
Heritrix User Manual

Internet Archive
Kristinn Sigur#sson
Michael Stack
Igor Ranitovic

Table of Contents

1. Introduction	1
2. Installing and running Heritrix	2
2.1. Obtaining and installing Heritrix	2
2.2. Running Heritrix	3
2.3. Security Considerations	7
3. Web based user interface	7
4. A quick guide to running your first crawl job	8
5. Creating jobs and profiles	9
5.1. Crawl job	9
5.2. Profile	10
6. Configuring jobs and profiles	11
6.1. Modules (Scope, Frontier, and Processors)	12
6.2. Submodules	19
6.3. Settings	24
6.4. Overrides	30
6.5. Refinements	31
7. Running a job	32
7.1. Web Console	33
7.2. Pending jobs	35
7.3. Monitoring a running job	36
7.4. Editing a running job	37
8. Analysis of jobs	38
8.1. Completed jobs	38
8.2. Logs	39
8.3. Reports	42
9. Outside the user interface	43
9.1. Generated files	43
9.2. Helpful scripts	45
9.3. Recovery of Frontier State and recover.gz	46
9.4. Checkpointing	47
9.5. Remote Monitoring and Control	49
9.6. Experimental FTP Support	49
9.7. Duplication Reduction Processors	50
A. Common Heritrix Use Cases	50
Glossary	52

1. Introduction

Heritrix is the Internet Archive's open-source, extensible, web-scale, archival-quality web crawler.

This document explains how to create, configure and run crawls using Heritrix. It is intended for users

of the software and presumes that they possess at least a general familiarity with the concept of web crawling.

For a general overview on Heritrix, see [An Introduction to Heritrix](#) [/An Introduction to Heritrix.pdf].

If you want to build Heritrix from source or if you'd like to make contributions and would like to know about contribution conventions, etc., see instead the [Developer's Manual](#) [http://crawler.archive.org/articles/developer_manual/index.html].

2. Installing and running Heritrix

This chapter will explain how to set up Heritrix.

Because Heritrix is a pure Java program it can (in theory anyway) be run on any platform that has a Java 5.0 VM. However we are only committed to supporting its operation on Linux and so this chapter only covers setup on that platform. Because of this, what follows assumes basic Linux administration skills. Other chapters in the user manual are platform agnostic.

This chapter also only covers installing and running the prepackaged binary distributions of Heritrix. For information about downloading and compiling the source see the [Developer's Manual](#) [http://crawler.archive.org/articles/developer_manual/index.html].

2.1. Obtaining and installing Heritrix

The packaged binary can be downloaded from the project's sourceforge home page [<http://sourceforge.net/projects/archive-crawler>]. Each release comes in four flavors, packaged as .tar.gz or .zip and including source or not.

For installation on Linux get the file `heritrix-???.tar.gz` (where ??? is the most recent version number).

The packaged binary comes largely ready to run. Once downloaded it can be untarred into the desired directory.

```
% tar xzf heritrix-???.tar.gz
```

Once you have downloaded and untarred the correct file you can move on to the next step.

2.1.1. System requirements

2.1.1.1. Java Runtime Environment

The Heritrix crawler is implemented purely in Java. This means that the only true requirement for running it is that you have a JRE installed (Building will require a JDK).

The Heritrix crawler, since release 1.10.0, makes use of Java 5.0 features so your JRE must be at least of a 5.0 (1.5.0+) pedigree.

We currently include all of the free/open source third-party libraries necessary to run Heritrix in the distribution package. See dependencies [<http://crawler.archive.org/dependencies.html>] for the complete list (Licenses for all of the listed libraries are listed in the dependencies section of the raw project.xml at the root of the `src` download or on Sourceforge).

2.1.1.1.1. Installing Java

If you do not have Java installed you can download Java from:

- **Sun** -- java.sun.com [<http://java.sun.com/>]
- **IBM** -- www.ibm.com/java [<http://www.ibm.com/java>]

2.1.1.2. Hardware

A default java heap is 256MB RAM, which is usually suitable for crawls that range over hundreds of hosts. Assign more -- see Section 2.2.1.3, "JAVA_OPTS" for how -- of your available RAM to the heap if you are crawling thousands of hosts or experience Java out-of-memory problems.

2.1.1.3. Linux

The Heritrix crawler has been built and tested primarily on Linux. It has seen some informal use on Macintosh, Windows 2000 and Windows XP, but is not tested, packaged, nor supported on platforms other than Linux at this time.

2.2. Running Heritrix

To run Heritrix, first do the following:

```
% export HERITRIX_HOME=/PATH/TO/BUILT/HERITRIX
```

...where \$HERITRIX_HOME is the location of your untarred heritrix.???.tar.gz.

Next run:

```
% cd $HERITRIX_HOME
% chmod u+x $HERITRIX_HOME/bin/heritrix
% $HERITRIX_HOME/bin/heritrix --help
```

This should give you usage output like the following:

```
Usage: heritrix --help
Usage: heritrix --nowui ORDER.XML
Usage: heritrix [--port=#] [--run] [--bind=IP,IP...] --admin=LOGIN:PASSWORD \
[ORDER.XML]
Usage: heritrix [--port=#] --selftest[=TESTNAME]
Version: @VERSION@
Options:
  -b,--bind          Comma-separated list of IP addresses or hostnames for web
                    server to listen on. Set to / to listen on all available
                    network interfaces. Default is 127.0.0.1.
  -a,--admin          Login and password for web user interface administration.
                    Required (unless passed via the 'heritrix.cmdline.admin'
                    system property). Pass value of the form 'LOGIN:PASSWORD'.
  -h,--help          Prints this message and exits.
  -n,--nowui          Put heritrix into run mode and begin crawl using ORDER.XML. Do
                    not put up web user interface.
  -p,--port          Port to run web user interface on. Default: 8080.
  -r,--run            Put heritrix into run mode. If ORDER.XML begin crawl.
  -s,--selftest       Run the integrated selftests. Pass test name to test it only
                    (Case sensitive: E.g. pass 'Charset' to run charset selftest).

Arguments:
  ORDER.XML          Crawl order to run.
```

Launch the crawler with the UI enabled by doing the following:

```
% $HERITRIX_HOME/bin/heritrix --admin=LOGIN:PASSWORD
```

This will start up Heritrix printing out a startup message that looks like the following:

```
[b116-dyn-60 619] heritrix-0.4.0 > ./bin/heritrix
Tue Feb 10 17:03:01 PST 2004 Starting heritrix...
Tue Feb 10 17:03:05 PST 2004 Heritrix 0.4.0 is running.
Web UI is at: http://b116-dyn-60.archive.org:8080/admin
Login and password: admin/letmein
```

Note

By default, as of version 1.10.x, Heritrix binds to localhost only. This means that you need to be running Heritrix on the same machine as your browser to access the Heritrix UI. Read about the `--bind` argument above if you need to access the Heritrix UI over a network.

See Section 3, “Web based user interface” and Section 4, “A quick guide to running your first crawl job” to get your first crawl up and running.

2.2.1. Environment variables

Below are environment variables that effect Heritrix operation.

2.2.1.1. HERITRIX_HOME

Set this environment variable to point at the Heritrix home directory. For example, if you've unpacked Heritrix in your home directory and Heritrix is sitting in the `heritrix-1.0.0` directory, you'd set `HERITRIX_HOME` as follows. Assuming your shell is bash:

```
% export HERITRIX_HOME=~/.heritrix-1.0.0
```

If you don't set this environment variable, the Heritrix start script makes a guess at the home for Heritrix. It doesn't always guess correctly.

2.2.1.2. JAVA_HOME

This environment variable may already exist. It should point to the Java installation on the machine. An example of how this might be set (assuming your shell is bash):

```
% export JAVA_HOME=/usr/local/java/jre/
```

2.2.1.3. JAVA_OPTS

Pass options to the Heritrix JVM by populating the `JAVA_OPTS` environment variable with values. For example, if you want to have Heritrix run with a larger heap, say 512 megs, you could do either of the following (assuming your shell is bash):

```
% export JAVA_OPTS="-Xmx512M"
% $HERITRIX_HOME/bin/heritrix
```

Or, you could do it all on the one line as follows:

```
% JAVA_OPTS="-Xmx512m" $HERITRIX_HOME/bin/heritrix
```

2.2.2. System properties

Below we document the system properties passed on the command-line that can influence Heritrix's behavior. If you are using the `/bin/heritrix` script to launch Heritrix you may have to edit it to change/set these properties or else pass them as part of `JAVA_OPTS`.

2.2.2.1. heritrix.properties

Set this property to point at an alternate `heritrix.properties` file -- e.g.: `-Dheritrix.properties=/tmp/alternate.properties` -- when you want heritrix to use a properties file other than that found at `conf/heritrix.properties`.

2.2.2.2. heritrix.context

Provide an alternate context for the Heritrix admin UI. Usually the admin webapp is mounted on root: i.e. `/`.

2.2.2.3. heritrix.development

Set this property when you want to run the crawler from eclipse. This property takes no arguments. When this property is set, the `conf` and `webapps` directories will be found in their development locations and startup messages will show on the text console (standard out).

2.2.2.4. heritrix.home

Where heritrix is homed usually passed by the heritrix launch script.

2.2.2.5. heritrix.out

Where stdout/stderr are sent, usually `heritrix_out.log` and passed by the heritrix launch script.

2.2.2.6. heritrix.version

Version of heritrix set by the heritrix build into `heritrix.properties`.

2.2.2.7. heritrix.jobdir

Where to drop heritrix jobs. Usually empty. Default location is `${HERITRIX_HOME} / jobs`.

2.2.2.8. heritrix.conf

Specify an alternate configuration directory other than the default `$HERITRIX_HOME/conf`.

2.2.2.9. heritrix.cmdline

This set of system properties are rarely used. They are for use when Heritrix has NOT been started from the command-line -- e.g. its been embedded in another application -- and the startup configuration that is set usually by command-line options, instead needs to be done via system properties alone.

2.2.2.9.1. heritrix.cmdline.admin

Value is a colon-delimited String user name and password for admin GUI

2.2.2.9.2. heritrix.cmdline.nowui

If set to true, will prevent embedded web server crawler control interface from starting up.

2.2.2.9.3. heritrix.cmdline.order

If set to a string file path, will use the specified crawl order XML file.

2.2.2.9.4. heritrix.cmdline.port

Value is the port to run the GUI on.

2.2.2.9.5. heritrix.cmdline.run

If true, crawler is set into run mode on startup.

2.2.2.10. javax.net.ssl.trustStore

Heritrix has its own trust store at `conf/heritrix.cacerts` that it uses if the `FetcherHTTP` is configured to use a trust level of other than *open* (open is the default setting). In the unusual case where you'd like to have Heritrix use an alternate truststore, point at the alternate by supplying the `JSSE javax.net.ssl.trustStore` property on the command line: e.g.

2.2.2.11. java.util.logging.config.file

The Heritrix `conf` directory includes a file named `heritrix.properties`. A section of this file specifies the default Heritrix logging configuration. To override these settings, point `java.util.logging.config.file` at a properties file with an alternate logging configuration. Below we reproduce the default `heritrix.properties` for reference:

```
# Basic logging setup; to console, all levels
handlers= java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level= ALL

# Default global logging level: only warnings or higher
.level= WARNING

# currently necessary (?) for standard logs to work
crawl.level= INFO
runtime-errors.level= INFO
uri-errors.level= INFO
progress-statistics.level= INFO
recover.level= INFO

# HttpClient is too chatty... only want to hear about severe problems
org.apache.commons.httpclient.level= SEVERE
```

Here's an example of how you might specify an override:

```
% JAVA_OPTS="-Djava.util.logging.config.file=heritrix.properties" \
./bin/heritrix --no-wui order.xml
```

Alternatively you could edit the default file.

2.2.2.12. java.io.tmpdir

Specify an alternate tmp directory. Default is `/tmp`.

2.2.2.13. com.sun.management.jmxremote.port

What port to start up JMX Agent on. Default is 8849. See also the environment variable `JMX_PORT`.

2.3. Security Considerations

The crawler is a large and active network application which presents security implications, both local to the machine where it operates, and remotely for machines it contacts.

2.3.1. Local to the Crawling Machine

It is important to recognize that the web UI (discussed in Section 3, “Web based user interface”) and JMX agent (discussed in Section 9.5, “Remote Monitoring and Control”) allow remote control of the crawler process in ways that might potentially disrupt a crawl, change the crawler's behavior, read or write locally-accessible files, and perform or trigger other actions in the Java VM or local machine.

The administrative login and password are currently only a very mild protection against unauthorized access, unless you take additional steps to prevent access to the crawler machine. We strongly recommend some combination of the following practices:

First, use network configuration tools, like a firewall, to only allow trusted remote hosts to contact the web UI and, if applicable, JMX agent ports. (The default web UI port is 8080; JMX is 8849.)

Second, use a strong and unique username/password combination to secure the web UI and JMX agent. However, keep in mind that the default administrative web server uses plain HTTP for access, so these values are susceptible to eavesdropping in transit if network links between your browser and the crawler are compromised. (An upcoming update will change the default to HTTPS.) Also, setting the username/password on the command-line may result in their values being visible to other users of the crawling machine, and they are additionally printed to the console and `heritrix_out.log` for operator reference.

Third, run the crawler as a user with the minimum privileges necessary for its operation, so that in the event of unauthorized access to the web UI or JMX agent, the potential damage is limited.

Successful unauthorized access to the web UI or JMX agent could trivially end or corrupt a crawl, or change the crawler's behavior to be a nuisance to other network hosts. By adjusting configuration paths, unauthorized access could potentially delete, corrupt, or replace files accessible to the crawler process, and thus cause more extensive problems on the crawler machine.

Another potential risk is that some worst-case or maliciously-crafted crawled content might, in combination with crawler bugs, disrupt the crawl or other files or operations of the local system. For example, in the past, even without malicious intent, some rich-media content has caused runaway memory use in 3rd-party libraries used by the crawler, resulting in a memory-exhaustion condition that can stop or corrupt a crawl in progress. Similarly, atypical input patterns have at times caused runaway CPU use by crawler link-extraction regular expressions, severely slowing crawls. Crawl operators should monitor their crawls closely and stay informed via the project discussion list and bug database for any newly discovered similar bugs.

3. Web based user interface

After Heritrix has been launched from the command line, the web based user interface (WUI) becomes accessible.

The URI to access the WUI is printed on the text console from which the program was launched (typically `http://<host>:8080/admin/`).

The WUI is password protected. There is no default login for access; one must be specified using either the `'-a/--admin'` command-line option at startup or by setting the `'heritrix.cmdline.admin'` system property. The currently valid username and password combination will be printed out to the console, along with the access URL for the WUI, at startup.

The WUI can be accessed via any web browser. While we've endeavoured to make certain that it func-

tions in all recent browsers, Mozilla 5 or newer is recommended. IE 6 or newer should also work without problems.

The initial login page takes the username/password combination discussed above. Logins will time out after a period of non-use.

Caution

By default, communication with the WUI is not done over an encrypted HTTPS connection! Passwords will be submitted over the network in plain text, so you should take additional steps to protect your crawler administrative interface from unauthorized access, as described in the Section 2.3, “Security Considerations” section.

4. A quick guide to running your first crawl job

Once you've installed Heritrix and logged into the WUI (see above) you are presented with the web Console page. Near the top there is a row of tabs.

Step 1. Create a job

To create a new job choose the Jobs tab, this will take you to the Jobs page. Once there you are presented with three options for creating a new job. Select 'With defaults'. This will create a new job based on the default profile (see Section 5.2, “Profile”).

On the screen that comes next you will be asked to supply a name, description and a seed list for the new job.

For a name supply a short text with no special characters or spaces (except dash and underscore). You can skip the description if you like. In the seeds list type in the URL of the sites you are interested in harvesting. One URL to a line.

Creating a job is covered in greater detail in Section 5, “Creating jobs and profiles”.

Step 2. Configure the job

Once you've entered this information in you are ready to go to the configuration pages. Click the *Modules* button in the row of buttons at the bottom of the page.

This will take you to the modules configuration page (more details in Section 6.1, “Modules (Scope, Frontier, and Processors)”). For now we are only interested in the option second from the top named **Select crawl scope**. It allows you to specify the limits of the crawl. By default it is limited to the domains that your seeds span. This may be suitable for your purposes. If not you can choose a broad scope (not limited to the domains of its seeds) or the more restrictive host scope that limits the crawl to the hosts that its seeds span. For more on scopes refer to Section 6.1.1, “Crawl Scope”.

To change scopes, select the new one from the combobox and click the *Change* button.

Next turn your attention to the second row of tabs at the top of the page, below the usual tabs. You are currently on the far left tab. Now select the tab called *Settings* near the middle of the row.

This takes you to the Settings page. It allows you to configure various details of the crawl. Exhaustive coverage of this page can be found in Section 6.3, “Settings”. For now we are only interested in the two settings under **http-headers**. These are the `user-agent` and `from` field of the HTTP headers in the crawlers requests. You must set them to valid values before a crawl can be run. The current values up-per-case what needs replacing. If you have trouble with that please refer to Section 6.3.1.3, “HTTP headers” for what's regarded as valid values.

Once you've set the **http-headers** settings to proper values (and made any other desired changes), you can click the *Submit job* tab at the far right of the second row of tabs. The crawl job is now configured and ready to run.

Configuring a job is covered in greater detail in Section 6, “Configuring jobs and profiles”.

Step 3. Running the job

Submitted new jobs are placed in a queue of pending jobs. The crawler does not start processing jobs from this queue until the crawler is started. While the crawler is stopped, jobs are simply held.

To start the crawler, click on the Console tab. Once on the Console page, you will find the option *Start* at the top of the **Crawler Status** box, just to the right of the indicator of current status. Clicking this option will put the crawling into *Crawling Jobs* mode, where it will begin crawling any next pending job, such as the job you just created and configured.

The Console will update to display progress information about the on-going crawl. Click the *Refresh* option (or the top-left Heritrix logo) to update this information.

For more information about running a job see Section 7, “Running a job”.

Detailed information about evaluating the progress of a job can be found in Section 8, “Analysis of jobs”.

5. Creating jobs and profiles

In order to run a crawl a configuration must be created that defines it. In Heritrix such a configuration is called a **crawl job**.

5.1. Crawl job

A crawl job encompasses the configurations needed to run a single crawl. It also contains some additional elements such as file locations, status etc.

Once logged onto the WUI new jobs can be created by going to the *Jobs* tab. Once the Jobs page loads users can create jobs by choosing of the following three options:

- 1. Based on existing job**

This option allows the user to create a job by basing it on any existing job, regardless of whether it has been crawled or not. Can be useful for repeating crawls or recovering a crawl that had problems. (See Section 9.3, “Recovery of Frontier State and recover.gz”)

- 2. Based on a profile**

This option allows the user to create a job by basing it on any existing profiles.

- 3. With defaults**

This option creates a new crawl job based on the default profile.

Options 1 and 2 will display a list of available options. Initially there are two profiles and no existing jobs.

All crawl jobs are created by basing them on profiles (see Section 5.2, “Profile”) or existing jobs.

Once the proper profile/job has been chosen to base the new job on, a simple page will appear asking for the new job's:

1. **Name**

The name must only contain letters, numbers, dash (-) and underscore (_). No other characters are allowed. This name will be used to identify the crawl in the WUI but it need not be unique. The name can not be changed later

2. **Description**

A short description of the job. This is a freetext input box and can be edited later.

3. **Seeds**

The seed URIs to use for the job. This list can be edited later along with the general configurations.

Below these input fields there are several buttons. The last one *Submit job* will immediately submit the job and (assuming it is properly configured) it will be ready to run (see Section 7, "Running a job"). The other buttons will take the user to the relevant configuration pages (those are covered in detail in Section 6, "Configuring jobs and profiles"). Once all desired changes have been made to the configuration, click the '*Submit job*' tab (usually displayed top and bottom right) to submit it to the list of waiting jobs.

Note

Changes made afterwards to the original jobs or profiles that a new job is based on will **not** in any way affect the newly created job.

Note

Jobs based on the default profile provided with Heritrix are not ready to run *as is*. Their HTTP header information must be set to valid values. See Section 6.3.1.3, "HTTP headers" for details.

5.2. Profile

A profile is a template for a crawl job. It contains all the configurations that a crawl job would, but is not considered to be 'crawlable'. That is Heritrix will not allow you to directly crawl a profile, only jobs based on profiles. The reason for this is that while profiles may in fact be complete, they may also not be.

A common example is leaving the HTTP headers (*user-agent*, *from*) in an illegal state in a profile to force the user to input valid data. This applies to the default (*default*) profile that comes with Heritrix. Other examples would be leaving the seeds list empty, not specifying some processors (such as the writer/indexer) etc.

In general there is less error checking of profiles.

To manage profiles, go to the *Profiles* tab in the WUI. That page will display a list of existing profiles. To create a new profile select the option of creating a "New profile based on it" from the existing profile to use as a template. Much like jobs, profiles can only be created based on other profiles. It is not possible to create profiles based on existing jobs.

The process from there on mirrors the creation of jobs. A page will ask for the new profiles name, description and seeds list. Unlike job names, profile names *must be unique* from other profile names - jobs and a profile can share the same name - otherwise the same rules apply.

The user then proceeds to the configuration pages (see Section 6, “Configuring jobs and profiles”) to modify the behavior of the new profile from that of the parent profile.

Note

Even though profiles are based on other profiles, changes made to the original profiles afterwards will **not** affect the new ones.

6. Configuring jobs and profiles

Creating crawl jobs (Section 5.1, “Crawl job”) and profiles (Section 5.2, “Profile”) is just the first step. Configuring them is a more complicated process.

The following section applies equally to configuring crawl jobs and profiles. It does not matter if new ones are being created or existing ones are being edited. The interface is almost entirely the same, only the *Submit job / Finished* button will vary.

Note

Editing options for jobs being crawled are somewhat limited. See Section 7.4, “Editing a running job” for more.

Each page in the configuration section of the WUI will have a secondary row of tabs below the general ones. This secondary row is often replicated at the bottom of longer pages.

This row offers access to different parts of the configuration. While configuring the global level (more on global vs. overrides and refinements in Section 6.4, “Overrides” and Section 6.5, “Refinements”) the following options are available (left to right):

- Modules (Section 6.1, “Modules (Scope, Frontier, and Processors)”)

Add/remove/set configurable modules, such as the crawl Scope (Section 6.1.1, “Crawl Scope”), Frontier (Section 6.1.2, “Frontier”), or Processors (Section 6.1.3, “Processing Chains”).

- Submodules (Section 6.2, “Submodules”)

Here you can:

- Add/remove/reorder URL canonicalization rules (Section 6.2.1, “URL Canonicalization Rules”)
- Add/remove/reorder filters (Section 6.2.2, “Filters”)
- Add/remove login credentials (Section 6.2.3, “Credentials”)
- Settings (Section 6.3, “Settings”)

Configure settings on Heritrix modules

- Overrides (Section 6.4, “Overrides”)

Override settings on Heritrix modules based on domain

- Refinements (Section 6.5, “Refinements”)

Refine settings on Heritrix modules based on arbitrary criteria

- Submit job / Finished

Clicking this tab will take the user back to the Jobs or Profiles page, saving any changes.

The *Settings* tab is probably the most frequently used page as it allows the user to fine tune the settings of any Heritrix module used in a job or profile.

It is safe to navigate between these, it will not cause new jobs to be submitted to the queue of pending jobs. That only happens once the *Submit job* tab is clicked. Navigating out of the configuration pages using the top level tabs will cause new jobs to be lost. Any changes made are saved when navigating within the configuration pages. There is no undo function, once made changes can not be undone.

6.1. Modules (Scope, Frontier, and Processors)

Heritrix has several types of pluggable modules. These modules, while having a fixed interface usually have a number of provided implementations. They can also be third party plugins. The "Modules" tab allows the user to set several types of these pluggable modules.

Once modules have been added to the configuration they can be configured in greater detail on the Settings tab (Section 6.3, "Settings"). If a module can contain within it multiple other modules, these can be configured on the Submodules tab.

Note

Modules are referred to by their Java class names (org.archive.crawler.frontier.BdbFrontier).

This is done because these are the only names we can be assured of being unique.

See Developer's Manual [http://crawler.archive.org/articles/developer_manual/index.html] for information about creating and adding custom modules to Heritrix.

6.1.1. Crawl Scope

A crawl scope is an object that decides for each discovered URI if it is within the scope of the current crawl.

Several scopes are provided with Heritrix:

- **BroadScope**

This scope allows for limiting the depth of a crawl (how many links away Heritrix should crawl) but does not impose any limits on the hosts, domains, or URI paths crawled.

- **SurtPrefixScope**

A highly flexible and fairly efficient scope which can crawl within defined domains, individual hosts, or path-defined areas of hosts, or any mixture of those, depending on the configuration.

It considers whether any URI is inside the primary focus of the scope by converting the URI to its SURT form, and then seeing if that SURT form begins with any of a number of SURT prefixes. (See the glossary definitions for detailed information about the SURT form of a URI and SURT prefix comparisons.)

The operator may establish the set of SURT prefixes used either by letting the SURT prefixes be implied from the supplied seed URIs, specifying an external file with a listing of SURT prefixes, or both.

This scope also enables a special syntax within the seeds list for adding SURT prefixes separate

from seeds. Any line in the seeds list beginning with a '+' will be considered a SURT prefix specification, rather than a seed. Any URL you put after the '+' will only be used to deduce a SURT prefix -- it will not be independently scheduled. You can also put your own literal SURT prefix after the '+'.

For example, each of the following SURT prefix directives in the seeds box are equivalent:

```
+http://(org,example,      # literal SURT prefix
+http://example.org       # regular URL implying same SURT prefix
+example.org              # URL fragment with implied 'http' scheme
```

When you use this scope, it adds 3 hard-to-find-in-the-UI attributes -- `surts-source-file`, `seeds-as-surt-prefixes`, and `surts-dump-file` -- to the end of the scope section, just after `transitiveFilter` but before `http-headers`.

Use the `surts-source-file` setting to supply an external file from which to infer SURT prefixes, if desired. Any URLs in this file will be converted to the implied SURT prefix, and any line beginning with a '+' will be interpreted as a literal, precise SURT prefix. Use the `seeds-as-surt-prefixes` setting to establish whether SURT prefixes should be deduced from the seeds, in accordance with the rules given at the SURT prefix glossary entry. (The default is 'true', to deduce SURT prefixes from seeds.)

To see what SURT prefixes were actually used -- perhaps merged from seed-deduced and externally-supplied -- you can specify a file path in the `surts-dump-file` setting. The sorted list of actual SURT prefixes used will be written to that file for reference. (Note that redundant entries will be removed from this dump. If you have SURT prefixes `<http://(org,>` and `<http://(org,archive,>`, only the former will actually be used, because all SURT form URIs prefixed by the latter are also prefixed by the former.)

See also the crawler wiki on SurtScope [<http://crawler.archive.org/cgi-bin/wiki.pl?SurtScope>].

- **FilterScope**

A highly configurable scope. By adding different filters in different combinations this scope can be configured to provide a wide variety of behaviour.

After selecting this filter, you must then go to the *Filters* tab and add the filters you want to run as part of your scope. Add the filters at the *focusFilter* label and give them a meaningful name. The `URIRegexFilter` probably makes most sense in this context (The `ContentTypeRegexFilter` won't work at scope time because we don't know the content-type till after we've fetched the document).

After adding the filter(s), return to the *Settings* tab and fill in any configuration required of the filters. For example, say you added the `URIRegexFilter`, and you wanted only 'www.archive.org' hosts to be in focus, fill in a regex like the following: `^(?:http|dns)www.archive.org/\.*` (Be careful you don't rule out prerequisites such as `dns` or `robots.txt` when specifying your scope filter).

The following scopes are available, but the same effects can be achieved more efficiently, and in combination, with `SurtPrefixScope`. When `SurtPrefixScope` can be more easily understood and configured, these scopes may be removed entirely.

- **DomainScope**

This scope limits discovered URIs to the set of domains defined by the provided seeds. That is any URI discovered belonging to a domain from which one of the seed came is within scope. Like al-

ways it is possible to apply depth restrictions.

Using the seed 'archive.org', a domain scope will fetch 'audio.archive.org', 'movies.archive.org', etc. It will fetch all discovered URIs from 'archive.org' and from any subdomain of 'archive.org'.

- **HostScope**

This scope limits discovered URIs to the set of hosts defined by the provided seeds.

If the seed is 'www.archive.org', then we'll only fetch items discovered on this host. The crawler will not go to 'audio.archive.org' or 'movies.archive.org'.

- **PathScope**

This scope goes yet further and limits the discovered URIs to a section of paths on hosts defined by the seeds. Of course any host that has a seed pointing at its root (i.e. `www.sample.com/index.html`) will be included in full where as a host whose only seed is `www.sample2.com/path/index.html` will be limited to URIs under `/path/`.

Note

Internally Heritrix defines everything up to the right most slash as the path when doing path scope so for example, the URLs `http://members.aol.com/bigbird` and `http://members.aol.com/~bigbird` will treat as in scope any URL that begins `members.aol.com`. If your intent is to only include all below the path `bigbird`, add a slash on the end, using a form such as `http://members.aol.com/bigbird/` or `http://members.aol.com/bigbird/index.html` instead.

Scopes usually allow for some flexibility in defining depth and possible transitive includes (that is getting items that would usually be out of scope because of special circumstance such as their being embedded in the display of an included resource). Most notably, every scope can have additional filters applied in two different contexts (some scopes may only have one these contexts).

1. **Focus**

URIs matching these filters will be considered to be within scope

2. **Exclude**

URIs matching these filters will be considered to be out of scope.

Custom made Scopes may have different sets of filters. Also some scopes have filters hardcoded into them. This allows you to edit their settings but not remove or replace them. For example most of the provided scopes have a `Transclusion` filter hardcoded into them that handles transitive items (URIs that normally shouldn't be included but because of special circumstance they will be included).

For more about Filters see Section 6.2.2, "Filters".

6.1.1.1. Problems with the current Scopes

Our original Scope classes -- `PathScope`, `HostScope`, `DomainScope`, `BroadScope` -- all could be thought of as fitting a specific pattern: A URI is included if and only if:

```
protected final boolean innerAccepts(Object o) {
```

```
    return ((isSeed(o) || focusAccepts(o)) || additionalFocusAccepts(o) ||  
            transitiveAccepts(o)) && !excludeAccepts(o);  
}
```

More generally, the *focus* filter was meant to rule things in by prima facia/regexp-pattern analysis; the *transitive* filter rule extra items in by dynamic path analysis (for example, off site embedded images); and the *exclusion* filter rule things out by any number of chained exclusion rules. So in a typical crawl, the *focus* filter drew from one of these categories:

- **broad** : accept all
- **domain**: accept if on same 'domain' (for some definition) as seeds
- **host**: accept if on exact host as seeds
- **path**: accept if on same host and a shared path-prefix as seeds

The *transitive* filter configuration was based on the various link-hops and embed-hop thresholds set by the operator.

The *exclusion* filter was in fact a compound chain of filters, OR'ed together, such that any one of them could knock a URI out of consideration. However, a number of aspects of this arrangement have caused problems:

1. To truly understand what happens to an URI, you must understand the above nested boolean-construct.
2. Adding mixed focuses -- such as all of this one host, all of this other domain, and then just these paths on this other host -- is not supported by these classes, nor easy to mix-in to the *focus* filter.
3. Constructing and configuring the multiple filters required many setup steps across several WUI pages.
4. The reverse sense of the *exclusion* filters -- if URIs are accepted by the filter, they are excluded from the crawl -- proved confusing, exacerbated by the fact that 'filter' itself can commonly mean either 'filter in' or 'filter out'.

As a result of these problems, the SurtPrefixScope was added, and further major changes are planned. The first steps are described in the next section, Section 6.1.1.2, “DecidingScope”. These changes will also affect whether and how filters (see Section 6.2.2, “Filters”) are used.

6.1.1.2. DecidingScope

To address the shortcomings above, and generally make alternate scope choices more understandable and flexible, a new mechanism for scoping and filtering has been introduced in Heritrix 1.4. This new approach is somewhat like (and inspired by) HTTrack's 'scan rules'/filters, Alexa's mask/ignore/void syntax for adjusting recurring crawls, or the Nutch 'regex-urlfilter' facility, but may be a bit more general than any of those.

This new approach is available as a DecidingScope, which is modelled as a series of DecideRules. Each DecideRule, when presented with an Object (most often a URI of some form), may respond with one of three decisions:

- ACCEPT: the object is ruled in
- REJECT: the object is ruled out
- PASS: the rule has no opinion; retain whatever previous decision was made

To define a Scope, the operator configures an ordered series of DecideRules. A URI under consideration begins with no assumed status. Each rule is applied in turn to the candidate URI. If the rule decides ACCEPT or REJECT, the URI's status is set accordingly. After all rules have been applied, if the URI's status is ACCEPT it is "in scope" and scheduled for crawling; if its status is REJECT it is discarded.

There are no branches, but much of what nested conditionals can achieve is possible, in a form that should be easier to follow than arbitrary expressions.

The current list of available DecideRules includes:

```
AcceptDecideRule -- ACCEPTs all (establishing an early default)
RejectDecideRule -- REJECTs all (establishing an early default)
TooManyHopsDecideRule(max-hops=N) -- REJECTs all with hopsPath.length()>N, PAS
PrerequisiteAcceptDecideRule -- ACCEPTs any with 'P' as last hop, PASSes other
MatchesRegExpDecideRule(regexp=pattern) -- ACCEPTs (or REJECTs) all matching a
NotMatchesRegExpDecideRule(regexp=pattern) -- ACCEPTs (or REJECTs) all *not* m
PathologicalPathDecideRule(max-reps=N) -- REJECTs all mathing problem patterns
TooManyPathSegmentsDecideRule(max-segs=N) -- REJECTs all with too many path-se
TransclusionDecideRule(extra-hops=N) -- ACCEPTs anything with up to N non-navl
SurtPrefixedDecideRule(use-seeds=bool;use-file=path) -- ACCEPTs (or REJECTs) a
NotSurtPrefixedDecideRule(use-seeds=bool;use-file=path) -- ACCEPTs (or REJECTs)
OnHostsDecideRule(use-seeds=bool;use-file=path) -- ACCEPTs (or REJECTs) anythi
NotOnHostsDecideRule(use-seeds=bool;use-file=path) -- ACCEPTs (or REJECTs) any
OnDomainsDecideRule(use-seeds=bool;use-file=path) -- ACCEPTs (or REJECTs) anyt
NotOnDomainsSetDecideRule(use-seeds=bool;use-file=path) -- ACCEPTs (or REJECTs)
MatchesFilePatternDecideRule -- ACCEPTs (or REJECTs) URIs matching a chosen pr
NotMatchesFilePatternDecideRule -- ACCEPTs (or REJECTs) URIs *not* matching a
```

...covering just about everything our previous focus- and filter- based classes did. By ordering exclude and include actions, combinations that were awkward before -- or even impossible without writing custom code -- becomes straightforward.

For example, a previous request that was hard for us to accomodate was the idea: "crawl exactly these X hosts, and get offsite images if only on the same domains." That is, don't wander off the exact hosts to follow navigational links -- only to get offsite resources that share the same domain.

Our relevant function-of-seeds tests -- host-based and domain-based -- were exclusive of each other (at the 'focus' level) and difficult to mix-in with path-based criteria (at the 'transitive' level).

As a series of DecideRules, the above request can be easily achieved as:

```
RejectDecideRule
OnHostsDecideRule(use-seeds=true)
TranscludedDecideRule(extra-hops=2)
NotOnDomainsDecideRule(REJECT,use-seeds=true);
```

A good default set of DecideRules for many purposes would be...


```
RejectDecideRule           // reject by default
SurtPrefixedDecideRule     // accept within SURT prefixes established
TooManyHopsDecideRule      // but reject if too many hops from seeds
TransclusionDecideRule      // notwithstanding above, accept if within
PathologicalPathDecideRule // but reject if pathological repetitions
TooManyPathSegmentsDecideRule // ...or if too many path-segments
PrerequisiteAcceptDecideRule // but always accept a prerequisite of other
```

In Heritrix 1.10.0, the default profile was changed to use the above set of DecideRules (Previous to this, the operator had to choose the 'deciding-default' profile, since removed).

The naming, behavior, and user-interface for DecideRule-based scoping is subject to significant change based on feedback and experience in future releases.

Enable `FINE` logging on the class `org.archive.crawler.deciderules.DecideRuleSequence` to watch each deciderules finding on each processed URI.

6.1.2. Frontier

The Frontier is a pluggable module that maintains the internal state of the crawl. What URIs have been discovered, crawled etc. As such its selection greatly effects, for instance, the order in which discovered URIs are crawled.

There is only one Frontier per crawl job.

Multiple Frontiers are provided with Heritrix, each of a particular character.

6.1.2.1. BdbFrontier

The default Frontier in Heritrix as of 1.4.0 and later is the BdbFrontier(Previously, the default was the Section 6.1.2.2, “HostQueuesFrontier”). The BdbFrontier visits URIs and sites discovered in a generally breadth-first manner, it offers configuration options controlling how it throttles its activity against particular hosts, and whether it has a bias towards finishing hosts in progress ('site-first' crawling) or cycling among all hosts with pending URIs.

Discovered URIs are only crawled once, except that robots.txt and DNS information can be configured so that it is refreshed at specified intervals for each host.

The main difference between the BdbFrontier and its precursor, Section 6.1.2.2, “HostQueuesFrontier”, is that BdbFrontier uses BerkeleyDB Java Edition to shift more running Frontier state to disk.

6.1.2.2. HostQueuesFrontier

The forerunner of the Section 6.1.2.1, “BdbFrontier”. Now deprecated mostly because its custom disk-based data structures could not move as much Frontier state out of main memory as the BerkeleyDB Java Edition approach. Has same general characteristics as the Section 6.1.2.1, “BdbFrontier”.

6.1.2.3. DomainSensitiveFrontier

A subclass of the Section 6.1.2.2, “HostQueuesFrontier” written by Oskar Grenholm. The DSF allows specifying an upper-bound on the number of documents downloaded per-site. It does this by exploiting Section 6.4, “Overrides” adding a filter to block further fetching once the crawler has attained per-site limits.

6.1.2.4. AdaptiveRevisitingFrontier

The AdaptiveRevisitingFrontier -- a.k.a *AR Frontier* -- will repeatedly visit all encountered URIs. Wait time between visits is configurable and varies based on wait intervals specified by a WaitEvaluator processor. It was written by Kristinn Sigurdsson.

Note

This Frontier is still experimental, in active development and has not been tested extensively.

In addition to the WaitEvaluator (or similar processor) a crawl using this Frontier will also need to use the ChangeEvaluator processor: i.e. this Frontier requires that ChangeEvaluator and WaitEvaluator or equivalents are present in the processing chain.

ChangeEvaluator should be at the very top of the extractor chain.

WaitEvaluator -- or an equivalent -- needs to be in the post processing chain.

The ChangeEvaluator has no configurable settings. The WaitEvaluator however has numerous settings to adjust the revisit policy.

- Initial wait. A waiting period before revisiting the first time.
- Increase and decrease factors on unchanged and changed documents respectively. Basically if a document has not changed between visits, its wait time will be multiplied by the "unchanged-factor" and if it has changed, the wait time will be divided by the "changed-factor". Both values accept real numbers, not just integers.
- Finally, there is a 'default-wait-interval' for URIs where it is not possible to judge changes in content. Currently this applies only to DNS lookups.

If you want to specify different wait times and factors for URIs based on their mime types, this is possible. You have to create a Refinement (Section 6.5, "Refinements") and use the ContentType criteria. Simply use a regular expression that matches the desired mime type as its parameter and then override the applicable parameters in the refinement.

By setting the 'state' directory to the same location that another AR crawl used, it should resume that crawl (minus some stats).

6.1.3. Processing Chains

When a URI is crawled it is in fact passed through a series of processors. This series is split for convenience between five chains and the user can add, remove and reorder the processors on each of these chains.

Each URI taken off the Frontier queue runs through the Processing Chains listed in the diagram shown below. URIs are always processed in the order shown in the diagram unless a particular processor throws a fatal error or decides to stop the processing of the current URI for some reason. In this circumstance, processing skips to the end, to the Post-processing chain, for cleanup.

Each processing chain is made up of zero or more individual processors. For example, the extractor processing chain might comprise the ExtractorHTML, an ExtractorJS, and the ExtractorUniversal processors. Within a processing step, the order in which processors are run is the order in which processors are listed on the modules page.

Generally, particular processors only make sense within the context of one particular processing chain.

For example, it wouldn't make sense to run the `FetchHTTP` processor in the Post-processing chain. This is however not enforced, so users must take care to construct logical processor chains.

Most of the processors are fairly self explanatory, however the first and last two merit a little bit more attention.

In the `Pre-fetch processing` chain the following two processors should be included (or replacement modules that perform similar operations):

- **Preselector**

Last check if the URI should indeed be crawled. Can for example recheck scope. Useful if scope has been changed after the crawl starts (This processor is not strictly necessary).

- **PreconditionEnforcer**

Ensures that all preconditions for crawling a URI have been met. These currently include verifying that DNS and robots.txt information has been fetched for the URI. Should always be included.

Similarly the `Post Processing` chain has the following special purpose processors:

- **CrawlStateUpdater**

Updates the per-host information that may have been affected by the fetch. This is currently robots and IP address info. Should always be included.

- **LinksScoper**

Checks all links extracted from the current download against the crawl scope. Those that are out of scope are discarded. Logging of discarded URLs can be enabled.

- **FrontierScheduler**

'Schedules' any URLs stored as `CandidateURIs` found in the current `CrawlURI` with the frontier for crawling. Also schedules prerequisites if any.

6.1.4. Statistics Tracking

Any number of statistics tracking modules can be attached to a crawl. Currently only one is provided with Heritrix. The `StatisticsTracker` module that comes with Heritrix writes the `progress-statistics.log` file and provides the WUI with the data it needs to display progress information about a crawl. It is strongly recommended that any crawl running with the WUI use this module.

6.2. Submodules

On the Submodules tab, configuration points that take variable-sized listings of components can be configured. Components can be added, ordered, and removed. Examples of such components are listings of canonicalization rules to run against each URL discovered, Section 6.2.2, "Filters" on processors, and credentials. Once submodules are added under the Submodules tab, they will show in subsequent re-drawings of the Settings tab. Values which control their operation are configured over under the Settings tab.

6.2.1. URL Canonicalization Rules

Heritrix keeps a list of already seen URLs and before fetching, does a look up into this 'already seen' or 'already included' list to see if the URL has already been crawled. Often an URL can be written in multiple ways but the page fetched is the same in each case. For example, the page that is at `http://www.archive.org/index.html` is the same page as is at `http://WWW.ARCHIVE.ORG/` though the URLs differ (In this case by case only). Before going to the 'already included' list, Heritrix makes an effort at equating the likes of `http://www.archive.org/index.html` and `http://ARCHIVE.ORG/` by running each URL through a set of canonicalization rules. Heritrix uses the result of this canonicalization process when it goes to test if an URL has already been seen.

An example of a canonicalization rule would lowercase all URLs. Another might strip the 'www' prefix from domains.

The URL Canonicalization Rules screen allows you to specify canonicalization rules and the order in which they are run. A default set lowercases, strips wwws, removes sessionids and does other types of fixup such as removal of any userinfo. The URL page works in the same manner as the Section 6.2.2, "Filters" page.

To watch the canonicalization process, enable `org.archive.crawler.url.Canonicalizer` logging in `heritrix.properties` (There should already be a commented out directive in the properties file. Search for it). Output will show in `heritrix_out.log`. Set the logging level to INFO to see just before and after the transform. Set level to FINE to see the result of each rule's transform.

Canonicalization rules can be added as an override so an added rule only works in the overridden domain.

Canonicalization rules are NOT run if the URI-to-check is the fruit of a redirect. We do this for the following reason. Lets say the www canonicalization rule is in place (the rule that equates 'archive.org' and 'www.archive.org'). If the crawler first encounters 'archive.org' but the server at archive.org wants us to come in via 'www.archive.org', it will redirect us to 'www.archive.org'. The alreadyseen database will have been marked with 'archive.org' on the original access of 'archive.org'. The www canonicalization rule runs and makes 'archive.org' of 'www.archive.org' which has already been seen. If we always ran canonicalization rules regardless, we wouldn't ever crawl 'www.archive.org'.

6.2.1.1. URL Canonicalization Use Case: Stripping Site-Particular Session IDs

Say site `x.y.z` is returning URLs with a session ID key of `cid` as in `http://x.y.z/index.html?cid=XYZ123112232112229BCDEFFA0000111`. Say the session ID value is always 32 characters. Say also, for simplicity's sake, that it always appears on the end of the URL.

6.2.1.1.1. Solution

Add a `RegexRule` override for the domain `x.y.z`. To do this, pause the crawl, add an override for `x.y.z` by clicking on the `overrides` tab in the main menu bar and filling in the domain `x.y.z`. Once in the override screen, click on the URL tab in the override menu bar -- the new bar that appears below the main bar when in override mode -- and add a `RegexRule` canonicalization rule. Name it `cidStripper`. Adjust where you'd like it to appear in the running of canonicalization rules (Towards the end should be fine). Now browse back to the override settings. The new canonicalization rule `cidStripper` should appear in the settings page list of canonicalization rules. Fill in the `RegexRule` `matching-regex` with something like the following: `^(.+)?:cid=[0-9a-zA-Z]{32}?$` (Match a tail of 'cid=SOME_32_CHAR_STR' grouping all that comes before this tail). Fill into the `format` field `${1}` (This will copy the first group from the regex if the regex matched). To see the rule in operation, set the logging level for `org.archive.crawler.url.Canonicalizer` in `heritrix.properties` (Try uncommenting the line `org.archive.crawler.url.Canonicalizer.level = INFO`). Study the output and adjust your regex accordingly.

See also msg1611 [<http://groups.yahoo.com/group/archive-crawler/message/1611>] for another's experience getting regex to work.

6.2.2. Filters

Filters are modules that take a CrawlURI and determine if it matches the criteria of the filter. If so it returns true, otherwise it returns false.

Filters are used in a couple of different contexts in Heritrix.

Their use in scopes has already been discussed in Section 6.1.1, “Crawl Scope” and the problems with using them that in Section 6.1.1.1, “Problems with the current Scopes”.

Note

A DecidingFilter was added in 1.4.0 to address problems with current filter model. DecideRules can be added into a DecidingFilter with the filter decision the result of all included DecideRule set processing. There are DecideRule equivalents for all Filter-types mentioned below. See Section 6.1.1.2, “DecidingScope” for more on the particulars of DecideRules and on the new Deciding model in general.

Aside from scopes, filters are also used in processors. Filters applied to processors always filter URIs *out*. That is to say that any URI matching a filter on a processor will effectively skip over that processor.

This can be useful to disable (for instance) link extraction on documents coming from a specific section of a given website.

6.2.2.1. Adding, removing and reordering filters

The Submodules page of the configuration section of the WUI lists existing filters along with the option to remove, add, or move Filters up or down in the listing.

Adding a new filters requires giving it a unique name (for that list), selecting the class type of the filter from a combobox and clicking the associated add button. After the filter is added, its custom settings, if any, will appear in the Settings page of the configuration UI.

Since filters can in turn contain other filters (the OrFilter being the best example of this) these lists can become quite complex and at times confusing.

6.2.2.2. Provided filters

The following is an overview of the most useful of the filters provided with Heritrix.

6.2.2.2.1. org.archive.crawler.filter.OrFilter

Contains any number of filters and returns true if any of them returns true. A logical OR on its filters basically.

6.2.2.2.2. org.archive.crawler.filter.URIRegExpFilter

Returns true if a URI matches the regular expression set for it. See Regular expressions for more about regular expressions in Heritrix.

6.2.2.2.3. org.archive.crawler.filter.ContentTypeRegExpFilter

This filter runs a regular expression against the response Content-Type header. Returns true if content type matches the regular expression. ContentType regexp filter cannot be used until after fetcher processors have run. Only then is the Content-Type of the response known. A good place for this filter is the writer step in processing. See Regular expressions for more about regular expressions in Heritrix.

6.2.2.2.4. org.archive.crawler.filter.SurtPrefixFilter

Returns true if a URI is prefixed by one of the SURT prefixes supplied by an external file.

6.2.2.2.5. org.archive.crawler.filter.FilePatternFilter

Compares suffix of a passed URI against a regular expression pattern, returns true for matches.

6.2.2.2.6. org.archive.crawler.filter.PathDepthFilter

Returns true for all CrawlURI passed in with a path depth less or equal to its `max-path-depth` value.

6.2.2.2.7. org.archive.crawler.filter.PathologicalPathFilter

Checks if a URI contains a repeated pattern.

This filter checks if a pattern is repeated a specific number of times. The use is to avoid crawler traps where the server adds the same pattern to the requested URI like:

```
http://host/img/img/img/img....
```

Returns true if such a pattern is found. Sometimes used on a processor but is primarily of use in the exclude section of scopes.

6.2.2.2.8. org.archive.crawler.filter.HopsFilter

Returns true for all URIs passed in with a Link hop count greater than the `max-link-hops` value.

Generally only used in scopes.

6.2.2.2.9. org.archive.crawler.filter.TransclusionFilter

Filter which returns true for CrawlURI instances which contain more than zero but fewer than `max-trans-hops` embed entries at the end of their Discovery path.

Generally only used in scopes.

6.2.3. Credentials

In this section you can add login credentials that will allow Heritrix to gain access to areas of websites requiring authentication. As with all modules they are only added here (supplying a unique name for each credential) and then configured on the settings page (Section 6.3, “Settings”).

One of the settings for each credential is its `credential-domain` and thus it is possible to create all credentials on the global level. However since this can cause excessive unneeded checking of credentials it is recommended that credentials be added to the appropriate domain override (see Section 6.4, “Overrides” for details). That way the credential is only checked when the relevant domain is being crawled.

Heritrix can do two types of authentication: RFC2617 [<http://www.faqs.org/rfcs/rfc2617.html>] (BASIC and DIGEST Auth) and POST and GET of an HTML Form.

Logging

To enable text console logging of authentication interactions (for example for debugging), set the `FetchHTTP` and `PrconditionEnforcer` log levels to `fine`

```
org.archive.crawler.fetcher.FetchHTTP.level = FINE
```

```
org.archive.crawler.prefetch.PreconditionEnforcer.level = FINE
```

This is done by editing the `heritrix.properties` file under the `conf` directory as described in Section 2.2.2.1, “`heritrix.properties`”.

6.2.3.1. RFC2617 [<http://www.faqs.org/rfcs/rfc2617.html>] (BASIC and DIGEST Auth)

Supply credential-domain [#cd], realm, login, and password.

The way that the RFC2617 authentication works in Heritrix is that in response to a 401 response code (Unauthorized), Heritrix will use a key made up of the Credential Domain plus Realm to do a lookup into its Credential Store. If a match is found, then the credential is loaded into the CrawlURI and the CrawlURI is marked for immediate retry.

When the requested CrawlURI comes around again, this time through, the found credentials are added to the request. If the request succeeds -- result code of 200 -- the credentials are promoted to the CrawlServer and all subsequent requests made against this CrawlServer will preemptively volunteer the credential. If the credential fails -- we get another 401 -- then the URI is let die a natural 401 death.

6.2.3.1.1. credential-domain

This equates to the canonical root URI of RFC2617; effectively, in our case, its the CrawlServer name or URI authority [<http://java.sun.com/j2se/1.4.2/docs/api/java/net/URI.html>] (domain plus port if other than port 80). Examples of credential-domain would be: 'www.archive.org' or 'www.archive.org:8080', etc.

6.2.3.1.2. realm

Realm as per RFC2617 [<http://www.faqs.org/rfcs/rfc2617.html>]. The realm string must match exactly the realm name presented in the authentication challenge served up by the web server

6.2.3.1.3. Known Limitations

One Realm per Credential Domain Only

Currently, you can only have one realm per credential domain.

Digest Auth works for Apache

... but your mileage may vary going up against other servers (See [914301] Logging in (HTTP POST, Basic Auth, etc.) [http://sourceforge.net/tracker/index.php?func=detail&aid=914301&group_id=73833&atid=539102] to learn more).

6.2.3.2. HTML Form POST or GET

Supply credential-domain [#cdh], http-method [httpmethod], login-uri [loginuri], and form-items [formitems], .

Before a uri is scheduled, we look for preconditions. Examples of preconditions are the getting of the the dns record for the server that hosts the uri and the fetching of the `robots.txt`: i.e. we don't fetch any uri unless we first have gotten the `robots.txt` file. The HTML Form Credentials are done as a precondition. If there are HTML Form Credentials for a particular crawlserver in the credential store, the uri specified in the HTML Form Credential login-uri field is scheduled as a precondition for the site, after the fetching of the dns and robots preconditions.

6.2.3.2.1. credential-domain

Same as the Rfc22617 Credential credential-domain [#cd].

6.2.3.2.2. login-url

Relative or absolute URI to the page that the HTML Form submits to (Not the page that contains the HTML Form).

6.2.3.2.3. form-items

Listing of HTML Form key/value pairs. Don't forget to include the form submit button.

6.2.3.2.4. Known Limitations

Site is crawled logged in or not; cannot do both

If a site has an HTML Form Credential associated, the next thing done after the getting of the dns record and the robots.txt is that a login is performed against all listed HTML Form Credential login-uris. This means that the crawler will only ever view sites that have HTML Form Credentials from the 'logged-in' perspective. There is no way currently of telling the crawler to crawl the site 'non-logged-in' and then, when done, log in and crawl the site anew only this time from the 'logged-in' perspective (At least, not as part of the one crawl job).

No means of verifying or rerunning login

The login is run once only and the crawler continues whether the login succeeded or not. There is no means of telling the crawler retry upon unsuccessful authentication. Neither is there a means for the crawler to report success or otherwise (The operator is expected to study logs to see whether authentication ran successfully).

6.3. Settings

This page presents a semi-treelike representation of all the modules (fixed and pluggable alike) that make up the current configuration and allows the user to edit any of their settings. Go to the Modules and SubModules tabs to add, remove, replace modules mentioned here in the Settings page.

The first option presented directly under the top tabs is whether to hide or display 'expert settings'. Expert settings are those settings that are rarely changed and should only be changed by someone with a clear understanding of their implication. This document will not discuss any of the expert settings.

The first setting is the description of the job previously discussed. The seed list is at the bottom of the page. Between the two are all the other possible settings.

Module names are presented in bold and a short explanation of them is provided. As discussed in the previous three chapters some of them can be replaced, removed or augmented.

Behind each module and settings name a small question mark is present. By clicking on it a more detailed explanation of the relevant item pops up. For most settings users should refer to that as their primary source of information.

Some settings provide a fixed number of possible 'legal' values in combo boxes. Most are however typical text input fields. Two types of settings require a bit of additional attention.

- **Lists**

Some settings are a list of values. In those cases a list is printed with an associated *Remove* button

and an input box is printed below it with an *Add* button. Only those items in the list box are considered in the list itself. A value in the input box does not become a part of the list until the user clicks *Add*. There is no way to edit existing values beyond removing them and replacing them with correct values. It is also not possible to reorder the list.

- **Simple typed maps**

Generally Maps in the Heritrix settings framework contain program modules (such as the processors for example) and are therefore edited elsewhere. However maps that only accept simple data types (Java primitives) can be edited here.

They are treated as a key, value pair. Two input boxes are provided for new entries with the first one representing the key and the second the value. Clicking the associated *Add* button adds the entry to the map. Above the input boxes a list of existing entries is displayed along with a *Remove* option. Simple maps can not be reordered.

Changes on this page are not saved until you navigate to another part of the settings framework or you click the submit job/finished tab.

If there is a problem with one of the settings a red star will appear next to it. Clicking the star will display the relevant error message.

6.3.1. Basic settings

Some settings are always present. They form the so called crawl order. The root of the settings hierarchy that other modules plug into.

6.3.1.1. Crawl limits

In addition to limits imposed on the scope of the crawl it is possible to enforce arbitrary limits on the duration and extent of the crawl with the following settings:

- **max-bytes-download**

Stop after a fixed number of bytes have been downloaded. 0 means unlimited.

- **max-document-download**

Stop after downloading a fixed number of documents. 0 means unlimited.

- **max-time-sec**

Stop after a certain number of seconds have elapsed. 0 means unlimited.

For handy reference there are 3600 seconds in an hour and 86400 seconds in a day.

Note

These are not hard limits. Once one of these limits is hit it will trigger a graceful termination of the crawl job, that means that URIs already being crawled will be completed. As a result the set limit will be exceeded by some amount.

6.3.1.2. max-toe-threads

Set the number of toe threads (see Toe Threads).

If running a domain crawl smaller than 100 hosts a value approximately twice the number of hosts should be enough. Values larger than 150-200 are rarely worthwhile unless running on machines with exceptional resources.

6.3.1.3. HTTP headers

Currently Heritrix supports configuring the `user-agent` and `from` fields in the HTTP headers generated when requesting URIs from web servers.

6.3.1.3.1. from

The `from` attribute must contain a valid e-mail address.

6.3.1.3.2. user-agent

The initial user-agent template you see when you first start heritrix will look something like the following:

```
Mozilla/5.0 (compatible; heritrix/0.11.0 +PROJECT_URL_HERE
```

You must change at least the `PROJECT_URL_HERE` and put in place a website that webmasters can go to to view information on the organization or person running a crawl.

The `user-agent` string must adhere to the following format:

```
[optional-text] ([optional-text] +PROJECT_URL [optional-text]) [optional-text]
```

The parenthesis and plus sign before the URL must be present. Other examples of valid user agents would include:

```
my-heritrix-crawler (+http://mywebsite.com)
Mozilla/5.0 (compatible; bush-crawler +http://whitehouse.gov)
Mozilla/5.0 (compatible; os-heritrix/0.11.0 +http://loc.gov on behalf to the Librarian)
```

6.3.1.4. Robots honoring policy

There are five types of policies offered on how to deal with `robots.txt` rules:

1. **classic**

Simply obey the `robots.txt` rules. Recommended unless you have special permission to collect a site more aggressively.

2. **ignore**

Completely ignore `robots.txt` rules.

3. **custom**

Obey user set, custom, `robots.txt` rules instead of those discovered on the relevant site.

Mostly useful in overrides.

4. **most-favored**

Obey the rules set in the `robots.txt` for the robot that is allowed to access the most or has the least restrictions. Can optionally masquerade as said robot.

5. **most-favored-set**

Same as 4, but limit the robots whose rules we can follow to a given set.

Note

Choosing options 3-5 requires setting additional information in the fields below the policy combobox. For options 1 and 2 those can be ignored.

6.3.2. Scope settings

The different scopes do share a few common settings. (See Section 6.1.1, “Crawl Scope” for more on scopes provided with Heritrix.)

- **max-link-hops**

Maximum number of links followed to get to the current URI. Basically counts 'L's in the Discovery path.

- **max-trans-hops**

Maximum number of non link hops at the end of the current URI's Discovery path. Generally we don't want to follow embeds, redirects and the like for more than a few (default 5) hops in a row. Such deep embedded structures are usually crawler traps. Since embeds are usually treated with higher priority than links, getting stuck in this type of trap can be particularly harmful to the crawl.

Additionally scopes may possess many more settings, depending on what filters are attached to them. See the related pop-up help in the WUI for information on those.

6.3.3. Frontier settings

The Frontier provided with Heritrix has a few settings of particular interest.

6.3.3.1. Politeness

A combination of four settings controls the politeness of the Frontier. Before we cover them it is important to note that at any given time only one URI from any given Host is being processed. The following politeness rules all revolve around imposing additional wait time between the end of processing one URI and until the next one starts.

- **delay-factor**

Imposes a delay between URIs from the same host that is a multiple of the amount of time it took to fetch the last URI downloaded from that host.

For example if it took 800 milliseconds to fetch the last URI from a host and the delay-factor is 5 (a very high value) then the Frontier will wait 4000 milliseconds (4 seconds) before allowing another URI from that host to be processed.

This value can be set to 0 for maximum impoliteness. It is never possible to have multiple concurrent URIs being processed from the same host.

- **max-delay-ms**

This setting allows the user to set a maximum upper limit on the 'in between URIs' wait created by the delay factor. If set to 1000 milliseconds then in the example used above the Frontier would only hold URIs from that host for 1 second instead of 4 since the delay factor exceeded this ceiling value.

- **min-delay-ms**

Similar to the maximum limit, this imposes a minimum limit to the politeness. This can be useful to ensure, for example, that at least 100 milliseconds elapse between connections to the same host. In a case where the delay factor is 2 and it only took 20 milliseconds to get a URI this would come into effect.

- **min-interval-ms**

An alternate way of putting a floor on the delay, this specifies the minimum number of milliseconds that must elapse from the *start of processing* one URI until the next one after it starts. This can be useful in cases where sites have a mix of large files that take an excessive amount of time and very small files that take virtually no time.

In all cases (this can vary from URI to URI) the more restrictive (delaying) of the two floor values is imposed.

6.3.3.2. Retry policy

The Frontier imposes a policy on retrying URIs that encountered errors that usually are transitory (socket timeouts etc.) . Fetcher processors may also have their own policies on certain aspects of this.

- **max-retries**

How often to retry URIs that encounter possible transient errors.

- **retry-delay-seconds**

How long to wait between such retries.

6.3.3.3. Bandwidth limits

The Frontier allows the user to limit bandwidth usage. This is done by holding back URIs when bandwidth usage has exceeded limits. As a result individual spikes of bandwidth usage can occur that greatly exceed this limit. This only limits overall bandwidth usage over a longer period of time (minutes).

- **total-bandwidth-usage-KB-sec**

Maximum bandwidth to use in Kilobytes per second.

- **max-per-host-bandwidth-usage-KB-sec**

Maximum bandwidth to use in dealing with any given host. This is a form of politeness control as it limits the load Heritrix places on a host.

6.3.4. Processors settings

A couple of the provided processors have settings that merit some extra attention.

As has been noted elsewhere each processor has a setting named **enabled**. This is set to true by default, but can be set to false to effectively remove it from consideration. Processors whose enabled setting is set to false will not be applied to any applicable URI (this is of greatest use in overrides and refinements).

6.3.4.1. HTTP Fetcher

- **timeout-seconds**

If a fetch is not completed within this many seconds, the HTTP fetcher will terminate it.

Should generally be set quite high.

- **max-length-bytes**

Maximum number of bytes to download per document. Will truncate file once this limit is reached.

By default this value is set to an extremely large value (in the exabyte range) that will never be reached in practice.

Its also possible to add in filters that are checked after the download of the HTTP response headers but before the response content is read. Use `midfetch-filters` to abort the download of content-types other than those wanted (Aborted fetches have an annotation `midFetchAbort` appended to the `crawl.log` entry). Note that unless the same filters are applied at the writer processing step, the response headers -- but not the content -- will show in ARC files.

6.3.4.2. Archiver

The ARC writer processor can be configured somewhat. This mostly relates to how the ARC files are written to disk.

- **compress**

Write compressed ARC files true or false.

Note

Each item that is added to the ARC file will be compressed individually.

- **prefix**

A prefix to the ARC files filename. See Section 9.1.9, “ARC files” for more on ARC file naming.

- **max-size-bytes**

Maximum size per ARC file. Once this size is reached no more documents will be added to an ARC file, another will be created to continue the crawl. This is of course not a hard limit and the last item added to an ARC file will push its size above this limit. If exceptionally large items are being downloaded the size of an ARC file may exceed this value by a considerable amount since items will never be split between ARC files.

- **path**

Path where ARC files should be written. Can be a list of absolute paths. If relative paths, will be relative to the `jobdirectory`. It can be safely configured mid-crawl to point elsewhere if current location is close to full. If multiple paths, then we'll choose from the list of paths in a round-robin fashion.

- **pool-max-active**

The Archiver maintains a pool of ARC files which are each ready to accept a downloaded documents, to prevent ARC writing from being a bottleneck in multithreaded operation. This setting establishes the maximum number of such files to keep ready. Default is 5. For small crawls that you want to confine to a single ARC file, this should be set to 1.

- **pool-max-wait**

The maximum amount of time to wait on the Archiver's pool element.

6.4. Overrides

Overrides provide the ability to override individual settings on a per domain basis. The overrides page provides an iterative list of domains that contain override settings, that is values for parameters that override values in the global configuration.

It is best to think of the general global settings as the root of the settings hierarchy and they are then overridden by top level domains (com, net, org, etc) who are in turn overridden by domains (yahoo.com, archive.org, etc.) who can further be overridden by subdomains (crawler.archive.org). There is no limit for how deep into the subdomains the overrides can go.

When a URI is being processed the settings for its host is first looked up. If the needed setting is not available there, its super domains are checked until the setting is found (all settings exist at the global level at the very least).

Creating a new override is done by simply typing in the domain in the input box at the bottom of the page and clicking the *Create / Edit* button. Alternatively if overrides already exist the user can navigate the hierarchy of existing overrides, edit them and create new overrides on domains that don't already have them.

Once an override has been created or selected for editing the user is taken to a page that closely resembles the settings page discussed in Section 6.3, "Settings". The main difference is that those settings that can not be overridden (file locations, number of threads etc.) are printed in a non-editable manner. Those settings that can be edited now have a checkbox in front of them. If they are being overridden at the current level that checkbox should be checked. Editing a setting will cause the checkmark to appear. Removing the checkmark effectively removes the override on that setting.

Once on the settings page the second level tabs will change to override context. The new tabs will be similar to the general tabs and will have:

- **URL**

Add URL Canonicalization Rules to the override. It is not possible to remove inherited filters or interject new filters among them. New filters will be added after existing filters.

- **Filters**

Add filters to the override. It is not possible to remove inherited filters or interject new filters among

them. New filters will be added after existing filters.

Inherited filters though have the option to locally disable them. That can be set on the settings page.

- **Credentials**

Add credentials to the override. Generally credentials should always be added to an override of the domain most relevant to them. See Section 6.2.3, “Credentials” for more details.

- **Settings**

Page allowing the user to override specific settings as discussed above.

- **Refinements**

Manage refinements for the override. See Section 6.5, “Refinements”

- **Done with override**

Once the user has finished with the override, this option will take him back to the overrides overview page.

It is not possible to add, remove or reorder existing modules on an override. It is only possible to add filters and credentials. Those added will be inherited to sub domains of the current override domain. Those modules that are added in an override will not have a checkbox in front of their settings on the override settings page since the override is effectively their 'root'.

Finally, due to how the settings framework is structured there is negligible performance penalty to using overrides. Lookups for settings take as much time whether or not overrides have been defined. For URIs belonging to domains without overrides no performance penalty is incurred.

6.5. Refinements

Refinements are similar to overrides (see Section 6.4, “Overrides”) in that they allow the user to modify the settings under certain circumstances. There are however two major differences.

1. Refinements are applied based on arbitrary criteria rather than encountered URIs domain.

Currently it is possible to set criteria based on the time of day, a regular expression matching the URI and the port number of the URI.

2. They incur a performance penalty.

This effect is small if their numbers are few but for each URI encountered there must be a check made to see if it matches any of the existing criteria of defined refinements.

This effect can be mitigated by applying refinements to overrides rather than the global settings.

Refinements can be applied either to the global settings or to any override. If applied to an override they can affect any settings, regardless of whether the parent override has modified it.

It is not possible to create refinements on refinements.

Clicking the *Refinements* tab on either the global settings or an override brings the user to the refinements overview page. The overview page displays a list of existing refinements on the current level and

allows the user to create new ones.

To create a new refinement the user must supply a unique name for it (name is limited to letters, numbers, dash and underscore) and a short description that will be displayed underneath it on the overview page.

Once created, refinements can be either removed or edited.

Choosing the edit option on an override brings the user to the criteria page. Aside from the criteria tab replacing the refinements tab, the second level tabs will have the same options as they do for overrides and their behavior will be the same. Clicking the 'Done with refinement' tab will bring the user back to the refinements overview page.

6.5.1. Criteria

The criteria page displays a list of the current criteria and the option to add any of the available criteria types to the list. It is also possible to remove existing criteria.

Note

URIs must match **all** set criteria for the refinement to take effect.

Currently the following criteria can be applied:

- **Port number**

Match only those URIs for the given port number.

Default port number for HTTP is 80 and 443 for HTTPS.

- **Time of day**

If this criteria is applied the refinement will be in effect between the hours specified each day.

The format for the input boxes is HHMM (hours and minutes).

An example might be: From 0200, To 0600. This refinement would be in effect between 2 and 6 am each night. Possibly to ease the politeness requirements during these hours when load on websites is generally low.

Note

As with all times in Heritrix these are **always GMT** times.

- **Regular expression**

The refinement will only be in effect for those URIs that match the given regular expression.

Note

See Regular expressions for more on them.

7. Running a job

Once a crawl job has been created and properly configured it can be run. To start a crawl the user must go to the web Console page (via the Console tab).

7.1. Web Console

The web Console presents an overview of the current status of the crawler.

7.1.1. Crawler Status Box

The following information is always provided:

- **Crawler Status**

Is the crawler in *Holding Jobs* or *Crawling Jobs* mode? If holding, no new jobs pending or created will be started (but a job already begun will continue). If crawling, the next pending or created job will be started as soon as possible, for example when a previous job finishes. For more detail see "Holding Jobs" vs. "Crawling Jobs".

To the right of the current crawler status, a control link reading either "Start" or "Hold" will toggle the crawler between the two modes.

- **Jobs**

If a current job is in progress, its status and name will appear. Alternatively, "None running" will appear to indicate no job is in progress because the crawler is holding, or "None available" if no job is in progress because no jobs have been queued.

Below the current job info, the number of jobs pending and completed is shown. The completed count includes those that failed to start for some reason (see Section 7.3.2, "Job failed to start" for more on misconfigured jobs).

- **Alerts**

Total number of alerts, and within brackets new alerts, if any.

See Section 7.3.4, "Alerts" for more on alerts.

- **Memory**

The amount of memory currently used, the size of the Java heap, and the maximum size to which the heap can possibly grow are all displayed, in kilobytes (KB).

7.1.2. Job Status Box

If a job is in-progress -- running, paused, or between job states -- the following information is also provided in a second area underneath the *Crawler Status Box*.

- **Job Status**

The current status of the job in progress. Jobs being crawled are usually running or paused.

To the right of the current status, controls for pausing/resuming or terminating the current job will appear as appropriate.

When a job is terminated, its status will be marked as 'Ended by operator'. All currently active threads will be allowed to finish behind the scenes even though the WUI will report the job being terminated at once. If the crawler is in "Crawling Jobs" mode, a next pending job, if any, will start immediately.

When a running job is paused, it may take some time for all the active threads to enter a paused state. Until then the job is considered to be still running and 'pausing'. It is possible to resume from this interim state.

Once paused a job is considered to be suspended and time spent in that state does not count towards elapsed job time or rates.

- **Rates**

The number of URIs successfully processed per second is shown, both the rate in the latest sampling interval and (in parentheses) the average rate since the crawl began. The sampling interval is typically about 20 seconds, and is adjustable via the "interval-seconds" setting. The latest rate of progress can fluctuate considerably, as the crawler workload varies and housekeeping memory and file operations occur -- especially if the sampling interval has been set to a low value.

Also shown is the rate of successful content collection, in KB/sec, for the latest sampling interval and (in parentheses) the average since the crawl began. (See Bytes, KB and statistics.)

- **Time**

The amount of time that has elapsed since the crawl began (excluding any time spent paused) is displayed, as well as a very crude estimate of the require time remaining. (This estimate does not yet consider the typical certainty of discovering more URIs to crawl, and ignored other factors, so should not be relied upon until it can be improved in future releases.)

- **Load**

A number of measures are shown of how busy or loaded the job has made the crawler. The number of active threads, compared to the total available, is shown. Typically, if only a small number of threads are active, it is because activating more threads would exceed the configured politeness settings, given the remaining URI workload. (For example, if all remaining URIs are on a single host, no more than one thread will be active -- and often none will be, as polite delays are observed between requests.)

The *congestion ratio* is a rough estimate of how much additional capacity, as a multiple of current capacity, would be necessary to crawl the current workload at the maximum rate allowable by politeness settings. (It is calculated by comparing the number of internal queues that are progressing with those that are waiting for a thread to become available.)

The *deepest queue* number indicates the longest chain of pending URIs that must be processed sequentially, which is a better indicator of the work remaining than the total number of URIs pending. (A thousand URIs in a thousand independent queues can complete in parallel very quickly; a thousand in one queue will take longer.)

The *average depth* number indicates the average depth of the last URI in every active sequential queue.

- **Totals**

A progress bar indicates the relative percentage of completed URIs to those known and pending. As with the remaining time estimate, no consideration is given to the likelihood of discovering additional URIs to crawl. So, the percentage completed can shrink as well as grow, especially in broader crawls.

To the left of the progress bar, the total number of URIs successfully downloaded is shown; to the right, the total number of URIs queued for future processing. Beneath the bar, the total of downloaded plus queued is shown, as well as the uncompressed total size of successfully downloaded data in kilobytes. See Bytes, KB and statistics. (Compressed ARCs on disk will be somewhat smaller

than this figure.)

- **Paused Operations**

When the job is paused, additional options will appear such as *View or Edit Frontier URIs*.

The *View or Edit Frontier URIs* option takes the operator to a page allowing the lookup and deletion of URIs in the frontier by using a regular expression, or addition of URIs from an external file (even URIs that have already been processed).

Some of this information is replicated in the head of each page (see Section 7.3.3, “All page status header”).

7.1.3. Console Bottom Operations

7.1.3.1. Refresh

Update the status display. The status display does not update itself and quickly becomes out of date as crawling proceeds. This also refreshes the options available if they've changed as a result of a change in the state of the job being crawled.

7.1.3.2. Shut down Heritrix

It is possible to shut down Heritrix through this option. Doing so will terminate the Java process running Heritrix and the only way to start it up again will be via the command line as this also disables the WUI.

The user is asked to confirm this action twice to prevent accidental shut downs.

This option will try to terminate any current job gracefully but will only wait a very short time for active threads to finish.

7.2. Pending jobs

At any given time there can be any number of crawl jobs waiting for their turn to be crawled.

From the *Jobs* tab the user can access a list of these pending jobs (it also possible to get to them from the header, see Section 7.3.3, “All page status header”).

The list displays the name of each job, its status (currently all pending jobs have the status 'Pending') and offers the following options for each job:

- **View order**

Opens up the actual XML configuration file in a separate window. Of interest to advanced users only.

- **Edit configuration**

Takes the user to the Settings page of the jobs configurations (see Section 6.3, “Settings”).

- **Journal**

Takes the user to the job's Journal (see Section 7.4.1, “Journal”).

- **Delete**

Deletes the job (will only be marked as deleted, does not delete it from disk).

7.3. Monitoring a running job

In addition to the logs and reports generally available on all jobs (see Section 8.2, “Logs” and Section 8.3, “Reports”) some information is provided only for jobs being crawled.

Note

The Crawl Report (see Section 8.3.1, “Crawl report”) contains one bit of information only available on active crawls. That is the amount of time that has elapsed since a URI belonging to each host was last finished.

7.3.1. Internal reports on ongoing crawl

The following reports are only available while the crawler is running. They provide information about the internal status of certain parts of the crawler. Generally this information is only of interest to advanced users who possess detailed knowledge of the internal workings of said modules.

These reports can be accessed from the Reports tab when a job is being crawled.

7.3.1.1. Frontier report

A report on the internal state of the frontier. Can be unwieldy in size or the amount of time/memory it takes to compose in large crawls (with thousands of hosts with pending URIs).

7.3.1.2. Thread report

Contains information about what each thread is doing and how long it has been doing it. Also allows users to terminate threads that have become stuck. Terminated threads will not actually be removed from memory, Java does not provide a way of doing that. Instead they will be isolated from the rest of the program running and the URI they are working on will be reported back to the frontier as if it had failed to be processed.

Caution

Terminating threads should only be done by advanced users who understand the effect of doing so.

7.3.1.3. Processors report

A report on each processor. Not all processors provide reports. Typically these are numbers of URIs handled, links extracted etc.

This report is saved to a file at the end of the crawl (see Section 9.1.6, “processors-report.txt”).

7.3.2. Job failed to start

If a job is misconfigured in such a way that it is not possible to do any crawling it might seem as if it never started. In fact what happens is that the crawl is started but on the initialization it is immediately terminated and sent to the list of completed jobs (Section 8.1, “Completed jobs”). In those instances an explanation of what went wrong is displayed on the completed jobs page. An alert will also be created.

A common cause of this is forgetting to set the HTTP header's `user-agent` and `from` attributes to

valid values (see Section 6.3.1.3, “HTTP headers”).

If no processors are set on the job (or the modules otherwise badly misconfigured) the job may succeed in initializing but immediately exhaust the seed list, failing to actually download anything. This will not trigger any errors but a review of the logs for the job should highlight the problem. So if a job terminates immediately after starting without errors, the configuration (especially modules) should be reviewed for errors.

7.3.3. All page status header

At the top of every page in the WUI, right next to the Heritrix logo, is a brief overview of the crawler's current status.

The three lines contain the following information (starting at the top left and working across and down).

First bit of information is the current time when the page was displayed. This is useful since the status of the crawler will continue to change after a page loads, but those changes will not be reflected on the page until it is reloaded (usually manually by the user). As always this time is in GMT.

Right next to it is the number of current and new alerts.

The second line tells the user if the crawler is in "Crawling Jobs" or "Holding Jobs" mode. (See "Holding Jobs" vs. "Crawling Jobs"). If a job is in progress, its status and name will also be shown.

At the beginning of the final line the number of pending and completed jobs are displayed. Clicking on either value takes the user to the related overview page. Finally if a job is in progress, total current URIs completed, elapsed time, and URIs/sec figures are shown.

7.3.4. Alerts

The number of existing and new alerts is displayed both in the Console (Section 7.1, “Web Console”) and the header of each page (Section 7.3.3, “All page status header”).

Clicking on the link made up of those numbers takes the user to an overview of the alerts. The alerts are presented as messages, with unread ones clearly marked in bold and offering the user the option of reading them, marking as read and deleting them.

Clicking an alert brings up a screen with its details.

Alerts are generated in response to an error or problem of some form. Alerts have severity levels that mirror the Java log levels.

Serious exception that occur will have a *Severe* level. These may be indicative of bugs in the code or problems with the configuration of a crawl job.

7.4. Editing a running job

The configurations of a job can be edited while it is running. This option is accessed from the *Jobs* tab (Current job/Edit configuration). When selected the user is taken to the settings section of the job's configuration (see sSection 6.3, “Settings”).

When a configuration file is edited, the old version of it is saved to a new file (new file is named <oldFilename>_<timestamp>.xml) before it is updated. This way a record is kept of any changes. This record is only kept for changes made *after* crawling begins.

It is not possible to edit all aspects of the configuration after crawling starts. Most noticeably the Modules section is disabled. Also, although not enforced by the WUI, making changes to certain settings (in particular filenames, directory locations etc.) will have no effect (doing so will typically not harm the crawl,

it will simply be ignored).

However most settings can be changed. This includes the number of threads being used and the seeds list and although it is not possible to remove modules, most have the option to disable them. Settings a modules `enabled` attribute to `false` effectively removes them from the configuration.

If changing more than an existing atomic value -- for example, adding a new filter -- it is good practice to pause the crawl first, as some modifications to composite configuration entities may not occur in a thread-safe manner with respect to ongoing crawling otherwise.

7.4.1. Journal

The user can add notes to a journal that is kept for each job. No entries are made automatically in the journal, it is only for user added comments.

It can be useful to use it to document reasons behind configuration changes to preserve that information along with the actual changes.

The journal can be accessed from the Pending jobs page (Section 7.2, “Pending jobs”) for pending jobs, the Jobs tab for currently running jobs and the Completed jobs page (Section 8.1, “Completed jobs”) for completed jobs.

The journal is written to a plain text file that is stored along with the logs.

8. Analysis of jobs

Heritrix offers several facilities for examining the details of a crawl. The reports and logs are also available at run time.

8.1. Completed jobs

In the *Jobs* tab (and page headers) is a listing of how many completed jobs there are along with a link to a page that lists them.

The following information / options are provided for each completed job:

- **UID**

Each job has a unique (generated) ID. This is actually a time stamp. It differentiates jobs with the same name from one another.

This ID is used (among other things) for creating the job's directory on disk.

- **Job name**

The name that the user gave the job.

- **Status**

Status of the job. Indicates how it ended.

- **Options**

In addition the following options are available for each job.

- *Crawl order*

Opens up the actual XML file of the jobs configuration in a separate window. Generally only of interest to advanced users.

- *Crawl report*

Takes the user to the job's Crawl report (Section 8.3.1, “Crawl report”).

- *Seeds report*

Takes the user to the job's Seeds report (Section 8.3.2, “Seeds report”).

- *Seed file*

Displays the seed

- *Logs*

Takes the user to the job's logs (Section 8.2, “Logs”).

- *Journal*

Takes the user to the Journal page for the job (Section 7.4.1, “Journal”). Users can still add entries to it.

- *Delete*

Marks the job as deleted. This will remove it from the WUI but not from disk.

Note

It is not possible to directly access the configuration for completed jobs in the same way as you can for new, pending and running jobs. Instead users can look at the actual XML configuration file *or* create a new job based on the old one. The new job (and it need never be run) will perfectly mirror the settings of the old one.

8.2. Logs

Heritrix writes several logs as it crawls a job. Each crawl job has its own set of these logs.

The location where logs are written can be configured (expert setting). Otherwise refer to the `crawl-manifest.txt` for on disk location of logs (Section 9.1.2, “crawl-manifest.txt”).

Logs can be manually rotated. Pause the crawl and at the base of the screen a **Rotate Logs** link will appear. Clicking on **Rotate Logs** will move aside all current crawl logs appending a 14-digit GMT timestamp to the moved-aside logs. New log files will be opened for the crawler to use in subsequent crawling.

The WUI offers users four ways of viewing these logs by:

1. **Line number**

View a section of a log that starts at a given line number and the next X lines following it. X is configurable, is 50 by default.

2. **Time stamp**

View a section of a log that starts at a given time stamp and the next X lines following it. X is configurable, is 50 by default. The format of the time stamp is the same as in the logs (YYYY-MM-DDTHH:MM:SS.SSS). It is not necessary to add more detail to this than is desired. For instance the entry 2004-04-25T08 will match the first entry made after 8 am on the 25 of April, 2004.

3. Regular expression

Filter the log based on a regular expression. Only lines matching it (and optionally lines following it that are indented - usually meaning that they are related to the previous ones) are displayed.

This can be an expensive operation on really big logs, requiring a lot of time for the page to load.

4. Tail

Allows users to just look at the last X lines of the given log. X can be configured, is 50 by default.

8.2.1. crawl.log

For each URI tried will get an entry in the `crawl.log` regardless of success or failure.

Below is a two line extract from a `crawl.log`:

```
2004-07-21T23:29:40.438Z    200          310 http://127.0.0.1:9999/selftest/Charset/c
2004-07-21T23:29:40.502Z    200          225 http://127.0.0.1:9999/selftest/MaxLinkHo
```

The *1st* column is a timestamp in ISO8601 format, to millisecond resolution. The time is the instant of logging. The *2nd* column is the fetch status code. Usually this is the HTTP status code but it can also be a negative number if URL processing was unexpectedly terminated. See Status codes for a listing of possible values.

The *3rd* column is the size of the downloaded document in bytes. For HTTP, Size is the size of the content-only. It excludes the size of the HTTP response headers. For DNS, its the total size of the DNS response. The *4th* column is the URI of the document downloaded. The *5th* column holds breadcrumb codes showing the trail of downloads that got us to the current URI. See Discovery path for description of possible code values. The *6th* column holds the URI that immediately referenced this URI ('referrer'). Both of the latter two fields -- the discovery path and the referrer URL -- will be empty for such as the seed URIs.

The *7th* holds the document mime type, the *8th* column has the id of the worker thread that downloaded this document, the *9th* column holds a timestamp (in RFC2550/ARC condensed digits-only format) indicating when a network fetch was begun, and if appropriate, the millisecond duration of the fetch, separated from the begin-time by a '+' character.

The *10th* field is a SHA1 digest of the content only (headers are not digested). The *11th* column is the 'source tag' inherited by this URI, if that feature is enabled. Finally, the *12th* column holds "annotations", if any have been set. Possible annotations include: the number of times the URI was tried (This field is '-' if the download was never retried); the literal `lenTrunc` if the download was truncated because it exceeded configured limits; `timeTrunc` if the download was truncated because the download time exceeded configured limits; or `midFetchTrunc` if a midfetch filter determined the download should be truncated.

8.2.2. local-errors.log

Errors that occur when processing a URI that can be handled by the processors (usually these are network related problems trying to fetch the document) are logged here.

Generally these can be safely ignored, but can provide insight to advanced users when other logs and/or reports have unusual data.

8.2.3. progress-statistics.log

This log is written by the `StatisticsTracker` (Section 6.1.4, “Statistics Tracking”).

At configurable intervals a line about the progress of the crawl is written to this file.

The legends are as follows:

- **timestamp**

Timestamp indicating when the line was written, in ISO8601 format.

- **discovered**

Number of URIs discovered to date.

- **queued**

Number of URIs queued at the moment.

- **downloaded**

Number of URIs downloaded to date

- **doc/s(avg)**

Number of documents downloaded per second since the last snapshot. In parenthesis since the crawl began.

- **KB/s(avg)**

Amount in Kilobytes downloaded per second since the last snapshot. In parenthesis since the crawl began.

- **dl-failures**

Number of URIs that Heritrix has failed to download to date.

- **busy-thread**

Number of toe threads currently busy processing a URI.

- **mem-use-KB**

Amount of memory currently assigned to the Java Virtual Machine.

8.2.4. runtime-errors.log

This log captures unexpected exceptions and errors that occur during the crawl. Some may be due to hardware limitation (out of memory, although that error may occur without being written to this log), but most are probably because of software bugs, either in Heritrix's core but more likely in one of the plug-

gable classes.

8.2.5. uri-errors.log

Contains errors in dealing with encountered URIs. Usually its caused by erroneous URIs. Generally only of interest to advanced users trying to explain unexpected crawl behavior.

8.2.6. recover.gz

The recover.gz file is a gzipped journal of Frontier events. It can be used to restore the Frontier after a crash to roughly the state it had before the crash. See Section 9.3, “Recovery of Frontier State and recover.gz” to learn more.

8.3. Reports

Heritrix's WUI offers a couple of reports on ongoing and completed crawl jobs.

Both are accessible via the Reports tab.

Note

Although jobs are loaded after restarts of the software, their statistics are not reloaded with them. That means that these reports are only available as long as Heritrix is not shut down. All of the information is however replicated in report files at the end of each crawl for permanent storage.

8.3.1. Crawl report

At the top of the crawl report some general statistics about the crawl are printed out. All of these replicate data from the Console so you should refer to Section 7.1, “Web Console” for more information on them.

Next in line are statistics about the number of URIs pending, discovered, currently queued, downloaded etc. Question marks after most of the values provides pop up descriptions of those metrics.

Following that is a breakdown of the distribution of status codes among URIs. It is sorted from most frequent to least. The number of URIs found for each status code is displayed. Only successful fetches are counted here.

A similar breakdown for file types (mime types) follows. In addition to the number of URIs per file type, the amount of data for that file type is also displayed.

Last a breakdown per host is provided. Number of URIs and amount of data for each is presented. The time that has elapsed since the last URI was finished for each host is also displayed for ongoing crawls. This value can provide valuable data on what hosts are still being actively crawled. Note that this value is only available while the crawl is in progress since it has no meaning afterwards. Also any pauses made to the crawl may distort these values, at least in the short term following resumption of crawling. Most noticeably while paused all of these values will continue to grow.

Especially in broad crawls, this list can grow very large.

8.3.2. Seeds report

This report lists all the seeds in the seeds file and also any *discovered* seeds if that option is enabled (that is treat redirects from seeds as new seeds). For each seed the status code for the fetch attempt is presented in verbose form (that is with minimum textual description of its meaning). Following that is the

seeds disposition, a quick look at if the seed was successfully crawled, not attempted, or failed to crawl.

Successfully crawled seeds are any that Heritrix had no internal errors crawling, the seed may never the less have generated a 404 (file not found) error.

Failure to crawl might be because of a bug in Heritrix or an invalid seed (commonly DNS lookup will have failed).

If the report is examined before the crawl is finished there might still be seeds not yet attempted. Especially if there is trouble getting their prerequisites or if the seed list is exceptionally large.

9. Outside the user interface

While it is possible to do a great many things via Heritrix's WUI it is worth taking a look at some of what is not available in it.

9.1. Generated files

In addition to the logs discussed above (see Section 8.2, “Logs”) the following files are generated. Some of the information in them is also available via the WUI.

9.1.1. heritrix_out.log

Captures what is written to the standard out and standard error streams of the program. Mostly this consists of low level exceptions (usually indicative of bugs) and also some information from third party modules who do their own output logging.

This file is created in the same directory as the Heritrix JAR file. It is not associated with any one job, but contains output from all jobs run by the crawler.

9.1.2. crawl-manifest.txt

A manifest of all files (excluding ARC and other data files) created while crawling a job.

An example of this file might be:

```
L+ /Heritrix/jobs/quickbroad-20040420191411593/disk/crawl.log
L+ /Heritrix/jobs/quickbroad-20040420191411593/disk/runtime-errors.log
L+ /Heritrix/jobs/quickbroad-20040420191411593/disk/local-errors.log
L+ /Heritrix/jobs/quickbroad-20040420191411593/disk/uri-errors.log
L+ /Heritrix/jobs/quickbroad-20040420191411593/disk/progress-statistics.log
L- /Heritrix/jobs/quickbroad-20040420191411593/disk/recover.gz
R+ /Heritrix/jobs/quickbroad-20040420191411593/disk/seeds-report.txt
R+ /Heritrix/jobs/quickbroad-20040420191411593/disk/hosts-report.txt
R+ /Heritrix/jobs/quickbroad-20040420191411593/disk/mimetype-report.txt
R+ /Heritrix/jobs/quickbroad-20040420191411593/disk/responsecode-report.txt
R+ /Heritrix/jobs/quickbroad-20040420191411593/disk/crawl-report.txt
R+ /Heritrix/jobs/quickbroad-20040420191411593/disk/processors-report.txt
C+ /Heritrix/jobs/quickbroad-20040420191411593/job-quickbroad.xml
C+ /Heritrix/jobs/quickbroad-20040420191411593/settings/org/settings.xml
C+ /Heritrix/jobs/quickbroad-20040420191411593/seeds-quickbroad.txt
```

The first character of each line indicates the type of file. L for logs, R for reports and C for configuration files.

The second character - a plus or minus sign - indicates if the file should be included in a standard bundle of the job (see Section 9.2.1, “manifest_bundle.pl”). In the example above the `recover.gz` is marked

for exclusion because it is generally only of interest if the job crashes and must be restarted. It has negligible value once the job is completed (See Section 9.3, “Recovery of Frontier State and recover.gz”).

After this initial legend the filename with full path follows.

This file is generated in the directory indicated by the *'disk'* attribute of the configuration at the very end of the crawl.

9.1.3. crawl-report.txt

Contains some useful metrics about the completed jobs. This report is created by the `StatisticsTracker` (see Section 6.1.4, “Statistics Tracking”)

Written at the very end of the crawl only. See `crawl-manifest.txt` for its location.

9.1.4. hosts-report.txt

Contains an overview of what hosts were crawled and how many documents and bytes were downloaded from each.

This report is created by the `StatisticsTracker` (see Section 6.1.4, “Statistics Tracking”) and is written at the very end of the crawl only. See `crawl-manifest.txt` for its location.

9.1.5. mimetype-report.txt

Contains an overview of the number of documents downloaded per mime type. Also has the amount of data downloaded per mime type.

This report is created by the `StatisticsTracker` (see Section 6.1.4, “Statistics Tracking”) and is written at the very end of the crawl only. See `crawl-manifest.txt` for its location.

9.1.6. processors-report.txt

Contains the processors report (see Section 7.3.1.3, “Processors report”) generated at the very end of the crawl.

9.1.7. responsecode-report.txt

Contains an overview of the number of documents downloaded per status code (see Status codes), covers successful codes only, does not tally failures, see `crawl.log` for that information.

This report is created by the `StatisticsTracker` (see Section 6.1.4, “Statistics Tracking”) and is written at the very end of the crawl only. See `crawl-manifest.txt` for its location.

9.1.8. seeds-report.txt

An overview of the crawling of each seed. Did it succeed or not, what status code was returned.

This report is created by the `StatisticsTracker` (see Section 6.1.4, “Statistics Tracking”) and is written at the very end of the crawl only. See `crawl-manifest.txt` for its location.

9.1.9. ARC files

Assuming that you are using the ARC writer that comes with Heritrix a number of ARC files will be generated containing the crawled pages.

It is possible to specify the location of these files on the ARCWriter processor in settings page. Unless

this is set as an absolute path this is a path relative to the *job* directory.

ARC files are named as follows:

```
[prefix]-[12-digit-timestamp]-[series#-padded-to-5-digits]-[crawler-hostname].ar
```

The `prefix` is set by the user when he configures the ARCWriter processor. By default it is `IAH`.

If you see an ARC file with an extra `.open` suffix, this means the ARC is currently in use being written to by Heritrix (It usually has more than one ARC open at a time).

Files with a `.invalid` are files Heritrix had trouble writing to (Disk full, bad disk, etc.). On `IOException`, Heritrix closes the problematic ARC and gives it the `.invalid` suffix. These files need to be checked for coherence.

For more on ARC files refer to the ARCwriter Javadoc [<http://crawler.archive.org/apidocs/org/archive/io/arc/ARCWriter.html>] and to the ARC Writer developer documentation [http://crawler.archive.org/articles/developer_manual/arcs.html].

9.2. Helpful scripts

Heritrix comes bundled with a few helpful scripts for Linux.

9.2.1. manifest_bundle.pl

This script will bundle up all resources referenced by a crawl manifest file (Section 9.1.2, “crawl-manifest.txt”). Output bundle is an uncompressed or compressed tar ball. Directory structure created in the tar ball is as follow:

- Top level directory (crawl name)
- Three default subdirectories (configuration, logs and reports directories)
- Any other arbitrary subdirectories

Usage:

```
manifest_bundle.pl crawl_name manifest_file [-f output_tar_file] [-z] [ -flag di
-f output tar file. If omitted output to stdout.
-z compress tar file with gzip.
-flag is any upper case letter. Default values C, L, and are R are set to
configuration, logs and reports
```

Example:

```
manifest_bundle.pl testcrawl crawl-manifest.txt -f \
/O/testcrawl/manifest-bundle.tar.gz -z -F filters
```

Produced tar ball for this example:

```
/O/testcrawl/manifest-bundle.tar.gz
```

Bundled directory structure for this example:

```
| -testcrawl
|   - configurations
|   - logs
|   - reports
|   - filters
```

9.2.2. hoppath.pl

This perl script, found in \$HERITRIX_HOME/bin recreates the hop path to the specified url. The hop path is a path of links (URLs) that we followed to get to the specified url.

Usage:

```
Usage: hoppath.pl crawl.log URI_PREFIX
crawl.log    Full-path to Heritrix crawl.log instance.
URI_PREFIX   URI we're querying about. Must begin 'http(s)://' or 'dns:'.
              Wrap this parameter in quotes to avoid shell interpretation
              of any '&' present in URI_PREFIX.
```

Example:

```
% hoppath.pl crawl.log 'http://www.house.gov/'
```

Result:

```
2004-02-25-02-36-06 - http://www.house.gov/house/MemberWWW_by_State.html
2004-02-25-02-36-06  L http://www.house.gov/search97cgi/s97.cgi
2004-02-25-03-30-38  L http://www.house.gov/
```

The L in the above example refers to the type of link followed (see Discovery path).

9.2.3. RecoverLogMapper

org.archive.crawler.util.RecoveryLogMapper is similar to Section 9.2.2, “hoppath.pl”. It was contributed by Mike Schwartz. RecoverLogMapper parses a Heritrix recovery log file, (See Section 9.3, “Recovery of Frontier State and recover.gz”), and builds maps that allow a caller to look up any seed URL and get back an Iterator of all URLs successfully crawled from given seed. It also allows lookup on any crawled URL to find the seed URL from which the crawler reached that URL (through 1 or more discovered URL hops, which are collapsed in this lookup).

9.2.4. cmdline-jmxclient

This jar file is checked in as a script. It enables command-line control of Heritrix if Heritrix has been started up inside of a SUN 1.5.0 JDK. See the cmdline-jmxclient project [<http://crawler.archive.org/cmdline-jmxclient/>] to learn more about this script's capabilities and how to use it. See also Section 9.5, “Remote Monitoring and Control”.

9.3. Recovery of Frontier State and recover.gz

During normal running, the Heritrix Frontier by default keeps a journal. The journal is kept in the jobs logs directory. Its named `recover.gz`. If a crawl crashes, the `recover.gz` journal can be used to recreate approximately the status of the crawler at the time of the crash. Recovery can take a long time in some cases, but is usually much quicker than repeating a crawl.

To run the recovery process, relaunch the crashed crawler. Create a new crawl order job based on the crawl that crashed. If you choose the "recover-log" link from the list of completed jobs in the 'Based on recovery' page, the new job will automatically be set up to use the original job's recovery journal to bootstrap its Frontier state (of completed and queued URIs). Further, if the recovered job attempts to reuse any already-full 'logs' or 'state' directories, new paths for these directories will be chosen with as many '-R' suffixes as are necessary to specify a new empty directory.

(If you simply base your new job on the old job without using the 'Recover' link, you must manually enter the full path of the original crawl's recovery journal into the `recover-path` setting, near the end of all settings. You must also adjust the 'logs' and 'state' directory settings, if they were specified using absolute paths that would cause the new crawl to reuse the directories of the original job.)

After making any further adjustments to the crawl settings, submit the new job. The submission will hang a long time as the `recover.gz` file is read in its entirety by the frontier. (This can take hours for a crawl that has run for a long time, and during this time the crawler control panel will appear idle, with no job pending or in progress, but the machine will be busy.) Eventually the submit and crawl job launch should complete. The crawl should pick up from close to where the crash occurred. There is no marking in the logs that this crawl was started by reading a recover log (Be sure to mark this was done in the crawl journal).

The recovery log is gzipped because it gets very large otherwise and because of the repetition of terms, it compresses very well. On abnormal termination of the crawl job, if you look at the `recover.gz` file with `gzip`, `gzip` will report `unexpected end of file` if you try to ungzip it. `Gzip` is complaining that the file write was abnormally terminated. But the `recover.gz` file will be of use restoring the frontier at least to where the `gzip` file went bad (`Gzip` zips in 32k blocks; the worst loss would be the last 32k of gzipped data).

Java's `Gzip` support (up through at least Java 1.5/5.0) can compress arbitrarily large input streams, but has problems when decompressing any stream to output larger than 2GB. When attempting to recover a crawl that has a recovery log that, when uncompressed, would be over 2GB, this will trigger a `FatalConfigurationException` alert, with detail message "Recover.log problem: java.io.IOException: Corrupt GZIP trailer". Heritrix will accept either compressed or uncompressed recovery log files, so a work-around is to first uncompress the recovery log using another non-Java tool (such as the 'gunzip' available in Linux and Cygwin), then refer to this uncompressed recovery log when recovering. (Reportedly, Java 6.0 "Mustang" will fix this Java bug with un-gzipping large files.)

See also below, the related recovery facility, Section 9.4, "Checkpointing", for an alternate recovery mechanism.

9.4. Checkpointing

Checkpointing [Checkpointing], the crawler writes a representation of current state to a directory under `checkpoints-path`, named for the checkpoint. Checkpointed state includes serialization of main crawler objects, copies of current set of bdbje log files, etc. The idea is that the checkpoint directory contains all that is required recovering a crawler. Checkpointing also rotates off the crawler logs including the `recover.gz` log, if enabled. Log files are NOT copied to the checkpoint directory. They are left under the logs directory but are distinguished by a suffix. The suffix is the checkpoint name (e.g. `crawl.log.000031` where 000031 is the checkpoint name).

Note

Currently, only the `BdbFrontier` using the bdbje-based already-seen or the bloom filter already-seen is checkpointable.

To run a checkpoint, click the checkpoint button in the UI or invoke `checkpoint` from JMX. This launches a thread to run through the following steps: If crawling, pause the crawl; run actual checkpoint; if was crawling when checkpoint was invoked, resume the crawl. Dependent on the size of the crawl,

checkpointing can take some time; often the step that takes longest is pausing the crawl, waiting on threads to get into a paused, checkpointable state. While checkpointing, the status will show as CHECKPOINTING. When the checkpoint has completed -- the crawler will resume crawling (Of if in PAUSED state when checkpointing was invoked, will return to the PAUSED state).

Recovery from a checkpoint has much in common with the recovery of a crawl using the recover.log (See ???). To recover, create a job. Then before launching, set the `crawl-order/recover-path` to point at the checkpoint directory you want to recover from. Alternatively, browse to the `Jobs->Based on a recovery` screen and select the checkpoint you want to recover from. After clicking, a new job will be created that takes the old jobs' (end-of-crawl) settings and autofills the `recover-path` with the right directory-path (The renaming of logs and `crawl-order/state-path` "state" dirs so they do not clash with the old as is described above in Section 9.3, "Recovery of Frontier State and recover.gz" is also done). The first thing recover does is copy into place the saved-off bdbje log files. Again, recovery can take time -- an hour or more if a crawl of millions.

Checkpointing is currently experimental. The recover-log technique is tried-and-true. Once checkpointing is proven reliable, faster, and more comprehensive, it will become the preferred method recovering a crawler).

9.4.1. Expert mode: Fast checkpointing

The bulk of the time checkpointing is taken up copying off the bdbje logs. For example, checkpointing a crawl that had downloaded 18million items -- it had discovered > 130million (bloom filter) -- the checkpointing took about 100minutes to complete of which 90 plus minutes were spent copying the ~12k bdbje log files (One disk only involved). Set log level on `org.archive.util.FileUtils` to FINE to watch the java bdbje log file-copy.

Since copying off bdbje log files can take hours, we've added an *expert mode* checkpoint that bypasses bdbje log copying. The upside is your checkpoint completes promptly -- in minutes, even if the crawl is large -- but downside is recovery takes more work: to recover from a checkpoint, the bdbje log files need to be manually assembled in the checkpoint `bdb-logs` subdirectory. You'll know which bdbje log files make up the checkpoint because Heritrix writes the checkpoint list of bdbje logs into the checkpoint directory to a file named `bdbj-logs-manifest.txt`. To prevent bdbje removing log files that might be needed assembling a checkpoint made at sometime in the past, when running expert mode checkpointing, we configure bdbje not to delete logs when its finished with them; instead, bdbje gives logs its no longer using a `.del` suffix. Assembling a checkpoint will often require renaming files with the `.del` suffix so they have the `.jdb` suffix in accordance with the `bdbj-logs-manifest.txt` list (See below for more on this).

Note

With this expert mode enabled, the crawler `crawl-order/state-path` "state" directory will grow without bound; a process external to the crawler can be set to prune the state directory of `.del` files referenced by checkpoints since superseded).

To enable the no-files copy checkpoint, set the new expert mode setting `checkpoint-copy-bdbje-logs` to false.

To recover using a checkpoint that has all but the bdbje log files present, you will need to copy all logs listed in `bdbj-logs-manifest.txt` to the `bdbje-logs` checkpoint subdirectory. In some cases this will necessitate renaming logs with the `.del` to instead have the `.jdb` ending as suggested above. One thing to watch for is copying too many logs into the bdbje logs subdirectory. The list of logs must match exactly whats in the manifest file. Otherwise, the recovery will fail (For example, see [1325961] `resurrectOneQueueState` has keys for items not in allqueues [http://sourceforge.net/tracker/index.php?func=detail&aid=1325961&group_id=73833&atid=539099]).

On checkpoint recovery, Heritrix copies bdbje log files from the referenced checkpoint `bdb-logs` subdirectory to the new crawl's `crawl-order/state-path` "state" directory. As noted above, this can

take some time. Of note, if a bdbje log file already exists in the new crawls' `crawl-order/state-path "state"` directory, checkpoint recover will not overwrite the existing bdbje log file. Exploit this property and save on recovery time by using native unix `cp` to manually copy over bdbje log files from the checkpoint directory to the new crawls' `crawl-order/state-path "state"` directory before launching a recovery (Or, at the extreme, though it will trash your checkpoint, set the checkpoint's `bdb-logs` subdirectory as the new crawls `crawl-order/state-path "state"` directory).

9.4.2. Automated Checkpointing

To have Heritrix run a checkpoint on a period, uncomment (or add) to `heritrix.properties` a line like:

```
org.archive.crawler.framework.Checkpointer.period = 2
```

This will install a Timer Thread that will run on an interval (Units are in hours). See `heritrix_out.log` to see log of installation of the timer thread that will run the checkpoint on a period and to see log of everytime it runs (Assuming `org.archive.crawler.framework.Checkpointer.level` is set to INFO).

9.5. Remote Monitoring and Control

As of release 1.4.0, Heritrix will start up the JVM's JMX Agent if deployed in a SUN 1.5.0 JVM. It password protects the JMX Agent using whatever was specified as the Heritrix admin password so to login, you'll use 'monitorRole' or 'controlRole' for login and the Heritrix admin password as password. By default, the JMX Agent is started up on port 8849 (To change any of the JMX settings, set the `JMX_OPTS` environment variable).

On startup, Heritrix looks if any JMX Agent running in current context and registers itself with the first JMX Agent found publishing attributes and operations that can be run remotely. If running in a SUN 1.5.0 JVM where the JVM JMX Agent has been started, Heritrix will attach to the JVM JMX Agent (If running inside JBOSS, Heritrix will register with the JBOSS JMX Agent).

To see what Attributes and Operations are available via JMX, use the SUN 1.5.0 JDK `jconsole` application -- its in `$JAVA_HOME/bin` -- or use Section 9.2.4, "cmdline-jmxclient".

To learn more about the SUN 1.5.0 JDK JMX managements and `jconsole`, see Monitoring and Management Using JMX [<http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html>]. This O'Reilly article is also a good place for getting started : Monitoring Local and Remote Applications Using JMX 1.2 and JConsole [<http://www.onjava.com/pub/a/onjava/2004/09/29/tigerjmx.html>].

9.6. Experimental FTP Support

As of release 1.10.0, Heritrix has experimental support for crawling FTP servers. To enable FTP support for your crawls, there is a configuration file change you will have to manually make.

Specifically, you will have to edit the `$HERITRIX_HOME/conf/heritrix.properties` file. Remove `ftp` from the `org.archive.net.UURIFactory.ignored-schemes` property list. Also, you must add `ftp` to the `org.archive.net.UURIFactory.schemes` property list.

After that change, you should be able to add the FetchFTP processor to your crawl using the Web UI. Just create a new job, click "Modules", and add FetchFTP under "Fetchers."

Note that FetchFTP is a little unusual in that it works both as a fetcher and as an extractor. If an FTP URI refers to a directory, and if FetchFTP's `extract-from-dirs` property is set to true, then FetchFTP will extract one link for every line of the directory listing. Similarly, if the `extract-parent` property is true, then FetchFTP will extract the parent directory from every FTP URI it encounters.

Also, remember that FetchFTP is experimental. As of 1.10, FetchFTP has the following known limitations:

1. FetchFTP can only store directories if the FTP server supports the `NLIST` command. Some older systems may not support `NLIST`.
2. Similarly, FetchFTP uses passive mode transfer, to work behind firewalls. Not all FTP servers support passive mode, however.
3. Heritrix currently has no means of determining the mime-type of a document unless an HTTP server explicitly mentions one. Since FTP has no concept of metadata, all documents retrieved using FetchFTP have a mime-type of `no-type`.
4. In the absence of a mime-type, many of the postprocessors will not work. For instance, `HTMLExtractor` will not extract links from an HTML file fetched with FetchFTP.

Still, FetchFTP can be used to archive an FTP directory of tarballs, for instance. If you discover any additional problems using FetchFTP, please inform the [<archive-crawler@yahoogroups.com>](mailto:archive-crawler@yahoogroups.com) mailing list.

9.7. Duplication Reduction Processors

Starting in release 1.12.0, a number of Processors can cooperate to carry forward URI content history information between crawls, reducing the amount of duplicate material downloaded or stored in later crawls. For more information, see the project wiki's notes on using the new duplication-reduction functionality [<http://webteam.archive.org/confluence/display/Heritrix/Feature+Notes++1.12.0>].

A. Common Heritrix Use Cases

There are many different ways you may perform a web crawl. Here we have listed several use cases which will allow you to become familiar with some of Heritrix's more frequently used crawling parameters.

A.1. Avoiding Too Much Dynamic Content

Suppose you want to crawl only pages from a particular host (<http://www.foo.org/>), and you want to avoid crawling too many pages of the dynamically generated calendar. Let's say the calendar is accessed by passing a year, month and day to the calendar directory, as in <http://www.foo.org/calendar?year=2006&month=3&day=12>.

When you first create the job for this crawl, you will specify a single seed URI: <http://www.foo.org/>. By default, your new crawl job will use the `DecidingScope`, which will contain a default set of `DecideRules`. One of the default rules is the `SurtPrefixedDecideRule`, which tells Heritrix to accept any URIs that match our seed URI's SURT prefix, [http://\(org,foo,www,\)/](http://(org,foo,www,)/). Subsequently, if the URI <http://foo.org/> is encountered, it will be rejected since its SURT prefix [http://\(org,foo,\)/](http://(org,foo,)/) does not match the seed's SURT prefix. To allow both foo.org and www.foo.org, you could use the two seeds <http://foo.org/> and <http://www.foo.org/>. To allow every subdomain of foo.org, you could use the seed <http://foo.org> (note the absence of a trailing slash).

You will need to delete the `TransclusionDecideRule` since this rule has the potential to lead Heritrix onto another host. For example, if a URI returned a 301 (moved permanently) or 302 (found) response code and a URI with a different host name, Heritrix would accept this URI using the `TransclusionDecideR-`

ule. Removing this rule will keep Heritrix from straying off of our `www.foo.org` host.

A few of the rules like `PathologicalPathDecideRule` and `TooManyPathSegmentsDecideRule` will allow Heritrix to avoid some types of crawler traps. The `TooManyHopsDecideRule` will keep Heritrix from following too many links away from the seed so the calendar doesn't trap Heritrix in an infinite loop. By default, the hop path is set to 15, but you can change that on the Settings screen.

Alternatively, you may add the `MatchesFilePathDecideRule`. Set `use-preset-pattern` to `CUSTOM` and set `regex` to something like:

```
. *foo\.org(?:/calendar) .* | . *foo\.org/calendar/?year=200[56] . *
```

Finally, you'll need to set the `user-agent` and `from` fields on the Settings screen, and then you may submit the job and monitor the crawl.

A.2. Only Store Successful HTML Pages

Suppose you wanted to only grab the first 50 pages encountered from a set of seeds and archive only those pages that return a 200 response code and have the `text/html` MIME type. Additionally, you only want to look for links in HTML resources.

When you create your job, use the `DecidingScope` with the default set of `DecideRules`.

In order to examine HTML documents only for links, you will need to remove the following extractors that tell Heritrix to look for links in style sheets, JavaScript, and Flash files:

1. `ExtractorCSS`
2. `ExtractorJS`
3. `ExtractorSWF`

You should leave in the `ExtractorHTTP` since it is useful in locating resources that can only be found using a redirect (301 or 302).

You can limit the number of files to download by setting `max-document-download` on the Settings screen. Setting this value to 50 will probably not have the results you intend. Since each DNS response and `robots.txt` file is counted in this number, you'll likely want to use the value of `50 * number of seeds * 2`.

Next, you will need to add filters to the `ARCWriterProcessor` so that it only records documents with a 200 status code and a mime-type of `text/html`. The first filter to add is the `ContentTypeRegExpFilter`; set its `regex` setting to `text/html . *`. Next, add a `DecidingFilter` to the `ARCWriterProcessor`, then add `FetchStatusDecideRule` to the `DecidingFilter`.

You'll probably want to apply the above filters to the `mid-fetch-filters` setting of `FetchHTTP` as well. That will prevent `FetchHTTP` from downloading the content of any non-html or non-successful documents.

Once you have entered the desired settings, start the job and monitor the crawl.

A.3. Mirroring .html Files Only

Suppose you only want to crawl URLs that match `http://foo.org/bar/*.html`, and you'd like to save the crawled files in a file/directory format instead of saving them in ARC files. Suppose you also know that you are crawling a web server that is case sensitive (`http://foo.org/bar/abc.html` and `http://foo.org/bar/ABC.HTML` are pointing to two different resources).

You would first need to create a job with the single seed `http://foo.org/bar/`. You'll need to add the `MirrorWriterProcessor` on the Modules screen and delete the `ARCWriterProcessor`. This will store your files in a directory structure that matches the crawled URIs, and the files will be stored in the crawl job's `mirror` directory.

Your job should use the `DecidingScope` with the following set of `DecideRules`:

1. `RejectDecideRule`
2. `SurtPrefixedDecideRule`
3. `TooManyHopsDecideRule`
4. `PathologicalPathDecideRule`
5. `TooManyPathSegmentsDecideRule`
6. `NotMatchesFilePatternDecideRule`
7. `PrerequisiteAcceptDecideRule`

We are using the `NotMatchesFilePatternDecideRule` so we can eliminate crawling any URIs that don't end with `.html`. It's important that this `DecideRule` be placed immediately before `PrerequisiteAcceptDecideRule`; otherwise the DNS and robots.txt prerequisites will be rejected since they won't match the regexp.

On the Setting screen, you'll want to set the following for the `NotMatchesFilePatternDecideRule`:

1. decision: REJECT
2. use-preset-pattern: CUSTOM
3. regexp: `.*(\/\..html)$`

Note that the regexp will accept URIs that end with `/` as well as `.html`. If we don't accept the `/`, the seed URI will be rejected. This also allows us to accept URIs like `http://foo.org/bar/dir/` which are likely pointing to `index.html`. A stricter regexp would be `.*\.html$`, but you'll need to change your seed URI if you use it. One thing to be aware of: if Heritrix encounters the URI `http://foo.org/bar/dir` where `dir` is a directory, the URI will be rejected since it is missing the terminating slash.

Finally you'll need to allow Heritrix to differentiate between `abc.html` and `ABC.HTML`. Do this by removing the `LowercaseRule` under `uri-canonicalization-rules` on the Submodules screen.

Once you have entered the desired settings, start the job and monitor the crawl.

Glossary

Some definitions

Bytes, KB and statistics

Heritrix adheres to the following conventions for displaying byte and bit amounts:

Legend Type
B Bytes
KB Kilobytes - 1 KB = 1024 B
MB Megabytes - 1 MB = 1024 KB
GB Gigabytes - 1 GB = 1024 MB

b bits
Kb Kilobits - 1 Kb = 1000 b
Mb Megabits - 1 Mb = 1000 Kb
Gb Gigabits - 1 Gb = 1000 Mb

This also applies to all logs.

Checkpointing

Heritrix checkpointing has been heavily influenced by what Mercator provided. In one of the papers on Mercator [<http://citeseer.nj.nec.com/najork01highperformance.html>] it is described this way: "Checkpointing is an important part of any long-running process such as a web crawl. By checkpointing we mean writing a representation of the crawler's state to stable storage that, in the event of a failure, is sufficient to allow the crawler to recover its state by reading the checkpoint and to resume crawling from the exact state it was in at the time of the checkpoint. By this definition, in the event of a failure, any work performed after the most recent checkpoint is lost, but none of the work up to the most recent checkpoint. In Mercator, the frequency with which the background thread performs a checkpoint is user-configurable; we typically checkpoint anywhere from 1 to 4 times per day."

See Section 9.4, "Checkpointing" for discussion of the Heritrix implementation.

CrawlURI

A URI and its associated data such as parent URI, number of links from seed etc.

Dates and times

All times in Heritrix are GMT assuming the clock and timezone on the local system are correct.

This means that all dates/times in logs are GMT, all dates and times shown in the WUI are GMT and any times or dates entered by the user need to be in GMT.

Discovered URIs

That is any URI that has been confirmed be within 'scope'. This includes those that have been processed, are being processed and have finished processing. Does not include URIs that have been 'forgotten' (deemed out of scope when trying to fetch, most likely due to operator changing scope definition).

Note

This only counts discovered URIs. Since the same URI can (at least in most frontiers) be fetched multiple times, this number may be somewhat lower then the combined queued, in process and finished items combined due to duplicate URIs being queued and processed. This variance is likely to be especially high in Frontiers implementing 'revisit' strategies.

Discovery path

Each URI has a discovery path. The path contains one character for

each link or embed followed from the seed.

The character legend is as follows.

```
R - Redirect
E - Embed
X - Speculative embed (aggressive/Javascript link extrac
L - Link
P - Prerequisite (as for DNS or robots.txt before anothe
```

The discovery path of seeds is an empty string.

Frontier	A Frontier is a pluggable module in Heritrix that maintains the internal state of the crawl. See Section 6.1.2, “Frontier”.
"Holding Jobs" vs. "Crawling Jobs"	<p>The mode <i>Crawling Jobs</i> generally means that the crawler will start executing a job as soon as one is made available in the pending jobs queue (as long as there is not a job already running).</p> <p>If the crawler is in the <i>Holding Jobs</i> mode, jobs added to the pending jobs queue will be held; they will not be started, even if there are no jobs currently being run.</p>
Host	<p>A host can serve multiple domains or a single domain can be served by multiple hosts. For our purposes so far, host == hostname in URI. DNS is not considered; it is volatile and may be unavailable. So when Heritrix gets the URIs...</p> <pre>http://www.example.com http://search.example.com http://201.199.7.15</pre> <p>...even if they all point to the 201.199.7.15 IP, they are 3 different logical hosts (at the level of the URI/HTTP protocol).</p> <p>Conformant HTTP proxies behave similarly, we think, even if they know <code>www.example.com == 201.199.7.15</code>, they will not consider them interchangeable.</p> <p>This is not ideal for politeness where we'd want politeness rules to apply to the physical host rather than the logical.</p>
Link hop count	<p>Number of link follow from the seed to the current URI. Seeds have a link hop count of 0.</p> <p>This number is equal to counting the 'L's in a URIs discovery path.</p>
Pending URIs	<p>Number of URIs that are awaiting detailed processing.</p> <p>Number of discovered URIs that have not been inspected for scope or duplicates. Depending on the implementation of the Frontier this might always be zero. It may also be an adjusted number that tries to account for duplicates by estimation.</p>
Politeness	Politeness refers to attempts by the crawler software to limit load on a site. Without politeness restrictions the crawler might otherwise overwhelm smaller sites and even cause moderately sized sites to slow down significantly.

Unless you have express permission to crawl a site aggressively you should apply strict politeness rules to any crawl.

Queued URIs

Number of URIs queued up and waiting for processing.

This includes any URIs that failed but will be retried. Basically this is any discovered URI that has not either been processed or is being processed.

Regular expressions

All regular expressions used by Heritrix are Java regular expressions.

Java regular expressions differ from those used in Perl, for example, in several ways. For detailed info on Java regular expressions see the Java API for `java.util.regex.Pattern` on Sun's home page (java.sun.com [<http://java.sun.com>]).

For API of Java SE v1.4.2 see <http://java.sun.com/j2se/1.4.2/docs/api/index.html>. It is recommended you lookup the API for the version of Java that is being used to run Heritrix.

Server

A server is a service on a Host. There might be more than one service on a host differentiated by port number.

Status codes

Each crawled URI gets a status code. This code (or number) is an indication of what happened when Heritrix tried to fetch the URI.

Codes ranging from 200 to 599 are standard HTTP response codes and information about their meanings is available at the World Wide Web consortium's web page [<http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>].

Other status codes used by Heritrix (From `org.archive.crawler.datamodel.FetchStatusCodes` [<http://crawler.archive.org/xref/org/archive/crawler/datamodel/FetchStatusCodes.html#38>]):

Code Meaning

- 1 Successful DNS lookup
- 0 Fetch never tried (perhaps protocol unsupported or
- 1 DNS lookup failed
- 2 HTTP connect failed
- 3 HTTP connect broken
- 4 HTTP timeout (before any meaningful response received)
- 5 Unexpected runtime exception; see `runtime-errors.1`
- 6 Prerequisite domain-lookup failed, precluding fetch
- 7 URI recognized as unsupported or illegal
- 8 Multiple retries all failed, retry limit reached
- 50 Temporary status assigned URIs awaiting precondition logs may be a bug
- 60 Failure status assigned URIs which could not be queued by Frontier (and may in fact be unfetchable)
- 61 Prerequisite robots.txt-fetch failed, precluding a
- 62 Some other prerequisite failed, precluding a fetch
- 63 A prerequisite (of any type) could not be scheduled for fetch attempt
- 3000 Severe Java 'Error' conditions (`OutOfMemoryError`, etc.) during URI processing.
- 4000 'chaff' detection of traps/content of negligible value

```
-4001 Too many link hops away from seed
-4002 Too many embed/transitive hops away from last URI
-5000 Out of scope upon reexamination (only happens if s
      crawl)
-5001 Blocked from fetch by user setting
-5002 Blocked by a custom processor
-5003 Blocked due to exceeding an established quota
-5004 Blocked due to exceeding an established runtime
-6000 Deleted from Frontier by user
-7000 Processing thread was killed by the operator (perh
      hung condition)
-9998 Robots.txt rules precluded fetch
```

Note

Codes and explanations are also available under the Help link in the web UI.

Please note that status codes defined by Heritrix may be subject to change between versions, especially new codes may be added to tackle a wider array of situations.

SURT

SURT stands for Sort-friendly URI Reordering Transform, and is a transformation applied to URIs which makes their left-to-right representation better match the natural hierarchy of domain names.

A URI `<scheme://domain.tld/path?query>` has SURT form `<scheme://(tld,domain,)/path?query>`.

Conversion to SURT form also involves making all characters lowercase, and changing the 'https' scheme to 'http'. Further, the '/' after a URI authority component -- for example, the third slash in a regular HTTP URI -- will only appear in the SURT form if it appeared in the plain URI form. (This convention proves important when using real URIs as a shorthand for SURT prefixes, as described below.)

SURT form URIs are typically not used to specify exact URIs for fetching. Rather, SURT form is useful when comparing or sorting URIs. For example, URIs in SURT format sort into natural groups -- all 'archive.org' URIs will be adjacent, regardless of what subdomains like 'books.archive.org' or 'movies.archive.org' are used.

Most importantly, a SURT form URI, or a truncated version of a SURT form URI, can be used as a SURT prefix. A SURT prefix will often correspond to all URIs within a common 'area' of interest for crawling. For example, the prefix `<http://(is,>` will be shared by all URIs in the '.is' top-level domain.

SURT prefix

A URI in SURT form, especially if truncated, may be of use as a "SURT prefix", a shared prefix string of all SURT form URIs in the same 'area' of interest for web crawling.

For example, the prefix `<http://(is,>` will be shared by all SURT form URIs in the '.is' top-level domain. The prefix `<http://(org.archive,www,)/movies>` (which is also a valid full SURT form URI) will be shared by all URIs at `www.archive.org` with a path beginning '/movies'.

A collection of sorted SURT prefixes is an efficient way to specify a desired crawl scope: any URI whose SURT form starts with any of the prefixes should be included.

A small set of conventions can be also be used to calculate an "implied SURT prefix" from a regular URI, such as a URI supplied as a crawl seed. These conventions are:

1. Convert the URI to its SURT form.
2. If there are at least 3 slashes ('/') in the SURT form, remove everything after the last slash. As examples, `<http://(org,example,www,)/main/subsection/>` is unchanged; `<http://(org,example,www,)/main/subsection>` is truncated to `<http://(org,example,www,)/main/>`; `<http://(org,example,www,)/>` is unchanged; and `<http://(org,example,www,)>` is unchanged.
3. If the resulting form ends in an off-parenthesis (')'), remove the off-parenthesis. So each of the above examples except for the last is unchanged, while the last `<http://(org,example,www,)>` becomes `<http://(org,example,www,>`.

This allows many seed URIs, in their usual form, to imply the most useful SURT prefixes for crawling related URIs -- with the presence or absence of a trailing '/' on URIs without further path-info being a subtle indicator as to whether subdomains of the supplied domain should be included.

For example, seed `<http://www.archive.org/>` will become SURT form and implied SURT prefix `<http://(org,archive,www,)/>`, and is the prefix of all SURT form URIs on `www.archive.org`. However, any subdomain URI like `<http://homepages.www.archive.org/directory>` would be ruled out, because its SURT form `<http://(org,archive,www,homepages,)/directory>` does not begin with the full SURT prefix, including the ')', deduced from the seed.

In contrast, seed `<http://www.archive.org>` (note the lack of trailing slash) will become SURT form `<http://(org,archive,www,)>`, and implied SURT prefix `<http://(org,archive,www,>` (note the lack of trailing ')'). This will be the prefix of all URIs on `www.archive.org`, as well as any subdomain URIs like `<http://homepages.www.archive.org/directory>`, because the full SURT prefix appears in subdomain URI SURT forms.

Toe Threads

When crawling Heritrix employs a configurable number of Toe Threads to process each URI.

Each of these threads will request a URI from the Frontier (Section 6.1.2, "Frontier"), apply each of the set Processors (Section 6.1.3, "Processing Chains") to it and finally report it as completed back to the Frontier.