

Loggerithim

The Documentation

Cory 'G' Watson

Loggerithim: The Documentation

by Cory 'G' Watson

Copyright © 2002 Cory Watson

Table of Contents

Preface	vii
Platform Notes	vii
Contact Information	vii
Getting Help	vii
Acknowledgments	vii
1. Introduction	1
What is Loggerithim?	1
Features	1
Metrics	2
2. Installing Loggerithim	3
Requirements	3
Perl Modules	3
Automated Install	4
Installing the loggeragent	5
Starting the loggeragent	5
3. Getting Started	7
Starting Loggerithim	7
Some Basics	7
Speaking Of Hosts...	7
Results	8
Preferences	8
4. A Closer Look	9
Configuration	9
Groups, Users, and Contacts	9
Smeeplets	10
Attributes	10
Thresholds	10
Jobs	10
Events	11
Reports	11
5. Understanding Loggerithim	12
Access	12
Handler	12
Pages	12
Smeeper	12
EventHandler	12
6. Extending Loggerithim	14
LastSampled Smeeplet	14
HylaFAXCheck Smeeplet	15
Index	17

List of Tables

1.1. Monitored Metrics	2
4.1. Configuration Options	9

List of Examples

6.1. LastSampled Smeeplet Beginning	14
6.2. LastSampled Smeeplet Continued	14
6.3. LastSampled Smeeplet End	15
6.4. HylaFAX Smeeplet	15

Preface

Time travel has long been the stuff of science fiction. For as long as people have been able to break things, the ability to manipulate time has been on their minds. Stories, books, and even movies have repeatedly visited this subject.

Managing the various devices involved in our IT infrastructures could certainly benefit from the use of time travel. The ability to go back and see what caused an outage, or how things ran before the last patch would be useful.

For as long as people have had to perform labor, they have desired to offload it onto others. Computers have helped us in this, usually generating more work than they save, but that is a different story all together.

Having a robot that can intelligently do your bidding (well, withing reason) would also be beneficial. A robot that can speak to your computers and give them orders might come in handy.

Simply put, Loggerithim aims to be your time machine and robot, allowing you to go back some amount of time and see how things were. While we are at it, we'll also use this flow of information to check for problems. We will have the opportunity to give the machine some instructions if there are problems, or maybe use that same ability to detect problems before they happen.

Platform Notes

Loggerithim is developed on RedHat Linux 7.2 for x86, Solaris 6 for Sparc, Solaris 8 for Sparc, and MacOS X Jaguar (10.2). Perl 5.8.0, Apache 1.3.26, mod_perl 1.27, and PostgreSQL 7.3.3 round out the major software portions. Testing is done on RedHat Linux 8.0 as well.

All the components of Loggerithim are written with portability across UNIX boxen to the best of the author's understanding, which may be lacking on many platforms. Still, all the pieces *should* play well together on any OS where Perl and C work properly.

Contact Information

This documentation is a living entity. It's contents are probably littered with misspellings and grammatical errors. Maybe some of the instructions are wrong. Please communicate any errors you find or suggestions you have to the mailing list at <loggerithim-users@lists.sourceforge.net> or directly to the author at <gphat@cafes.net>.

Getting Help

Loggerithim can be a complicated beast. If you need help, visit the web page at <http://loggerithim.sourceforge.net>.

Acknowledgments

Despite the fact that you currently only see a single name on the documentation and source code of Loggerithim, quite a few people have had a hand in making it work. Most notably is my employer, InPHact. The management there has been kind enough to allow me to release my project to the public under an Open Source License. Specific thanks go to Helen, Rick, and Howard for being understanding managers and helpful critics. Thanks to Canton for incessant proofreadings and long shpleels about how this or that worked. Also to Mike for constantly telling me that pieces were not good enough (not that he has stopped yet), and Kevin for inspiring my early mod_perl adventures. Finally, my lovely wife Jennifer for pretending to care when I show her the new whiz-bang feature I cooked up.

Chapter 1. Introduction

What is Loggerithim?

So you've heard all these great things about Loggerithim. You've seen the commercials, you've read the reviews, but you don't really know what it is. Well, all that is about to change.

Loggerithim's primary mission in life is to collect data from your machines and present it to you in a way that makes sense. At the same time, Loggerithim can send commands to it's agents to do just about anything you can imagine. It saves the data it collects and allows you to look back in time to see how your machines were behaving last week, last month, or last year. Reports can be run, Notifications can be sent, and Hosts can be compared. Basically, Loggerithim tries to make the life of a System Administrator as easy as possible.

In the following chapters we will cover the installation, configuration, and use of Loggerithim.

Warning

This is the first public release of Loggerithim. It is highly recommended that you take some time to evaluate and test it before deciding to dedicate any real resources to it. This will change as more people use it and smooth the sharp, rusty edges.

Features

Now we know what Loggerithim is supposed to do at a high-level, let's outline the features that it will provide.

- *Web-based* - Using mod_perl and Apache, Loggerithim does not require any heavy client software. This enables you to check the status of your assets from any network-connected machine with a browser. Loggerithim should work in any browser, as there are no fancy HTML or Javascript features to cause problems.
- *Remote Management* - Loggerithim's monitoring agent is capable of executing commands sent from the main server, enabling you to automate to your hearts content. The agent is also extensible, allowing you to write your own functionality.
- *Security* - All of Loggerithim's communication is performed over SSL encryption, fortified with certificate based authentication.
- *Department and System Organization* - The Hosts monitored by Loggerithim are organized into Departments and Systems. Department examples would be 'Production' or 'QA'. Systems could be 'Mail', 'DNS', or the name of your custom application. Loggerithim will then display Hosts in the same Department and System together.
- *Attributes and Profiles for Hosts* - Attributes describe the capabilities of a Host. These Attributes allow Loggerithim to treat a host as a simple entity, and to abstract the data that can be reaped from it. Profiles are an initial set of Attributes to be assigned to a new host.
- *Status at a Glance* - Important metrics are color coded on the main page to give the status of a System in a single glance.
- *Agent-Based Metric Collection* - Key metrics are gathered from agents on monitored Hosts at configurable intervals. These metrics are stored in Loggerithim's database for whatever use we can cook up. If the agents can't contact the central server, they will save the metrics and report them when a connection can be established.
- *Plugins* - Users of Loggerithim can leverage the power of Perl to gather custom metrics, check for problems, or whatever else they can dream up.
- *Cross-Platform* - The loggeragent currently runs on Linux and Solaris, with initial support for OS X and Windows. Other platforms are easy to add.

- *Notification* - Notify users when things happen. Events automatically adjust severity based on the number and importance of events.
- *Per Host, Per Metric, Per Key Thresholds* - Set Thresholds on individual Hosts, individual Metrics for a Host, or even individual keys of a Host. That's a lot of individuals.

Metrics

Out of the box, Loggerithim monitors the following performance indicators, called metrics.

Table 1.1. Monitored Metrics

Memory	Real Memory and Swap. Buffered and Cached memory are included in Linux.
CPU	Idle, User, and System time
Load	1, 5, and 15 minute averages
VM Subsystem	Paging and Swapping
Storage	Utilization
Interfaces	Traffic in and out.
Misc.	Uptime, OS/Kernel Version

The metrics that come standard will not satisfy everyone who wants to use Loggerithim, but they will more than likely be the common denominator. They were chosen because they are common between all systems, and because each of them are useful indicators of system performance.

Loggerithim is by no means limited to these metrics. Any statistic can be added to the loggerithim agent, or collected using a Smeeplet. Most any metric can be added without modifying any of the existing code. This will be discussed in detail in chapter 6, *Extending Loggerithim*.

Chapter 2. Installing Loggerithim

Requirements

Loggerithim is written in Perl, makes use of a number of Perl Modules, and runs under mod_perl. Obviously, each of these is required, unless you plan on hacking your own port of Loggerithim. The installation of mod_perl and Apache is beyond the scope of this discussion, but instructions can be found in at the mod_perl website, <http://perl.apache.org>.

Loggerithim is developed on Linux, RedHat 7.2 specifically. It should work just fine on any distribution, as development is also done on Gentoo Linux. The loggeragent only runs on Solaris (2.6 and 2.8 tested) and Linux (x86 and PPC tested). There is initial support for MacOS X and Windows.

Perl Modules

Loggerithim requires the following modules to be installed. All are available from CPAN.

- *GD* - The GD library allows us to manipulate graphics, which is the key to our graphs.
- *GD::Polyline* - The Polyline extension to GD gives us more flexibility in the creation of the graphs.
- *DBI* - Tim Bunce's wonderful module that provides a layer of abstraction between Perl and the many databases available. Loggerithim uses this to speak to it's database, as well as any others you might choose to converse with.
- *Apache::DBI* - Loggerithim uses this module to avoid recreating database connections for each Apache request.
- *Apache::Session* - Provides session support.
- *HTML::Template* - Template System.
- *IPC::SharedCache* - Used to cache prepared templates and objects, greatly improving performance. Depends on *IPC::ShareLite*, which should be handled by CPAN automagically.
- *Time::HiRes* - Provides a high-resolution timer for timing operations.
- *Apache::Cookie* - Cookie API. This requires *libapreq*, which should be handled by CPAN automagically.
- *Math::NumberCruncher* - Fancy math functions. Depends on *Math::BigInt*, which should be handled by CPAN automagically.
- *Net::SMTP* - Allows Loggerithim send email.
- *Net::SSLeay* - Used to communicate with the agents.
- *DBD::Pg* - DBI Driver for PostgreSQL.
- *MD5* - Hashing for Cookies.
- *Cache::Cache* - If caching is enabled, Loggerithim will use this module to cache objects.
- *XML::Simple* - Metrics are sent as XML, and we need to be able to parse it, hence *XML::Simple*.
- *XML::Writer* - Commands are sent to agents using XML, so we need a way to write it.
- *XML::Writer::String* - *XML::Writer* only likes to output to *IO::Handle* objects, and this class allows us to capture it's output.

Many of these modules may have their own requirements, asking you to install a handful of other packages. This is expected, and is OK. Also, the versions should be irrelevant, as development is usually done with whatever is most recent on CPAN.

Automated Install

Loggerithim makes use of Autoconf and Automake to simplify it's installation.

Before you can install Loggerithim, you must install PostgreSQL, Apache, and mod_perl. Then, uncompress the Loggerithim tarball and run the autoconf script.

```
#./configure
```

The script will try and locate your apache, openssl, and perl binaries. If that succeeds, each of the modules Loggerithim requires will be checked.

```
Checking for GD... okChecking for GD::Polyline... okChecking for DBI... ok
```

If all the modules are found, the configure script will finish by writing Makefiles. The next step is to type 'make'. This will create manual pages for all the Loggerithim objects.

You can now install Loggerithim by typing 'make install'. This will place the required files in the appropriate places. The installation is now complete, but there are still some things that need to be done.

With all the files installed, now it is time to create the database. Execute 'make database' to start this. It will first ask you where the database is running:

```
On what host is your database running?: [localhost]
```

If your PostgreSQL database is running locally, then you can take the default (localhost) by pressing enter. If it is running on another host, put that host's hostname or IP address in. Next, you will be asked for the port on which PostgreSQL is listening. Unless you have changed it, just press enter for the default.

Now you will be prompted for the username and password of a user that can connect to the PostgreSQL server to create the Loggerithim database. This user will need need the ability to create databases and users. You can use the **createuser** command to add a user with these privileges. You may have to find the createuser command on your system, as it may not be in your normal \$PATH.

```
$createuser -P
```

Answer yes to the questions concerning creating users and databases.

Tip

This user is basically a root user for the PostgreSQL system, and you may already have one setup on your system. We will leave that as an exercise. This is probably the best way to do things.

After you've given the password to user, you will be prompted to provide a password for the loggerithim PostgreSQL user. This user will be used by the application to manipulate the loggerithim database. It will be entered into Loggerithim's configuration file.

The installer will now create all the views, tables, and sequences for the Loggerithim database. You will then be prompted for the username and password of your Administrative user. This user will be created so that you can login to the web interface.

Finally, you will be asked for an SMTP server and a cookie 'secret'. The SMTP server is for sending alerts, and the cookie secret is used to stop people from modifying cookie values on their machines.

Next, we need to customize the configuration file that Loggerithim includes in Apache's `httpd.conf`. This file is `@prefix@/loggerithim/lr-httpd.conf`, where `@prefix@` is `/usr/local`, or something else if you customized it when you ran `./configure` earlier.

After you open this file, you should see a few places where `domain.com` was used. Replace this with your real information.

Next, we need to enable name-based virtual hosting in our `httpd.conf`. Find the line that looks like `#NameVirtualHost *` and remove the pound sign from the beginning of the line to uncomment it.

Loggerithim needs certificates to communicate with its agents, so now run 'make certificates' and answer all the questions to generate the proper certificates. You will be asked the same question a number of times, just bear with it. It will all be over soon.

Lastly, schedule the three perl scripts in the `/etc/cron.loggerithim` directory to run every minute. This is a simple thing to do for most admins, so we will only provide an example of how to do this in RedHat. Edit your `/etc/crontab` file and make a new entry.

```
* * * * * root run-parts /etc/cron.loggerithim
```

Finally, start the loggerserver via its `init.d` script.

```
# /etc/init.d/loggerserver start
```

Tip

You may want to setup this service to be started when your machine starts. Try `ntsysv` on RedHat or `rcconf` on Debian to accomplish this.

Installing the loggeragent

The loggeragent is distributed as source code, but includes targets for an RPM or Solaris package so that you create a master RPM for all your monitored hosts.

To compile the source, simply run `./configure` in the untarred loggeragent code directory (but don't do it yet). You must supply an option `'--with-cert-directory=some_directory'`, where `some_directory` is the path to the certificates generated by a Loggerithim install. By default, this is `/usr/local/loggerithim/certificates`. If you do not want to install the various libraries that the loggeragent requires on each machine that you plan to monitor, you should also supply the `'--enable-static'` option to configure. This will produce a static binary that has all the libraries included. This is probably the best way to do things.

After you have run configure, edit `res/loggeragent.xml` and change the `'metrichost'` to the hostname of the machine that you will be running the Loggerithim backend on, specifically the loggerserver. You might want to change the `'metricinterval'` setting if you would like your machines to report metrics more or less often than 600 seconds. Loggerithim is written to support a 10 minute (600 second) interval, so be warned that there may be adverse effects if you change it. Consult the mailing list with questions.

After changing those settings, save the file and run `'make package'` to generate the appropriate package for the machine you are on. Use this package to install the agent on the machines you plan to monitor.

Starting the loggeragent

Use the init.d script installed by the package to start the loggeragent.
#!/etc/init.d/loggeragent start

Chapter 3. Getting Started

Starting Loggerithim

Loggerithim has two parts that must be started, the loggerserver and the web interface. The loggerserver was installed during the automated install, and it should have an init.d script that will automatically start it at boot time. To start the loggerserver, use this command:

```
#!/etc/init.d/loggerserver start
```

The web interface can be started by simply starting apache. We will assume you know how to do this already.

With the loggerserver running, your agents should be able to send metrics. If you visit the URL you've setup for Loggerithim, you should see a login screen.

Some Basics

If you have not logged in yet, do so now.

You should be greeted by a relatively sparse web page. This is normal, because Loggerithim doesn't have any information about your Hosts yet. Before we start adding Departments, Systems, and Hosts it is important that we take a look at each of these entities and make sure we understand them.

Hosts are entities that Loggerithim monitors. These can be machines that are running the loggeragent daemon, or devices that are monitored via Smeeplets.

Systems are an organizational unit used to group Hosts. For example, say you have an application composed of a database server and application server called WidgetMaker. You would create a System called WidgetMaker and make your database and application server a part of it.

Metrics are counters or values gathered from Hosts. Generally, this term is used to represent a pile of values sent as one big message.

Departments are organizational units that hold Systems. Using our WidgetMaker example again, you might have both a Development and Production WidgetMaker Department.

Departments and Systems have no actual relationship, they are simply tied to the Host. To setup your WidgetMaker environment you would create a Production and Development Department, a WidgetMaker system, and select the appropriate values in the dropboxes for adding a Host.

Now that we understand Departments and Systems, feel free to go to the Admin section and add some Departments and Systems. Use the 'Add A System' and 'Add a Department' links.

Speaking Of Hosts...

Now that we have covered Departments and Systems, we can work with Hosts. To monitor a machine's metrics, you need to have installed and started the loggeragent, as described in Chapter 2, *Installing the loggeragent..*

Next, add the Host to Loggerithim through the Admin section. The important fields are Hostname, IP, Active, and Purpose. When a host connects to pass in some metrics, the loggerserver tries to find the host by IP address. This means that you need to have the IP match the address that the host will be sending metrics through. If that fails, the loggerserver will try to find a host with a hostname that matches the hostname provided in the metric. If a Host has more than one hostname, use the hostname that the box gives when you execute the **hostname** command. The Active box is a flag used to determine if the box should be displayed in Loggerithim's menu. If Active is not checked, the box will not be displayed.

The Purpose field is an explanation of the box's responsibilities. You can use 'SMTP Server', rather than trying to decipher whatever silly naming scheme your manager is forcing on you.

The Profile field determines what Attributes a host has. For example, if you choose the canned Profile 'Linux' for a Host, then all the Attributes for Linux will be added to that Host. For example, if you have a pile of Cisco routers that you want to monitor, you could define a Profile that defines a set of metrics that you want to use with them. Then, when you add them to Loggerithim, you can simply choose the 'Cisco' Profile you created rather than having to add a ton of Attributes to the Host.

Changing a Profile after you've used it to add Hosts does NOT change the Attributes of existing Hosts. It is only used as a guide when the Hosts are added.

Be sure to choose a System and Department, which you added in the last section. The Database information can be ignored for now, it is used when extending Loggerithim.

Results

If you installed, configured, and started the loggeragent and also added a Host representing that box into Loggerithim, data should start collecting. Don't be worried if you click one of the links and don't get any pretty results. It takes a few samples for Loggerithim to get a box figured out. Some metrics, like CPU and network traffic, require three readings to calculate their values. Continue adding Hosts to your setup until you've added all the Hosts you want to monitor.

When you do see your first graph with real data, it should be fairly simple to read. The dropbox can be changed to graph a different number of hours, days, weeks, or months.

If you see a vertical red line on the graph, this signifies midnight. This should help you orient yourself when you are looking at a graph.

The dashed lines and the numbers drawn in the right hand margin signify the average for a particular line.

Don't forget that the logserver logs it's actions to syslog. You can tail your messages file to watch any errors that might show up.

Preferences

The user preferences are as follows:

Password - Self-explanatory, I hope.

Department & System - When you visit Loggerithim, it will show you the Department and System of your choice.

Color - If enabled, Loggerithim will color code some of the key metrics for a Hosts on the main page. This allows you to glance at a System and see it's state. This can be expensive if you monitor alot of Hosts, so it's configurable.

With what we've covered so far, it is pretty easy to keep an eye on your boxes. In the next chapter we'll dig deeper and learn about Loggerithim's more advanced capabilities.

Chapter 4. A Closer Look

The ability to use Loggerithim to it's potential requires that you understand the pieces that make it tick. The sections of this chapter outline the major ideas that are required to understand Loggerithim. Knowing them enables a user to add new Hosts, monitor specific values for anamolies, and send notifications to other users.

Configuration

Loggerithim reads it's configuration parameters from `/etc/loggerithim.xml`. Here is a table of options avail# able and what they do.

Table 4.1. Configuration Options

Option	Meaning
db_pass	Password for connecting to the loggerithim database
smtp_server	SMTP server for sending notifications
annotation	If set to a true value (i.e. 1) Loggerithim will attempt to fetch annotations from an instance of it's sister applica# tion, Catalyst.
secret	The 'secret' to use as a seed for encoding session inform# ation in the client cookie.
caching	If set to a true value (i.e. 1) Loggerithim will cache some of it's often used objects with Cache::FileCache
debug	If set to a true value (i.e. 1) Loggerithim will be verbose.

The Loggerithim installer will automatically create `/etc/loggerithim.xml` and populate the `db_pass`, `smtp_host`, and `secret` values. It is strongly recommended that you enable caching, as this yields huge speed im# provements. The `Cache::Cache` module is required for this feature.

Groups, Users, and Contacts

Groups are collections of Users who have the same privileges. Groups allow fine-grained control over who can do what, and where they can do it.

To add a group, click 'Add A Group' in the Admin section. After you have filled in the Group name and submitted it, click 'List Groups' in the Admin section and click the edit icon for the Group you just created. You can now choose to limit this group to a specific Department and/or System. This will cause any permissions you grant to only be ap# plicable to that System and/or Group.

The checkboxes allow you to specify which objects in Loggerithim this user can fiddle with. You can grant read, write, and FIXME permissions on each object.

Users are the people that use Loggerithim. They have a username, password, and fullname. To add a User, click 'Add a User' in the Admin section. After you have filled in the information and submitted it, that user should be able to login using the password you specified.

Contacts are the ways that Loggerithim gets in touch with users. Contacts are used to notify users of events in Log# gerithim and to facilitate a user who wants different Event notifications sent to different places.

Contacts have a type and a value. The type is currently either 'email', 'pager', or 'other' and the value is an email ad# dress. Loggerithim can only send events to an email address, but most places have email to pager gateways, or something similar.

Contacts are added by going to the admin page, 'List Users', then clicking the new Contact icon (looks like a sparkling business card) for the appropriate User.

Contacts might seem useless at the moment, but they come in very handy when we cover Jobs and Thresholds later.

Smeeplets

Smeeplets are Loggerithim plugins. Loggerithim comes with two Smeeplets `FileSystemSpace` and `LastSampled`. `FileSystemSpace` checks the last filesystem Metrics when it runs and generates events if any of the filesystems break Thresholds. `LastSampled` generates an event for any Host that hasn't been updated in more than twenty minutes.

Smeeplets are one of the most powerful parts of Loggerithim, as they allow anyone with Perl programming experience to create a custom monitor for their environment. This capability is extensively discussed in Chapter Five, *Extending Loggerithim*

So how do you schedule a Smeeplet to run? We'll reveal this when we cover Jobs.

Attributes

Attributes are traits of a host. Rather than try and define an Attribute, it's easier to give an example.

If you write a Smeeplet that checks the percentage of free memory a host last reported, you may not want to run it against all of the boxes you monitor. When you create your Smeeplet, its install script can create an Attribute called 'MemoryMonitor'. When you create a Job to run that Smeeplet (covered in the next section), you specify what Attribute a host must have to have in order for this Job to run against it. Next, add this Attribute to all the hosts whose memory you want checked.

To add an Attribute to a host, go to Admin, 'List Hosts', click the add attribute icon (looks like a sparkling wrench) and select it from the drop-box.

All hosts automatically have an Attribute called 'Default'. This is used when you want a Job to run against all hosts.

Thresholds

A Threshold is a value that must be exceeded to generate an event. Thresholds are added to Hosts, because 90% CPU utilization might be bad on one host, but not another.

To add a Threshold to a host, list the hosts and click the 'Add Threshold' link. Next, we must choose a Resource. If you were to choose 'stoUsed' Resource, a value of '90' and a key of '/', then Loggerithim would notify you if the amount of used space exceeded 90% on the '/' filesystem because the `FileSystemSpace` Smeeplet checks for this Threshold.

Adding a `~` to the front of the key will cause it to be matched as a regex against the filesystem name. For example, if a machine's job is to cache things and you have filesystems `/cache1`, `/cache2`, and `/cache3`, these filesystems are supposed to be full. If `/cache1` is at 100%, it's not a bad thing. A Threshold can be created with a severity of 0, value of 101, and a key of `~cache`. This would effectively cause Loggerithim to ignore any filesystem that contains the word 'cache'.

Notice: If you have more than one threshold for the same resource, they stack. The most immediate example of this is the aforementioned `FileSystemSpace` Smeeplet. By default it generates Events at Thresholds of 85% and 90% consumption. If a filesystem is at 91%, two Events will be generated, but their severities will be combined. When you view this from the Events page, it will be easily understandable.

Jobs

Jobs are scheduled, recurring tasks that handle the running of Smeeplets. When you create a Job you specify an Attribute, and any hosts that have this Attribute will have the Job's Smeeplet run against them.

To add a Job, go to admin, 'Add A Job', and choose an Attribute, Smeeplet, and enter a cron-style time. See the man page for crontab(5) if you need to learn about cron times.

For example, to schedule the FileSystemSpace Smeeplet to run at 7:15am, and 3:15pm, we would go to the Admin page, and Add A Job. Select the 'Default' attribute and the 'FileSystemSpace' smeeplet from the list. Then set the interval to '15 7,15 * * *'. This tells Loggerithim to run the FileSystemSpace Smeeplet against all the Hosts with the Default attribute at 7:15 and 15:15.

Some Jobs check values against Host's Thresholds. If these Thresholds are exceeded, an Event is generated. A notification of the Event is then sent to any Contacts that are specified for the Job. To add a Contact to a Job, choose 'List Jobs' from the Admin menu, and click the 'Add Contact' link beside the Job you'd like to add the Contact to.

You can remove a Contact from a Job by choosing 'List Contacts' from the list of Jobs. Click the 'Remove Contact' link of the Contact you would like to remove to stop notifying that person when this Job gets verbose.

Events

Events are things that happen in Loggerithim. They are generated by Jobs, and are used to notify you of the results of those Jobs. The EventHandler cron job handles sending Event notifications to the Contacts specified in the Job. When Events are created, they are assigned a severity by the Smeeplet that creates them. If more than one Event is created for a given Job on a given host, then they are considered a Child Event. Loggerithim calculates a relative severity by combining the severities of the parent and all child Events. Events whose subsequent child Events increase in severity will have a higher relative severity than one whose child events go down in severity. This allows Events that either get worse or fix themselves to express their severity to the person who is notified.

Reports

Reports aren't used at the moment. It is likely they will be reworked. If you are interested in them, see the code in the Reporters directory of Loggerithim or join the Loggerithim mailing lists.

Chapter 5. Understanding Loggerithim

Before we start talking about writing new parts to Loggerithim, we should cover some of the important pieces that work in the background.

Access

Loggerithim::Access is a PerlAccessHandler. It verifies that any user trying to view a page other than the error or login page has a valid session.

Handler

The Handler is the circulatory system of Loggerithim, make sure that users are properly distributed to the pages they desire. Our tour begins by pointing out that Loggerithim::Handler begins life as a normal mod_perl PerlHandler. Please follow along in the source for the best understanding.

First, we build an array of pages Loggerithim will be answering for. This is done by opening the Page directory and getting a list of all the .pm files. For each Page object found, its object is created and stored in a hash. Doing this at startup saves us from having to instantiate the page at request time.

At request time, the handler() method is called. The Apache::Request object is shifted in and the uri() method is called on that object to give us the filename requested. A regular expression next validates that the request has a proper page name in it. That page name is then fetched from the page object cache.

Assuming that the page was in the cache, the parameters that were POSTed or GETed are stored in a hash and the page's handler() is called. The Request object and the parameters from any forms/urls/etc are passed to the page.

The page returns a hashref suitable for use with HTML::Template. Any special actions (Cookie Creation, Redirect#tion or setting the no_cache flag) are performed if necessary. The HTML::Template object is then created and supplied with the arguments to create the requested page. The content type is sent to the client along with the output of the template.

Pages

The pages are made up of two methods; new() and handler(). The new() method is a normal Perl constructor. handler() returns a hash reference of all the things going into the template, as well as a few special parameters that the Handler takes care of.

Smeeper

Smeeper.pl runs minutely and checks for any Jobs that need to be run. If one is found, its Smeeplet is executed. The Smeeplet queries the database for the Thresholds relevant to its parent Job and uses them to determine its state. The Event object will create a child Event if the Event has already been stored in the table. The act of create a child Event sets the hushed flag to 0 for the parent Event. The hushed flag will be discussed in the next section.

EventHandler

The EventHandler also runs minutely. It scans the events table for any Events with hushed and squelched flags that are both set to 0. It calculates a relative severity using the following equation:

```
foreach child Event of this parent {
  modifier += (Parent Severity) - (Child Severity) + 1;
}
```

In other words, for each child the relative severity will increase by one point plus the difference between it and the parent. This means that if a child Event suddenly crosses from Warning to Danger, this change will be taken into consideration, accelerating the natural escalation of an Event. Events otherwise gain one point of severity for each child Event. If the severity of the child decreases, it will have a negative effect on the parent. This will either keep the relative severity the same or decrease it. This allows the changing state of the situation to effect the overall severity of the problem. Therefore problems that get solved by themselves will not continue to pester the people it notifies.

Chapter 6. Extending Loggerithim

The ability to add your own code to Loggerithim is one of its most powerful features. In this chapter we will follow the creation of two Smeeplets.

LastSampled Smeeplet

The `last_sampled` field of the Hosts table gives us the last time that the Host had any data successfully collected, either from an agent or from a Smeeplet. This could give us insight into a host that is failing to send metrics to us, which obviously needs attention.

A new Smeeplet, which we will call LastSampled, will turn this idea into a valuable new tool. First, we must do the boring stuff:

Example 6.1. LastSampled Smeeplet Beginning

```
package Loggerithim::Smeeplets::LastSampled
use strict;

use base ("Loggerithim::Smeeplets::Base");

sub run {
    my $self = shift();
    my $hostid = shift();
    my $jobid = shift();

    my $job = Loggerithim::Job->new($jobid);
    my $host = Loggerithim::Host->new($hostid);
```

That code is going to be common to most any Smeeplet. You may need to 'use' other libraries if you use other features of Loggerithim's classes, that are not included by the `Loggerithim::Smeeplets::Base` class. The only non-standard class that we included is `Loggerithim::Date`, which will give us some convenience methods for fiddling with the date we are going to get from `last_sampled`. `run()` is called by Smeeper to actually run the code. `run()` is passed the `hostid` of the Host it should run against, and the `id` of the Job that is responsible for this run.

You may notice there is no `new()` method for this class. Again, `Loggerithim::Smeeplets::Base` saves us some work by providing a stock `new()` that does, well, nothing special. If you need to do anything cute in your constructor, just add your own.

With that out of the way, we can write some real code:

Example 6.2. LastSampled Smeeplet Continued

```
my $db = Loggerithim::Database->new();
my $dbh = $db->connect();

my $laststh = $dbh->prepare("SELECT last_sampled FROM hosts WHERE hostid=?");
my $laststh->execute($hostid);
my $lastref = $laststh->fetchrow_arrayref();
my $update = Loggerithim::Date->unixFormat($lastref->[0]);
```

We get a database connection from `Loggerithim::Database`, and we execute a query that will give us the `last_sampled` value for the host in question. We then convert the date from a Postgres format into a Unix timestamp, or the number of seconds since epoch.

Example 6.3. LastSampled Smeeplet End

```

my $twentyAgo = time() - (60 * 20);
if($update < $twentyAgo) {
    my $name = $host->hostname();
    my $ip = $host->ip();
    my $message = "$name:$hostid ($ip) has not been updated since ".$lastref->
    my $event = Loggerithim::Event->new();
    $event->hostid($hostid);
    $event->jobid($jobid);
    $event->severity(5);
    $event->identifier("LAST UPDATED");
    $event->text($message);
    $event->commit();
}
$laststh->finish();
$dbh->disconnect();
}
1;

```

Next, we calculate what the Unix timestamp was 20 minutes ago, and if our value from the Host is older, then we create an Event.

That wasn't so bad, was it? Our next example will check the availability of a network service.

HylaFAXCheck Smeeplet

Smeeplets can also be used to check for the availability of a service. For example, you might want to check on the availability of your HylaFAX service.

We will start with the usual, boring stuff.

Example 6.4. HylaFAX Smeeplet

```

package Loggerithim::Smeeplets::HylaFAXCheck;
use strict;

use base ("Loggerithim::Smeeplets::Base");
use IO::Socket;

sub run {
    my $self = shift();
    my $hostid = shift();
    my $jobid = shift();

```

```

my $job = Loggerithim::Job->new($jobid);
my $host = Loggerithim::Host->new($hostid);

my $fail;
my $remote = IO::Socket::INET->new(
    Proto      => "tcp",
    PeerAddr   => $host->ip(),
    PeerPort   => 4559,
) or $fail = 1;

if($fail) {
    my $name     = $host->hostname();
    my $ip       = $host->ip();
    my $message  = "HylaFAX services on $name:$hostid ($ip) are not responding.";
    my $event    = Loggerithim::Event->new();
    $event->hostid($hostid);
    $event->jobid($jobid);
    $event->severity(9);
    $event->identifier("HYLAFAX CHECK");
    $event->text($message);
    $event->commit();
}

print $remote "QUIT\n";
close($remote);
return;
}
1;

```

In this example we use IO::Socket to try and connect to the HylaFAX service on port 4559. If the connection fails, we create an Event. If it succeeds, we go back to twiddling our thumbs.

Index

A

Attribute
 Default, 10
Attributes, 10
 adding, 10

C

Configuration, 9
Contacts, 9
 adding, 9

D

Departments, 7

E

EventHandler, 12
Events, 11
 child, 11

G

Groups, 9
 adding, 9

H

Host
 purpose, 7
Hosts, 7
 adding, 7

J

Job, 10
 adding, 10
 adding Contacts, 11
 removing Contacts, 11

L

Loggerithim::Access, 12
Loggerithim::Handler, 12

M

Metrics, 7

P

Profile, 7

S

Smeeper, 12
Smeeplets, 10
 scheduling, 10
Systems, 7

T

Thresholds, 10
 Adding to a Host, 10

U

User, 9
 preferences, 8