Harvest User's Manual

Darren R. Hardy, Michael F. Schwartz, Duane Wessels, Kang-Jin Lee

2002-10-29

Harvest User's Manual was edited by Kang-Jin Lee and covers Harvest version 1.8. It was originally written by Darren R. Hardy, Michael F. Schwartz and Duane Wessels for Harvest 1.4.pl2 in 1996-01-31.

Contents

Chapter 1

Introduction to Harvest

HARVEST is an integrated set of tools to gather, extract, organize, and search information across the Internet. With modest effort users can tailor Harvest to digest information in many different formats, and offer custom search services on the Internet.

A key goal of Harvest is to provide a flexible system that can be configured in various ways to create many types of indexes.

Harvest also allows users to extract structured (attribute-value pair) information from many different information formats and build indexes that allow these attributes to be referenced during queries (e.g., searching for all documents with a certain regular expression in the title field).

An important advantage of Harvest is that it allows users to build indexes using either manually constructed templates (for maximum control over index content) or automatically extracted data constructed templates (for easy coverage of large collections), or using a hybrid of the two methods.

Harvest is designed to make it easy to distribute the search system on a pool of networked machines to handle higher load.

1.1 Copyright

The core of Harvest is licensed under *GPL* `<../../COPYING>`. Additional components distributed with Harvest are also under GPL or similar license. Glimpse, the current default fulltext indexer has a different license. Here is a clarification of *Glimpse' copyright status* `<../glimpse-license-status>` kindly posted by *Golda Velez* `<mailto:gvelez@tucson.com>` to *comp.infosystems.harvest* `<news: comp.infosystems.harvest>`.

1.2 Online Harvest Resources

This manual is available at *harvest.sourceforge.net/harvest/doc/html/manual.html*.

More information about Harvest is available at *harvest.sourceforge.net*.

Chapter 2

Subsystem Overview

Harvest consists of several subsystems. The *Gatherer* subsystem collects indexing information (such as keywords, author names, and titles) from the resources available at *Provider* sites (such as FTP and HTTP servers). The *Broker* subsystem retrieves indexing information from one or more Gatherers, suppresses duplicate information, incrementally indexes the collected information, and provides a WWW query interface to it.

You should start using Harvest simply, by installing a single "stock" (i.e., not customized) Gatherer and Broker on one machine to index some of the FTP, World Wide Web, and NetNews data at your site.

After you get the system working in this basic configuration, you can invest additional effort as warranted. First, as you scale up to index larger volumes of information, you can reduce the CPU and network load to index your data by distributing the gathering process. Second, you can customize how Harvest extracts, indexes, and searches your information, to better match the types of data you have and the ways your users would like to interact with the data.

We discuss how to distribute the gathering process in the next subsection. We cover various forms of customization in Section 4.5.3 (Customizing the type recognition, candidate selection, presentation unnesting, and summarizing steps) and in several parts of Section 5 (The Broker).

2.1 Distributing the Gathering and Brokering Processes

Harvest Gatherers and Brokers can be configured in various ways. Running a Gatherer remotely from a Provider site allows Harvest to interoperate with sites that are not running Harvest Gatherers, by using standard object retrieval protocols like FTP, Gopher, HTTP, and NNTP. However, as suggested by the bold lines in the left side of Figure 2.1 (2), this arrangement results in excess server and network load. Running a Gatherer locally is much more efficient, as shown in the right side of Figure 2.1 (2). Nonetheless, running a Gatherer remotely is still better than having many sites independently collect indexing information, since many Brokers or other search services can share the indexing information that the Gatherer collects.

If you have a number of FTP/HTTP/Gopher/NNTP servers at your site, it is most efficient to run a Gatherer on each machine where these servers run. On the other hand, you can reduce installation effort by running a Gatherer at just one machine at your site and letting it retrieve data from across the network.

Figure 2.1 (2) also illustrates that a Broker can collect information from many Gatherers (to build an index of widely distributed information). Brokers can also retrieve information from other Brokers,

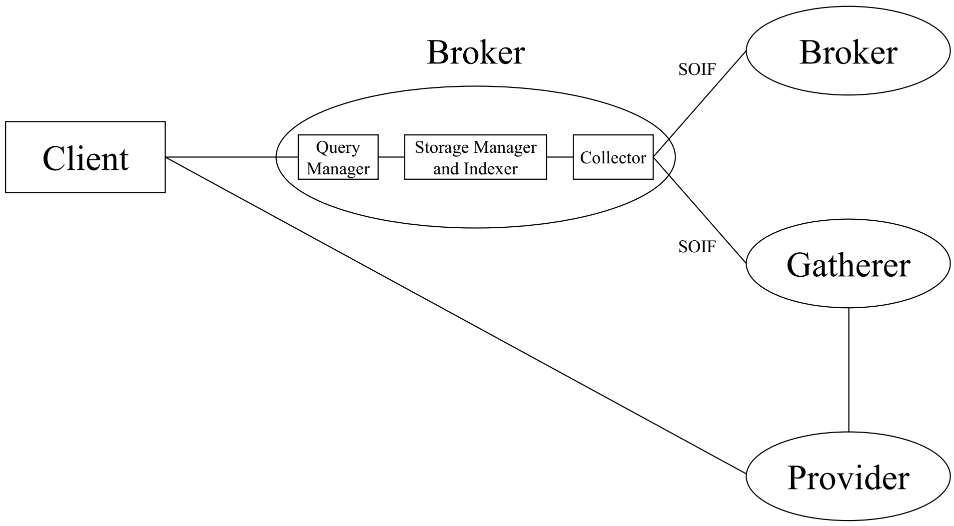

Figure 2.1: Harvest Software Components

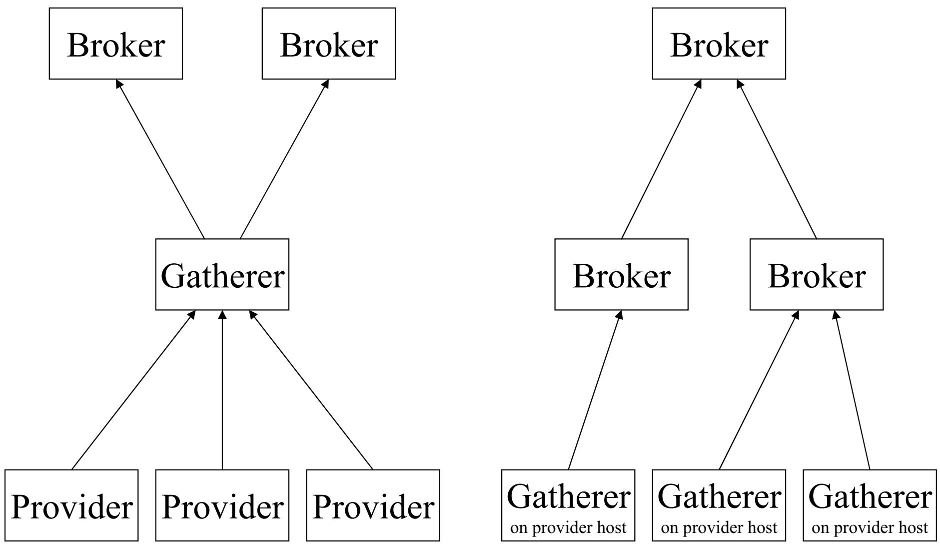

Figure 2.2: Harvest Configuration Options

in effect cascading indexed views from one another. Brokers retrieve this information using the query interface, allowing them to filter or refine the information from one Broker to the next.

Chapter 3

Installing the Harvest Software

3.1 Requirements for Harvest Servers

3.1.1 Hardware

A good machine for running a typical Harvest server will have a reasonably fast processor, 1-2 GB of free disk, and 128 MB of RAM. A slower CPU will work but it will slow down the Harvest server. More important than CPU speed, however, is memory size. Harvest uses a number of processes, some of which provide needed "plumbing" (e.g., `search.cgi`), and some of which improve performance (e.g., the `glimpseserver` process). If you do not have enough memory, your system will page too much, and drastically reduce performance. The other factor affecting RAM usage is how much data you are trying to index in a Harvest Broker. The more data, the more disk I/O will be performed at query time, the more RAM it will take to provide a reasonable sized disk buffer pool.

The amount of disk you'll need depends on how much data you want to index in a single Broker. (It is possible to distribute your index over multiple Brokers if it gets too large for one disk.) A good rule of thumb is that you will need about 10% as much disk to hold the Gatherer and Broker databases as the total size of the data you want to index. The actual space needs will vary depending on the type of data you are indexing. For example, PostScript achieves a much higher indexing space reduction than HTML, because so much of the PostScript data (such as page positioning information) is discarded when building the index.

3.1.2 Platforms

To run a Harvest server, you need an UNIX-like Operating System.

3.1.3 Software

To use Harvest, you need the following software packages:

- All Harvest servers require: Perl v5.0 or higher.
- The Harvest Broker and Gatherer require: GNU `gzip` v1.2.4 or higher.
- The Harvest Broker requires: HTTP server.

To build Harvest from the source distribution you may need to install one or more of the following software packages:

- Compiling Harvest requires: GNU `gcc` v2.5.8 or higher.
- Compiling the Harvest Broker requires: `flex` v2.4.7 or higher and `bison` v1.22 or higher.

The sources for `gcc`, `gzip`, `flex`, and `bison` are available at the *GNU FTP server* `<ftp://ftp.gnu.org/>`.

3.2 Requirements for Harvest Users

Anyone with a web browser (e.g., Internet Explorer, Lynx, Mozilla, Netscape, Opera, etc.) can access and use Harvest servers.

3.3 Retrieving and Installing the Harvest Software

3.3.1 Distribution types

Currently we offer only source distribution of Harvest. The *source distribution* contains all of the source code for the Harvest software. There are no *binary distributions* of Harvest.

You can retrieve the Harvest source distributions from the Harvest download site *prdownloads.sourceforge.net/harvest/*.

3.3.2 Harvest components

Harvest components are in the *components* directory. To use a component, follow the instructions included in the desired component directory.

3.3.3 User-contributed software

There is a collection of unsupported user-contributed software in *contrib* directory. If you would like to contribute some software, please send email to *lee@arco.de* `<mailto:lee@arco.de>`.

3.4 Building the Source Distribution

The source distribution can be extracted in any directory. The following command will extract the gnu-zipped source archive:

```
% gzip -dc harvest-x.y.z.tar.gz | tar xf -
```

For archives compressed with bzip2, use:

```
% bzip2 -dc harvest-x.y.z.tar.bz2 | tar xf -
```

Harvest uses GNU's *autoconf* package to perform needed configuration at installation time. If you want to override the default installation location of */usr/local/harvest*, change the "prefix" variable when invoking "configure". If desired, you may edit *src/common/include/config.h* before compiling to change various Harvest compile-time limits and variables. To compile the source tree type `make`.

For example, to build and install the entire Harvest system into */usr/local/harvest* directory, type:

```
% ./configure
% make
% make install
```

You may see some compiler warning messages, which you can ignore.

Building the entire Harvest distribution will take few minutes on a reasonably fast machine. The compiled source tree takes approximately 25 megabytes of disk space.

Later, after the installed software working, you can remove the compiled code (".o" files) and other intermediate files by typing `make clean`. If you want to remove the configure-generated Makefiles, type `make distclean`.

3.5 Additional installation for the Harvest Broker

3.5.1 Checking the installation for HTTP access

The Broker interacts with your HTTP server in a number of ways. You should make sure that the HTTP server can properly access the files it needs. In many cases, the HTTP server will run under a different userid than the owner of the Harvest files.

First, make sure the HTTP server userid can read the *query.html* files in each broker directory. Second, make sure the HTTP server userid can access and execute the CGI programs in *$HARVEST_HOME/cgi-bin/*. The `search.cgi` script reads files from the *$HARVEST_HOME/cgi-bin/lib/* directory, so check that as well. Finally, check the files in *$HARVEST_HOME/lib/*. Some of the CGI Perl scripts require "include" files in this directory.

3.5.2 Required modifications to your HTTP server

The Harvest Broker requires that an HTTP server is running, and that the HTTP server "knows" about the Broker's files. Below are some examples of how to configure various HTTP servers to work with the Harvest Broker.

3.5.3 Apache httpd

Requires a **ScriptAlias** and an **Alias** entry in *httpd.conf*, e.g.:

```
ScriptAlias /Harvest/cgi-bin/ Your-HARVEST_HOME/cgi-bin/
Alias /Harvest/ Your-HARVEST_HOME/
```

WARNING: The **ScriptAlias** entry must appear *before* the **Alias** entry.

Additionally, it might be necessary to configure Apache httpd to follow *symbolic links*. To do this, add following to your *httpd.conf*:

```
<Directory Your-HARVEST_HOME>
        Options FollowSymLinks
</Directory>
```

3.5.4 Other HTTP servers

Install the HTTP server and modify its configuration file so that the */Harvest* directory points to *$HARVEST_HOME*. You will also need to configure your HTTP server so that it knows that the directory */Harvest/cgi-bin* contains valid CGI programs. If the default behaviour of your HTTP server is not to follow symbolik links, you will need to configure it so that it will follow symbolic links in the */Harvest* directory.

3.6 Upgrading versions of the Harvest software

3.6.1 Upgrading from version 1.6 to version 1.8

You *can not* install version 1.8 on top of version 1.6. For example, the change from version 1.6 to version 1.8 included some reorganization of the executables, and hence simply installing version 1.8 on top of version 1.6 would cause you to use old executables in some cases.

To upgrade from Harvest version 1.6 to 1.8, do:

1. Move your old installation to a temporary location.
2. Install the new version as directed by the release notes.
3. Then, for each Gatherer and Broker that you were running under the old installation, migrate the server into the new installation.

   Gatherers:
   you need to move the Gatherer's directory into *$HARVEST_HOME/gatherers*. Section 4.3 (RootNode specifications) describes the Gatherer workload specifications if you want to modify your Gatherer's configuration file.

   Brokers:
   rebuild your broker by using `CreateBroker` and merge in any customizations you have made to your old Broker.

3.6.2 Upgrading from version 1.5 to version 1.6

There are no known incompatibilities between versions 1.5 and 1.6.

3.6.3 Upgrading from version 1.4 to version 1.5

You *can not* install version 1.5 on top of version 1.4. For example, the change from version 1.4 to version 1.5 included some reorganization of the executables, and hence simply installing version 1.5 on top of version 1.4 would cause you to use old executables in some cases.

To upgrade from Harvest version 1.4 to 1.5, do:

1. Move your old installation to a temporary location.

2. Install the new version as directed by the release notes.

3. Then, for each Gatherer and Broker that you were running under the old installation, migrate the server into the new installation.

   Gatherers:

   you need to move the Gatherer's directory into *$HARVEST_HOME/gatherers*. Section 4.3 (RootNode specifications) describes the Gatherer workload specifications if you want to modify your Gatherer's configuration file.

   Brokers:

   you need to move the Broker's directory into *$HARVEST_HOME/brokers*. Remove any *.glimpse_ ** files from your Broker's directory and use the *admin.html* interface to force a full-index. You may want, however, to rebuild your broker by using `CreateBroker` so that you can use the updated *query.html* and related files.

3.6.4 Upgrading from version 1.3 to version 1.4

There are no known incompatibilities between versions 1.3 and 1.4.

3.6.5 Upgrading from version 1.2 to version 1.3

Version 1.3 is mostly backwards compatible with 1.2, with the following exception:

Harvest 1.3 uses Glimpse 3.0. The *.glimpse_ ** files in the broker directory created with Harvest 1.2 (Glimpse 2.0) are incompatible. After installing Harvest 1.3 you should:

1. Shutdown any running brokers.
2. Execute `rm .glimpse_*` in each broker directory.
3. Restart your brokers with `RunBroker`.
4. Force a full-index from the *admin.html* interface.

3.6.6 Upgrading from version 1.1 to version 1.2

There are a few incompatabilities between Harvest version 1.1 and version 1.2.

- The Gatherer has improved incremental gatherering support which is incompatible with version 1.1. To update your existing Gatherer, change into the Gatherer's *Data-Directory* (usually the *data* subdirectory), and run the following command:

```
% set path = ($HARVEST_HOME/lib/gatherer $path)
% cd data
% rm -f INDEX.gdbm
% mkindex
```

  This should create the *INDEX.gdbm* and *MD5.gdbm* files in the current directory.

- The Broker has a new log format for the *admin/LOG* file which is incompatible with version 1.1.

3.6.7 Upgrading to version 1.1 from version 1.0 or older

If you already have an older version of Harvest installed, and want to upgrade, you *can not* unpack the new distribution on top of the old one. For example, the change from version 1.0 to version 1.1 included some reorganization of the executables, and hence simply installing version 1.1 on top of version 1.0 would cause you to use old executables in some cases.

On the other hand, you may not want to start over from scratch with a new software version, as that would not take advantage of the data you have already gathered and indexed. Instead, to upgrade from Harvest version 1.0 to 1.1, do the following:

1. Move your old installation to a temporary location.
2. Install the new version as directed by the release notes.
3. Then, for each Gatherer and Broker that you were running under the old installation, migrate the server into the new installation.

   Gatherers:
   you need to move the Gatherer's directory into *$HARVEST_HOME/gatherers*. Section 4.3 (RootNode specifications) describes the new Gatherer workload specifications which were introduced in version 1.1; you may modify your Gatherer's configuration file to employ this new functionality.

   Brokers:
   you need to move the Broker's directory into *$HARVEST_HOME/brokers*. You may want, however, to rebuild your broker by using `CreateBroker` so that you can use the updated *query.html* and related files.

3.7 Starting up the system: RunHarvest and related commands

The simplest way to start the Harvest system is to use the `RunHarvest` command. `RunHarvest` prompts the user with a short list of questions about what data to index, etc., and then creates and runs a Gatherer and Broker with a "stock" (non-customized) set of content extraction and indexing mechanisms. Some more primitive commands are also available, for starting individual Gatherers and Brokers (e.g., if you want to distribute the gathering process). The Harvest startup commands are:

RunHarvest

Checks that the Harvest software is installed correctly, prompts the user for basic configuration information, and then creates and runs a Gatherer and a Broker. If you have *$HARVEST_HOME* set, then it will use it; otherwise, it tries to determine *$HARVEST_HOME* automatically. Found in the *$HARVEST_HOME* directory.

RunBroker

Runs a Broker. Found in the Broker's directory.

RunGatherer

Runs a Gatherer. Found in the Gatherer's directory.

CreateBroker

Creates a single Broker which will collect its information from other existing Brokers or Gatherers. Used by `RunHarvest`, or can be run by a user to create a new Broker. Uses *$HARVEST_HOME*, and defaults to */usr/local/harvest*. Found in the *$HARVEST_HOME/bin* directory.

There is no `CreateGatherer` command, but the `RunHarvest` command can create a Gatherer, or you can create a Gatherer manually (see Section 4.5.3 (Customizing the type recognition, candidate selection, presentation unnesting, and summarizing steps) or Section 8 (Gatherer Examples)). The layout of the installed Harvest directories and programs is discussed in Section 6 (Programs and layout of the installed Harvest software).

Among other things, the `RunHarvest` command asks the user what port numbers to use when running the Gatherer and the Broker. By default, the Gatherer will use port 8500 and the Broker will use the Gatherer port plus 1. The choice of port numbers depends on your particular machine – you need to choose ports that are not in use by other servers on your machine. You might look at your */etc/services* file to see what ports are in use (although this file only lists some servers; other servers use ports without registering that information anywhere). Usually the above port numbers will not be in use by other processes. Probably the easiest thing is simply to try using the default port numbers, and see if it works.

The remainder of this manual provides information for users who wish to customize or otherwise make more sophisticated use of Harvest than what happens when you install the system and run `RunHarvest`.

3.8 Harvest team contact information

If you have questions the about Harvest system or problems with the software, post a note to the USENET newsgroup *comp.infosystems.harvest* `<news:comp.infosystems.harvest>`. Please note your machine type, operating system type, and Harvest version number in your correspondence.

If you have bug fixes, ports to new platforms or other software improvements, please email them to the Harvest maintainer *lee@arco.de* `<mailto:lee@arco.de>`.

Chapter 4

The Gatherer

4.1 Overview

The Gatherer retrieves information resources using a variety of standard access methods (FTP, Gopher, HTTP, NNTP, and local files), and then summarizes those resources in various type-specific ways to generate structured indexing information. For example, a Gatherer can retrieve a technical report from an FTP archive, and then extract the author, title, and abstract from the paper to summarize the technical report. Harvest Brokers or other search services can then retrieve the indexing information from the Gatherer to use in a searchable index available via a WWW interface.

The Gatherer consists of a number of separate components. The `Gatherer` program reads a Gatherer configuration file and controls the overall process of enumerating and summarizing data objects.

The structured indexing information that the Gatherer collects is represented as a list of attribute-value pairs using the *Summary Object Interchange Format* (SOIF, see Section 7 (The Summary Object Interchange Format (SOIF))). The `gatherd` daemon serves the Gatherer database to Brokers. It hangs around, in the background, after a gathering session is complete. A stand-alone `gather` program is a client for the `gatherd` server. It can be used from the command line for testing, and is used by the Broker. The Gatherer uses a local disk cache to store objects it has retrieved. The disk cache is described in Section 4.7.6 (The local disk cache).

Even though the `gatherd` daemon remains in the background, a Gatherer does not automatically update or refresh its summary objects. Each object in a Gatherer has a Time-to-Live value. Objects remain in the database until they expire. See Section 4.7.5 (Periodic gathering and realtime updates) for more information on keeping Gatherer objects up to date.

Several example Gatherers are provided with the Harvest software distribution (see Section 8 (Gatherer Examples)).

4.2 Basic setup

To run a basic Gatherer, you need only list the Uniform Resource Locators (URLs, see *RFC1630* and *RFC1738*) from which it will gather indexing information. This list is specified in the Gatherer configuration file, along with other optional information such as the Gatherer's name and the directory in which it resides (see Section 4.7.1 (Setting variables in the Gatherer configuration file) for details on the optional information). Below is an example Gatherer configuration file:

```
#
```

```
#  sample.cf - Sample Gatherer Configuration File
#
Gatherer-Name:    My Sample Harvest Gatherer
Gatherer-Port:    8500
Top-Directory:    /usr/local/harvest/gatherers/sample

<RootNodes>
# Enter URLs for RootNodes here
http://www.mozilla.org/
http://www.xfree86.org/
</RootNodes>

<LeafNodes>
# Enter URLs for LeafNodes here
http://www.arco.de/~kj/index.html
</LeafNodes>
```

As shown in the example configuration file, you may classify an URL as a **RootNode** or a **LeafNode**. For a LeafNode URL, the Gatherer simply retrieves the URL and processes it. LeafNode URLs are typically files like PostScript papers or compressed “tar” distributions. For a RootNode URL, the Gatherer will expand it into zero or more LeafNode URLs by recursively enumerating it in an access method-specific way. For FTP or Gopher, the Gatherer will perform a recursive directory listing on the FTP or Gopher server to expand the RootNode (typically a directory name). For HTTP, a RootNode URL is expanded by following the embedded HTML links to other URLs. For News, the enumeration returns all the messages in the specified USENET newsgroup.

PLEASE BE CAREFUL when specifying RootNodes as it is possible to specify an enormous amount of work with a single RootNode URL. To help prevent a misconfigured Gatherer from abusing servers or running wildly, by default the Gatherer will only expand a RootNode into 250 LeafNodes, and will only include HTML links that point to documents that reside on the same server as the original RootNode URL. There are several options that allow you to change these limits and otherwise enhance the Gatherer specification. See Section 4.3 (RootNode specifications) for details.

The Gatherer is a *“robot”* and collects URLs starting from the URLs specified in RootNodes. It obeys the *robots.txt* convention and the *robots META tag*. It also is *HTTP Version 1.1* compliant and sends the *User-Agent* and *From* request fields to HTTP servers for accountability.

After you have written the Gatherer configuration file, create a directory for the Gatherer and copy the configuration file there. Then, run the `Gatherer` program with the configuration file as the only command-line argument, as shown below:

```
% Gatherer GathName.cf
```

The Gatherer will generate a database of the content summaries, a log file (*log.gatherer*), and an error log file (*log.errors*). It will also start the `gatherd` daemon which exports the indexing information automatically to Brokers and other clients. To view the exported indexing information, you can use the `gather` client program, as shown below:

```
% gather localhost 8500 | more
```

The **-info** option causes the Gatherer to respond only with the Gatherer summary information, which consists of the attributes available in the specified Gatherer's database, the Gatherer's host and name, the range of object update times, and the number of objects. Compression is the default, but can be disabled with the **-nocompress** option. The optional timestamp tells the Gatherer to send only the objects that have changed since the specified timestamp (in seconds since the UNIX "epoch" of January 1, 1970).

4.2.1 Gathering News URLs with NNTP

News URLs are somewhat different than the other access protocols because the URL generally does not contain a hostname. The Gatherer retrieves News URLs from an NNTP server. The name of this server must be placed in the environment variable *$NNTPSERVER*. It is probably a good idea to add this to your `RunGatherer` script. If the environment variable is not set, the Gatherer attempts to connect to a host named *news* at your site.

4.2.2 Cleaning out a Gatherer

Remember the Gatherer databases persists between runs. Objects remain in the databases until they expire. When experimenting with the gatherer, it is always a good idea to "clean out" the databases between runs. This is most easily accomplished by executing this command from the Gatherer directory:

```
% rm -rf data tmp log.*
```

4.3 RootNode specifications

The RootNode specification facility described in Section 4.2 (Basic setup) provides a basic set of default enumeration actions for RootNodes. Often it is useful to enumerate beyond the default limits, for example, to increase the enumeration limit beyond 250 URLs, or to allow site boundaries to be crossed when enumerating HTML links. It is possible to specify these and other aspects of enumeration, using the following syntax:

```
<RootNodes>
URL EnumSpec
URL EnumSpec
...
</RootNodes>
```

where *EnumSpec* is on a single line (using "\" to escape linefeeds), with the following syntax:

```
URL=URL-Max[,URL-Filter-filename]  \
Host=Host-Max[,Host-Filter-filename] \
Access=TypeList \
Delay=Seconds \
Depth=Number \
Enumeration=Enumeration-Program
```

The *EnumSpec* modifiers are all optional, and have the following meanings:

URL-Max

The number specified on the right hand side of the "URL=" expression lists the maximum number of LeafNode URLs to generate at all levels of depth, from the current URL. Note that *URL-Max* is the maximum number of URLs that are generated during the enumeration, and *not* a limit on how many URLs can pass through the candidate selection phase (see Section 4.5.3 (Customizing the candidate selection step)).

URL-Filter-filename

This is the name of a file containing a set of regular expression filters (see Section 4.3.1 (RootNode filters)) to allow or deny particular LeafNodes in the enumeration. The default filter is *$HARVEST_HOME/lib/gatherer/URL-filter-default* which excludes many image and sound files.

Host-Max

The number specified on the right hand side of the "Host=" expression lists the maximum number of hosts that will be touched during the RootNode enumeration. This enumeration actually counts hosts by IP address so that aliased hosts are properly enumerated. Note that this does not work correctly for multi-homed hosts, or for hosts with rotating DNS entries (used by some sites for load balancing heavily accessed servers).

Note: Prior to Harvest Version 1.2 the "Host=..." line was called "Site=...". We changed the name to "Host=" because it is more intuitively meaningful (being a host count limit, not a site count limit). For backwards compatibility with older Gatherer configuration files, we will continue to treat "Site=" as an alias for "Host=".

Host-Filter-filename

This is the name of a file containing a set of regular expression filters to allow or deny particular hosts in the enumeration. Each expression can specify both a host name (or IP address) and a port number (in case you have multiple servers running on different ports of the same server and you want to index only one). The syntax is "hostname:port".

Access

If the RootNode is an HTTP URL, then you can specify which access methods across which to enumerate. Valid access method types are: **FILE, FTP, Gopher, HTTP, News, Telnet,** or **WAIS**. Use a "|" character between type names to allow multiple access methods. For example, "**Access=HTTP|FTP|Gopher**" will follow HTTP, FTP, and Gopher URLs while enumerating an HTTP RootNode URL.

Note: We do not support cross-method enumeration from Gopher, because of the difficulty of ensuring that Gopher pointers do not cross site boundaries. For example, the Gopher URL *gopher://powell.cs.colorado.edu:7005/1ftp3aftp.cs.washington.edu40pub/* would get an FTP directory listing of ftp.cs.washington.edu:/pub, even though the host part of the URL is powell.cs.colorado.edu.

Delay

This is the number of seconds to wait between server contacts. It defaults to one second, when not specified otherwise. **Delay=3** will let the gatherer sleep 3 seconds between server contacts.

Depth

This is the maximum number of levels of enumeration that will be followed during gathering. **Depth=0** means that there is *no* limit to the depth of the enumeration. **Depth=1** means the specified URL will be retrieved, and all the URLs referenced by the specified URL will be retrieved; and so on for higher Depth values. In other words, the enumeration will follow links up to *Depth* steps away from the specified URL.

Enumeration-Program

This modifier adds a very flexible way to control a Gatherer. The Enumeration-Program is a filter which reads URLs as input and writes new enumeration parameters on output. See section 4.3.2 (Generic Enumeration program description) for specific details.

By default, *URL-Max* defaults to 250, *URL-Filter* defaults to no limit, *Host-Max* defaults to 1, *Host-Filter* defaults to no limit, *Access* defaults to HTTP only, *Delay* defaults to 1 second, and *Depth* defaults to zero. There is no way to specify an unlimited value for *URL-Max* or *Host-Max*.

4.3.1 RootNode filters

Filter files use the standard UNIX regular expression syntax (as defined by the POSIX standard), not the csh "globbing" syntax. For example, you would use ".*abc" to indicate any string ending with "abc", not "*abc". A filter file has the following syntax:

```
Deny  regex
Allow regex
```

The *URL-Filter* regular expressions are matched only on the URL-path portion of each URL (the scheme, hostname and port are excluded). For example, the following URL-Filter file would allow all URLs except those containing the regular expression "*/gatherers/*":

```
Deny  /gatherers/
Allow .
```

Another common use of URL-filters is to prevent the Gatherer from travelling "up" a directory. Automatically generated HTML pages for HTTP and FTP directories often contain a link for the parent directory "..". To keep the gatherer below a specific directory, use a URL-filter file such as:

```
Allow ^/my/cool/sutff/
Deny  .
```

The *Host-Filter* regular expressions are matched on the "hostname:port" portion of each URL. Because the port is included, you cannot use "**$**" to anchor the end of a hostname. Beginning with version 1.3, IP addresses may be specified in place of hostnames. A class B address such as 128.138.0.0 would be written as "**^128\.138\..***" in regular expression syntax. For example:

```
Deny   bcn.boulder.co.us:8080
Deny   bvsd.k12.co.us
Allow  ^128\.138\..*
Deny   .
```

The order of the **Allow** and **Deny** entries is important, since the filters are applied sequentially from first to last. So, for example, if you list "**Allow .***" first, no subsequent **Deny** expressions will be used, since this **Allow** filter will allow all entries.

4.3.2 Generic Enumeration program description

Flexible enumeration can be achieved by giving an **Enumeration=Enumeration-Program** modifier to a RootNode URL. The *Enumeration-Program* is a filter which takes URLs on standard input and writes new RootNode URLs on standard output.

The output format is different than specifying a RootNode URL in a Gatherer configuration file. Each output line must have nine fields separated by spaces. These fields are:

```
URL
URL-Max
URL-Filter-filename
Host-Max
Host-Filter-filename
Access
Delay
Depth
Enumeration-Program
```

These are the same fields as described in section 4.3 (RootNode specifications). Values must be given for each field. Use */dev/null* to disable the URL-Filter-filename and Host-Filter-filename. Use `/bin/false` to disable the Enumeration-Program.

4.3.3 Example RootNode configuration

Below is an example RootNode configuration:

```
      <RootNodes>
(1)   http://harvest.cs.colorado.edu/              URL=100,MyFilter
(2)   http://www.cs.colorado.edu/                  Host=50 Delay=60
(3)   gopher://gopher.colorado.edu/                Depth=1
(4)   file://powell.cs.colorado.edu/home/hardy/    Depth=2
(5)   ftp://ftp.cs.colorado.edu/pub/cs/techreports/ Depth=1
(6)   http://harvest.cs.colorado.edu/~hardy/hotlist.html \
              Depth=1 Delay=60
(7)   http://harvest.cs.colorado.edu/~hardy/ \
              Depth=2 Access=HTTP|FTP
      </RootNodes>
```

Each of the above RootNodes follows a different enumeration configuration as follows:

1. This RootNode will gather up to 100 documents that pass through the URL name filters contained within the file *MyFilter*.
2. This RootNode will gather the documents from up to the first 50 hosts it encounters while enumerating the specified URL, with no limit on the Depth of link enumeration. It will also wait for 60 seconds between each retrieval.
3. This RootNode will gather only the documents from the top-level menu of the Gopher server at *gopher.colorado.edu*.
4. This RootNode will gather all documents that are in the */home/hardy* directory, or that are in any subdirectory of */home/hardy*.
5. This RootNode will gather only the documents that are in the */pub/techreports* directory which, in this case, is some bibliographic files rather than the technical reports themselves.
6. This RootNode will gather all documents that are within 1 step away from the specified RootNode URL, waiting 60 seconds between each retrieval. This is a good method by which to index your hotlist. By putting an HTML file containing "hotlist" pointers as this RootNode, this enumeration will gather the top-level pages to all of your hotlist pointers.

7. This RootNode will gather all documents that are at most 2 steps away from the specified RootNode URL. Furthermore, it will follow and enumerate any HTTP or FTP URLs that it encounters during enumeration.

4.3.4 Gatherer enumeration vs. candidate selection

In addition to using the *URL-Filter* and *Host-Filter* files for the RootNode specification mechanism described in Section 4.3 (RootNode specifications), you can prevent documents from being indexed through customizing the *stoplist.cf* file, described in Section 4.5.3 (Customizing the type recognition, candidate selection, presentation unnesting, and summarizing steps). Since these mechanisms are invoked at different times, they have different effects. The *URL-Filter* and *Host-Filter* mechanisms are invoked by the Gatherer's "RootNode" enumeration programs. Using these filters as stop lists can prevent unwanted objects from being retrieved across the network. This can dramatically reduce gathering time and network traffic.

The *stoplist.cf* file is used by the *Essence* content extraction system (described in Section 4.5 (Extracting data for indexing: The Essence summarizing subsystem)) *after* the objects are retrieved, to select which objects should be content extracted and indexed. This can be useful because Essence provides a more powerful means of rejecting indexing candidates, in which you can customize based not only file naming conventions but also on file contents (e.g., looking at strings at the beginning of a file or at UNIX "magic" numbers), and also by more sophisticated file-grouping schemes (e.g., deciding not to extract contents from object code files for which source code is available).

As an example of combining these mechanisms, suppose you want to index the ".ps" files linked into your WWW site. You could do this by having a *stoplist.cf* file that contains "HTML", and a RootNode *URL-Filter* that contains:

```
Allow \.html
Allow \.ps
Deny  .*
```

As a final note, independent of these customizations the Gatherer attempts to avoid retrieving objects where possible, by using a local disk cache of objects, and by using the HTTP "If-Modified-Since" request header. The local disk cache is described in Section 4.7.6 (The local disk cache).

4.4 Generating LeafNode/RootNode URLs from a program

It is possible to generate RootNode or LeafNode URLs automatically from program output. This might be useful when gathering a large number of Usenet newsgroups, for example. The program is specified inside the RootNode or LeafNode section, preceded by a pipe symbol.

```
<LeafNodes>
|generate-news-urls.sh
</LeafNodes>
```

The script must output valid URLs, such as

```
news:comp.unix.voodoo
news:rec.pets.birds
http://www.nlanr.net/
...
```

In the case of RootNode URLs, enumeration parameters can be given after the program.

```
<RootNodes>
|my-fave-sites.pl Depth=1 URL=5000,url-filter
</RootNodes>
```

4.5 Extracting data for indexing: The Essence summarizing subsystem

After the Gatherer retrieves a document, it passes the document through a subsystem called *Essence* to extract indexing information. Essence allows the Gatherer to collect indexing information easily from a wide variety of information, using different techniques depending on the type of data and the needs of the particular corpus being indexed. In a nutshell, Essence can determine the type of data pointed to by a URL (e.g., PostScript vs. HTML), “unravel” presentation nesting formats (such as compressed “tar” files), select which types of data to index (e.g., don’t index Audio files), and then apply a type-specific extraction algorithm (called a *summarizer*) to the data to generate a content summary. Users can customize each of these aspects, but often this is not necessary. Harvest is distributed with a “stock” set of type recognizers, presentation unnesters, candidate selectors, and summarizers that work well for many applications.

Below we describe the stock summarizer set, the current components distribution, and how users can customize summarizers to change how they operate and add summarizers for new types of data. If you develop a summarizer that is likely to be useful to other users, please notify us via email at *lee@arco.de* `<mailto:lee@arco.de>` so we may include it in our Harvest distribution.

```
Type            Summarizer Function
--------------------------------------------------------------------
Bibliographic   Extract author and titles
Binary          Extract meaningful strings and manual page summary
C, CHeader      Extract procedure names, included file names, and comments
Dvi             Invoke the Text summarizer on extracted ASCII text
FAQ, FullText, README
                Extract all words in file
Font            Extract comments
HTML            Extract anchors, hypertext links, and selected fields
LaTex           Parse selected LaTex fields (author, title, etc.)
Mail            Extract certain header fields
Makefile        Extract comments and target names
ManPage         Extract synopsis, author, title, etc., based on ``-man'' macros
News            Extract certain header fields
Object          Extract symbol table
Patch           Extract patched file names
Perl            Extract procedure names and comments
PostScript      Extract text in word processor-specific fashion, and pass
                through Text summarizer.
RCS, SCCS       Extract revision control summary
RTF             Up-convert to HTML and pass through HTML summarizer
SGML            Extract fields named in extraction table
ShellScript     Extract comments
SourceDistribution
                Extract full text of README file and comments from Makefile
                and source code files, and summarize any manual pages
```

SymbolicLink	Extract file name, owner, and date created
TeX	Invoke the Text summarizer on extracted ASCII text
Text	Extract first 100 lines plus first sentence of each remaining paragraph
Troff	Extract author, title, etc., based on ''-man'', ''-ms'', ''-me'' macro packages, or extract section headers and topic sentences.
Unrecognized	Extract file name, owner, and date created.

4.5.1 Default actions of "stock" summarizers

The table in Section 4.5 (Extracting data for indexing: The Essence summarizing subsystem) provides a brief reference for how documents are summarized depending on their type. These actions can be customized, as discussed in Section 4.5.3 (Customizing the type recognition, candidate selection, presentation unnesting, and summarizing steps). Some summarizers are implemented as UNIX programs while others are expressed as regular expressions; see Section 4.5.3 (Customizing the summarizing step) or Section 8.4 (Example 4) for more information about how to write a summarizer.

4.5.2 Summarizing SGML data

It is possible to summarize documents that conform to the Standard Generalized Markup Language (SGML), for which you have a Document Type Definition (DTD). The World Wide Web's Hypertext Mark-up Language (HTML) is actually a particular application of SGML, with a corresponding DTD. (In fact, the Harvest HTML summarizer can use the HTML DTD and our SGML summarizing mechanism, which provides various advantages; see Section 4.5.2 (The SGML-based HTML summarizer).) SGML is being used in an increasingly broad variety of applications, for example as a format for storing data for a number of physical sciences. Because SGML allows documents to contain a good deal of structure, Harvest can summarize SGML documents very effectively.

The SGML summarizer (`SGML.sum`) uses the `sgmls` program by James Clark to parse the SGML document. The parser needs both a DTD for the document and a Declaration file that describes the allowed character set. The `SGML.sum` program uses a table that maps SGML tags to SOIF attributes.

Location of support files

SGML support files can be found in *$HARVEST_HOME/lib/gatherer/sgmls-lib/*. For example, these are the default pathnames for HTML summarizing using the SGML summarizing mechanism:

```
$HARVEST_HOME/lib/gatherer/sgmls-lib/HTML/html.dtd
$HARVEST_HOME/lib/gatherer/sgmls-lib/HTML/HTML.decl
$HARVEST_HOME/lib/gatherer/sgmls-lib/HTML/HTML.sum.tbl
```

The location of the DTD file must be specified in the `sgmls` catalog (*$HARVEST_HOME/lib/gatherer/sgmls-lib/catalog*). For example:

```
DOCTYPE   HTML   HTML/html.dtd
```

The `SGML.sum` program looks for the *.decl* file in the default location. An alternate pathname can be specified with the **-d** option to `SGML.sum`.

The summarizer looks for the *.sum.tbl* file first in the Gatherer's lib directory and then in the default location. Both of these can be overridden with the **-t** option to `SGML.sum`.

The SGML to SOIF table

The translation table provides a simple yet powerful way to specify how an SGML document is to be summarized. There are four ways to map SGML data into SOIF. The first two are concerned with placing the *content* of an SGML tag into a SOIF attribute.

A simple SGML-to-SOIF mapping looks like this:

```
<TAG>              soif1,soif2,...
```

This places the content that occurs inside the tag "TAG" into the SOIF attributes "soif1" and "soif2". It is possible to select different SOIF attributes based on SGML attribute values. For example, if "ATT" is an attribute of "TAG", then it would be written like this:

```
<TAG,ATT=x>         x-stuff
<TAG,ATT=y>         y-stuff
<TAG>               stuff
```

The second two mappings place values of SGML attributes into SOIF attributes. To place the value of the "ATT" attribute of the "TAG" tag into the "att-stuff" SOIF attribute you would write:

```
<TAG:ATT>           att-stuff
```

It is also possible to place the value of an SGML attribute into a SOIF attribute named by a different SOIF attribute:

```
<TAG:ATT1>          $ATT2
```

When the summarizer encounters an SGML attribute not listed in the table, the content is passed to the parent tag and becomes a part of the parent's content. To force the content of some tag *not* to be passed up, specify the SOIF attribute as "ignore". To force the content of some tag to be passed to the parent in addition to being placed into a SOIF attribute, list an addition SOIF attribute named "parent".

Please see Section 4.5.2 (The SGML-based HTML summarizer) for examples of these mappings.

Errors and warnings from the SGML Parser

The `sgmls` parser can generate an overwhelming volume of error and warning messages. This will be especially true for HTML documents found on the Internet, which often do not conform to the strict HTML DTD. By default, errors and warnings are redirected to */dev/null* so that they do not clutter the Gatherer's log files. To enable logging of these messages, edit the `SGML.sum` Perl script and set **$syntax_check = 1**.

Creating a summarizer for a new SGML-tagged data type

To create an SGML summarizer for a new SGML-tagged data type with an associated DTD, you need to do the following:

1. Write a shell script named FOO.sum which simply contains

```
#!/bin/sh
exec SGML.sum FOO $*
```

2. Modify the essence configuration files (as described in Section 4.5.3 (Customizing the type recognition step)) so that your documents get typed as FOO.
3. Create the directory *$HARVEST_HOME/lib/gatherer/sgmls-lib/FOO/* and copy your DTD and Declaration there as FOO.dtd and FOO.decl. Edit *$HARVEST_HOME/lib/gatherer/sgmls-lib/catalog* and add FOO.dtd to it.
4. Create the translation table FOO.sum.tbl and place it with the DTD in *$HARVEST_HOME/lib/gatherer/sgmls-lib/FOO/*.

At this point you can test everything from the command line as follows:

```
% FOO.sum myfile.foo
```

The SGML-based HTML summarizer

Harvest can summarize HTML using the generic SGML summarizer described in Section 4.5.2 (Summarizing SGML data). The advantage of this approach is that the summarizer is more easily customizable, and fits with the well-conceived SGML model (where you define DTDs for individual document types and build interpretation software to understand DTDs rather than individual document types). The downside is that the summarizer is now pickier about syntax, and many Web documents are not syntactically correct. Because of this pickiness, the default is for the HTML summarizer to run with syntax checking outputs disabled. If your documents are so badly formed that they confuse the parser, this may mean the summarizing process dies unceremoniously. If you find that some of your HTML documents do not get summarized or only get summarized in part, you can turn syntax-checking output on by setting **$syntax_check = 1** in `$HARVEST_HOME/lib/gatherer/SGML.sum`. That will allow you to see which documents are invalid and where.

Note that part of the reason for this problem is that Web browsers do not insist on well-formed documents. So, users can easily create documents that are not completely valid, yet display fine.

Below is the default SGML-to-SOIF table used by the HTML summarizer:

```
HTML ELEMENT   SOIF ATTRIBUTES
------------   -----------------------
    <A>             keywords,parent
    <A:HREF>        url-references
    <ADDRESS>       address
    <B>             keywords,parent
    <BODY>          body
    <CITE>          references
    <CODE>          ignore
    <EM>            keywords,parent
    <H1>            headings
    <H2>            headings
    <H3>            headings
    <H4>            headings
    <H5>            headings
    <H6>            headings
    <HEAD>          head
```

```
<I>              keywords,parent
<IMG:SRC>        images
<META:CONTENT>   $NAME
<STRONG>         keywords,parent
<TITLE>          title
<TT>             keywords,parent
<UL>             keywords,parent
```

The pathname to this file is *$HARVEST_HOME/lib/gatherer/sgmls-lib/HTML/HTML.sum.tbl*.

Individual Gatherers may do customized HTML summarizing by placing a modified version of this file in the Gatherer *lib* directory. Another way to customize is to modify the `HTML.sum` script and add a **-t** option to the SGML.sum command. For example:

```
SGML.sum -t $HARVEST_HOME/lib/my-HTML.table HTML $*
```

In HTML, the document title is written as:

```
<TITLE>My Home Page</TITLE>
```

The above translation table will place this in the SOIF summary as:

```
title{13}:  My Home Page
```

Note that "keywords,parent" occurs frequently in the table. For any specially marked text (bold, emphasized, hypertext links, etc.), the words will be copied into the keywords attribute and also left in the content of the parent element. This keeps the body of the text readable by not removing certain words.

Any text that appears inside a pair of CODE tags will not show up in the summary because we specified "ignore" as the SOIF attribute.

URLs in HTML anchors are written as:

```
<A HREF="http://harvest.cs.colorado.edu/">
```

The specification for **<A:HREF>** in the above translation table causes this to appear as:

```
url-references{32}: http://harvest.cs.colorado.edu/
```

Adding META data to your HTML

One of the most useful HTML tags is META. This allows the document writer to include arbitrary metadata in an HTML document. A Typical usage of the META element is:

```
<META NAME="author" CONTENT="Joe T. Slacker">
```

By specifying "**<META:CONTENT>** $NAME" in the translation table, this comes out as:

```
author{15}: Joe T. Slacker
```

Using the META tags, HTML authors can easily add a list of keywords to their documents:

```
<META NAME="keywords" CONTENT="word1 word2">
<META NAME="keywords" CONTENT="word3 word4">
```

Other examples

A very terse HTML summarizer could be specified with a table that only puts emphasized words into the keywords attribute:

```
HTML ELEMENT   SOIF ATTRIBUTES
------------   -----------------------
    <A>             keywords
    <B>             keywords
    <EM>            keywords
    <H1>            keywords
    <H2>            keywords
    <H3>            keywords
    <I>             keywords
    <META:CONTENT>  $NAME
    <STRONG>        keywords
    <TITLE>         title,keywords
    <TT>            keywords
```

Conversely, a full-text summarizer can be easily specified with only:

```
HTML ELEMENT   SOIF ATTRIBUTES
------------   -----------------------
    <HTML>          full-text
    <TITLE>         title,parent
```

4.5.3 Customizing the type recognition, candidate selection, presentation unnesting, and summarizing steps

The Harvest Gatherer's actions are defined by a set of configuration and utility files, and a corresponding set of executable programs referenced by some of the configuration files.

If you want to customize a Gatherer, you should create *bin* and *lib* subdirectories in the directory where you are running the Gatherer, and then copy *$HARVEST_HOME/lib/gatherer/*.cf* and *$HARVEST_HOME/lib/gatherer/magic* into your *lib* directory. Then add to your Gatherer configuration file:

```
Lib-Directory:         lib
```

The details about what each of these files does are described below. The basic contents of a typical Gatherer's directory is as follows (note: some of the file names below can be changed by setting variables in the Gatherer configuration file, as described in Section 4.7.1 (Setting variables in the Gatherer configuration file)):

```
RunGatherd*    bin/          GathName.cf    log.errors     tmp/
RunGatherer*   data/         lib/           log.gatherer

bin:
MyNewType.sum*

data:
All-Templates.gz    INFO.soif    PRODUCTION.gdbm    gatherd.log
INDEX.gdbm          MD5.gdbm     gatherd.cf
```

```
lib:
bycontent.cf   byurl.cf      quick-sum.cf
byname.cf      magic         stoplist.cf

tmp:
```

The `RunGatherd` and `RunGatherer` are used to export the Gatherer's database after a machine reboot and to run the Gatherer, respectively. The *log.errors* and *log.gatherer* files contain error messages and the output of the *Essence* typing step, respectively (Essence will be described shortly). The *GathName.cf* file is the Gatherer's configuration file.

The *bin* directory contains any summarizers and any other program needed by the summarizers. If you were to customize the Gatherer by adding a summarizer, you would place those programs in this *bin* directory; the `MyNewType.sum` is an example.

The *data* directory contains the Gatherer's database which `gatherd` exports. The Gatherer's database consists of the *All-Templates.gz, INDEX.gdbm, INFO.soif, MD5.gdbm* and *PRODUCTION.gdbm* files. The *gatherd.cf* file is used to support access control as described in Section 4.7.4 (Controlling access to the Gatherer's database). The *gatherd.log* file is where the `gatherd` program logs its information.

The *lib* directory contains the configuration files used by the Gatherer's subsystems, namely Essence. These files are described briefly in the following table:

```
bycontent.cf    Content parsing heuristics for type recognition step
byname.cf       File naming heuristics for type recognition step
byurl.cf        URL naming heuristics for type recognition step
magic           UNIX ``file'' command specifications (matched against
                bycontent.cf strings)
quick-sum.cf    Extracts attributes for summarizing step.
stoplist.cf     File types to reject during candidate selection
```

Customizing the type recognition step

Essence recognizes types in three ways (in order of precedence): by URL naming heuristics, by file naming heuristics, and by locating *identifying* data within a file using the UNIX `file` command.

To modify the type recognition step, edit *lib/byname.cf* to add file naming heuristics, or *lib/byurl.cf* to add URL naming heuristics, or *lib/bycontent.cf* to add by-content heuristics. The by-content heuristics match the output of the UNIX `file` command, so you may also need to edit the *lib/magic* file. See Section 8.3 (Example 3) and 8.4 (Example 4) for detailed examples on how to customize the type recognition step.

Customizing the candidate selection step

The *lib/stoplist.cf* configuration file contains a list of types that are rejected by Essence. You can add or delete types from *lib/stoplist.cf* to control the candidate selection step.

To direct Essence to index only certain types, you can list the types to index in *lib/allowlist.cf*. Then, supply Essence with the −**allowlist** flag.

The file and URL naming heuristics used by the type recognition step (described in Section 4.5.3 (Customizing the type recognition step)) are particularly useful for candidate selection when gathering remote data. They allow the Gatherer to avoid retrieving files that you don't want to index (in contrast, recognizing types by locating identifying data within a file requires that the file be retrieved first). This approach can save quite a bit of network traffic, particularly when used in combination with enumerated *RootNode* URLs. For example, many sites provide each of their files in both a compressed and uncompressed form. By building a *lib/allowlist.cf* containing only the Compressed types, you can avoid retrieving the uncompressed versions of the files.

Customizing the presentation unnesting step

Some types are declared as "nested" types. Essence treats these differently than other types, by running a presentation unnesting algorithm or "Exploder" on the data rather than a Summarizer. At present Essence can handle files nested in the following formats:

1. binhex
2. uuencode
3. shell archive ("shar")
4. tape archive ("tar")
5. bzip2 compressed ("bzip2")
6. compressed
7. GNU compressed ("gzip")
8. zip compressed archive

To customize the presentation unnesting step you can modify the Essence source file *src/gatherer/essence/unnest.c*. This file lists the available presentation encodings, and also specifies the unnesting algorithm. Typically, an external program is used to unravel a file into one or more component files (e.g. `bzip2, gunzip, uudecode,` and `tar`).

An *Exploder* may also be used to explode a file into a stream of SOIF objects. An Exploder program takes a URL as its first command-line argument and a file containing the data to use as its second, and then generates one or more SOIF objects as output. For your convenience, the *Exploder* type is already defined as a nested type. To save some time, you can use this type and its corresponding `Exploder.unnest` program rather than modifying the Essence code.

See Section 8.2 (Example 2) for a detailed example on writing an Exploder. The *unnest.c* file also contains further information on defining the unnesting algorithms.

Customizing the summarizing step

Essence supports two mechanisms for defining the type-specific extraction algorithms (called *Summarizers*) that generate content summaries: a UNIX program that takes as its only command line argument the filename of the data to summarize, and line-based regular expressions specified in *lib/quick-sum.cf*. See Section 8.4 (Example 4) for detailed examples on how to define both types of Summarizers.

The UNIX Summarizers are named using the convention `TypeName.sum` (e.g., `PostScript.sum`). These Summarizers output their content summary in a SOIF attribute-value list (see Section 7 (The Summary Object Interchange Format (SOIF))). You can use the `wrapit` command to wrap raw output into the SOIF format (i.e., to provide byte-count delimiters on the individual attribute-value pairs).

There is a summarizer called `FullText.sum` that you can use to perform full text indexing of selected file types, by simply setting up the *lib/bycontent.cf* and *lib/byname.cf* configuration files to recognize the desired file types as FullText (i.e., using "FullText" in column 1 next to the matching regular expression).

4.6 Post-Summarizing: Rule-based tuning of object summaries

It is possible to "fine-tune" the summary information generated by the Essence summarizers. A typical application of this would be to change the *Time-to-Live* attribute based on some knowledge about the objects. So an administrator could use the post-summarizing feature to give quickly-changing objects a lower TTL, and very stable documents a higher TTL.

Objects are selected for post-summarizing if they meet a specified condition. A condition consists of three parts: An attribute name, an operation, and some string data. For example:

```
city == 'New York'
```

In this case we are checking if the *city* attribute is equal to the string 'New York'. For exact string matching, the string data must be enclosed in single quotes. Regular expressions are also supported:

```
city ~ /New York/
```

Negative operators are also supported:

```
city != 'New York'
city !~ /New York/
```

Conditions can be joined with '&&' (logical and) or '||' (logical or) operators:

```
city == 'New York' && state != 'NY';
```

When all conditions are met for an object, some number of instructions are executed on it. There are four types of instructions which can be specified:

1. Set an attribute exactly to some specific string.

   Example:

   ```
   time-to-live = "86400"
   ```

2. Filter an attribute through some program. The attribute value is given as input to the filter. The output of the filter becomes the new attribute value.

   Example:

   ```
   keywords | tr A-Z a-z
   ```

3. Filter multiple attributes through some program. In this case the filter must read and write attributes in the SOIF format.

   Example:

```
address,city,state,zip ! cleanup-address.pl
```

4. A special case instruction is to delete an object. To do this, simply write:

```
delete()
```

4.6.1 The Rules file

The conditions and instructions are combined together in a “rules” file. The format of this file is somewhat similar to a Makefile; conditions begin in the first column and instructions are indented by a tab-stop.

Example:

```
type == 'HTML'
        partial-text | cleanup-html-text.pl

URL ~ /users/
        time-to-live = "86400"
        partial-text ! extract-owner.sh

type == 'SOIFStream'
        delete()
```

This rules file is specified in the gatherer.cf file with the Post-Summarizing tag, e.g.:

```
Post-Summarizing: lib/myrules
```

4.6.2 Rewriting URLs

Until version 1.4 it was not possible to rewrite the URL-part of an object summary. It is now possible, but only by using the “pipe” instruction. This may be useful for people wanting to run a Gatherer on *file://* URLs, but have them appear as *http://* URLs. This can be done with a post-summarizing rule such as:

```
url ~ 'file://localhost/web/htdocs/'
        url | fix-url.pl
```

And the 'fix-url.pl' script might look like:

```
#!/usr/local/bin/perl -p
s'file://localhost/web/htdocs/'http://www.my.domain/';
```

4.7 Gatherer administration

4.7.1 Setting variables in the Gatherer configuration file

In addition to customizing the steps described in Section 4.5.3 (Customizing the type recognition, candidate selection, presentation unnesting, and summarizing steps), you can customize the Gatherer by setting variables in the Gatherer configuration file. This file consists of two parts: a list of variables that specify information about the Gatherer (such as its name, host, and port number), and two lists of URLs (divided into **RootNodes** and **LeafNodes**) from which to collect indexing information. Section 4.2 (Basic setup) shows an example Gatherer configuration file. In this section we focus on the variables that the user can set in the first part of the Gatherer configuration file.

Each variable name starts in the first column, ends with a colon, then is followed by the value. The following table shows the supported variables:

```
Access-Delay:          Default delay between URLs accesses.
Data-Directory:        Directory where GDBM database is written.
Debug-Options:         Debugging options passed to child programs.
Errorlog-File:         File for logging errors.
Essence-Options:       Any extra options to pass to Essence.
FTP-Auth:              Username/password for protected FTP documents.
Gatherd-Inetd:         Denotes that gatherd is run from inetd.
Gatherer-Host:         Full hostname where the Gatherer is run.
Gatherer-Name:         A Unique name for the Gatherer.
Gatherer-Options:      Extra options for the Gatherer.
Gatherer-Port:         Port number for gatherd.
Gatherer-Version:      Version string for the Gatherer.
HTTP-Basic-Auth:       Username/password for protected HTTP documents.
HTTP-Proxy:            host:port of your HTTP proxy.
Keep-Cache:            ``yes'' to not remove local disk cache.
Lib-Directory:         Directory where configuration files live.
Local-Mapping:         Mapping information for local gathering.
Log-File:              File for logging progress.
Post-Summarizing:      A rules-file for post-summarizing.
Refresh-Rate:          Object refresh-rate in seconds, default 1 week.
Time-To-Live:          Object time-to-live in seconds, default 1 month.
Top-Directory:         Top-level directory for the Gatherer.
Working-Directory:     Directory for tmp files and local disk cache.
```

Notes:

- We recommend that you use the **Top-Directory** variable, since it will set the **Data-Directory**, **Lib-Directory**, and **Working-Directory** variables.
- Both **Working-Directory** and **Data-Directory** will have files in them after the Gatherer has run. The **Working-Directory** will hold the local-disk cache that the Gatherer uses to reduce network I/O, and the **Data-Directory** will hold the GDBM databases that contain the content summaries.
- You should use full rather than relative pathnames.
- All variable definitions *must* come before the RootNode or LeafNode URLs.
- Any line that starts with a "#" is a comment.
- **Local-Mapping** is discussed in Section 4.7.2 (Local file system gathering for reduced CPU load).
- **HTTP-Proxy** will retrieve HTTP URLs via a proxy host. The syntax is **hostname:port**; for example, **proxy.yoursite.com:3128**.
- **Essence-Options** is particularly useful, as it lets you customize basic aspects of the Gatherer easily.

- The only valid **Gatherer-Options** is **–save-space** which directs the Gatherer to be more space efficient when preparing its database for export.
- The `Gatherer` program will accept the **-background** flag which will cause the Gatherer to run in the background.

The Essence options are:

```
Option                  Meaning
--------------------------------------------------------------------
--allowlist filename    File with list of types to allow
--fake-md5s             Generates MD5s for SOIF objects from a .unnest program
--fast-summarizing      Trade speed for some consistency.  Use only when
                        an external summarizer is known to generate clean,
                        unique attributes.
--full-text             Use entire file instead of summarizing.  Alternatively,
                        you can perform full text indexing of individual file
                        types by using the FullText.sum summarizer.
--max-deletions n       Number of GDBM deletions before reorganization
--minimal-bookkeeping   Generates a minimal amount of bookkeeping attrs
--no-access             Do not read contents of objects
--no-keywords           Do not automatically generate keywords
--stoplist filename     File with list of types to remove
--type-only             Only type data; do not summarize objects
```

A particular note about full text summarizing: Using the Essence **–full-text** option causes files not to be passed through the Essence content extraction mechanism. Instead, their entire content is included in the SOIF summary stream. In some cases this may produce unwanted results (e.g., it will directly include the PostScript for a document rather than first passing the data through a PostScript to text extractor, providing few searchable terms and large SOIF objects). Using the individual file type summarizing mechanism described in Section 4.5.3 (Customizing the summarizing step) will work better in this regard, but will require you to specify how data are extracted for each individual file type. In a future version of Harvest we will change the Essence **–full-text** option to perform content extraction before including the full text of documents.

4.7.2 Local file system gathering for reduced CPU load

Although the Gatherer's work load is specified using URLs, often the files being gathered are located on a local file system. In this case it is much more efficient to gather directly from the local file system than via FTP/Gopher/HTTP/News, primarily because of all the UNIX forking required to gather information via these network processes. For example, our measurements indicate it causes from 4-7x more CPU load to gather from FTP than directly from the local file system. For large collections (e.g., archive sites containing many thousands of files), the CPU savings can be considerable.

Starting with Harvest Version 1.1, it is possible to tell the Gatherer how to translate URLs to local file system names, using the **Local-Mapping** Gatherer configuration file variable (see Section 4.7.1 (Setting variables in the Gatherer configuration file)). The syntax is:

```
Local-Mapping: URL_prefix local_path_prefix
```

This causes all URLs starting with **URL_prefix** to be translated to files starting with the prefix **local_path_prefix** while gathering, but to be left as URLs in the results of queries (so the objects can be retrieved as usual). Note that no regular expressions are supported here. As an example, the specification

```
Local-Mapping: http://harvest.cs.colorado.edu/~hardy/ /homes/hardy/public_html/
Local-Mapping: ftp://ftp.cs.colorado.edu/pub/cs/ /cs/ftp/
```

would cause the URL *http://harvest.cs.colorado.edu/~hardy/Home.html* to be translated to the local file name */homes/hardy/public_html/Home.html*, while the URL *ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.Conf.ps.Z* would be translated to the local file name */cs/ftp/techreports/schwartz/Harvest.Conf.ps.Z*.

Local gathering will work over NFS file systems. A local mapping will fail if: the local file cannot be opened for reading; or the local file is not a regular file; or the local file has execute bits set. So, for directories, symbolic links and CGI scripts, the server is always contacted rather than the local file system. Lastly, the Gatherer does not perform any URL syntax translations for local mappings. If your URL has characters that should be escaped (as in *RFC1738*), then the local mapping will fail. Starting with version 1.4 patchlevel 2 Essence will print *[L]* after URLs which were successfully accessed locally.

Note that if your network is highly congested, it may actually be faster to gather via HTTP/FTP/Gopher than via NFS, because NFS becomes very inefficient in highly congested situations. Even better would be to run local Gatherers on the hosts where the disks reside, and access them directly via the local file system.

4.7.3 Gathering from password-protected servers

You can gather password-protected documents from HTTP and FTP servers. In both cases, you can specify a username and password as a part of the URL. The format is as follows:

```
 ftp://user:password@host:port/url-path
http://user:password@host:port/url-path
```

With this format, the "user:password" part is kept as a part of the URL string all throughout Harvest. This may enable anyone who uses your Broker(s) to access password-protected documents.

You can keep the username and password information "hidden" by specifying the authentication information in the Gatherer configuration file. For HTTP, the format is as follows:

```
HTTP-Basic-Auth: realm username password
```

where **realm** is the same as the **AuthName** parameter given in an Apache httpd *httpd.conf* or *.htaccess* file. In other httpd server configuration, the realm value is sometimes called **ServerId**.

For FTP, the format in the gatherer.cf file is

```
FTP-Auth: hostname[:port] username password
```

4.7.4 Controlling access to the Gatherer's database

You can use the *gatherd.cf* file (placed in the **Data-Directory** of a Gatherer) to control access to the Gatherer's database. A line that begins with **Allow** is followed by any number of domain or host names that are allowed to connect to the Gatherer. If the word **all** is used, then all hosts are matched. **Deny** is the opposite of **Allow**. The following example will only allow hosts in the **cs.colorado.edu** or **usc.edu** domain access the Gatherer's database:

```
Allow  cs.colorado.edu usc.edu
Deny   all
```

4.7.5 Periodic gathering and realtime updates

The `Gatherer` program does not automatically do any periodic updates – when you run it, it processes the specified URLs, starts up a `gatherd` daemon (if one isn't already running), and then exits. If you want to update the data periodically (e.g., to capture new files as they are added to an FTP archive), you need to use the UNIX `cron` command to run the `Gatherer` program at some regular interval.

To set up periodic gathering via `cron`, use the `RunGatherer` command that `RunHarvest` will create. An example `RunGatherer` script follows:

```
#!/bin/sh
#
#  RunGatherer - Runs the ATT 800 Gatherer (from cron)
#
HARVEST_HOME=/usr/local/harvest; export HARVEST_HOME
PATH=${HARVEST_HOME}/bin:${HARVEST_HOME}/lib/gatherer:${HARVEST_HOME}/lib:$PATH
export PATH
NNTPSERVER=localhost; export NNTPSERVER
cd /usr/local/harvest/gatherers/att800
exec Gatherer "att800.cf"
```

You should run the `RunGatherd` command from your system startup (e.g. */etc/rc.local*) file, so the Gatherer's database is exported each time the machine reboots. An example `RunGatherd` script follows:

```
#!/bin/sh
#
#  RunGatherd - starts up the gatherd process (from /etc/rc.local)
#
HARVEST_HOME=/usr/local/harvest; export HARVEST_HOME
PATH=${HARVEST_HOME}/lib/gatherer:${HARVEST_HOME}/bin:$PATH; export PATH
exec gatherd -d /usr/local/harvest/gatherers/att800/data 8500
```

4.7.6 The local disk cache

The Gatherer maintains a local disk cache of files it gathers to reduce network traffic from restarting aborted gathering attempts. However, since the remote server must still be contacted whenever `Gatherer` runs, please do not set your cron job to run `Gatherer` frequently. A typical value might be weekly or monthly, depending on how congested the network and how important it is to have the most current data.

By default, the Gatherer's local disk cache is deleted after each successful completion. To save the local disk cache between Gatherer sessions, define **Keep-Cache: yes** in your Gatherer configuration file (Section 4.7.1 (Setting variables in the Gatherer configuration file)).

If you want your Broker's index to reflect new data, then you must run the Gatherer *and* run a Broker collection. By default, a Broker will perform collections once a day. If you want the Broker to collect data as soon as it's gathered, then you will need to coordinate the timing of the completion of the Gatherer and the Broker collections.

If you run your Gatherer frequently and you use the **Keep-Cache: yes** in your Gatherer configuration file, then the Gatherer's local disk cache may interfere with retrieving updates. By default, objects in the local disk cache expire after 7 days; however, you can expire objects more quickly by setting the **$GATHERER_CACHE_TTL** environment variable to the number of seconds for the Time-To-Live (TTL) before you run the Gatherer, or you can change `RunGatherer` to remove the Gatherer's *tmp* directory after each Gatherer run. For example, to expire objects in the local disk cache after one day:

```
% setenv GATHERER_CACHE_TTL 86400       # one day
% ./RunGatherer
```

The Gatherer's local disk cache size defaults to 32 MBs, but you can change this value by setting the **$HARVEST_MAX_LOCAL_CACHE** environment variable to the number of MBs before you run the Gatherer. For example, to have a maximum cache of 10 MB you can do as follows:

```
% setenv HARVEST_MAX_LOCAL_CACHE 10       # 10 MB
% ./RunGatherer
```

If you have access to the software that creates the files that you are indexing (e.g., if all updates are funneled through a particular editor, update script, or system call), you can modify this software to schedule realtime Gatherer updates whenever a file is created or updated. For example, if all users update the files being indexed using a particular program, this program could be modified to run the Gatherer upon completion of the user's update.

Note that, when used in conjunction with `cron`, the Gatherer provides a powerful data "mirroring" facility. You can use the Gatherer to replicate the contents of one or more sites, retrieve data in multiple formats via multiple protocols (FTP, HTTP, etc.), optionally perform a variety of type- or site-specific transformations on the data, and serve the results very efficiently as compressed SOIF object summary streams to other sites that wish to use the data for building indexes or for other purposes.

4.7.7 Incorporating manually generated information into a Gatherer

You may want to inspect the quality of the automatically-generated SOIF templates. In general, Essence's techniques for automatic information extraction produce imperfect results. Sometimes it is possible to customize the summarizers to better suit the particular context (see Section 4.5.3 (Customizing the summarizing step)). Sometimes, however, it makes sense to augment or change the automatically generated keywords with manually entered information. For example, you may want to add *Title* attributes to the content summaries for a set of PostScript documents (since it's difficult to parse them out of PostScript automatically).

Harvest provides some programs that automatically clean up a Gatherer's database. The `rmbinary` program removes any binary data from the templates. The `cleandb` program does some simple validation of SOIF objects, and when given the **-truncate** flag it will truncate the *Keywords* data field to 8 kilobytes. To help in manually managing the Gatherer's databases, the `gdbmutil` GDBM database management tool is provided in *$HARVEST_HOME/lib/gatherer*.

In a future release of Harvest we will provide a forms-based mechanism to make it easy to provide manual annotations. In the meantime, you can annotate the Gatherer's database with manually generated information by using the `mktemplate`, `template2db`, `mergedb`, and `mkindex` programs. You first need to create a file (called, say, *annotations*) in the following format:

```
@FILE { url1
Attribute-Name-1:       DATA
Attribute-Name-2:       DATA
...
Attribute-Name-n:       DATA
}

@FILE { url2
Attribute-Name-1:       DATA
Attribute-Name-2:       DATA
...
Attribute-Name-n:       DATA
}

...
```

Note that the *Attributes* must begin in column 0 and have one tab after the colon, and the *DATA* must be on a single line.

Next, run the `mktemplate` and `template2db` programs to generate SOIF and then GDBM versions of these data (you can have several files containing the annotations, and generate a single GDBM database from the above commands):

```
% set path = ($HARVEST_HOME/lib/gatherer $path)
% mktemplate annotations [annotations2 ...] | template2db annotations.gdbm
```

Finally, you run `mergedb` to incorporate the annotations into the automatically generated data, and `mkindex` to generate an index for it. The usage line for `mergedb` is:

```
mergedb production automatic manual [manual ...]
```

The idea is that *production* is the final GDBM database that the Gatherer will serve. This is a *new* database that will be generated from the other databases on the command line. *automatic* is the GDBM database that a Gatherer automatically generated in a previous run (e.g., *WORKING.gdbm* or a previous *PRODUCTION.gdbm*). *manual* and so on are the GDBM databases that you manually created. When mergedb runs, it builds the *production* database by first copying the templates from the *manual* databases, and then merging in the attributes from the *automatic* database. In case of a conflict (the same attribute with different values in the *manual* and *automatic* databases), the *manual* values override the *automatic* values.

By keeping the automatically and manually generated data stored separately, you can avoid losing the manual updates when doing periodic automatic gathering. To do this, you will need to set up a script to remerge the manual annotations with the automatically gathered data after each gathering.

An example use of `mergedb` is:

```
% mergedb PRODUCTION.new PRODUCTION.gdbm annotations.gdbm
% mv PRODUCTION.new PRODUCTION.gdbm
% mkindex
```

If the manual database looked like this:

```
@FILE { url1
my-manual-attribute:  this is a neat attribute
}
```

and the automatic database looked like this:

```
@FILE { url1
keywords:   boulder colorado
file-size:  1034
md5:        c3d79dc037efd538ce50464089af2fb6
}
```

then in the end, the production database will look like this:

```
@FILE { url1
my-manual-attribute:  this is a neat attribute
keywords:   boulder colorado
file-size:  1034
md5:        c3d79dc037efd538ce50464089af2fb6
}
```

4.8 Troubleshooting

Debugging

Extra information from specific programs and library routines can be logged by setting debugging flags. A debugging flag has the form **-Dsection,level**. *Section* is an integer in the range 1-255, and *level* is an integer in the range 1-9. Debugging flags can be given on a command line, with the **Debug-Options:** tag in a gatherer configuration file, or by setting the environment variable **$HARVEST_DEBUG**.

Examples:

```
Debug-Options: -D68,5 -D44,1
% httpenum -D20,1 -D21,1 -D42,1 http://harvest.cs.colorado.edu/
% setenv HARVEST_DEBUG '-D20,1 -D23,1 -D63,1'
```

Debugging sections and levels have been assigned to the following sections of the code:

```
section  20, level 1, 5, 9        Common liburl URL processing
section  21, level 1, 5, 9        Common liburl HTTP routines
section  22, level 1, 5           Common liburl disk cache routines
section  23, level 1              Common liburl FTP routines
section  24, level 1              Common liburl Gopher routines
section  25, level 1              urlget - standalone liburl program.
section  26, level 1              ftpget - standalone liburl program.
section  40, level 1, 5, 9        Gatherer URL enumeration
section  41, level 1              Gatherer enumeration URL verification
section  42, level 1, 5, 9        Gatherer enumeration for HTTP
section  43, level 1, 5, 9        Gatherer enumeration for Gopher
section  44, level 1, 5           Gatherer enumeration filter routines
section  45, level 1              Gatherer enumeration for FTP
section  46, level 1              Gatherer enumeration for file:// URLs
section  48, level 1, 5           Gatherer enumeration robots.txt stuff
section  60, level 1              Gatherer essence data object processing
section  61, level 1              Gatherer essence database routines
section  62, level 1              Gatherer essence main
section  63, level 1              Gatherer essence type recognition
section  64, level 1              Gatherer essence object summarizing
```

```
section  65, level 1              Gatherer essence object unnesting
section  66, level 1, 2, 5        Gatherer essence post-summarizing
section  67, level 1              Gatherer essence object-ID code
section  69, level 1, 5, 9        Common SOIF template processing
section  70, level 1, 5, 9        Broker registry
section  71, level 1              Broker collection routines
section  72, level 1              Broker SOIF parsing routines
section  73, level 1, 5, 9        Broker registry hash tables
section  74, level 1              Broker storage manager routines
section  75, level 1, 5           Broker query manager routines
section  75, level 4              Broker query_list debugging
section  76, level 1              Broker event management routines
section  77, level 1              Broker main
section  78, level 9              Broker select(2) loop
section  79, level 1, 5, 9        Broker gatherer-id management
section  80, level 1              Common utilities memory management
section  81, level 1              Common utilities buffer routines
section  82, level 1              Common utilities system(3) routines
section  83, level 1              Common utilities pathname routines
section  84, level 1              Common utilities hostname processing
section  85, level 1              Common utilities string processing
section  86, level 1              Common utilities DNS host cache
section 101, level 1              Broker PLWeb indexing engine
section 102, level 1, 2, 5        Broker Glimpse indexing engine
section 103, level 1              Broker Swish indexing engine
```

Symptom

The Gatherer *doesn't pick up all the objects* pointed to by some of my RootNodes.

Solution

The Gatherer places various limits on enumeration to prevent a misconfigured Gatherer from abusing servers or running wildly. See section 4.3 (RootNode specifications) for details on how to override these limits.

Symptom

Local-Mapping did not work for me - it retrieved the objects via the usual remote access protocols.

Solution

A local mapping will fail if:

- the local filename cannot be opened for reading; or,
- the local filename is not a regular file; or,
- the local filename has execute bits set.

So for directories, symbolic links, and CGI scripts, the HTTP server is always contacted. We don't perform URL translation for local mappings. If your URL's have funny characters that must be escaped, then the local mapping will also fail. Add debug option **-D20,1** to understand how local mappings are taking place.

Symptom

Using the –**full-text** option I see a lot of *raw data* in the content summaries, with few keywords I can search.

Solution

At present –**full-text** simply includes the full data content in the SOIF summaries. Using the individual file type summarizing mechanism described in Section 4.5.3 (Customizing the summarizing step) will work better in this regard, but will require you to specify how data are extracted for each individual file type. In a future version of Harvest we will change the Essence –**full-text** option to perform content extraction before including the full text of documents.

Symptom

No indexing terms are being generated in the SOIF summary for the META tags in my HTML documents.

Solution

This probably indicates that your HTML is not syntactically well-formed, and hence the SGML-based HTML summarizer is not able to recognize it. See Section 4.5.2 (Summarizing SGML data) for details and debugging options.

Symptom

Gathered data are *not being updated*.

Solution

The Gatherer does not automatically do periodic updates. See Section 4.7.5 (Periodic gathering and realtime updates) for details.

Symptom

The Gatherer puts *slightly different URLs* in the *SOIF* summaries than I specified in the Gatherer *configuration file*.

Solution

This happens because the Gatherer attempts to put URLs into a canonical format. It does this by removing default port numbers and similar cosmetic changes. Also, by default, Essence (the content extraction subsystem within the Gatherer) removes the standard stoplist.cf types, which includes HTTP-Query (the cgi-bin stuff).

Symptom

There are *no Last-Modification-Time* or *MD5 attributes* in my gatherered SOIF data, so the Broker can't do duplicate elimination.

Solution

If you gather remote, manually-created information, it is pulled into Harvest using "exploders" that translate from the remote format into SOIF. That means they don't have a direct way to fill in the Last-Modification-Time or MD5 information per record. Note also that this will mean one update to the remote records would cause all records to look updated, which will result in more network load for Brokers that collect from this Gatherer's data. As a solution, you can compute MD5s for all objects, and store them as part of the record. Then, when you run the exploder you only generate timestamps for the ones for which the MD5s changed - giving you real last-modification times.

Symptom

The Gatherer substitutes a "%7e" for a "~" in all the user directory URLs.

Solution

The Gatherer conforms to *RFC1738*, which says that a tilde inside a URL should be encoded as "%7e", because it is considered an "unsafe" character.

Symptom

When I search using keywords I know are in a document I have indexed with Harvest, the *document isn't found*.

Solution

Harvest uses a content extraction subsystem called *Essence* that by default does not extract every keyword in a document. Instead, it uses heuristics to try to select promising keywords. You can change what keywords are selected by customizing the summarizers for that type of data, as discussed in Section 4.5.3 (Customizing the type recognition, candidate selection, presentation unnesting, and summarizing steps). Or, you can tell *Essence* to use full text summarizing if you feel the added disk space costs are merited, as discussed in Section 4.7.1 (Setting variables in the Gatherer configuration file).

Symptom

I'm running Harvest on HP-UX, but the `essence` process in the Gatherer *takes too much memory*.

Solution

The supplied regular expression library has memory leaks on HP-UX, so you need to use the regular expression library supplied with HP-UX. Change the *Makefile* in *src/gatherer/essence* to read:

```
REGEX_DEFINE    = -DUSE_POSIX_REGEX
REGEX_INCLUDE   =
REGEX_OBJ       =
REGEX_TYPE      = posix
```

Symptom

I built the configuration files to *customize* how Essence types/content extracts data, but it *uses the standard typing/extracting* mechanisms anyway.

Solution

Verify that you have the **Lib-Directory** set to the *lib/* directory that you put your configuration files. **Lib-Directory** is defined in your Gatherer configuration file.

Symptom

I am having problems *resolving host names* on SunOS.

Solution

In order to gather data from hosts outside of your organization, your system must be able to resolve fully qualified domain names into IP addresses. If your system cannot resolve hostnames, you will see error messages such as "Unknown Host." In this case, either:

- the hostname you gave does not really exist; or
- your system is not configured to use the DNS.

To verify that your system is configured for DNS, make sure that the file */etc/resolv.conf* exists and is readable. Read the resolv.conf(5) manual page for information on this file. You can verify that DNS is working with the `nslookup` command.

Some sites may use Sun Microsystem's Network Information Service (NIS) instead of, or in addition to, DNS. We believe that Harvest works on systems where NIS has been properly configured. The NIS servers (the names of which you can determine from the `ypwhich` command) must be configured to query DNS servers for hostnames they do not know about. See the **-b** option of the `ypxfr` command.

Symptom

I cannot get the Gatherer to work across our *firewall gateway*.

Solution

Harvest only supports retrieving HTTP objects through a proxy. It is not yet possible to request Gopher and FTP objects through a firewall. For these objects, you may need to run Harvest internally (behind the firewall) or on the firewall host itself.

If you see the "Host is unreachable" message, these are the likely problems:

- your connection to the Internet is temporarily down due to a circuit or routing failure; or
- you are behind a firewall.

If you see the "Connection refused" message, the likely problem is that you are trying to connect with an unused port on the destination machine. In other words, there is no program listening for connections on that port.

The Harvest gatherer is essentially a WWW client. You should expect it to work the same as any Web browser.

Chapter 5

The Broker

5.1 Overview

The Broker retrieves and manages indexing information from Gatherers and other Brokers, and provides a WWW query interface to the indexing information.

5.2 Basic setup

The Broker is automatically started by the `RunHarvest` command. Other relevant commands are described in Section 3.7 (Starting up the system: RunHarvest and related commands).

In the current section we discuss various ways users can customize and tune the Broker, how to administrate the Broker, and the various Broker programming interfaces.

As suggested in Figure 2 (1), the Broker uses a flexible indexing interface that supports a variety of indexing subsystems. The default Harvest Broker uses *Glimpse* as indexer, but other indexers such as Swish, and WAIS (both *freeWAIS and commercial WAIS* `<ftp://ftp.cnidr.org/pub/software/freewais/>`), also work with the Broker (see Section 5.8 (Using different index/search engines with the Broker)).

To create a new Broker, run the `CreateBroker` program. It will ask you a series of questions about how you'd like to configure your Broker, and then automatically create and configure it. To start your Broker, use the `RunBroker` program that `CreateBroker` generates. The Broker should be started when your system reboots. To prevent a collection while starting the broker, use the **-nocol** option. There are a number of ways you can customize or tune the Broker, discussed in Sections 5.7 (Tuning Glimpse indexing in the Broker) and 5.8 (Using different index/search engines with the Broker). You may also use the `RunHarvest` command, discussed in Section 3.7 (Starting up the system: RunHarvest and related commands), to create both a Broker and a Gatherer.

5.3 Querying a Broker

The Harvest Broker can handle many types of queries. The queries handled by a particular Broker depend on what index/search engine is being used inside of it (e.g., WAIS does not support some of the queries that Glimpse does). In this section we describe the full syntax. If a particular Broker does not support a certain type of query, it will return an error when the user requests that type of query.

The simplest query is a single keyword, such as:

```
lightbulb
```

Searching for common words (like "computer" or "html") may take a lot of time.

Particularly for large Brokers, it is often helpful to use more powerful queries. Harvest supports many different index/search engines, with varying capabilities. At present, our most powerful (and commonly used) search engine is *Glimpse*, which supports:

- case-insensitive and case-sensitive queries;
- matching parts of words, whole words, or multiple word phrases (like "resource discovery");
- Boolean (AND/OR) combinations of keywords;
- approximate matches (e.g., allowing spelling errors);
- structured queries (which allow you to constrain matches to certain attributes);
- displaying matched lines or entire matching records (e.g., for citations);
- specifying limits on the number of matches returned; and
- a limited form of regular expressions (e.g., allowing "wild card" expressions that match all words ending in a particular suffix).

The different types of queries (and how to use them) are discussed below. Note that you use the same syntax regardless of what index/search engine is running in a particular Broker, but that not all engines support all of the above features. In particular, some of the Brokers use WAIS, which sometimes searches faster than Glimpse but supports only Boolean keyword queries and the ability to specify result set limits.

The different options - case-sensitivity, approximate matching, the ability to show matched lines vs. entire matching records, and the ability to specify match count limits - can all be specified with buttons and menus in the Broker query forms.

A structured query has the form:

```
tag-name : value
```

where *tag-name* is a Content Summary attribute name, and *value* is the search value within the attribute. If you click on a Content Summary, you will see what attributes are available for a particular Broker. A list of common attributes is shown in Section 7.2 (List of common SOIF attribute names).

Keyword searches and structured queries can be combined using Boolean operators (AND and OR) to form complex queries. Lacking parentheses, logical operation precedence is based left to right. For multiple word phrases or regular expressions, you need to enclose the string in double quotes, e.g.,

```
"internet resource discovery"
```

or

```
"discov.*"
```

Double quotes should also be used when searching for non-alphanumeric characters.

5.3.1 Example queries

Simple keyword search query:

Arizona

This query returns all objects in the Broker containing the word *Arizona*.

Boolean query:

Arizona AND desert

This query returns all objects in the Broker that contain both words anywhere in the object in any order.

Phrase query:

"*Arizona desert*"

This query returns all objects in the Broker that contain *Arizona desert* as a phrase. Notice that you need to put double quotes around the phrase.

Boolean queries with phrases:

"*Arizona desert*" *AND windsurfing*

This query returns all objects in the Broker that contain *Arizona desert* as a phrase and the word windsurfing.

Simple Structured query:

Title : windsurfing

This query returns all objects in the Broker where the *Title* attribute contains the value *windsurfing*.

Complex query:

"*Arizona desert*" *AND (Title : windsurfing)*

This query returns all objects in the Broker that contain the phrase *Arizona desert* and where the *Title* attribute of the same object contains the value *windsurfing*.

5.3.2 Regular expressions

Some types of regular expressions are supported by Glimpse. A regular expression search can be much slower that other searches. The following is a partial list of possible patterns. (For more details see the *Glimpse documentations*.)

- *^joe* will match "joe" at the beginning of a line.
- *joe$* will match "joe" at the end of a line.
- *[a-ho-z]* matches any character between "a" and "h" or between "o" and "z".
- . matches any single character except newline.
- *c** matches zero or more occurrences of the character "c".
- *.** matches any number of characters except newline.
- * matches the character "*". (\ escapes any of the above special characters.)

Regular expressions are currently limited to approximately 30 characters, not including meta characters. Regular expressions will generally not cross word boundaries (because only words are stored in the index). So, for example, "*lin.*ing*" will find “linking” or “flinching,” but not “linear programming.”

5.3.3 Query options selected by menus or buttons

The query page may have following checkboxes to allow some control of the query specification.

Case insensitive:

By selecting this checkbox the query will become case insensitive (lower case and upper case letters don't differ). Otherwise, the query will be case sensitive. The default is case insensitive.

Keywords match on word boundaries:

By selecting this checkbox, keywords will match on word boundaries. Otherwise, a keyword will match part of a word (or phrase). For example, "network" will match “networking”, "sensitive" will match “insensitive”, and "Arizona desert" will match “Arizona desertness”. The default is to match keywords on word boundaries.

Number of errors allowed:

Glimpse allows the search to contain a number of errors. An error is either a deletion, insertion, or substitution of a single character. The Best Match option will find the match(es) with the least number of errors. The default is 0 (zero) errors.

Note: The previous three options do not apply to attribute names. Attribute names are always case insensitive and allow no errors.

5.3.4 Filtering query results

Harvest allows to filter the results of a query by any query term using any attribute defined in the 7.2 (List of common SOIF attribute names). This is done by defining **filter** parameters in the query form. It is possible to define more that one filter parameter; they will be concatenated by boolean **AND**. Filter parameters consist of two parts, separated by the pipe symbol “|”. The first part is a query expression which is attached to the user query using **AND** before sending the request to the broker. The optional second part is a HTML text that shall be displayd on the results page, to give the user some information on the applied filter.

Example:

```
<SELECT NAME="filter">
<OPTION VALUE=''>No Filter
<OPTION VALUE='uri: "xyz\.edu"|Seach only xyz.edu'>Search xyz.edu only
<OPTION VALUE='type: html|HTML documents only'>Search HTML documents only
</SELECT>
```

The first option returns an unfiltered output. The second option returns only pages found on pages with “xyz.edu” in their URL. The third option returns only HTML-documents. See the advanced search page of the broker for more examples.

5.3.5 Result set presentation

The query page may have following checkboxes allow some control of presentation of the query return.

Display matched lines (from content summaries):

By selecting this checkbox, the result set presentation will contain the lines of the Content Summary that matched the query. Otherwise, the matched lines will not be displayed. The default is to display the matched lines.

Display object descriptions (if available):

Some objects have short, one-line descriptions associated with them. By selecting this checkbox, the descriptions will be presented. Otherwise, the object descriptions will not be displayed. The default is to display object descriptions.

Display links to indexed content summary:

This checkbox allows you to set whether links to the indexed content summaries are displayed or not. The default is not to display links to inexed content summaries.

5.4 Customizing the Broker's Query Result Set

It is possible for the Harvest administrator to customize how the Broker query result set is generated, by modifying a configuration file that is interpreted by the `search.cgi` Perl program at query result time.

`search.cgi` allows you to customize almost every aspect of its HTML output. The file *$HARVEST_HOME/cgi-bin/lib/search.cf* contains the default output definitions. Individual brokers can be customized by creating a similar file which overrides the default definitions.

5.4.1 The search.cf configuration file

Definitions are enclosed within SGML-like beginning and ending tags. For example:

```
<HarvestUrl>
http://harvest.sourceforge.net/
</HarvestUrl>
```

The last newline character is removed from each definition, so that the above becomes the string "http://harvest.sourceforge.net/."

Variable substitution occurs on every definition before it is output. A number of specific variables are defined by `search.cgi` which can be used inside a definition. For example:

```
<BrokerLoad>
Sorry, the Broker at <STRONG>$host, port $port</STRONG>
is currently too heavily loaded to process your request.
Please try again later.<P>
</BrokerLoad>
```

When this definition is printed out, the variables *$host* and *$port* would be replaced with the hostname and port of the broker.

Defined Variables

The following variables are defined as soon as the query string is processed. They can be used before the broker returns any results.

```
$maxresult    The maximum number of matched lines to be returned
$host         The broker hostname
$port         The broker port
$query        The query string entered by the user
$bquery       The whole query string sent to the broker
```

These variables are defined for each matched object returned by the broker.

```
$objectnum   The number of the returned object
$desc        The description attribute of the matched object
$opaque      ALL the matched lines from the matched object
$url         The original URL of the matched object
$A           The access method of $url (e.g.: http)
$H           The hostname (including port) from $url
$P           The path part of $url
$D           The directory part of $P
$F           The filename part of $P
$cs_url      The URL of the content summary in the broker database
$cs_a        Access part of $cs_url
$cs_h        Hostname part of $cs_url
$cs_p        Path part of $cs_url
$cs_d        Directory part of $cs_p
$cs_f        Filename part of $cs_p
```

List of Definitions

Below is a partial list of definitions. A complete list can be found in the search.cf file. Only definitions likely to be customized are described here.

<Timeout>

Timeout value for `search.cgi`. If the broker doesn't respond within this time, `search.cgi` will exit.

<ResultHeader>

The first part of the result page. Should probably contain the HTML **<TITLE>** element and the user query string.

<ResultTrailer>

The last part of the result page. The default has URL references to the broker home page and the Harvest project home page.

<ResultSetBegin>

This is output just before looping over all the matched objects.

<ResultSetEnd>

This is output just after ending the loop over matched objects.

<PrintObject>

This definition prints out a matched object. It should probably include the variables *$url*, *$cs_url*, *$desc*, and *$opaque*.

<EndBrokerResults>

Printed between **<ResultSetEnd>** and **<ResultTrailer>** if the query was successful. Should probably include a count of matched objects and/or matched lines.

<FailBrokerResults>

Similar to **<EndBrokerResults>** but prints if the broker returns an error in response to the query.

<ObjectNumPrintf>

A `printf` format string for the object number (*$objectnum*).

<TruncateWarning>

Prints a warning message if the result set was truncated at the maximum number of matched lines.

These following definitions are somewhat different because they are evaluated as Perl instructions rather than strings.

<MatchedLineSub>

Evaluated for every matched line returned by the broker. Can be used to indent matched lines or to remove the leading "Matched line" and attribute name strings.

<InitFunction>

Evaluated near the beginning of the `search.cgi` program. Can be used to set up special variables or read data files.

<PerObjectFunction>

Evaluated for each object just before **<PrintObject>** is called.

<FormatAttribute>

Evaluated for each SOIF attribute requested for matched objects (see Section 5.4.4 (Displaying SOIF attributes in results)). *$att* is set to the attribute name, and *$val* is set to the attribute value.

5.4.2 Example search.cf customization file

The following definitions demonstrate how to change the `search.cgi` output. The **<PerObjectFunction>** ensures that the description is not empty. It also prepends the string "matched data:" before any matched lines. The **<PrintObject>** specification prints the object number, description, and indexing data all on the first line. The description is wrapped around HMTL anchor tags so that it is a link to the object originally gathered. The words "indexing data" are a link to the displaySOIF program which will format the content summary for HTML browsers. The object number is formatted as a number in parenthesis such that the whole thing takes up four spaces.

The **<MatchedLineSub>** definition includes four substitution expressions. The first removes the words "Matched line:" from the beginning of each matched line. The second removes SOIF attributes of the form "*partial-text{43}:*" from the beginning of a line. The third displays the attribute names (e.g. *partial-text#*) in italics. The last expression indents each line by five spaces to align it with the description line. The definition for <**EndBrokerResults**> slightly modifies the report of how many objects were matched.

```
# Demo to show some of the customization features for the Harvest output
# More information can be found in the manual at:
# http://harvest.sourceforge.net/harvest/doc/html/manual.html


# The PerObjectFunction is Perl code evaluated for every hit
<PerObjectFunction>
# Create description
# Is the descriptions provided by Harvest very short (e.g. missing <TITLE>)?
if (length($desc) < 5) {
  # Yes: use filename ($F) instead
  $description = "<I>File:</I> $F";
} else {
  # No: use description provided by Harvest
  $description = $desc;
}

# Format matched lines ("opaque data") if data is present
if ($opaque ne '') {
  $opaque = "<strong>matched lines:</strong><BR>$opaque"
}
</PerObjectFunction>


# PrintObject defines the apperance of hits
<PrintObject>
$objectnum <A HREF="$url"><STRONG>$description</STRONG></A> \
[<A HREF="$cs_a://$cs_h/Harvest/cgi-bin/displaySOIF.cgi?object=$cs_p">\
indexing data</A>]
<pre>
     $opaque
</pre>\n
</PrintObject>


# Format the appearance of the hit number
<ObjectNumPrintf>
(%2d)
</ObjectNumPrintf>


# Format the appearance of every matched line
<MatchedLineSub>
s/^Matched line: *//;            # Remove "Matched line:"
s/^([\w-]+# )[\w-]+{\d+}:\t/\1/; # Remove SOIF attributes of the form "partial-text{43}:"
s/^([\w-]+#)/<I>\1<\/I>/;        # Format attribute names as italics
s/^.*/     $&/;                  # Add spaces to indent text
</MatchedLineSub>
```

```
# Modifies the report of how many objects were matched
<EndBrokerResults>
<STRONG>Found $nopaquelines matched lines, $nobjects objects.</STRONG>
<P>\n
</EndBrokerResults>
```

5.4.3 Integrating your customized configuration file

The `search.cgi` configuration files are kept in *$HARVEST_HOME/cgi-bin/lib*. The name of a customized file is listed in the *query.html* form, and passed as an option to the `search.cgi` program.

The simplest way to specify the customized file is by placing an <**INPUT**> tag in the HTML form:

```
<INPUT TYPE="hidden" NAME="brokerqueryconfig" VALUE="custom.cf">
```

Another way is to allow users to select from different customizations with a <**SELECT**> list:

```
<SELECT NAME="brokerqueryconfig">
<OPTION VALUE=""> Default
<OPTION VALUE="custom1.cf"> Customized
<OPTION VALUE="custom2.cf" SELECTED> Highly Customized
</SELECT>
```

5.4.4 Displaying SOIF attributes in results

It is possible to request SOIF attributes from the HTML query form. A simple approach is to include a select list in the query form:

```
<SELECT MULTIPLE NAME="attribute">
<OPTION VALUE="title">
<OPTION VALUE="author">
<OPTION VALUE="date">
<OPTION VALUE="subject">
</SELECT>
```

In this manner, the user may control which attributes get displayed. The layout of these attributes when the results are displayed in HTML is controlled by the <**FormatAttribute**> specification in the *search.cf* file described in Section 5.4.1 (The search.cf configuration file).

5.5 World Wide Web interface description

To allow Web browsers to easily interface with the Broker, we implemented a World Wide Web interface to the Broker's query manager and administrative interfaces. This WWW interface, which includes several HTML files and a few programs that use the *Common Gateway Interface* (CGI), consists of the following:

- HTML files that use *Forms* support to present a graphical user interface (GUI) to the user;
- CGI programs that act as a gateway between the user and the Broker; and
- Help files for the user.

Users go through the following steps when using a Broker to locate information:

1. The user issues a query to the Broker.
2. The Broker processes the query, and returns the query results to the user.
3. The user can then view content summaries from the result set, or access the URLs from the result set directly.

To provide a WWW-queryable interface, the Broker needs to run in conjunction with an HTTP server. Section 3.5 (Additional installation for the Harvest Broker) describes how to configure your HTTP server to work with Harvest.

You can run the Broker on a different machine than your HTTP server runs on, but if you want users to be able to view the Broker's content summaries then the Broker's files will need to be accessible to your HTTP server. You can NFS mount those files or manually copy them over. You'll also need to change the *Brokers.cf* file to point to the host that is running the Broker.

5.5.1 HTML files for graphical user interface

`CreateBroker` creates some HTML files to provide GUIs to the user:

query.html

Contains the GUI for the query interface. `CreateBroker` will install different *query.html* files for Glimpse, Swish, and WAIS, since each subsystem requires different defaults and supports different functionality (e.g., WAIS doesn't support approximate matching like Glimpse). This is also the "home page" for the Broker and a link to this page is included at the bottom of all query results.

admin.html

Contains the GUI for the administrative interface. This file is installed into the *admin* directory of the Broker.

Brokers.cf

Contains the hostname and port information for the supported brokers. This file is installed into the *$HARVEST_HOME/brokers* directory. The *query.html* file uses the value of the "broker" FORM tag to pass the name of the broker to `search.cgi` which in turn retrieves the host and port information from *Brokers.cf*.

5.5.2 CGI programs

When you install the WWW interface (see Section 5 (The Broker)), a few programs are installed into your HTTP server's */Harvest/cgi-bin* directory:

`search.cgi`

This program takes the submitted query from *query.html*, and sends it to the specified Broker. It then retrieves the query results from the Broker, formats them in HTML, and sends the result set in HTML to the user.

`displaySOIF.cgi`

This program displays the content summaries from the Broker.

`BrokerAdmin.pl.cgi`

This program will take the submitted administrative command from *admin.html* and send it to the appropriate Broker. It retrieves the result of the command from the Broker and displays it to the user.

5.5.3 Help files for the user

The WWW interface to the Broker includes a few help files written in HTML. These files are installed on your HTTP server in the */Harvest/brokers* directory when you install the broker (see Section 5 (The Broker)):

queryhelp.html

Provides a tutorial on constructing Broker queries, and on using the *query.html* forms. *query.html* has a link to this help page.

adminhelp.html

Provides a tutorial on submitting Broker administrative commands using the *admin.html* form. *admin.html* has a link to this help page.

soifhelp.html

Provides a brief description of SOIF.

5.6 Administrating a Broker

Administrators have two basic ways for managing a Broker: through the *broker.conf* and *Collection.conf* configuration files, and through the interactive administrative interface. The interactive interface controls various facilities and operating parameters within the Broker. We provide a HTML interface page for these administrative commands. See Section 5.9 (Collector interface description: Collection.conf) for additional information on the Broker administrative and collector interfaces.

The *broker.conf* file is a list of variable names and their values, which consists of information about the Broker (such as the directory in which it lives) and the port on which it runs. The *Collection.conf* file (see Section 5.9 (Collector interface description: Collection.conf) for an example) is a list of collection points from which the Broker collects its indexing information. The `CreateBroker` program automatically generates both of these configuration files. You can manually edit these files if needed.

The `CreateBroker` program also creates the *admin.html* file, which is the WWW interface to the Broker's administrative commands. Note that all administrative commands require a password as defined in *broker.conf*.

Note: Changes to the Broker configuration are not saved when the Broker is restarted. Permanent changes to the Broker configuration should be made by manually editing the *broker.conf* file.

The administrative interface created by `CreateBroker` has the following window fields:

```
Command        Select an administrative command.  See below for a
               description of the commands.
Parameters     Specify parameters for those commands that need them.
Password       The administrative password.
Broker Host    The host where the broker is running.
Broker Port    The port where the broker is listening.
```

The administrative interface created by `CreateBroker` supports the following commands:

Add objects by file:

Add object(s) to the Broker. The parameter is a list of filenames that contain SOIF object to be added to the Broker.

Close log:

Flush all accumulated log information and close the current log file. Causes the Broker to stop logging. No parameters.

Compress Registry:

Performs garbage collection on the Registry file. No parameters.

Delete expired objects:

Deletes any object from the Broker whose *Time-to-Live* has expired. No parameters.

Delete objects by query:

Deletes any object(s) that matches the given query. The parameter is a query with the same syntax as user queries. Query flags are currently unsupported.

Delete objects by oid:

Deletes the object(s) identified by the given OID numbers. The parameter is a list of OID numbers. The OID numbers can be obtained by using the `dumpregistry` command.

Disable log type:

Disables logging information about a particular type of event. The parameter is an event type. See Enable log type for a list of events.

Enable log type:

Enables logging information about a particular type of events. The parameter is the name of an event type. Currently, event types are limited to the following:

```
Update                  Log updated objects.
Delete                  Log deleted objects.
Refresh                 Log refreshed objects.
Query                   Log user queries.
Query-Return            Log objects returned from a query.
Cleaned                 Log objects removed by the cleaner.
Collection              Log collection events.
Admin                   Log administrative events.
Admin-Return            Log the results of administrative events.
Bulk-Transfer           Log bulk transfer events.
Bulk-Return             Log objects sent by bulk transfers.
Cleaner-On              Log cleaning events.
Compressing-Registry    Log registry compression events.
All                     Log all events.
```

Flush log:

Flush all accumulated log information to the current log file. No parameters.

Generate statistics:

Generates some basic statistics about the Broker object database. No parameters.

Index changes:

Index only the objects that have been added recently. No parameters.

Index corpus:

Index the *entire* object database. No parameters.

Open log:

Open a new log file. If the file does not exist, create a new one. The parameter is the name (relative to the broker) of a file to use for logging.

Restart server:

Force the broker to reread the Registry and reindex the corpus. This does not actually kill the broker process. No parameters.

Rotate log file:

Rotates the current log file to LOG.YYYYMMDD. Opens a new log file. No parameters.

Set variable:

Sets the value of a broker configuration variable. Takes two parameters, the name of a configuration variable and the new value for the variable. The configuration variables that can be set are those that occur in the *broker.conf* file. The change only is valid until the broker process dies.

Shutdown server:

Cleanly shutdown the Broker. No parameters.

Start collection:

Perform collections. No parameters.

Delete older objects of duplicate URLs:

Occasionally a broker may end up with multiple summarizes for individual URLs. This can happen when the Gatherer changes its description, hostname, or port number. Use this command to search the broker for duplicated URLs. When two objects with the same URL are found, the object with the least-recent timestamp is removed.

5.6.1 Deleting unwanted Broker objects

If you build a Broker and then decide not to index some of that data (e.g., you decide it would make sense to split it into two different Brokers, each targetted to a different community), you need to change the Gatherer's configuration file, rerun the Gatherer, and then let the old objects time out in the Broker (since the Broker and Gatherer maintain separate databases). If you want to clean out the Broker's data sooner than that you can use the Broker's administrative interface in one of three ways:

1. Use the 'Remove object by name' command. This is only reasonable if you have a small number of objects to remove in the Broker.

2. Use the 'Remove object by query'. This might be the best option if, for example, you can construct a regular expression based on the URLs you want to remove.

3. Shutdown the server, manually remove the Broker's *objects/** files, and then restart the Broker. This is easiest, although if you have a large number of objects it will take longer to rebuild the index. A simple way to accomplish this is by "rebooting" the Broker by deleting all the current objects, and doing a full collection, as follows:

```
% mv objects objects.old
% rm -rf objects.old &
% broker ./admin/broker.conf -new
```

After removing objects, you should use the *Index corpus* command.

5.6.2 Command-line Administration

It is possible to perform administrative functions by using the `brkclient` program from the command-line and shell scripts. For example, to force a collection, run:

```
% brkclient localhost 8501 '#ADMIN #Password secret #collection'
```

See your broker's raw *admin.html* file for a complete list of administrative commands.

5.7 Tuning Glimpse indexing in the Broker

The Glimpse indexing system can be tuned in a variety of ways to suit your particular needs. Probably the most noteworthy parameter is indexing granularity, for which Glimpse provides three options: a tiny index (2-3% of the total size of all files – your mileage may vary), a small index (7-8%), and a medium-size index (20-30%). Search times are better with larger indexes. By changing the **GlimpseIndex-Option** in your Broker's *broker.conf* file, you can tune Glimpse to use one of these three indexing granularity options. By default, **GlimpseIndex-Option** builds a medium-size index using the `glimpseindex` program.

Note also that with Glimpse it is much faster to search with "show matched lines" turned off in the Broker query page.

Glimpse uses a "stop list" to avoid indexing very common words. This list is not fixed, but rather computed as the index is built. For a medium-size index, the default is to put any word that appears at least 500 times per Mbyte (on the average) in the stop-list. For a small-size index, the default is words that appear in at least 80% of all files (unless there are fewer than 256 files, in which case there is no stop-list). Both defaults can be changed using the **-S** option, which should be followed by the new number (average per Mbyte when **-b** indexing is used, or % of files when **-o** indexing is used). Tiny-size indexes do not maintain a stop-list (their effect is minimal).

`glimpseindex` includes a number of other options that may be of interest. You can find out more about these options (and more about Glimpse in general) in the *Glimpse documentations*. If you'd like to change how the Broker invokes the `glimpseindex` program, then edit the *src/broker/Glimpse/index.c* file from the Harvest source distribution.

5.7.1 The glimpseserver program

The Glimpse system comes with an auxiliary server called `glimpseserver`, which allows indexes to be read into a process and kept in memory. This avoids the added cost of reading the index and starting a large process for each search. `glimpseserver` is automatically started each time you run the Broker, or reindex the Broker's corpus. If you do not want to run `glimpseserver`, then set **GlimpseServer-Host** to "false" in your *broker.conf*.

5.8 Using different index/search engines with the Broker

By default, Harvest uses the Glimpse index/search subsystem. However, Harvest defines a flexible indexing interface, to allow Broker administrators to use different index/search subsystems to accommodate domain-specific requirements. For example, it might be useful to provide a relational database back-end.

At present we distribute code to support an interface to both the free and the commercial WAIS index/search engines, Glimpse, and Swish.

Below we discuss how to use other index/search engine instead of Glimpse in the Broker, and provide some brief discussion of how to integrate a new index/search engine into the Broker.

5.8.1 Using Swish as an indexer

Harvest includes support for using Swish as indexing engine with the Broker. Swish is a nice alternative to Glimpse if you need faster search support and are willing to lose the more powerful query features. It also is an alternative in cases of trouble with Glimpse' copyright status.

To use Swish with an existing Broker, you need to change the *Indexer-Type* variable in *broker.conf* to "Swish".

You can also specify that you want to use Swish for a Broker, when you use the `RunHarvest` command by running: `RunHarvest -swish`.

5.8.2 Using WAIS as an indexer

Support for using WAIS (both freeWAIS and WAIS Inc.'s index/search engine) as the Broker's indexing and search subsystem is included in the Harvest distribution. WAIS is a nice alternative to Glimpse if you need faster search support and are willing to lose the more powerful query features.

To use WAIS with an existing Broker, you need to change the *Indexer-Type* variable in *broker.conf* to "WAIS"; you can choose among the WAIS variants by setting the *WAIS-Flavor* variable in *broker.conf* to "Commercial-WAIS", "freeWAIS", or "WAIS". Otherwise, `CreateBroker` will ask you if you want to use WAIS, and where the WAIS programs (`waisindex`, `waissearch`, `waisserver`, and with the commercial version of WAIS `waisparse`) are located. When you run the Broker, a WAIS server will be started automatically after the index is built.

You can also specify that you want to use WAIS for a Broker, when you use the `RunHarvest` command by running: `RunHarvest -wais`.

5.9 Collector interface description: Collection.conf

The Broker retrieves indexing information from Gatherers or other Brokers through its *Collector* interface. A list of collection points is specified in the *admin/Collection.conf* configuration file. This file contains a collection point on each line, with 4 fields. The first field is the host of the remote

Gatherer or Broker, the second field is the port number on that host, the third field is the collection type, and the forth field is the query filter or – if there is no filter.

The Broker supports various types of collections as described below:

```
Type  Remote Process       Description      Compression?
--------------------------------------------------------
  0     Gatherer    Full collection each time     No
  1     Gatherer    Incremental collections       No
  2     Gatherer    Full collection each time     Yes
  3     Gatherer    Incremental collections       Yes
  4     Broker      Full collection each time     No
  5     Broker      Incremental collections       No
  6     Broker      Collection based on a query   No
  7     Broker      Incremental based on a query  No
```

The query filter specification for collection types 6 and 7 contains two parts: the **–QUERY keywords** portion and an optional **–FLAGS flags** portion. The **–QUERY** portion is passed on to the Broker as the keywords for the query (the keywords can be any Boolean and/or structured query); the **–FLAGS** portion is passed on to the Broker as the indexer-specific flags to the query. The following table shows the valid indexer-specific flags for the supported indexers:

```
Indexer         Flag                            Description
-----------------------------------------------------------------------------
All:            #desc                           Show Description Lines

Glimpse:        #index case insensitive         Case Insensitive
                #index case sensitive           Case sensitive
                #index error number             Allow "number" errors
                #index matchword                Matches on word boundaries
                #index maxresult number         Allow max of "number" results
                #opaque                         Show matched lines

Wais:           #index maxresult number         Allow max of "number" results
                #opaque                         Show scores and rankings
```

The following is an example *Collection.conf*, which collects information from 2 Gatherers (one compressed incrementals and the other uncompressed full transfers), and collects information from 3 Brokers (one incrementally based on a timestamp, and the others using query filters):

```
gatherer-host1.foo.com 8500 3 --
gatherer-host2.foo.com 8500 0 --
broker-host1.foo.com   8501 5 --
broker-host2.foo.com   8501 6 --QUERY (URL : document) AND gnu
broker-host3.foo.com   8501 7 --QUERY Harvest --FLAGS #index case sensitive
```

5.10 Troubleshooting

Symptom

The Broker is running but always returns *empty query results*.

Solution

Look at the log messages in the broker.out file in the Broker's directory for error messages. If your Broker didn't index the data, use the administrative interface to force the Broker to build the index (see Section 5.6 (Administrating a Broker)).

Symptom

When I query my Broker, I get a `"500 Server Error"`.

Solution

Generally, the "500" errors are related to a CGI program not working correctly or a misconfigured httpd server. Make sure that the userid running the HTTP server has access to the Harvest cgi-bin directory and the Perl include files in *$HARVEST_HOME/lib*. Refer to Section 3.5 (Additional installation for the Harvest Broker) for further details.

Symptom

I see *duplicate documents* in my Broker.

Solution

The Broker performs duplicate elimination based on a combination of MD5 checksums and Gatherer-Host, Name, Version. Therefore, you can end up with duplicate documents if your Broker collects from more than one Gatherer, each of which gathers from the (a subset of) the same URLs. (As an aside, the reason for this notion of duplicate elimination is to allow a single Broker to contain several different SOIF objects for the same URL, but summarized in different ways.)

Two solutions to the problem are:

1. Run your Gatherers on the same host.
2. Remove the duplicate URLs in a customized version of the `search.cgi` program by doing a string comparison of the URLs.

Symptom

The Broker takes a *long time* and does not answer queries.

Solution

Some queries are quite expensive, because they involve a great deal of I/O. For this reason we modified the Broker so that if a query takes longer than 5 minutes, the query process is killed. The best solution is to use a less expensive query, for example by using less common keywords.

Symptom

Some of the *query options* (such as structured or case sensitive queries) *aren't working*.

Solution

This usually means you are using an index/search engine that does not support structured queries (like the current Harvest support for commercial WAIS). If you are setting up your own Broker (rather than using someone else's Broker), see Section 5.8 (Using different index/search engines with the Broker) for details on how to switch to other index/search engines. Or, it could be that your `search.cgi` program is an old version and should be updated.

Symptom

I get *syntax errors* when I specify queries.

Solution

Usually this means you did not use double quotes where needed. See Section 5.3 (Querying a Broker).

Symptom

When I submit a query, I get an *answer faster than I can believe* it takes to perform the query, and the answer contains *garbage data*.

Solution

This probably indicates that your `httpd` is misconfigured. A common case is not putting the 'ScriptAlias' before the 'Alias' in your *conf/httpd.conf* file, when running the Apache `httpd`. See Section 3.5 (Additional installation for the Harvest Broker).

Symptom

When I make *changes* to the Broker configuration via the *administration interface*, they are *lost* after the Broker is restarted.

Solution

The Broker administration interface does not save changes across sessions. Permanent changes to the Broker configuration should be done through the *broker.conf* file.

Symptom

My Broker is *running very slowly*.

Solution

Performance tuning can be complicated, but the most likely problem is that you are running on a machine with insufficient RAM, and paging a lot because the query engine kicks pages out in order to access the needed index and data files. (In UNIX the disk buffer cache competes with program and data pages for memory.)

A simple way to tell is to run "vmstat 5" in one window, and after a couple of lines of output, issue a query from another window. This will print a line of measurements about the virtual memory status of your machine every 5 seconds. In particular, look at the "pi" and "po" columns. If the numbers suddenly jump into the 500-1,000 range after you issue the query, you are paging a lot.

Note that paging problems are accentuated by running simultaneous memory-intensive or disk I/O-intensive programs on your machine. Simultaneous queries to a single Broker should not cause a paging problem, because the Broker processes the queries sequentially.

It is best to run Brokers on an otherwise mostly unused machine with at least 128 MB of RAM (or more, if the above "vmstat" experiment indicates you are paging alot).

One other performance enhancer is to run an *httpd-accelerator* on your Broker machine, to intercept queries headed for your Broker. While it will not cache the results of queries, it will reduce load on the machine because it provides a very efficient means of returning results in the case of concurrent queries. Without the accelerator the results are sent back by a `search.cgi` UNIX process per query, and inefficiently time sliced by the UNIX kernel. With an accelerator the `search.cgi` processes exit quickly, and let the accelerator send the results back to the concurrent users. The accelerator will also reduce load for (non-query) retrievals of data from your httpd server.

Chapter 6

Programs and layout of the installed Harvest software

6.1 $HARVEST_HOME

The top directory of where you installed Harvest is known as *$HARVEST_HOME*. By default, *$HARVEST_HOME* is */usr/local/harvest*. The following files and directories are located in *$HARVEST_HOME*:

```
RunHarvest*     brokers/     gatherers/     tmp/
bin/            cgi-bin/     lib/
```

`RunHarvest` is the script used to create and run Harvest servers (see Section 3.7 (Starting up the system: RunHarvest and related commands)). `RunHarvest` has the same command line syntax as `Harvest`.

6.2 $HARVEST_HOME/bin

The *$HARVEST_HOME/bin* directory only contains programs that users would normally run directly. All other programs (e.g., individual summarizers for the Gatherer) as well as Perl library code are in the *lib* directory. The *bin* directory contains the following programs:

`CreateBroker`

Creates a Broker.

Usage: `CreateBroker [skeleton-tree [destination]]`

`Gatherer`

Main user interface to the Gatherer. This program is run by the `RunGatherer` script found in a Gatherer's directory.

Usage: `Gatherer [-manual|-export|-debug] file.cf`

`Harvest`

The program used by `RunHarvest` to create and run Harvest servers as per the user's description.

Usage: `Harvest [flags]`

where flags can be any of the following:

```
-novice        Simplest Q&A.  Mostly uses the defaults.
-glimpse       Use Glimpse for the Broker. (default)
-swish         Use Swish for the Broker.
-wais          Use WAIS for the Broker.
-dumbtty       Dumb TTY mode.
-debug         Debug mode.
-dont-run      Don't run the Broker or the Gatherer.
-fake          Doesn't build the Harvest servers.
-protect       Don't change the umask.
```

`broker`

The Broker program. This program is run by the `RunBroker` script found in a Broker's directory. Logs messages to both *broker.out* and to *admin/LOG*.

Usage: `broker [broker.conf file] [-nocol]`

`gather`

The client interface to the Gatherer.

Usage: `gather [-info] [-nocompress] host port [timestamp]`

6.3 $HARVEST_HOME/brokers

The *$HARVEST_HOME/brokers* directory contains images and logos in *images* directory, some basic tutorial HTML pages, and the skeleton files that `CreateBroker` uses to construct new Brokers. You can change the default values in these created Brokers by editing the files in *skeleton*.

6.4 $HARVEST_HOME/cgi-bin

The *$HARVEST_HOME/cgi-bin* directory contains the programs needed for the WWW interface to the Broker (described in Section 5.5.2 (CGI programs)) and configuration files for `search.cgi` in *lib* directory.

6.5 $HARVEST_HOME/gatherers

The *$HARVEST_HOME/gatherers* directory contains example Gatherers discussed in Section 8 (Gatherer Examples). `RunHarvest`, by default, will create the new Gatherer in this directory.

6.6 $HARVEST_HOME/lib

The *$HARVEST_HOME/lib* directory contains number of Perl library routines and other programs needed by various parts of Harvest, as follows:

chat2.pl, ftp.pl, socket.ph

Perl libraries used to communicate with remote FTP servers.

dateconv.pl, lsparse.pl, timelocal.pl

Perl libraries used to parse `ls` output.

`ftpget`

Program used to retrieve files and directories from FTP servers.

Usage: `ftpget [-htmlify] localfile hostname filename A,I username password`

`gopherget.pl`

Perl program used to retrieve files and menus from Gopher servers.

Usage: `gopherget.pl localfile hostname port command`

`harvest-check.pl`

Perl program to check whether gatherers and brokers are up.

Usage: `harvest-check.pl [-v]`

`md5`

Program used to compute MD5 checksums.

Usage: `md5 file [...]`

`newsget.pl`

Perl program used to retrieve USENET articles and group summaries from NNTP servers.

Usage: `newsget.pl localfile news-URL`

soif.pl, soif-mem-efficient.pl

Perl library used to process SOIF.

`urlget`

Program used to retrieve a URL.

Usage: `urlget URL`

`urlpurge`

Program to purge the local disk URL cache used by `urlget` and the Gatherer.

Usage: `urlpurge`

6.7 $HARVEST_HOME/lib/broker

The *$HARVEST_HOME/lib/broker* directory contains the search and index programs needed by the Broker, plus several utility programs needed for Broker administration, as follows:

`BrokerRestart`

This program will issue a restart command to a broker.

Usage: `BrokerRestart [-password passwd] host port`

`brkclient`

Client interface to the broker. Can be used to send queries or administrative commands to a broker.

Usage: `brkclient hostname port command-string` `dumpregistry`

Prints the Broker's Registry file in a human-readable format.

Usage: `dumpregistry [-count] [BrokerDirectory]`

`agrep, glimpse, glimpseindex, glimpseindex.bin, glimpseserver`

The Glimpse indexing and search system as described in Section 5 (The Broker).

`swish`

The Swish indexing and search program as an alternative to Glimpse.

`info-to-html.pl, mkbrokerstats.pl`

Perl programs used to generate Broker statistics and to create *stats.html*.

Usage: `gather -info host port | info-to-html.pl > host.port.html`

Usage: `mkbrokerstats.pl broker-dir > stats.html`

6.8 $HARVEST_HOME/lib/gatherer

The *$HARVEST_HOME/lib/gatherer* directory contains the default summarizers described in Section 4.5 (Extracting data for indexing: The Essence summarizing subsystem), plus various utility programs needed by the summarizers and the Gatherer, as follows:

URL-filter-default

Default URL filter as described in Section 4.3 (RootNode specifications).

bycontent.cf, byname.cf, byurl.cf, magic, stoplist.cf, quick-sum.cf

Essence configuration files as described in Section 4.5.3 (Customizing the type recognition, candidate selection, presentation unnesting, and summarizing steps).

`*.sum`

Essence summarizers as discussed in Section 4.5 (Extracting data for indexing: The Essence summarizing subsystem).

`HTML-sum.pl`

Alternative HTML summarizer written in Perl.

`HTMLurls`

Program to extract URLs from a HTML file.

Usage: `HTMLurls [-base-url url] filename`

`catdoc, xls2csv,` ***catdoc-lib***

Programs and files used by Microsoft Word summarizer.

`dvi2tty, print-c-comments, ps2txt, ps2txt-2.1, pstext, skim`

Programs used by various summarizers.

`gifinfo`

Program to support summarizers.

`l2h`

Program used by TeX summarizer.

`rast`, `smgls`, `sgmlsasp`, ***sgmls-lib***

Programs and files used by SGML summarizer.

`rtf2html`

Program used by RTF summarizer.

`wp2x`, `wp2x.sh`, ***wp2x-lib***

Programs and files used by WordPerfect summarizer.

`hexbin`, `unshar`, `uudecode`

Programs used to unnest nested objects.

`cksoif`

Programs used to check the validity of a SOIF stream (e.g., to ensure that there is not parsing errors).

Usage: `cksoif < INPUT.soif`

`cleandb`, `consoldb`, `expiredb`, `folddb`, `mergedb`, `mkgathererstats.pl`, `mkindex`, `rmbinary`

Programs used to prepare a Gatherer's database to be exported by `gatherd`.

`cleandb` ensures that all SOIF objects are valid, and deletes any that are not;

`consoldb` will consolidate n GDBM database files into a single GDBM database file;

`expiredb` deletes any SOIF objects that are no longer valid as defined by its *Time-to-Live* attribute;

`folddb` runs all of the operations needed to prepare the Gatherer's database for export by `gatherd`;

`mergedb` consolidates GDBM files as described in Section 4.7.7 (Incorporating manually generated information into a Gatherer);

`mkgathererstats.pl` generates the *INFO.soif* statistics file;

`mkindex` generates the cache of timestamps; and

`rmbinary` removes binary data from a GDBM database.

`enum`, `prepurls`, `staturl`

Programs used by `Gatherer` to perform the RootNode and LeafNode enumeration for the Gatherer as described in Section 4.3 (RootNode specifications).

`enum` performs a RootNode enumeration on the given URLs;

`prepurls` is a wrapper program used to pipe `Gatherer` and `essence` together;

`staturl` retrieves LeafNode URLs to determine if the URL has been modified or not.

`fileenum`, `ftpenum`, `ftpenum.pl`, `gopherenum-*`, `httpenum-*`, `newsenum`

Programs used by `enum` to perform protocol specific enumeration.

`fileenum` performs a RootNode enumeration on "file" URLs;

`ftpenum` calls `ftpenum.pl` to perform a RootNode enumeration on "ftp" URLs;

`gopherenum-breadth` performs a breadth first RootNode enumeration on "gopher" URLs;

`gopherenum-depth` performs a depth first RootNode enumeration on "gopher" URLs;

`httpenum-breadth` performs a breadth first RootNode enumeration on "http" URLs;

`httpenum-depth` performs a depth first RootNode enumeration on "http" URLs;

`newsenum` performs a RootNode enumeration on "news" URLs;

`essence`

The Essence content extraction system as described in Section 4.5.3 (Customizing the type recognition, candidate selection, presentation unnesting, and summarizing steps).

Usage: `essence [options] -f input-URLs` or `essence [options] URL ...`

where options are:

```
--dbdir directory       Directory to place database
--full-text             Use entire file instead of summarizing
--gatherer-host         Gatherer-Host value
--gatherer-name         Gatherer-Name value
--gatherer-version      Gatherer-Version value
--help                  Print usage information
--libdir directory      Directory to place configuration files
--log logfile           Name of the file to log messages to
--max-deletions n       Number of GDBM deletions before reorganization
--minimal-bookkeeping   Generates a minimal amount of bookkeeping attrs
--no-access             Do not read contents of objects
--no-keywords           Do not automatically generate keywords
--allowlist filename    File with list of types to allow
--stoplist filename     File with list of types to remove
--tmpdir directory      Name of directory to use for temporary files
--type-only             Only type data; do not summarize objects
--verbose               Verbose output
--version               Version information
```

`print-attr`

Reads in a SOIF stream from stdin and prints the data associated with the given attribute to stdout.

Usage: `cat SOIF-file | print-attr Attribute`

`gatherd, in.gatherd`

Daemons that exports the Gatherer's database. `in.gatherd` is used to run this daemon from inetd.

Usage: `gatherd [-db | -index | -log | -zip | -cf file] [-dir dir] port`

Usage: `in.gatherd [-db | -index | -log | -zip | -cf file] [-dir dir]`

`gdbmutil`

Program to perform various operations on a GDBM database.

```
Usage: gdbmutil consolidate [-d | -D] master-file file [file ...]
Usage: gdbmutil delete file key
Usage: gdbmutil dump file
Usage: gdbmutil fetch file key
Usage: gdbmutil keys file
Usage: gdbmutil print [-gatherd] file
Usage: gdbmutil reorganize file
Usage: gdbmutil restore file
Usage: gdbmutil sort file
Usage: gdbmutil stats file
Usage: gdbmutil store file key < data
```

`mktemplate`

Program to generate valid SOIF based on a more easily editable SOIF-like format (e.g., SOIF without the byte counts).

Usage: `mktemplate < INPUT.txt > OUTPUT.soif`

`quick-sum`

Simple Perl program to emulate Essence's *quick-sum.cf* processing for those who cannot compile Essence with the corresponding C code.

`template2db`

Converts a stream of SOIF objects (from stdin or given files) into a GDBM database.

Usage: `template2db database [tmpl tmpl...]`

`wrapit`

Wraps the data from stdin into a SOIF attribute-value pair with a byte count. Used by Essence summarizers to easily generate SOIf.

Usage: `wrapit [Attribute]`

`kill-gatherd`

Script to kill gatherd process.

6.9 $HARVEST_HOME/tmp

The *$HARVEST_HOME/tmp* directory is used by search.cgi to store search result pages.

Chapter 7

The Summary Object Interchange Format (SOIF)

Harvest Gatherers and Brokers communicate using an attribute-value stream protocol called the *Summary Object Interchange Format (SOIF)*, an example of which is available in Section 8.1 (Example 1). Gatherers generate content summaries for individual objects in SOIF, and serve these summaries to Brokers that wish to collect and index them. SOIF provides a means of bracketing collections of summary objects, allowing Harvest Brokers to retrieve SOIF content summaries from a Gatherer for many objects in a single, efficient compressed stream. Harvest Brokers provide support for querying SOIF data using structured attribute-value queries and many other types of queries, as discussed in Section 5.3 (Querying a Broker).

7.1 Formal description of SOIF

The SOIF Grammar is as follows:

```
SOIF            ::=  OBJECT SOIF | OBJECT
OBJECT          ::=  @ TEMPLATE-TYPE { URL ATTRIBUTE-LIST }
ATTRIBUTE-LIST  ::=  ATTRIBUTE ATTRIBUTE-LIST | ATTRIBUTE
ATTRIBUTE       ::=  IDENTIFIER {VALUE-SIZE} DELIMITER VALUE
TEMPLATE-TYPE   ::=  Alpha-Numeric-String
IDENTIFIER      ::=  Alpha-Numeric-String
VALUE           ::=  Arbitrary-Data
VALUE-SIZE      ::=  Number
DELIMITER       ::=  ":<tab>"
```

7.2 List of common SOIF attribute names

Each Broker can support different attributes, depending on the data it holds. Below we list a set of the most common attributes:

```
Abstract
     Brief abstract about the object.
Author
     Author(s) of the object.
```

```
Description
     Brief description about the object.
File-Size
     Number of bytes in the object.
Full-Text
     Entire contents of the object.
Gatherer-Host
     Host on which the Gatherer ran to extract information from the object.
Gatherer-Name
     Name of the Gatherer that extracted information from the object. (eg.
     Full-Text, Selected-Text, or Terse).
Gatherer-Port
     Port number on the Gatherer-Host that serves the Gatherer's information.
Gatherer-Version
     Version number of the Gatherer.
Update-Time
     The time that Gatherer updated the content summary for the object.
Keywords
     Searchable keywords extracted from the object.
Last-Modification-Time
     The time that the object was last modified.
MD5
     MD5 16-byte checksum of the object.
Refresh-Rate
     The number of seconds after Update-Time when the summary object is to
     be re-generated.  Defaults to 1 month.
Time-to-Live
     The number of seconds after Update-Time when the summary object is
     no longer valid.  Defaults to 6 months.
Title
     Title of the object.
Type
     The object's type. Some example types are:

             Archive
             Audio
             Awk
             Backup
             Binary
             C
             CHeader
             Command
             Compressed
             CompressedTar
             Configuration
             Data
             Directory
             DotFile
             Dvi
             FAQ
             FYI
             Font
             FormattedText
             GDBM
             GNUCompressed
```

GNUCompressedTar
HTML
Image
Internet-Draft
MacCompressed
Mail
Makefile
ManPage
Object
OtherCode
PCCompressed
Patch
Pdf
Perl
PostScript
RCS
README
RFC
RTF
SCCS
ShellArchive
Tar
Tcl
Tex
Text
Troff
Uuencoded
WaisSource

Update-Time
: The time that the summary object was last updated. REQUIRED field, no default.

URI
: Uniform Resource Identifier.

URL-References
: Any URL references present within HTML objects.

Chapter 8

Gatherer Examples

The following examples install into *$HARVEST_HOME/gatherers* by default (see Section 3 (Installing the Harvest Software)).

The Harvest distribution contains several examples of how to configure, customize, and run Gatherers. This section will walk you through several example Gatherers. The goal is to give you a sense of what you can do with a Gatherer and how to do it. You needn't work through all of the examples; each is instructive in its own right.

To use the Gatherer examples, you need the Harvest binary directory in your path, and *HARVEST_HOME* defined. For example:

```
% setenv HARVEST_HOME /usr/local/harvest
% set path = ($HARVEST_HOME/bin $path)
```

8.1 Example 1 - A simple Gatherer

This example is a simple Gatherer that uses the default customizations. The only work that the user does to configure this Gatherer is to specify the list of URLs from which to gather (see Section 4 (The Gatherer)).

To run this example, type:

```
% cd $HARVEST_HOME/gatherers/example-1
% ./RunGatherer
```

To view the configuration file for this Gatherer, look at *example-1.cf*. The first few lines are variables that specify some local information about the Gatherer (see Section 4.7.1 (Setting variables in the Gatherer configuration file)). For example, each content summary will contain the name of the Gatherer (**Gatherer-Name**) that generated it. The port number (**Gatherer-Port**) that will be used to export the indexing information, as is the directory that contains the Gatherer (**Top-Directory**). Notice that there is one RootNode URL and one LeafNode URL.

After the Gatherer has finished, it will start up the Gatherer daemon which will export the content summaries. To view the content summaries, type:

```
% gather localhost 9111 | more
```

The following SOIF object should look similar to those that this Gatherer generates.

```
@FILE { http://harvest.cs.colorado.edu/~schwartz/IRTF.html
Time-to-Live{7}:        9676800
Last-Modification-Time{1}:      0
Refresh-Rate{7}:        2419200
Gatherer-Name{25}:      Example Gatherer Number 1
Gatherer-Host{22}:      powell.cs.colorado.edu
Gatherer-Version{3}:    0.4
Update-Time{9}: 781478043
Type{4}:        HTML
File-Size{4}:   2099
MD5{32}:        c2fa35fd44a47634f39086652e879170
Partial-Text{151}:      research problems
Mic Bowman
Peter Danzig
Udi Manber
Michael Schwartz
Darren Hardy
talk
talk
Harvest
talk
Advanced
Research Projects Agency

URL-References{628}:
ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/RD.ResearchProblems.Jour.ps.Z
ftp://grand.central.org/afs/transarc.com/public/mic/html/Bio.html
http://excalibur.usc.edu/people/danzig.html
http://glimpse.cs.arizona.edu:1994/udi.html
http://harvest.cs.colorado.edu/~schwartz/Home.html
http://harvest.cs.colorado.edu/~hardy/Home.html
ftp://ftp.cs.colorado.edu/pub/cs/misc/schwartz/HPCC94.Slides.ps.Z
ftp://ftp.cs.colorado.edu/pub/cs/misc/schwartz/HPC94.Slides.ps.Z
http://harvest.cs.colorado.edu/harvest/Home.html
ftp://ftp.cs.colorado.edu/pub/cs/misc/schwartz/IETF.Jul94.Slides.ps.Z
http://ftp.arpa.mil/ResearchAreas/NETS/Internet.html

Title{84}:      IRTF Research Group on Resource Discovery
IRTF Research Group on Resource Discovery

Keywords{121}:  advanced agency bowman danzig darren hardy harvest manber mic
michael peter problems projects research schwartz talk udi

}
```

Notice that although the Gatherer configuration file lists only 2 URLs (one in the RootNode section and one in the LeafNode section), there are more than 2 content summaries in the Gatherer's database. The Gatherer expanded the RootNode URL into dozens of LeafNode URLs by recursively extracting the links from the HTML file at the RootNode *http://harvest.cs.colorado.edu/*. Then, for each LeafNode given to the Gatherer, it generated a content summary for it as in the above example summary for *http://harvest.cs.colorado.edu/~schwartz/IRTF.html*.

The HTML summarizer will extract structured information about the Author and Title of the file. It will also extract any URL links into the *URL-References* attribute, and any anchor tags into the *Partial-Text* attribute. Other information about the HTML file such as its MD5 (see *RFC1321*) and its size (*File-Size*) in bytes are also added to the content summary.

8.2 Example 2 - Incorporating manually generated information

The Gatherer is able to "explode" a resource into a stream of content summaries. This is useful for files that contain manually-generated information that may describe one or more resources, or for building a gateway between various structured formats and SOIF (see Section 7 (The Summary Object Interchange Format (SOIF))).

This example demonstrates an exploder for the Linux Software Map (LSM) format. LSM files contain structured information (like the author, location, etc.) about software available for the Linux operating system.

To run this example, type:

```
% cd $HARVEST_HOME/gatherers/example-2
% ./RunGatherer
```

To view the configuration file for this Gatherer, look at *example-2.cf*. Notice that the Gatherer has its own *Lib-Directory* (see Section 4.7.1 (Setting variables in the Gatherer configuration file) for help on writing configuration files). The library directory contains the typing and candidate selection customizations for Essence. In this example, we've only customized the candidate selection step. *lib/stoplist.cf* defines the types that Essence should not index. This example uses an empty *stoplist.cf* file to direct Essence to index all files.

The Gatherer retrieves each of the LeafNode URLs, which are all Linux Software Map files from the Linux FTP archive *tsx-11.mit.edu*. The Gatherer recognizes that a ".lsm" file is *LSM* type because of the naming heuristic present in *lib/byname.cf*. The *LSM* type is a "nested" type as specified in the Essence source code (*src/gatherer/essence/unnest.c*). Exploder programs (named `TypeName.unnest`) are run on nested types rather than the usual summarizers. The `LSM.unnest` program is the standard exploder program that takes an *LSM* file and generates one or more corresponding SOIF objects. When the Gatherer finishes, it contains one or more corresponding SOIF objects for the software described within each *LSM* file.

After the Gatherer has finished, it will start up the Gatherer daemon which will serve the content summaries. To view the content summaries, type:

```
% gather localhost 9222 | more
```

Because *tsx-11.mit.edu* is a popular and heavily loaded archive, the Gatherer often won't be able to retrieve the LSM files. If you suspect that something went wrong, look in *log.errors* and *log.gatherer* to try to determine the problem.

The following two SOIF objects were generated by this Gatherer. The first object is summarizes the *LSM* file itself, and the second object summarizes the software described in the *LSM* file.

```
@FILE { ftp://tsx-11.mit.edu/pub/linux/docs/linux-doc-project/man-pages-1.4.lsm
Time-to-Live{7}:        9676800
Last-Modification-Time{9}:      781931042
Refresh-Rate{7}:        2419200
Gatherer-Name{25}:      Example Gatherer Number 2
Gatherer-Host{22}:      powell.cs.colorado.edu
```

```
Gatherer-Version{3}:    0.4
Type{3}:        LSM
Update-Time{9}: 781931042
File-Size{3}:   848
MD5{32}:        67377f3ea214ab680892c82906081caf
}

@FILE { ftp://ftp.cs.unc.edu/pub/faith/linux/man-pages-1.4.tar.gz
Time-to-Live{7}:        9676800
Last-Modification-Time{9}:      781931042
Refresh-Rate{7}:        2419200
Gatherer-Name{25}:      Example Gatherer Number 2
Gatherer-Host{22}:      powell.cs.colorado.edu
Gatherer-Version{3}:    0.4
Update-Time{9}: 781931042
Type{16}:       GNUCompressedTar
Title{48}:      Section 2, 3, 4, 5, 7, and 9 man pages for Linux
Version{3}:     1.4
Description{124}:       Man pages for Linux.  Mostly section 2 is complete.  Section
3 has over 200 man pages, but it still far from being finished.
Author{27}:     Linux Documentation Project
AuthorEmail{11}:        DOC channel
Maintainer{9}:  Rik Faith
MaintEmail{16}: faith@cs.unc.edu
Site{45}:       ftp.cs.unc.edu
sunsite.unc.edu
tsx-11.mit.edu
Path{94}:       /pub/faith/linux
/pub/Linux/docs/linux-doc-project/man-pages
/pub/linux/docs/linux-doc-project
File{20}:       man-pages-1.4.tar.gz
FileSize{4}:    170k
CopyPolicy{47}: Public Domain or otherwise freely distributable
Keywords{10}:   man
pages

Entered{24}:    Sun Sep 11 19:52:06 1994
EnteredBy{9}:   Rik Faith
CheckedEmail{16}:       faith@cs.unc.edu
}
```

We've also built a Gatherer that explodes about a half-dozen index files from various PC archives into more than 25,000 content summaries. Each of these index files contain hundreds of a one-line descriptions about PC software distributions that are available via anonymous FTP.

8.3 Example 3 - Customizing type recognition and candidate selection

This example demonstrates how to customize the type recognition and candidate selection steps in the Gatherer (see Section 4.5.3 (Customizing the type recognition, candidate selection, presentation unnesting, and summarizing steps)). This Gatherer recognizes World Wide Web home pages, and is configured only to collect indexing information from these home pages.

To run this example, type:

```
% cd $HARVEST_HOME/gatherers/example-3
% ./RunGatherer
```

To view the configuration file for this Gatherer, look at *example-3.cf*. As in Section 8.2 (Example 2), this Gatherer has its own library directory that contains a customization for Essence. Since we're only interested in indexing home pages, we need only define the heuristics for recognizing home pages. As shown below, we can use URL naming heuristics to define a home page in *lib/byurl.cf*. We've also added a default *Unknown* type to make candidate selection easier in this file.

```
HomeHTML            ^http:.*/$
HomeHTML            ^http:.*[hH]ome\.html$
HomeHTML            ^http:.*[hH]ome[pP]age\.html$
HomeHTML            ^http:.*[wW]elcome\.html$
HomeHTML            ^http:.*/index\.html$
```

The *lib/stoplist.cf* configuration file contains a list of types not to index. In this example, *Unknown* is the only type name listed in stoplist.configuration, so the Gatherer will only reject files of the *Unknown* type. You can also recognize URLs by their filename (in *byname.cf*) or by their content (in *bycontent.cf* and *magic*); although in this example, we don't need to use those mechanisms. The default `HomeHTML.sum` summarizer summarizes each *HomeHTML* file.

After the Gatherer has finished, it will start up the Gatherer daemon which will serve the content summaries. You'll notice that only content summaries for HomeHTML files are present. To view the content summaries, type:

```
% gather localhost 9333 | more
```

8.4 Example 4 - Customizing type recognition and summarizing

This example demonstrates how to customize the type recognition and summarizing steps in the Gatherer (see Section 4.5.3 (Customizing the type recognition, candidate selection, presentation unnesting, and summarizing steps). This Gatherer recognizes two new file formats and summarizes them appropriately.

To view the configuration file for this Gatherer, look at *example-4.cf*. As in the examples in 8.2 (Example 2) and 8.3 (Example 3), this Gatherer has its own library directory that contains the configuration files for Essence. The Essence configuration files are the same as the default customization, except for *lib/byname.cf* which contains two customizations for the new file formats.

8.4.1 Using regular expressions to summarize a format

The first new format is the "ReferBibliographic" type which is the format that the `refer` program uses to represent bibliography information. To recognize that a file is in this format, we'll use the convention that the filename ends in ".referbib". So, we add that naming heuristic as a type recognition customization. Naming heuristics are represented as a regular expression against the filename in the *lib/byname.cf* file:

```
ReferBibliographic      ^.*\.referbib$
```

Now, to write a summarizer for this type, we'll need a sample ReferBibliographic file:

```
%A A. S. Tanenbaum
%T Computer Networks
%I Prentice Hall
%C Englewood Cliffs, NJ
%D 1988
```

Essence summarizers extract structured information from files. One way to write a summarizer is by using regular expressions to define the extractions. For each type of information that you want to extract from a file, add the regular expression that will match lines in that file to *lib/quick-sum.cf*. For example, the following regular expressions in *lib/quick-sum.cf* will extract the author, title, date, and other information from ReferBibliographic files:

```
ReferBibliographic      Author                  ^%A[ \t]+.*$
ReferBibliographic      City                    ^%C[ \t]+.*$
ReferBibliographic      Date                    ^%D[ \t]+.*$
ReferBibliographic      Editor                  ^%E[ \t]+.*$
ReferBibliographic      Comments                ^%H[ \t]+.*$
ReferBibliographic      Issuer                  ^%I[ \t]+.*$
ReferBibliographic      Journal                 ^%J[ \t]+.*$
ReferBibliographic      Keywords                ^%K[ \t]+.*$
ReferBibliographic      Label                   ^%L[ \t]+.*$
ReferBibliographic      Number                  ^%N[ \t]+.*$
ReferBibliographic      Comments                ^%O[ \t]+.*$
ReferBibliographic      Page-Number             ^%P[ \t]+.*$
ReferBibliographic      Unpublished-Info        ^%R[ \t]+.*$
ReferBibliographic      Series-Title            ^%S[ \t]+.*$
ReferBibliographic      Title                   ^%T[ \t]+.*$
ReferBibliographic      Volume                  ^%V[ \t]+.*$
ReferBibliographic      Abstract                ^%X[ \t]+.*$
```

The first field in *lib/quick-sum.cf* is the name of the type. The second field is the Attribute under which to extract the information on lines that match the regular expression in the third field.

8.4.2 Using programs to summarize a format

The second new file format is the "Abstract" type, which is a file that contains only the text of a paper abstract (a format that is common in technical report FTP archives). To recognize that a file is written in this format, we'll use the naming convention that the filename for "Abstract" files ends in ".abs". So, we add that type recognition customization to the *lib/byname.cf* file as a regular expression:

```
Abstract                ^.*\.abs$
```

Another way to write a summarizer is to write a program or script that takes a filename as the first argument on the command line, extracts the structured information, then outputs the results as a list of SOIF attribute-value pairs.

Summarizer programs are named `TypeName.sum`, so we call our new summarizer `Abstract.sum`. Remember to place the summarizer program in a directory that is in your path so that Gatherer can run it. You'll see below that `Abstract.sum` is a Bourne shell script that takes the first 50 lines of the file, wraps it as the "Abstract" attribute, and outputs it as a SOIF attribute-value pair.

```
#!/bin/sh
#
#  Usage: Abstract.sum filename
#
head -50 "$1" | wrapit "Abstract"
```

8.4.3 Running the example

To run this example, type:

```
% cd $HARVEST_HOME/gatherers/example-4
% ./RunGatherer
```

After the Gatherer has finished, it will start up the Gatherer daemon which will serve the content summaries. To view the content summaries, type:

```
% gather localhost 9444 | more
```

8.5 Example 5 - Using RootNode filters

This example demonstrates how to use RootNode filters to customize the candidate selection in the Gatherer (see Section 4.3.1 (RootNode filters)). Only items that pass RootNode filters will be retrieved across the network (see Section 4.3.4 (Gatherer enumeration vs. candidate selection)).

To run this example, type:

```
% cd $HARVEST_HOME/gatherers/example-5
% ./RunGatherer
```

After the Gatherer has finished, it will start up the Gatherer daemon which will serve the content summaries. To view the content summaries, type:

```
% gather localhost 9555 | more
```

Chapter 9

History of Harvest

9.1 History of Harvest

- 1996-01-31: Harvest 1.4pl2 was the last official release by Darren R. Hardy, Michael F. Schwartz, and Duane Wessels.
- 1997-04-21: Harvest 1.5 was released by Simon Wilkinson.
- 1998-06-12: Harvest 1.5.20 was released by Simon Wilkinson.
- 1999-05-26: Harvest-MathNet100.tar.gz released.
- 2000-01-14: harvest-modified-by-RL-Stajsic.tar.gz released.
- 2000-02-07: Harvest 1.6.1 was released by Kang-Jin Lee in cooperation with Simon Wilkinson.
- 2002-10-25: Harvest 1.8.0 was released by Harald Weinreich and Kang-Jin Lee.

9.2 History of Harvest User's Manual

- 1996-01-31: Harvest User's Manual for Harvest 1.4.pl2 was written by Darren R. Hardy, Michael F. Schwartz, and Duane Wessels. The document was written in LaTeX. The HTML (converted with LaTeX2HTML) and the Postscript versions were made available to the public.
- 2001-04-27: The HTML version of this document was updated and bundled with the Harvest distribution by Kang-Jin Lee. Notable changes were removing the sections about Harvest Object Cache and the Replicator which are not part of Harvest any more.
- 2002-01-28: This Harvest User's Manual was converted to linuxdoc. It is now available in PostScript, PDF, text and HTML format.