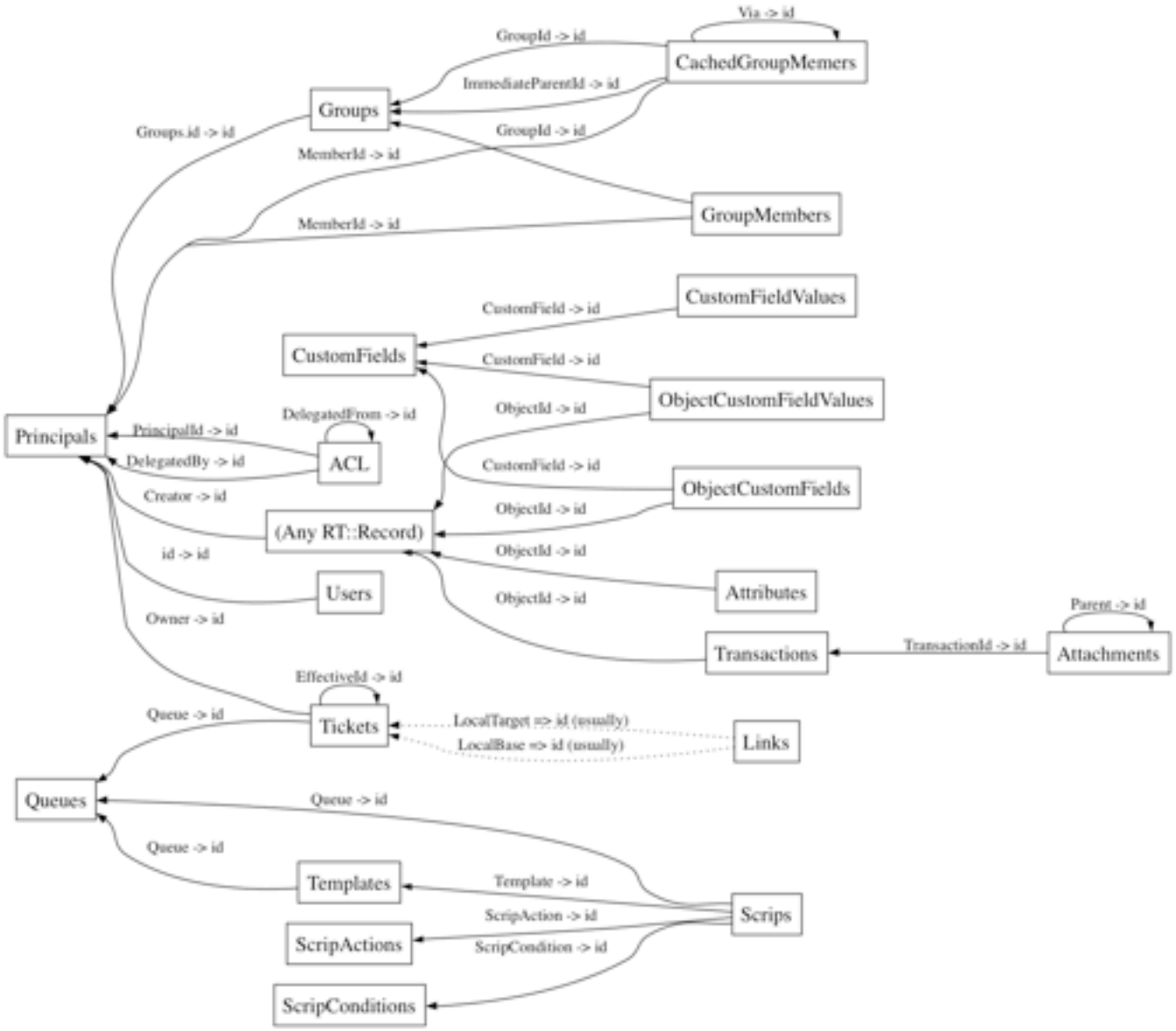
Schema– Ticketing (II)

- Queues
  - Highest level categorization of tickets
  - Per queue Scrips, Custom Fields and ACLs
- Scrips
  - Conditional actions that can happen on any ticket update
- Templates
  - Simple templates which can take embedded perl expressions
  - Used by Scrips

Schema– Ticketing (III)

- ScripActions
  - Used by Scrips to perform some action
    - Autorepy to Requestor
    - Notify Owner
    - Page Systems Administrators
- ScripConditions
  - Used by Scrips to decide when to fire
    - On Create
    - On Correspond
    - If message matches "explosion"

Schema Diagram



Working with your data

Writing directly to the database is wrong

- RT is a complex application with complex relationships between database tables
- Querying the database for reporting is sometimes OK, but usually unnecessary
- We've got a better way...

DBIx::SearchBuilder

- It's an object-relational mapper
- It hides SQL from your application
- It lets you transparently treat your database as perl objects
- Most RT objects are subclasses of DBIx::SearchBuilder or DBIx::SearchBuilder::Record
- It's a dessert topping and a floor wax

SearchBuilder is Easy

```
$t = new RT::Ticket( $RT::SystemUser );
$t->Load( 42 );
print $t->Subject;
```

Two kinds of object

- DBIx::SearchBuilder
  - Collections
  - perldoc DBIx::SearchBuilder
- DBIx::SearchBuilder::Record
  - Individual records
  - perldoc DBIx::SearchBuilder::Record

Record Objects

Create

```
my $user = RT::User->new($RT::SystemUser);
($id, $msg) = $user->Create(Name => 'jrv', EmailAddress =>
'jrv@s.ly');
```

Load

```
$user->LoadByCols(EmailAddress => 'jrv@s.ly');
```

Read

```
print $user->Name;
```

Update

```
$user->SetName('jesse');
```

Delete

```
$user->Delete();
```

Collection Objects

- Every record has a corresponding DBIx::SearchBuilder 'collection' object

- Complex searches without raw SQL

```
my $users = RT::Users->new($RT::SystemUser);
$users->Limit(FIELD => 'EmailAddress',
OPERATOR => 'LIKE',  VALUE => 'fsck.com');
while (my $user = $users->Next) {
    print "Found ",$user->EmailAddress,"\n";
}
```

Collection Objects

- The standard methods:
  - Limit a result set
    - Limit();
  - Iterate
    - Next();
  - Sort and Order
    - OrderBy();
  - Page
    - RowsPerPage();

The RT Core Objects

- DBIx::SearchBuilder::Record subclasses
  - Ticket
  - Queue
  - User
  - Group
  - And others
    - ACE, Attachment, Attribute, GroupMember, Transaction, Principal, Link, CustomField, CustomFieldValue, ObjectCustomFieldValue, ...

API

- RT's API is its core objects
- All RT tools use the same API we export to the world
    - rt-crontool
    - Web frontend
    - Database setup tools

Boilerplate Code

Simple samples

RT Tool Boilerplate

```
#!/usr/bin/perl -w
use strict;

use lib qw(/opt/rt3/local/lib /opt/rt3/lib);
use RT;

# Load the config file
RT::LoadConfig();

# Connect to the database and get RT::SystemUser
#  loaded
RT::Init();
```

Resolve a ticket

```
[...boilerplate...]
use RT::Interface::CLI qw(GetCurrentUser loc);
use RT::Ticket;

my $CurrentUser = GetCurrentUser();
die loc("No RT user found.")
 unless ($CurrentUser->Id);
my $ticketid = shift @ARGV;
my $ticket = RT::Ticket->new($CurrentUser);
$ticket->Load($ticketid);
die loc("Ticket not found") unless ($ticket->Id);
my ($trans, $msg) = $ticket->SetStatus('resolved');
print $msg;
```

Overlay Classes

Cleanly customize core classes

Overlay and Local Classes

- Core database-access classes are auto-generated
- When the database changes, you don't want to hand-hack code
- When you make changes to RT, you want your changes to persist seamlessly across minor version upgrades
- Most sites don't track local source changes
- Even if they do, merging sucks

How Overlays work

```
# User_Local.pm
no warnings qw/redefine/;
package RT::User;
use Site::UserDB;

sub RealName {
 my $self = shift;
 return Site::UserDB::LookupName(
$self->EmailAddress);
}
```

How Overlays work

```
    eval "require RT::Ticket_Overlay";
if ($@ && $@ !~ qr{^Can't locate RT/Ticket_Overlay.pm}) {
            die $@;
};

eval "require RT::Ticket_Vendor";
if ($@ && $@ !~ qr{^Can't locate RT/Ticket_Vendor.pm}) {
            die $@;
};

eval "require RT::Ticket_Local";
if ($@ && $@ !~ qr{^Can't locate RT/Ticket_Local.pm}) {
            die $@;
};
```

Principals and Users and Groups

Oh my!

Principals, Users and Groups

- Principals
  - Every User is a Principal
  - Every Group is a Principal
  - We can treat users and groups as equivalent for ACL checks and Group Membership
- Groups can contain users and groups
  - Groups can't contain themselves

Authentication

- RT has its own internal authentication system
- RT needs to have a user object for any user before they're allowed to access RT
- You can tie RT into your single sign on
  - RT::Authen::
    - Bitcard
    - OpenID

ACL System

- ACLs can apply to any DBIx::SB::Record
- Any Record object type can define what rights it supports
- Rights can be granted to any user or group
- Other systems that drop on top of RT can use the ACL system

Delegation

- Supports basic delegation of rights
- Doesn't support "partial" delegation of a given right
- Doesn't support "re-delegation of rights"
- When a user's right to do something is revoked, delegates also have right revoked

I18N and L10N

Internationalization and Localization

Internationalization

I18N

Building Blocks

- UTF8/Unicode
- Locale::MakeText
- Locale::MakeText::Lexicon

Overview

- Write Code
- Extract Strings to Message Catalog
- Translate Message Catalog

String Extraction

- Core
  - $self->loc("Created ticket [_1]", $ticket->Id);
- Code In Mason
  - loc("Created ticket [_1]", $ticket->Id);
- Text In Mason
  - <&|/l, $ticket->Id&>Created ticket [_1]</&>
- Getting the strings into the .po files
  - tool/extract-message-catalog

Localization

L10N

Adding a New Translation

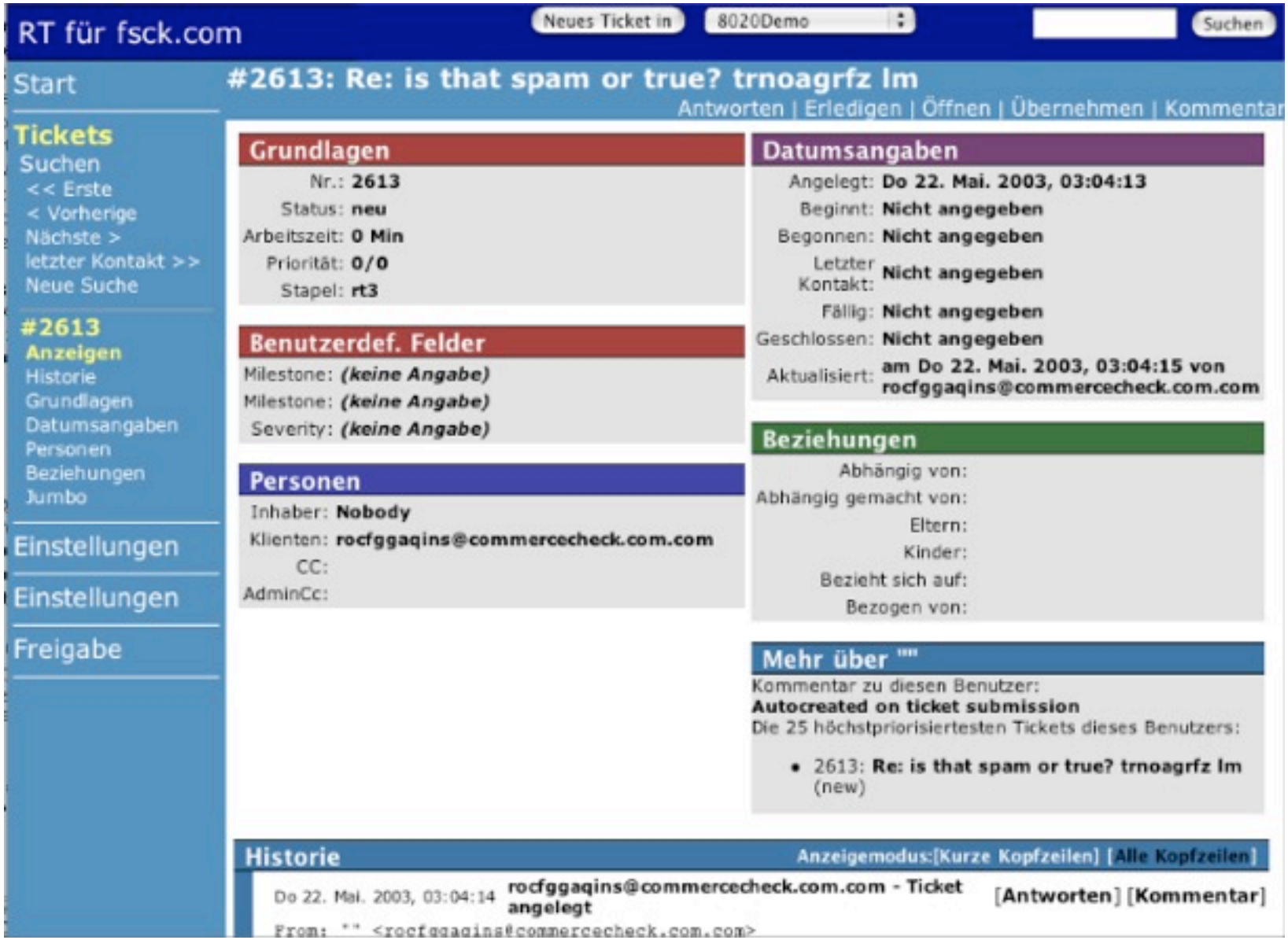
- Basic localization for languages without cases/ aspects
- Extract a fresh .po file
- Translate .po file
- Example string from nl.po file
    - #: html/Admin/Users/Modify.html:80
    - msgid "Access control"
    - msgstr "Toegangscontrole"
- Check your work

Gotchas

All languages are not created equal

- "1 Ticket(s) found" is ugly and wrong
- Handling this case is language specific
- English treats singular and plural separately
- Some languages treat zero specially
- Some languages have a "dual"
- Locale::Maketext::Lexicon lets you use perl to help your translations.

BEST PRACTICAL ™



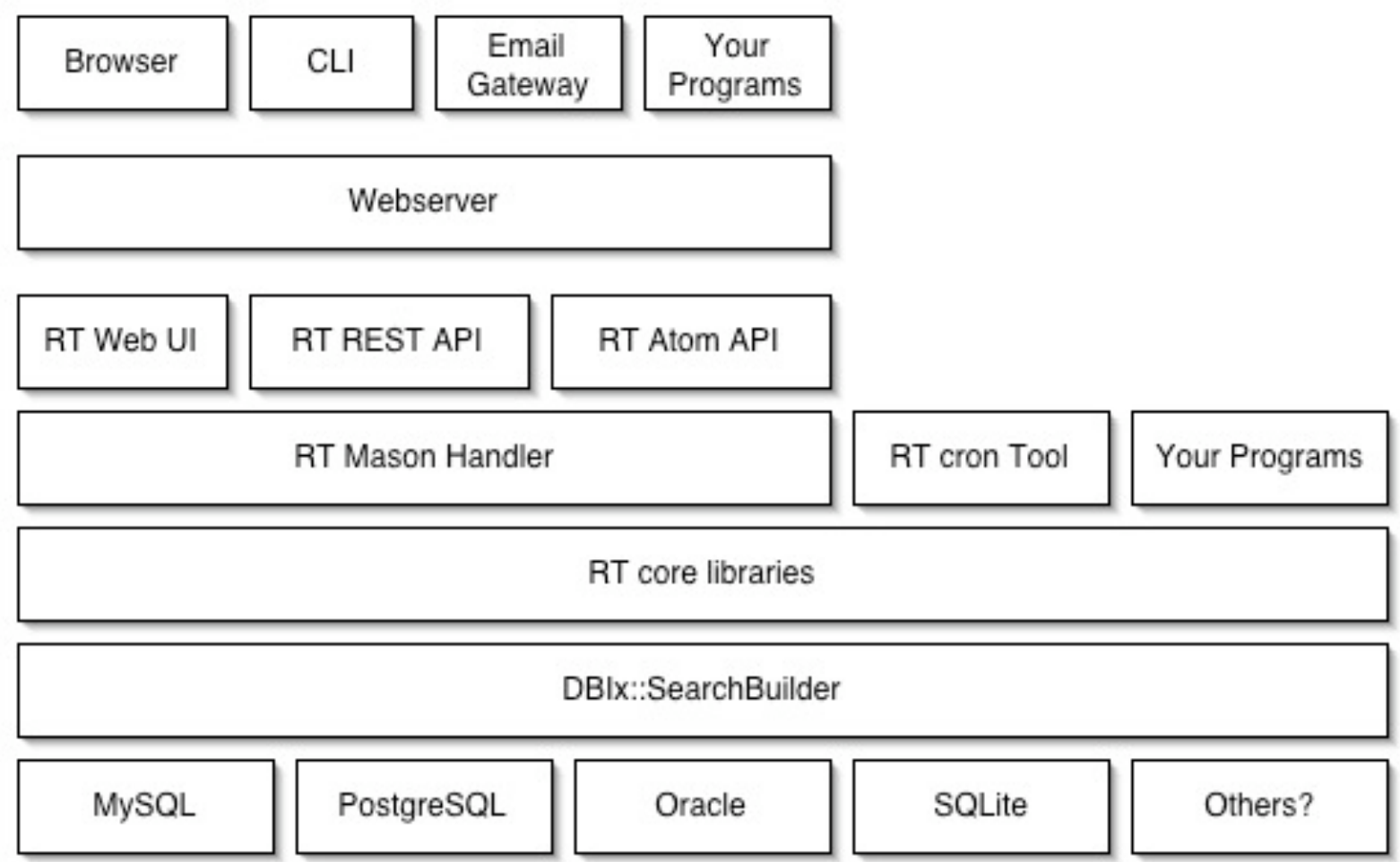
Corporatization

- Localization can be useful just within an organization.
- Every organization has its own jargon
- It's a "request", not a "ticket"
- PO files can be "overlaid" just like libraries

Reporting

Finding, Counting, and Extracting Tickets

The RT Layer Cake



Important Classes

- RT::Attachment
- RT::CustomField
- RT::Link
- RT::Queue
- RT::Ticket
- RT::Transaction
- RT::User

Collection Classes

- RT::Attachments
- RT::CustomFields
- RT::Links
- RT::Queues
- RT::Tickets
- RT::Transactions
- RT::Users

Finding Tickets

- Limit
  - DBIx::SearchBuilder-Like
- TicketSQL
  - SQL-like query language
- Direct Database Access
  - Try to avoid this if you can

TicketSQL

- A SQL WHERE clause style syntax for searching for tickets
- Basic Component:
  - Field + Operation + Value
- Boolean Logic and Grouping
  - AND / OR / ( )

TicketSQL: Search Terms

- Defined in Tickets_Overlay.pm
  - Status, Queue, Type, Creator, LastUpdatedBy, Owner, EffectiveId, id, InitialPriority, FinalPriority, Priority, TimeLeft, TimeWorked, MemberOf, DependsOn, RefersTo, HasMember, DependentOn, ReferredTo, Told, Starts, Started, Due, Resolved, LastUpdated, Created, Subject, Type, Content, ContentType, Filename, TransactionDate, Requestor, CC, AdminCC, Watcher, LinkedTo, CF

TicketSQL: Custom Fields

- "CF.{Field Name}"

TicketSQL: Usage

```
my $ts = RT::Tickets->new( $CurrentUser );
$ts->FromSQL( q[Queue = "perl5"] );
```

TicketSQL: Examples

```
Queue = "perl5"
AND ( "Status" = "new" OR Status = "open" )

Queue = "parrot" AND "CF.{Priority}" = "High"

Queue = "perl5" AND Owner = "jhi"
AND  Created < "2003-05-29"
```

TicketSQL: Complex Queries

- Created last week and owned by Jesse, or created This week and owned by Robert

  (Created > 'two weeks ago' AND Created < 'one week ago' AND Owner = "Jesse")

  OR (Created > 'one week ago' AND Owner = "Robert")

- Things in the Premium queue, or high priority things in the support queue, or things in the support queue more than two days old.

  (Queue = "support" AND Priority> 50) OR (Queue ="Premium")

  OR (Queue ="support" and Created < 'two days ago')

Finding Other Things (not Tickets)

- DBIx::SearchBuilder
  - For everything
  - Find all users named Robert

```
my $users = RT::Users($CurrentUser);
$users->Limit( FIELD => 'Name',
OPERATOR => 'LIKE', VALUE => 'Robert' );
```

Counting Results

- $users->Count()

Sorting Results

```
$users->OrderBy( FIELD => "Name", ORDER => "ASC" );
```

- Fields:
  - Defined in User.pm ($Class.pm)
- Orders:
  - ASC
  - DESC

Listing off Results

- Collections have an iterator

```
$users = new RT::Users( $CurrentUser );
$users->UnLimit();
while( my $u = $users->Next ) {
      print $u->Name,"\n";
}
```

Report Structure

- Query
  - What records do I want?
- Renderer
  - How do I want to output them?

Writing Queries

- Finding tickets

TicketSQL vs. Limit

- Concise
- Explicit
- Explicit grouping
- Easy to serialize
- If you know SQL, this is easy

- Can be verbose
- DWIM
- Implicit grouping
- Hacky to serialize
- Designed for iterative searches
- Requires understanding the concept

Recent Tickets: Limit

```
my $queue = new RT::Queue( $CurrentUser );
$queue->Load('perl5');
$recent = new RT::Tickets( $CurrentUser );
$recent->LimitQueue( VALUE => $queue, OPERATOR => '=' );
$recent->Limit( FIELD => 'Created',
                OPERATOR => '>',
                VALUE => 'one week ago' );
$recent->Limit( FIELD => 'Status',
                OPERATOR => '=',
                VALUE => 'New' );
$recent->Limit( FIELD => 'Status',
                OPERATOR => '=',
                VALUE => 'Open' );
```

Recent Tickets: TicketSQL

```
$recent = new RT::Tickets( $CurrentUser );
$recent->FromSQL(
"(Queue = 'perl5' and Created > 'one week ago')
  and (Status = 'new' or Status = 'open')"
);
```

Current Status

```
my %data = ();
my $q = "perl5";
for my $status (
qw[new open stalled resolved] )
{
    my $search = new RT::Tickets($User);
  $search->LimitQueue( VALUE => $q );
  $search->LimitStatus( VALUE => $status );
  my $c = $search->Count;
  $data{$status} = $c;
}
```

Writing Renderers

Loop over all the data and print it out formatted:

```
<table>
<tr><th>Name</th><th>Email</th></tr>
% while( my $u = $users->Next ) {
    <tr><td><% $u->Name %></td>
        <td><% $u->EmailAddress %></td></tr>
%}
</table>
```

Categorized Tickets

- Organize by Custom Field

Operating System	
aix	23
All	4
bsdos	7
cygwin	12
cygwin_nt	0
darwin	4
dec_osf	16
dgux	0
dos	0
dynixptx	0
freebsd	48

Severity	
abandoned	0
fatal	2
High	115
low	598
medium	379
none	11
Normal	1
unknown	1
Wishlist	6

Type	
5005threads	0
bounce	0
Bug	0
compiler	0
configure	1
core	450
dailybuild	0
docs	5
Documentation	0
duplicate	0
install	44

Perl Version	
1.0	0
5.000	0
5.002	0
5.003	0
5.004	0
5.004_00	0
5.004_01	0
5.004_02	0
5.004_03	0
5.004_04	3
5.004_05	1

Patch Status

Parrot Patch List

Open Patches

Title	Status
(801) [PATCH] PerlArray in scalar context	Pending
(15308) Dans Feedback Integrated into Documentation	Pending
(16077) 'assign' opcode and unmorphing	Applied
(16087) [PATCH] Scratchpad pmc	Pending
(16098) First draft of PerlScalar PMC	Pending
(16114) [PATCH] faster assembler	Pending
(16237) [PATCH] register window flush for sparc	Pending
(16258) [PATCH] The Great Renaming	Pending

Conference Organizing

Track Report

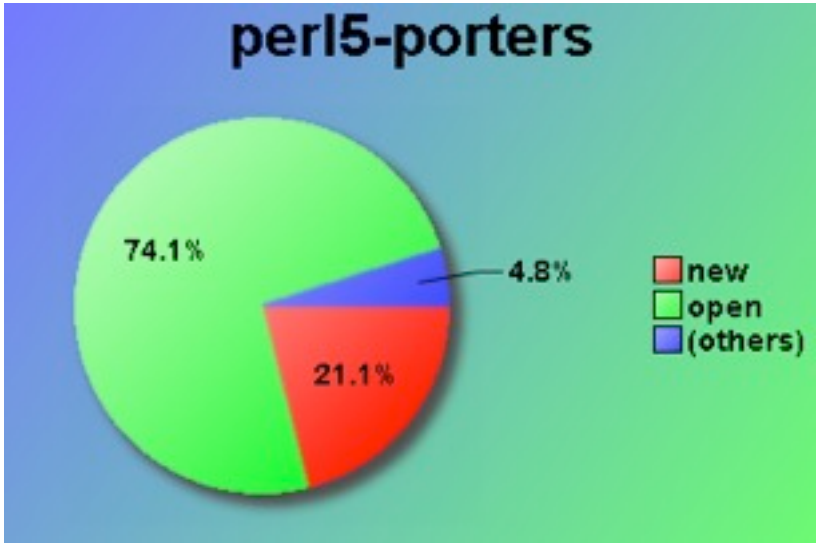
Track	Tutorial	Talk	Uncategorized
Perl	52	88	0
PHP	11	12	0
Python	23	42	0
Apache	10	30	0
Apps	24	62	0

Code Snippet

```
<%perl>
my $cfs = RT::CustomFields->new($RT::SystemUser);
$cfs->LimitToQueue("perl5");
while (my $cf = $cfs->Next) {
  my $cfn = $cf->Name();
  my $cf_values = $cf->Values;
  while (my $cfo = $cf_values->Next()) {
    my $cfv = $cfo->Name();
    my $query = qq[Queue = "perl5" AND "cf.{$cfn}" = "$cfv"
                AND ( Status = "New" OR Status = "Open")];
    my $z = new RT::Tickets( $CurrentUser );
    $z->FromSQL( $query );
</%perl>
# ... renderer ...
%  }
%}
```

Graphing

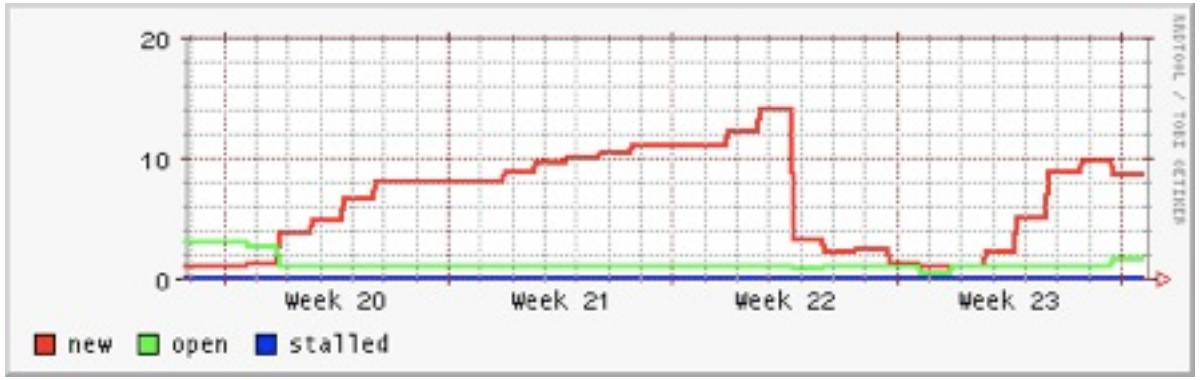
- Pie Chart
- Imager::Graph




```
my @status = qw[new open stalled];
for my $status (@status) {
  my $search = new RT::Tickets($User);
  $search->FromSQL(q[Queue="perl5" and Status="$status"]);
  push @data, $search->Count;
}
use Imager::Graph::Pie;
my $pie = Imager::Graph::Pie->new;
my $img1 = $pie->draw(data=>\@data);
$img1->write(file=>"out.png");
```

Graphing (II)

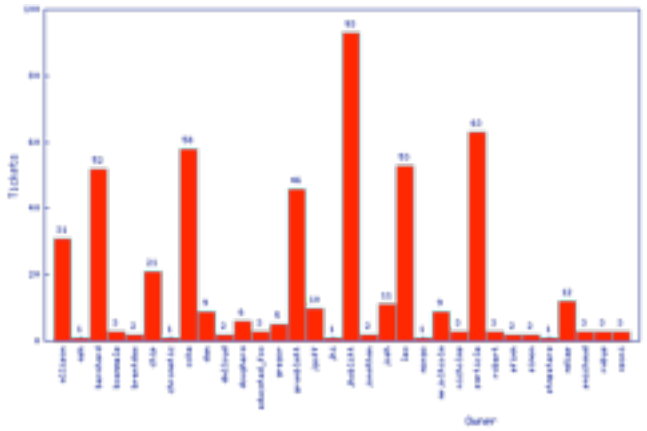
- Values Over Time
  - RRDtool

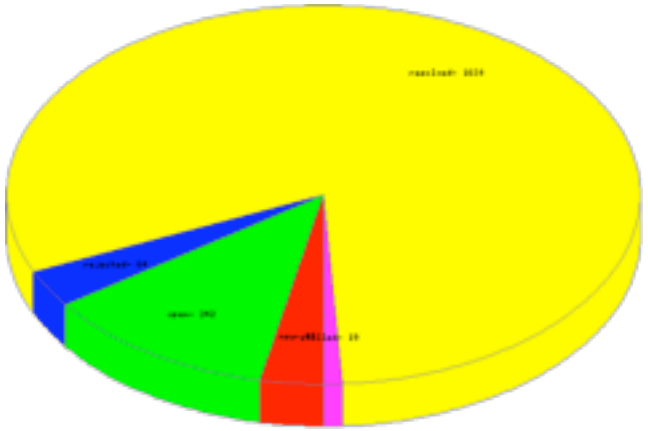


- More...
  - GD::Graph
  - Gnuplot

Built in Graphing Support

- From Search Results
  - Pie
  - Bar





BEST PRACTICAL ™

Custom RSS Feeds

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF ...><channel rdf:about="<%$RT::WebURL%>">
    <title>Tickets</title><link><%$RT::WebURL%></link>
</channel>
% while (my $Ticket = $Tickets->Next()) {
  <item rdf:about="<%$RT::WebURL%>/?q=<$Ticket->Id%>">
    <title><%$Ticket->Id%>: <%$Ticket->Subject%></title>
    <link><%$RT::WebURL%>/?q=<%$Ticket->Id%></link>
  </item>
% }
</rdf:RDF>
% $m->abort();
<%INIT>
my $Tickets = RT::Tickets->new($session{'CurrentUser'});
$Tickets->FromSQL( 'Owner = '.$session->{'CurrentUser'}->Name . 'AND
Status = "open"' );
</%INIT>
```

Ical Export

```
BEGIN:VCALENDAR
CALSCALE:GREGORIAN
VERSION:2.0
%while (my $ticket = $tix->Next) {
% my $start = Date::ICal->new(
epoch => $ticket->DueObj->Unix);
BEGIN:VEVENT
SUMMARY:#<%$ticket->Id%>: <%$ticket->Subject%>
DTSTART;VALUE=DATE-TIME:<% $start->ical%>
DTEND;VALUE=DATE-TIME:<% $start->ical %>
END:VEVENT
% }
END:VCALENDAR
```

Direct SQL Query

- Bending the rules
  - Hard to write
  - Hard to understand
  - Hard to maintain
- Things to remember
  - Ticket.EffectiveId
  - Disabled Flags

Reporting Extensions

- RT::Extension::ActivityReports
- RTx::Statistics

Approvals and Workflow

Approval Basics

- Create tickets based on a template
- Dependency Chains
- Often used for "Managerial Approval"
- Workflow Modeling
- "a ticket depends on its approvals"

Example Approval

- OnCreate / Create Tickets
- Template:
  - ===Create-Ticket: ManagerApproval
  - Subject: {$TOP->Subject} Approval
  - Queue: ___Approvals
  - Type: approval
  - Depended-On-By: {$TOP->Id}
  - Content: Please review {$TOP->OwnerObj->Name}'s ticket number {$TOP->Id} for approval.
  - ENDOFCONTENT

Enforcement

- RT requires all dependencies to be resolved before a Ticket can be resolved

CreateTicket Templates

- ===Create-Ticket: title
- Fields:
  - Queue, Subject, Status, Due, Starts, Started, Resolved, Owner, Requestor, CC, AdminCC, TimeWorked, TimeEstimated, TimeLeft, InitialPriority, Type, DependsOn, DependedOnBy,RefersTo, ReferredToBy, Members, MemberOf, Content, ContentType, CustomField-<id#>

Important Fields

- Queue
- Type
  - Ticket
  - Approval
- Content
  - ENDOFCONTENT

Approval Notifications

- Approvals queue configured by default
- To edit, you need to get there manually
  - $RT::WebUrl/Admin/Queues/Scrips.html?id=2

Default Approval Scrips

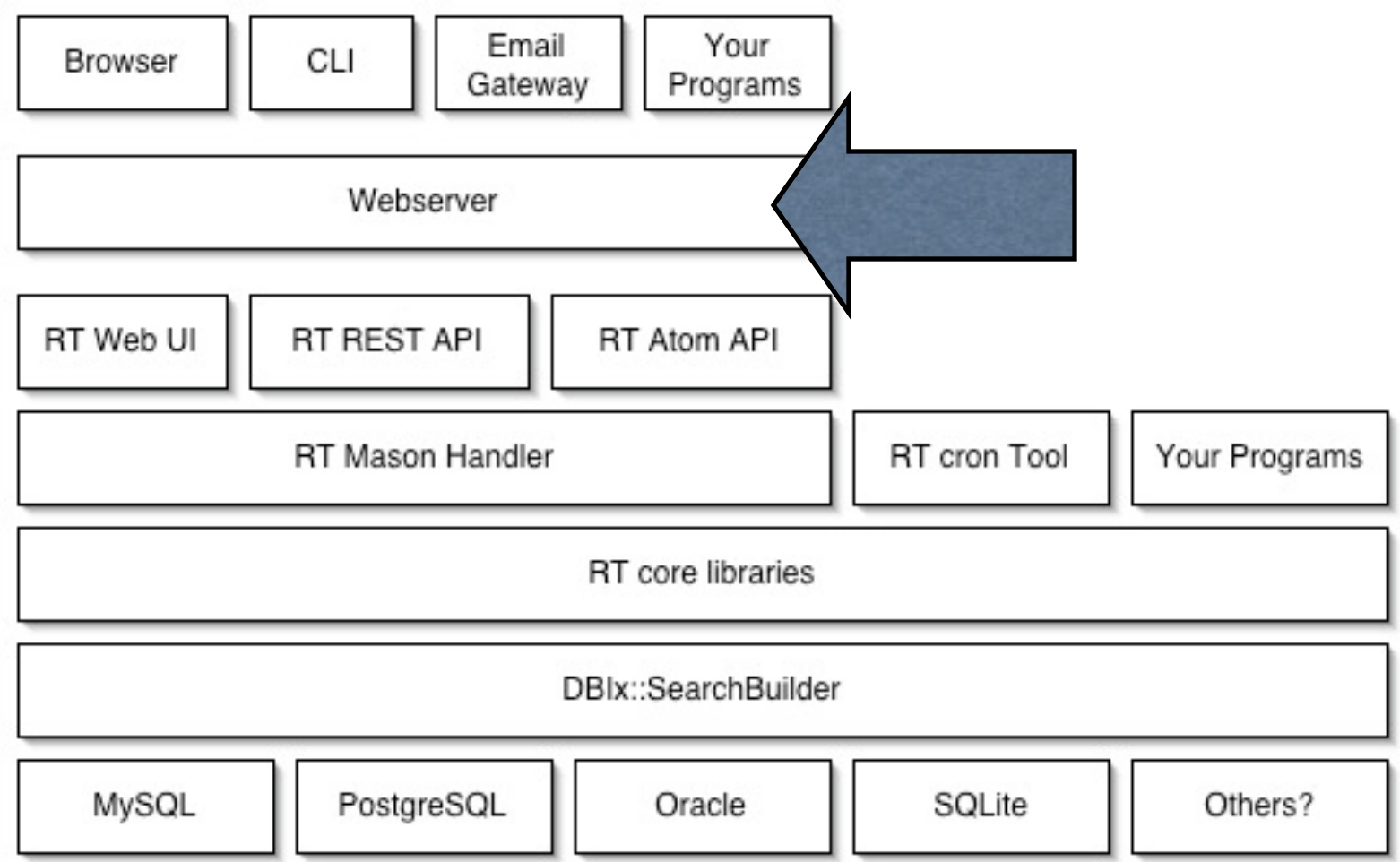
- When creating an approval ticket: notify the Owner and AdminCc
  - User Defined / Notify Owner / template New Pending Approval
- If an approver rejects: reject the original and delete pending approvals
  - On Status Change / User Defined / template Approval Rejected

Default Approval Scrips

- After any approver approves: add correspondence to the original ticket
  - On Resolve / User Defined / template Approval Passed
- After all approvers approve: add correspondence to the original ticket
  - On Resolve / User Defined / template All Approvals Passed

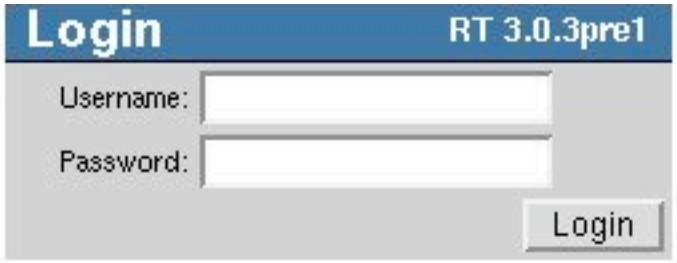
Authentication Tips and Tricks

The RT Layer Cake



Basics

- RT has a built in authentication system
- Users
- Groups
- Delegation of Rights



BEST PRACTICAL ™

Unified Authentication

- Simplify your life
  - Centralized Administration
- Simplify your users' lives
  - One username
  - One password

mod_auth_*

- LDAP
- Kerberos
- PAM
- NT Domain
- NDS
- Lotus Notes
- Radius

SMB
TACACS+
NIS/YP
SecureID
Text files
SQL Database

RT_SiteConfig.pm

- Trust REMOTE_USER
  - Set($WebExternalAuth, 1);
- Fallback to RT database
  - Set($WebFallbackToInternalAuth, 1);
- Autocreate Users
  - Set($WebExternalAuto, 1);

Apache Side

- Different for every module.
- Basic Format is
  - <Location /rt3>
    <Limit GET POST>
    AuthType Basic
    AuthName "MyCo RT"
    AuthUserFile /etc/httpd/rt-passwd
    require valid-user
    </Limit>
    </Location>

MySQL Auth

- <Location /rt3>
- SetHandler perl-script
- PerlHandler RT::Mason
- AuthName perl.org
- AuthType Basic
- AuthMySQLHost localhost
- AuthMySQLDB userdb
- AuthMySQLUser userdbuser
- AuthMySQLPassword userdbpass

RT as an Authentication Source

- Users table
  - Name
  - Password (MD5)
- Suitable for mod_auth_mysql
- Easy to use elsewhere

RT Auth Ideas

- mod_auth_rt*
- Authen::RT*
- pam_rt*
- Overlay RT::User->IsPassword()

- * denotes figment of Robert's imagination

Tuning and Debugging

Places to look when things go wrong

Logging (I)

- A _lot_ of information gets logged
  - RT_SiteConfig.pm:

- Set($LogToSyslog,   '');
- Set($LogToScreen,   'error');
- Set($LogToFile,   'debug');
- Set($LogDir,   '/opt/rt3/var/log');
- Set($LogToFileNamed, 'rt.log');

Logging (II)

- Levels:
  - debug, info, notice, warning, error, critical, alert, emergency

- Use:
  - $RT::Logger->warning("beware of dog");

Apache Error Log

- Captures STDERR
- Perl Warnings/Errors

Small Test Cases

- Small command line scripts are easier to debug than the Mason UI

Optimization and Tuning

Database Tuning

Bang for the Buck

- RT is database-bound
  - Making your database faster will make RT faster
- The Hardware Solution
  - More RAM
  - More RAM
  - Faster and more CPUs
  - Faster Disk

MySQL

- my.cnf
  - key_buffer=256M
  - table_cache=256
  - sort_buffer=2M
  - record_buffer=2M
  - thread_cache=8
  - thread_concurrency=4
- Higher is better... to a point
- MySQL 5.1 isn't ready

MySQL Query Cache

- Cache results at the database level
  - query_cache_size=32M
  - query_cache_type=1

Postgres

- Tell it to use more RAM
- VACUUM ANALYZE
  - Improves query analyzer
  - Query Analyzer highly configurable
  - Newer versions are better
- Use 8.1.3 or newer

Query Tuning

- Postgres and MySQL have very different query optimizers
- One sub-optimal query can take a disproportionate amount of time
- Use query logging to identify bottlenecks

MySQL: Logging

- to enable Query Logging...
  - my.cnf: log
  - safe_mysqld: --log
  - output: `hostname`.log
- to enable Slow Query Log...
  - my.cnf: log-slow-queries
  - my.cnf: long_query_time seconds (default: 10)
  - safe_mysqld: --log-slow-queries
  - output: `hostname`-slow.log

MySQL: Explain

How a SELECT is processed

```
mysql> explain SELECT  * FROM Scrips WHERE id = '13';
+--------+-------+---------------+---------+---------+-------+------+-------+
| table  | type  | possible_keys | key     | key_len | ref   | rows | Extra |
+--------+-------+---------------+---------+---------+-------+------+-------+
| Scrips | const | PRIMARY       | PRIMARY |       4 | const |    1 |       |
+--------+-------+---------------+---------+---------+-------+------+-------+
1 row in set (0.09 sec)
```

Good: where, index

Bad: temporary, filesort

MySQL Documentation, Chapter 5

Postgres: Logging

- Postgresql.conf
  - log_statement = true
  - log_duration = true
  - log_timestamp = true

Postgres: Explain

- Postgres' explain works like MySQL's.
- Different results

Oracle

- Take your DBA out to lunch
- ..often

Adding New Indexes

- If filesort, temporary, or table scan...
- An index might help
  - Single Column Index
  - Multiple Column Index
  - Different Databases
  - Per-site differences
- The wrong indexes can hurt

Perl DBI Logging

- DBI_TRACE environment variable
  - Before starting apache,
    - export DBI_TRACE=1=file.log
- Values
  - 0 - Trace disabled.
  - 1 - Trace DBI method calls returning with results or errors.
  - 2 - Trace method entry with parameters and returning with results.
  - 3+ - Even more details
- perldoc DBI

Thanks!

Contributing

- rt-bugs@bestpractical.com
  - If you find bugs, everyone wins if you report them
- rt-devel@lists.bestpractical.com
  - Lots of other folks use and hack on RT. Join the mailing list to share tips and tricks.

Recommended Reading

- RT Essentials
- Embedding Perl in HTML with Mason
- perldoc RT::StyleGuide
- perldoc RT::Ticket
- perldoc RT::Ticket_Overlay