HARVEST

EFFECTIVE USE OF INTERNET INFORMATION

Harvest User's Manual

Darren R. Hardy, *U. Colorado, Boulder*
Michael F. Schwartz, *U. Colorado, Boulder*
Duane Wessels, *U. Colorado, Boulder*

Version 1.4 patchlevel 2
January 31, 1996

University of Colorado at Boulder

Technical Report CU-CS-743-94
Department of Computer Science
University of Colorado
Boulder, Colorado 80309-0430

Acknowledgements

Harvest was designed and built by the Internet Research Task Force Research Group on Resource Discovery (IRTF-RD). IRTF-RD consists of Mic Bowman (Transarc Corp.), Peter Danzig (USC), Udi Manber (U. Arizona), and Michael Schwartz (IRTF-RD chair, U. Colorado). Darren Hardy and Duane Wessels are Professional Research Assistants on the project (at U. Colorado).

Many students have contributed to this project: Rajini Balay, William Camargo, Anawat Chankhunthod, Bhavna Chhabra, Gabe Dalbec, Dante De Lucia, Chanda Dharap, Burra Gopal, James Guyton, Allan Hundhausen, Paul Klark, Shih-Hao Li, Cheng-Che Lue, Dave Merkel, Chuck Neerdaels, John Noble, John Noll, Katia Obraczka, Mark Peterson, Erh-Yuan Tsai, and Kurt Worrell.

IRTF-RD is supported primarily by the Advanced Research Projects Agency (contract number DABT63-93-C-0052), with additional support from the Air Force Office of Scientific Research (award number F49620-93-1-0082), the National Science Foundation (grant numbers CCR-9002351, CCR-9301129, CDA-8914587, CDA-8914587AO2, NCR-9105372, and NCR-9204853), Hughes Aircraft Company (under NASA EOSDIS project subcontract number ECS-00009), two equipment grants from Sun Microsystems' Collaborative Research Program, and the University of Colorado's Office of the Vice Chancellor for Academic Affairs. The information contained in this document does not necessarily reflect the position or the policy of the U.S. Government or other sponsors of this research. No official endorsement should be inferred.

History of Free Software Status Originally we required sites to license the software in cases where they were going to build commercial products/services around Harvest. In June 1995 we changed this policy. We now allow people to use the *core* Harvest software (the code found in the Harvest "src/" directory) for free. We made this change in the interest of encouraging the widest possible deployment of the technology. The Harvest software is really a reference implementation of a set of protocols and formats, some of which we intend to standardize. We encourage commercial re-implementations of code complying to this set of standards.

[1] `http://harvest.cs.colorado.edu/`

Contents

1 Introduction to Harvest

HARVEST is an integrated set of tools to gather, extract, organize, search, cache, and replicate relevant information across the Internet [5]. 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, making very efficient use of Internet servers, network links, and index space on disk. Our measurements indicate that Harvest can reduce server load by a factor of over 6,000, network traffic by a factor of 60, and index space requirements by a factor of over 40 when building indexes compared with other systems, such as Archie, WAIS, and the World Wide Web Worm[2].

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 provides a data gathering architecture for constructing indexes. This stands in contrast to WHOIS++ [19] (which requires users to construct indexing templates manually) and GILS [1] (which does not define how index data are collected). Harvest 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.

For more detailed comparisons with related systems, see [4] or our online FAQ[3].

We provide an overview of the Harvest subsystems in the next section.

[2] `http://www.cs.colorado.edu/home/mcbryan/WWWW.html`

[3] `http://harvest.cs.colorado.edu/harvest/FAQ.html#compare-with-related-work`

2 Subsystem Overview

As illustrated in Figure 1, 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. The *Replicator* subsystem efficiently replicates Brokers around the Internet. Users can efficiently retrieve located information through the *Cache* subsystem. The Harvest Server Registry (HSR) is a distinguished Broker that holds information about each Harvest Gatherer, Broker, Cache, and Replicator in the Internet.

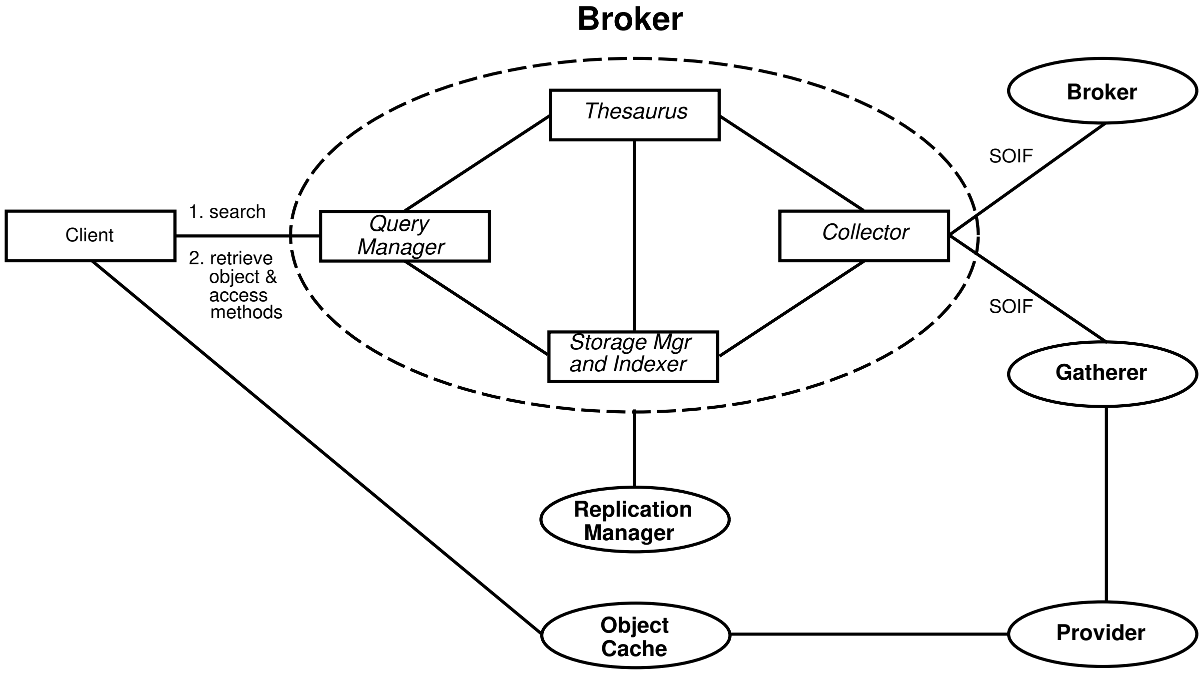

Figure 1: Harvest Software Components

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, Gopher, World Wide Web, and NetNews data at your site. You may also want to run an Object Cache (see Section 6), to reduce network traffic for accessing popular data.

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. Finally, you can reduce overloading on popular Brokers by running Replicators.

We discuss how to distribute the gathering process in the next subsection. We cover various forms of customization in Section 4.5.4 and in several parts of Section 5. We discuss Broker replication in Section 7.

Distributing the Gathering and Brokering Processes

As illustrated in Figure 2, 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, 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. 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.

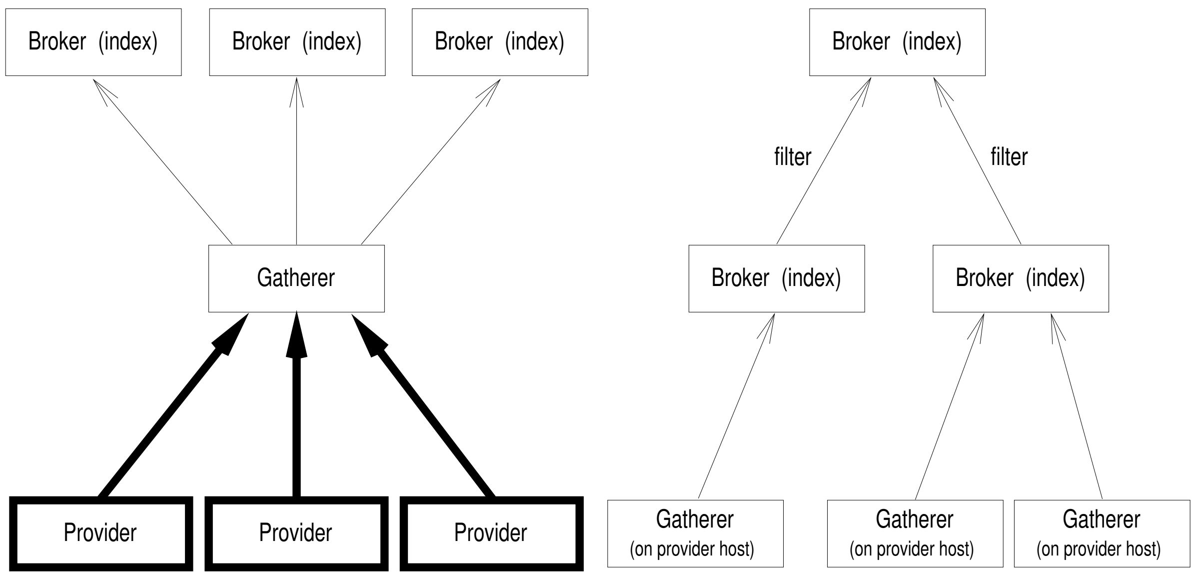

Figure 2: Harvest Configuration Options

Figure 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, 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.

You are welcome to browse our demonstration Gatherers[4] and Brokers[5].

[4] `http://harvest.cs.colorado.edu/Harvest/gatherers/`
[5] `http://harvest.cs.colorado.edu/Harvest/brokers/`

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 (e.g., Sun Sparc 5, DEC Alpha, Intel Pentium), 1-2 GB of free disk, and 64 MB of RAM. A slower CPU (e.g., a Sun 4) 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., `BrokerQuery.pl.cgi`), and some of which improve performance (e.g., the Object Cache and its parallel `dnsserver` processes, and 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.

You will need another 50MB of free disk space to run a Harvest Object Cache, or more if you want to run a widely shared cache (e.g., a company-wide "root" cache, under which there a number of subordinate caches).

3.1.2 Platforms

If you want to run a Harvest server, you will need a UNIX machine. Specifically, we support the following server platforms: **DEC's OSF/1 2.0 and 3.0**, **Sun's SunOS 4.1.x**, and **Sun's Solaris 2.3**.

At present we are concentrating our efforts on supporting the above platforms, although we may eventually support others. We have also incorporated source code changes provided by numerous Harvest users for the following platforms (and other ports are under way): AIX 3.2 using the AIX C compiler; FreeBSD; HP-UX 09.03 using the HP ANSI C compiler A.09.69; Linux 1.1.59; and IRIX 5.3. Note that we *do not support* these platforms. Binary distributions of non-supported platforms, if available, will be placed under the `contrib` directory on the FTP sites. If you have questions about the non-supported platform ports, you might post a note requesting help on the Harvest USENET newsgroup (*comp.infosystems.harvest*). If you port Harvest to a new system, please notify us via email at *harvest-dvl@cs.colorado.edu*.

3.1.3 Software

In addition to the above platform requirements, you may need to install one or more of the following software packages:

- All Harvest servers require: Perl v4.0 or higher (v5.0 is preferred).
- The Harvest Broker and Gatherer require: GNU `gzip` v1.2.4 or higher.
- The Harvest Broker requires: an HTTP server

If you want to build Harvest from the source distribution rather than using one of the binary distribution, then 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 and `bison` v1.22.

The source for Perl, `gcc`, `gzip`, `flex`, and `bison` are available at the GNU FTP server[6]. Information about various HTTP server software packages is available at the Sunsite FTP server[7].

3.2 Requirements for Harvest Users

Anyone with a World Wide Web client (e.g., NCSA Mosaic[8]) can access and use Harvest servers. World Wide Web clients are available for most platforms, including DOS, Windows, OS/2, Macintosh, and UNIX/X-Windows. Most of these clients will work over any high-speed modem (e.g., 9600 baud or better), or Ethernet connection. The World Wide Web consortium maintains a list of WWW clients[9].

3.3 Retrieving and Installing the Harvest Software

3.3.1 Distribution types

We offer two kinds of Harvest distributions: source and binary. The *source distribution* contains all of the source code for the Harvest software. You compile the source distribution on one of the supported platform, or port the source code to another platform. The *binary distributions* contain precompiled binaries for all of the Harvest programs. We offer binary distributions for the supported platforms, and we make available binary distributions for unsupported platforms which we receive from Harvest users. We offer source and binary distributions for the entire Harvest system, and for only the Harvest Object Cache subsystem (for those who only want to install a Cache).

For most users, the easiest option is to retrieve one of the binary distributions. We make the source code available primarily for users who wish to port Harvest to a new architecture or operating system, or to add functionality to the Harvest software.

You can retrieve the Harvest distributions from a number of distribution sites[10].

3.3.2 Optional Harvest components

You may want to install one of the *optional* Harvest components available from the Harvest distribution sites in the `components` directory. To install and use a component, follow the instructions included in the desired component distribution. As of March 1995, the following components are available: *WordPerfect*: Summarizers WordPerfect 5.1 files; and *Rainbow*: Summarizers MIF and RTF files generated by *FrameMaker*, *Microsoft Word*, and other word processors.

3.3.3 User-contributed software

There is a collection of unsupported user-contributed software available in the `contrib` directory under the top-level directory at each of the Harvest software distribution sites. If you would like to contribute some software to this directory, please send email to *harvest-dvl@cs.colorado.edu.*

3.3.4 Unpacking a binary distribution

Once you've retrieved the distribution, create a directory in which the Harvest software will live (typically `/usr/local/harvest`), then change your current directory to it. To unpack the binary distribution, run the following command to extract the software into the current directory:

```
% gzip -dc harvest-CPU-MACHINE-OS.tar.gz | tar xf -
```

[6] `ftp://ftp.gnu.ai.mit.edu/pub/gnu/`
[7] `http://sunsite.unc.edu/boutell/faq/www_faq.html#provide`
[8] `http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html`
[9] `http://www.w3.org/hypertext/WWW/Clients.html`
[10] `http://harvest.cs.colorado.edu/harvest/gettingsoftware.html`

WARNING: You must extract the Harvest software in an empty directory; do *not* extract it over any older version of Harvest. Finally, define the *HARVEST_HOME* environment variable. *HARVEST_HOME* is the Harvest directory where you extracted the distribution into.

```
% cd harvest-1.4
% setenv HARVEST_HOME `pwd`
```

3.4 Building the Source Distribution

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

```
% gzip -dc harvest-src.tar.gz | tar xf -
```

Harvest uses GNU's *autoconf* package to perform needed localizations at installation time. If you want to override the default installation location of `/usr/local/harvest`, change the "prefix" variable in *Makefile*[11] 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[12], type *make all.*

For example, to build the entire Harvest system, type:

```
% vi Makefile                       (only if changing the prefix)
% make reconfigure                  (only if prefix was changed)
% vi src/common/include/config.h    (only if desired)
% make all install
```

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

Building the entire Harvest distribution will take about 30 minutes on a DEC Alpha or on newer Sun SPARCstations, and almost an hour on older machines. 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 realclean`.

Building individual Harvest components

To build individual Harvest components (such as the Gatherer or the Broker), change into the `src` directory. If you wish to change the default installation from `/usr/local/harvest`, then edit the *Makefile* there and type `make reconfigure` to propagate the change. Finally, to build the Harvest component type `make component`, and to install the built component, type `make install-component`, where valid `component` names are *broker, cache, gatherer, indexer,* or *replicator.* For example, to build the Harvest Gatherer, type:

```
% cd src
% vi Makefile                       (only if changing the prefix)
% make reconfigure                  (only if prefix was changed)
% make gatherer install-gatherer
```

If you have problems compiling the software on your system, make sure you are using one of the supported platforms and compiler listed in Section 3.1.2.

[11] This is the top-level Makefile in the *harvest* directory created by extracting the Harvest distribution.

[12] A quick way to build and install the Harvest software into the top-level directory is: `make prefix=`pwd` all install`.

3.5 Installing the Harvest software

When using the source distribution, you need to type `make install` to install the Harvest programs and files into the *HARVEST_HOME* directory. When using the binary distribution, all of the programs and files already exist in the *HARVEST_HOME* directory. Simply, follow the on-line instructions[13].

3.5.1 Additional installation for the Harvest Broker

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 `$HARVEST_HOME/cgi-bin/`. The `BrokerQuery.pl.cgi` script reads files from the `cgi-bin/lib/` directory, so check that as well. Finally, check the files in `$HARVEST_HOME/lib/`. Some of the CGI Perl scripts require "include" files in this directory.

Required modifications to the Broker's CGI programs The Harvest Broker requires some additional installation so that the WWW interface to the Broker will function correctly.

You must edit `$HARVEST_HOME/cgi-bin/HarvestGather.cgi` if you need to change *HARVEST_HOME* from the default value

```
HARVEST_HOME=/files1/stuff/harvest
```

You may need to edit `$HARVEST_HOME/cgi-bin/BrokerQuery.pl.cgi`, to change the *HARVEST_HOME* variable near the top of that file. And if your Perl binary is not `/usr/local/bin/perl` then you'll need to change the first line of `BrokerQuery.pl.cgi` as appropriate.

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.

CERN httpd v3.0 Requires an *Exec* and a *Pass* entry in the `httpd.conf` configuration file, e.g.:

```
Exec /Harvest/cgi-bin/* Your-HARVEST_HOME/cgi-bin/*
Pass /Harvest/* Your-HARVEST_HOME/*
```

If you are running the CERN server in standalone mode, it may be necessary to send it a HUP signal so that the server re-reads its configuration files.

NCSA httpd v1.3 or v1.4; Apache httpd v0.8.8 Requires a *ScriptAlias* and an *Alias* entry in `conf/srm.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.

Alternatively, you can substitute the *ScriptAlias* line with the following line using *AddType* to identify CGI programs by filename extension. e.g.:

```
AddType application/x-httpd-cgi .cgi
```

Reportedly, the Apache v0.8.8 HTTP server requires this *AddType* line in addition to the *ScriptAlias* and *Alias* lines. If you are running the HTTP server in standalone mode, it may be necessary to send it a HUP signal so that the server re-reads its configuration files.

[13] `http://harvest.cs.colorado.edu/harvest/INSTRUCTIONS.html`

GN HTTP server Harvest can be made to work with the GN server with some moderate effort by editing the source code. GN requires that all CGI programs begin with the string `/CGI`. Below is a partial list of files where you will need to change `/Harvest/cgi-bin/` to `/CGI/Harvest/cgi-bin/`.

```
src/broker/WWW/BrokerQuery.pl
src/broker/WWW/BrokerQuery.cf
src/broker/WWW/HarvestGather
src/broker/example/brokers/skeleton/admin/admin.html.in
src/broker/example/brokers/skeleton/query-glimpse.html.in
src/broker/example/brokers/skeleton/query-wais.html.in
src/broker/example/brokers/soifhelp.html
src/Harvest
```

Also, you'll need to make a symbolic link for Harvest. e.g.:

```
% cd /your/root/data/dir
% ln -s $HARVEST_HOME Harvest
```

Plexus HTTP server Harvest does not work well with the Plexus server because Plexus will not recognize `/Harvest/cgi-bin/prog.cgi` as a CGI program. No work-around is known, aside from modifying the Harvest source as with the GN HTTP server.

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.

3.6 Upgrading versions of the Harvest software

3.6.1 Upgrading from version 1.3 to version 1.4

There are no known incompatibilities between versions 1.3 and 1.4.

3.6.2 Upgrading from version 1.2 to version 1.3

Version 1.3 is mostly backwards compatible with 1.2, with the following exceptions

Glimpse In Harvest v1.3 uses Glimpse v3.0. 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.

3.6.3 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.
- The Cache also has a new log format which is incompatible with version 1.1.

3.6.4 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 *cannot* 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 Harvest distribution as directed by the release notes.
3. Then, for each Gatherer or 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 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 (e.g., you will use this command if you follow the instructions in the binary Harvest distribution). `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). Some commands require that the user set the *HARVEST_HOME* environment variable, to indicate where Harvest is installed. 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 Broker and a Gatherer. 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.

`RunCache`
: Runs the Cache. Requires *$HARVEST_HOME*, or defaults to */usr/local/harvest*. Found in the *$HARVEST_HOME/bin* 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. Requires *$HARVEST_HOME*, or defaults to */usr/local/harvest*. Found in the *$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 4.5.4 or Appendix C). The layout of the installed Harvest directories and programs is discussed in Appendix A.

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.

Once you have successfully built a Harvest Gatherer, Broker, or Cache, please register your server(s) with the Harvest Server Registry (HSR) using our registration page[14]. The `RunHarvest` command will ask you if you'd like to register your servers with the HSR. If you answer yes, then you do not need to use the registration page.

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*. Please note your machine type, operating system type, and Harvest version number in your correspondence. You can access an archive[15] of postings to this newsgroup, and also search[16] this archive.

If you have bug fixes, ports to new platforms, or other software improvements, please email them to the Harvest development group at *harvest-dvl@cs.colorado.edu*.

[14] `http://harvest.cs.colorado.edu/Harvest/brokers/hsr/register-with-hsr.html`

[15] `http://harvest.cs.colorado.edu/archive/comp.infosystems.harvest`

[16] `http://harvest.cs.colorado.edu/Harvest/brokers/CIH/`

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 Appendix B). 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

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 for more information on keeping Gatherer objects up to date.

Several example Gatherers are provided with the Harvest software distribution (see Appendix C).

4.2 Basic setup

To run a basic Gatherer, you need only list the Uniform Resource Locators (URLs) [2, 3] 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 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
Top-Directory:    /usr/local/harvest/gatherers/sample

#  Specify the URLs from which to gather.
<RootNodes>
http://harvest.cs.colorado.edu/
</RootNodes>

<LeafNodes>
http://www.cs.colorado.edu/cucs/Home.html
http://www.cs.colorado.edu/~hardy/Home.html
</LeafNodes>
```

As shown in the example configuration file, you may classify a 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 for details.

Note: Harvest is not intended to operate as a "robot"[17], since it does not collect new URLs to retrieve other than those specified in RootNodes (of course, if you specify many high-level RootNodes you can make it operate as a robot, but that is not the intended use for the system). The Gatherer is HTTP Version 1.0[18] 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 export[19] 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 (see Appendix A for usage information):

```
% 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).

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.

Cleaning out a Gatherer Remember the Gatherer databases perists 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.*
```

[17] `http://web.nexor.co.uk/mak/doc/robots/robots.html`
[18] `http://www.w3.org/hypertext/WWW/Protocols/HTTP/HTTP2.html`
[19] The Gatherer leaves the `gatherd` daemon running in the background to export the database.

4.3 RootNode specifications

The RootNode specification facility described in Section 4.2 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. Starting with Harvest Version 1.1, it is possible to specify these and other aspects of enumeration, using the following syntax (which is backwards-compatible with Harvest Version 1.0):

```
<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 4.5.4).

URL-Filter-filename This is the name of a file containing a set of regular expression filters (discussed below) 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. [20] Valid access method types are: `FILE`, `FTP`, `Gopher`, `HTTP`, `News`, `Telnet`, or

[20] 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/-1ftp%3aftp.cs.washington.edu%40pub/* 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.

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.

Delay This is the number of seconds to wait 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 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 [21]. 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  .
```

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.

[21] In general we set these defaults to be conservative. However, we set the default Depth to unlimited to be backwards-compatible with the RootNode specification semantics defined in Harvest Version 1.0.

4.3.2 Generic Enumeration filter description

Flexible enumeration can be achieved by giving an *Enumeration=filter* modifier to a RootNode URL. The *filter* is a program 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 (9) 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. 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/homes/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 sites 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 */homes/hardy* directory, or that are in any subdirectory of */homes/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 Using extreme values – "robots"

Using extreme values with the RootNode specification mechanism it is possible to specify a Web "robot"[22]. We implore the user not to do this. Robots are very inefficient – they generate excessive load on network links and remote information servers, do not coordinate gathering effort, and will become decreasingly useful over time because they do not focus their content on a specific topic or community. The Harvest RootNode specification mechanism was developed to support gathering needs for topical collections, not to build robots.

NOTE: As of version 1.4 patchlevel 2, the Gatherer obeys the `robots.txt` convention.

4.3.5 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, you can prevent documents from being indexed through customizing the *stoplist.cf* file, described in Section 4.5.4. 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) **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 4.7.5.

4.4 Generating LeafNode/RootNode URLs from a program

As of version 1.4, it is now 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.

[22] `http://web.nexor.co.uk/mak/doc/robots/robots.html`

```
<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* [10, 11] 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)[23], "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.

Starting with Harvest Version 1.2 we are also integrating support for summarizers based on outside "component technologies" of both a free and a commercial nature.

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 (or an interface to a commercial system) that is likely to be useful to other users, please notify us via email at *harvest-dvl@cs.colorado.edu* so we may include it in our components distribution.

[23] While HTTP provides MIME types, other access methods (like FTP) do not. Essence can use either explicit information or heuristic "rules" to determine types.

Type	Summarizer Function
Audio	Extract file name
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
Framemaker	Up-convert to SGML and pass through SGML summarizer
Font	Extract comments
HTML	Extract anchors, hypertext links, and selected fields (see SGML)
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 SGML and pass through SGML summarizer
SGML	Extract fields named in extraction table (see Section 4.5.2)
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 above table provides a brief reference for how documents are summarized depending on their type. These actions can be customized, as discussed in Section 4.5.4. Some summarizers are implemented as UNIX programs while others are expressed as regular expressions; see Section 4.5.4 or Appendix C.4 for more information about how to write a summarizer.

4.5.2 Summarizing SGML data

Starting with Harvest Version 1.2, it is possible to summarize documents that conform to the Standard Generalized Markup Language (SGML) [12], for which you have a Document Type Definition (DTD). [24] 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 now uses the HTML DTD and our SGML summarizing mechanism, which provides various advantages; see Section 4.5.2.) SGML is being used in an increasingly broad variety of applications, for example as a format for storing data

[24] A good reference for learning how to write DTDs is [18].

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 Sections 4.5.2 and 4.5.3 below for concrete 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 and 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 4.5.4) 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 Starting with Version 1.2, Harvest summarizes HTML using the generic SGML summarizer described in Section 4.5.2. 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 syntically 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 uncerimoniously. 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 (like Netscape) do not insist on well-formed documents. So, users can easily create documents that are not completely valid, yet display fine. The problem should become less pronounced if/when people shift to creating HTML documents using HTML editors rather than editing the raw HTML themselves.

Below is the default SGML-to-SOIF table used by the HTML summarizer. 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 $*
```

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

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 Summarizer components distribution

Starting with Harvest Version 1.2 we began integrating support for summarizers based on outside "component technologies" of both a free and a commercial nature. The components distribution contains summarizers that are of interest to a more narrow audience (e.g., for extracting content from a particular seismological data format); are particularly large (e.g., summarizers where only the binary executables are being made available); or are just interface code to non-freely distributable systems (e.g., code for interfacing Harvest with a commercial database system that must be purchased from an outside organization).

The components are available from the *components* directory at the top-level of each of the Harvest distribution sites[25].

At present we only have a modest number of outside components. Specifically, we have added support for two popular PC data formats: Microsoft's Rich Text Format (RTF), and Framemaker's Maker Interchange Format (MIF). These summarizers are discussed below.

Using "Rainbow" to summarize MIF and RTF documents The MIF and RTF summarizers work by first converting into SGML and then using the Harvest SGML summarizer. The SGML conversion step is performed using a commercial software package (and DTD) called "Rainbow", which was developed by Electronic Book Technologies. Since converting these formats to SGML is an "up conversion", the translation process is necessarily imperfect. However, the Rainbow software works quite well, and we appreciate EBT's making this software freely available to the Internet community. (Please note

[25] `http://harvest.cs.colorado.edu/harvest/gettingsoftware.html`

that we are redistributing the EBT code and DTD as per their copyright restrictions; we have included a copy of EBT's copyright file in *$HARVEST_HOME/components/gatherer/Rainbow/copyrite.txt*).

Note that at present EBT only makes executable code available for DOS, SunOS, Solaris, HP-UX, and AIX. Therefore, at present we only provide bundled distributions of the code for SunOS and Solaris (the subset of the above platforms on which we currently support Harvest).

More information on Rainbow is also available[26].

The translation table Files converted from RTF/MIF to SGML by the `rbmaker` program look something like this:

```
<PARA PARATYPE="title">My Document Title</PARA>
<PARA PARATYPE="heading 1">Introduction</PARA>
<PARA PARATYPE="normal">The purpose of this ...<PARA>
```

We can separate these two paragraphs with the following translation table entries:

```
<PARA,PARATYPE=title>           title
<PARA,PARATYPE=heading 1>       headings,keywords
<PARA>                          body
```

Unfortunately, the type names title, heading 1, etc. are not standardized. Therefore, our default translation table may not do an adequate job. To learn which paragraph type names are contained in a document, you can add this to the table:

```
<PARA:PARATYPE>                 paragraph-types
```

and then examine the SOIF output.

4.5.4 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 files names below can be changed by setting variables in the Gatherer configuration file, as described in Section 4.7.1):

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

bin:
MyNewType.sum* Exploder.unnest*

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

[26] ftp://ftp.ebt.com/pub/nv/dtd/rainbow/

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

tmp:
cache-liburl/
```

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 or by the presentation unnesting steps. If you were to customize the Gatherer by adding a summarizer or a presentation unnesting program, you would place those programs in this *bin* directory; the `MyNewType.sum` and `Exploder.unnest` are examples (see Section 4.5.4).

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,* and *PRODUCTION.gdbm* files. The *gatherd.cf* file is used to support access control as described in Section 4.7.4. 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:

File or Directory	Description
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

We discuss each of the customizable steps in the subsections below.

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 Appendix C.3 and C.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 4.5.4) 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. uuencoded
2. tape archive ("tar")
3. shell archive ("shar")
4. compressed
5. GNU compressed ("gzip")
6. binhex[27].

To customize the presentation unnesting step you can modify the Essence source file *harvest/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., `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 Appendix C.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 Appendix C.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 Appendix B). 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

Beginning with version 1.3, 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'
```

[27] At present we have bundled software for decoding the binhex format, but not integrated it into Essence

In this case we are checking if the *city* attribute is equal to the string 'New York' The 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()
   ```

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
```

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 4.5.4, 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 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:

VARIABLE NAME	DESCRIPTION
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.
- *HTTP-Proxy* will retrieve HTTP URLs via a proxy host. The syntax is *hostname:port*; for example, *harvest-cache.cs.colorado.edu: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 as described in Section 4.5.4.
`--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 4.5.4 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). 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 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 rather than the local file system interface. Lastly, the Gatherer does not perform any URL syntax translations for local mappings. If your URL has characters that should be escaped (as in RFC 1738 [3]), 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 NCSA httpd *.htaccess* file. In the CERN httpd configuration, the realm value is 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
cd ${HARVEST_HOME}/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=${HARVEST_HOME}/lib/gatherer:$PATH; export PATH
gatherd -dir ${HARVEST_HOME}/gatherers/att800/data 8001
```

4.7.6 The local disk cache

The Gatherer maintains a local disk cache of files it gathered to reduce network traffic from restarting aborted gathering attempts[28]. 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).

[28]The Gatherer uses its own cache rather than the Harvest Cache (see Section 6.7) because gathering isn't likely to exhibit much locality, and would hence affect the Cache's performance.

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
```

One final note: 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 bytes 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 10485760        # 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 more powerful data "mirroring" facility than the often-used `mirror`[29] package. In particular, 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 4.5.4). 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:

[29] `ftp://src.doc.ic.ac.uk/packages/mirror/mirror.tar.gz`

```
@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 singe 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 Version 1.3 has an improved debugging facility. Extra information from specific programs and library routines can be logged by setting debugging flags. A debugging flag has the form `-D`*section*,*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           Common liburl URL processing
section  21, level 1, 5        Common liburl HTTP routines
section  22, level 1           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           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           Gatherer essence post-summarizing
section  69, level 1, 5, 9     Common SOIF template processing
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 102, level 1                Broker Glimpse 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 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 4.5.4 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 SGML-based HTML summarizer is not able to recognize it. See Section 4.5.2 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 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 http bookmarks, 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 gatherer remote, manually-created information (as in our PC Software Broker[30]), 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 Staring with Harvest Version 1.2 we changed the Gatherer to conform to RFC 1738 [3], which says that a tilde inside a URL should be encoded as "%7e" in, 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 4.5.4. 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.

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 **Essence dumps core** when run (from the Gatherer)

Solution Check if you're running a non-stock version of the Domain Naming System (DNS) under SunOS. There is a version that fixes some security holes, but is not compatible with the version of the DNS resolver library with which we link essence for the binary Harvest distribution. If this is indeed the problem, you can either run the binary Harvest distribution on a stock SunOS machine, or rebuild Harvest from source (more specifically, rebuild essence, linking with the non-stock DNS resolver library).

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:

[30] `http://harvest.cs.colorado.edu/brokers/pcindex/query.html`

(a) the hostname you gave does not really exist; or

(b) 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.

The Harvest executables for SunOS (4.1.3_U1) are statically linked with the stock resolver library from */usr/lib/libresolv.a*. If you seem to have problems with the statically linked executables, please try to compile Harvest from the source code (see Section 3). This will make use of your local libraries, which may have been modified for your particular organization.

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.

We would welcome reports of Harvest successfully working with NIS. Please email us at *harvest-dvl@cs.colorado.edu*.

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 or else on the firewall host itself.

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

(a) your connection to the Internet is temporarily down due to a circuit or routing failure; or

(b) 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 Mosaic, but without proxy support. We would be interested to hear about problems with Harvest and hostnames under the condition that the gatherer is unable to contact a host, yet you are able to use other network programs (Mosaic, telnet, ping) to that host without going through a proxy.

5 The Broker

5.1 Overview

5.2 Basic setup

The Broker is automatically started by the `RunHarvest` command. Other relevant commands are described in Section 3.7.

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 1, the Broker uses a flexible indexing interface that supports a variety of indexing subsystems. The default Harvest Broker uses Glimpse [14] as an indexer, but other indexers such as WAIS [13] (both freeWAIS[31] and WAIS, Inc.'s commercial version[32]) and Nebula [6] also work with the Broker (see Section 5.8).

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. Recent distributions include a `RunBrokers.sh` script in the *lib/broker* directory which can be executed from your system startup scripts. There are a number of ways you can customize or tune the Broker, discussed in Sections 5.7 and 5.8. You may also use the `RunHarvest` command, discussed in Section 3.7, 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. Please be considerate of other users.

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[33], 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

[31] `ftp://ftp.cnidr.org/pub/NIDR.tools/freewais/`
[32] `http://www.wais.com/`
[33] `http://glimpse.cs.arizona.edu:1994/`

- 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.[34]

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 Appendix B.2.

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.

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`

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*.

[34] We are currently working on adding Harvest support for some of the more powerful features in the commercial WAIS engine.

Query options selected by menus or buttons

These checkboxes 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 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 matching "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.

Result set presentation

These 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.

Verbose display: This checkbox allows you to set whether results are displayed listing the filename, host, path, and Content Summary each on separate lines, or just with two lines listing the filename (without a label) and the Content Summary (with a label). The default is verbose.

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 manual pages[35].)

- `^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 wild cards
- `\*` 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."

[35] `http://glimpse.cs.arizona.edu:1994/glimpse.html`

Default query settings

The Harvest Broker uses the following default query settings with Glimpse:

- case insensitive (except for the Harvest Server Registry, since case is important there)
- match on word boundaries
- 0 spelling errors allowed
- display matched lines
- display object descriptions
- verbose display
- maximum of 50 results

The Harvest Broker uses the following default query settings with WAIS:

- display object descriptions
- verbose display
- maximum of 50 results

5.4 Customizing the Broker's Query Result Set

Starting with Harvest Version 1.1, 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 `BrokerQuery.pl.cgi` Perl program at query result time. This makes it easier to customize the output than using the older C version of `BrokerQuery.cgi`. Although this version runs more slowly than the C version, the time difference is small compared with the overall time for performing the indexed search.

`BrokerQuery.pl.cgi` allows you to customize almost every aspect of its HTML output. The file *$HARVEST_HOME/cgi-bin/lib/BrokerQuery.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 *BrokerQuery.cf* configuration file

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

```
<HarvestUrl>
http://harvest.cs.colorado.edu/
</HarvestUrl>
```

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

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

```
<BrokerDown>
Sorry, the Broker at <STRONG>$host, port $port</STRONG>
is currently unavailable.  Please try again later.<P>
</BrokerDown>
```

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.

OPTION	MEANING
`$maxresult`	The maximum number of matched lines to be returned
`$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.

OPTION	MEANING
`$objectnum`	The number of the returned object
`$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 BrokerQuery.cf file. Only definitions likely to be customized are described here.

`<Timeout>`
: Timeout value for `BrokerQuery.pl.cgi`. If the broker doesn't respond within this time, `BrokerQuery.pl.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 `BrokerQuery.pl.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). `$att` will is set to the attribute name, and `$val` is set to the attribute value.

5.4.2 Example *BrokerQuery.cf* customization file

The following definitions demonstrate how to change the `BrokerQuery.pl.cgi` output. 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 DisplayObject 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.

```
<PrintObject>
$objectnum <A HREF="$url"><STRONG>$desc</STRONG></A> \
[<A HREF="$cs_a://$cs_h/Harvest/cgi-bin/DisplayObject.cgi?object=$cs_p">\
indexing data</A>]
$opaque

</PrintObject>
```

```
<ObjectNumPrintf>
(%2d)
</ObjectNumPrintf>

<MatchedLineSub>
s/^Matched line: *//;
s/^([\w-]+# )[\w-]+{\d+}:\t/\1/;
s/^([\w-]+#)/<I>\1<\/I>/;
s/^.*/     $&/;
</MatchedLineSub>

<EndBrokerResults>
<STRONG>Found $nopaquelines matched lines, $nobjects objects.</STRONG>
<P>\n
</EndBrokerResults>
```

5.4.3 Integrating your customized configuration file

The `BrokerQuery.pl.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 `BrokerQuery.pl.cgi` program.

The simplest way to specify the customized file is by placing and `<INPUT>` tag in the HTML form. For example:

```
<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

Since version 1.2 the Broker allowed specific attributes from matched objects to be returned in the result set. However, there was no real support for this in `BrokerQuery.pl.cgi`. With version 1.3 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. For example:

```
<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 *BrokerQuery.cf* file described in Section 5.4.1.

5.5 World Wide Web interface description

To allow popular 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[36] (CGI), consists of the following:

- HTML files that use Forms[37] 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. Since installing an HTTP server can sometimes be difficult, you might also want to look at the list of *Frequently Asked Questions*[38] on the subject. Section 3.5 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.

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 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* directory. The *query.html* file uses the value of the "broker" FORM tag to pass the name of the broker to *BrokerQuery.pl.cgi* which in turn retrieves the host and port information from *Brokers.cf*.

[36] `http://hoohoo.ncsa.uiuc.edu/cgi/overview.html`
[37] `http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-forms/overview.html`
[38] `http://sunsite.unc.edu/boutell/faq/www_faq.html#provide`

CGI programs

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

`BrokerQuery.pl.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. The result set contains links to the Broker's home page, links to the content summaries of the matched objects, and a link to the Harvest home page. Note that the older C version, `BrokerQuery.cgi`, is also included in the distribution.

`DisplayObject.cgi`
: This program displays the content summaries from the Broker. It retrieves a SOIF object from the Broker, then formats it in HTML for the user,which contains a link to the *soifhelp.html* page, a link to the URL of the object, and a link to the Harvest home page.

`BrokerAdmin.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.

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):

queryhelp.html
: Provides a tutorial on constructing Broker queries, and on using the Glimpse and WAIS *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. Each content summary that the user displays will have a link to this help page.

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 Sections 5.6 and 5.9 for more detailed 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 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.
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 to the 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. This is usually used only for Replica brokers.

Rotate log file:
Rotates the current log file to LOG.YYMMDD. 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.

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.

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 #Passowrd 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 3.0 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 manual pages[39]. 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.

The `glimpseserver` program

Starting with Harvest Version 1.1, 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, or a Latent Semantic Indexing [9] back-end.

At present we distribute code to support an interface to both the free and the commercial WAIS index/search engines, Glimpse, and Nebula [6] [40].

[39] `http://glimpse.cs.arizona.edu:1994/glimpsehelp.html`

[40] While Nebula was built by one of the Harvest investigators' research groups, we do not presently distribute the Nebula system with Harvest. We will do so in a future release of Harvest.

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

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 [41] and are willing to lose the more powerful query features. [42]

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`.

Using Verity as an indexer

Version 1.3 of Harvest includes support for using Verity Inc.'s Topic indexing engine with the broker. In order to use Topic with Harvest, a license must be purchased from Verity[43]. At this point, Harvest does not make use of all features in the Topic engine. However, does include a number of features that make it attractive:

- Background indexing: the broker will continue to service requests as new objects are added to the database.
- Matched lines (or Highlights): lines containing query terms are displayed with the result set.
- Result set ranking
- Flexible query operations such as proximity, stemming, and thesaurus.

For more information on Verity, or to retrieve the package, please see the `components/Verity` directory of the FTP site[44].

Using GRASS as an indexer

As a demonstration of Harvest's flexibility, we have created a broker for spatial data. This broker uses the GRASS GIS package[45] as its indexer.

A Gatherer was customized to extract coordinate and other information from DLG "tar" files available from the National Wetlands Inventory FTP site[46]. Each DLG file retrieved by the gatherer was imported into a GRASS database with the `v.in.dlg` program. Next a summary (SOIF template) was generated from the output of the GRASS `v.stats` command.

Only the eleven "`index.c`" interface functions needed to be changed in the broker. A `grassindex` program was written which took the brokers SOIF templates and generated a GRASS vector map. The

[41] WAIS indexes use fine-grained blocks and a sorted index to minimize search-time I/O, while Glimpse allows regular expressions and other features by searching the index sequentially, and allows indexing granularity to be adjusted to trade index size for search time.

[42] We are currently working on adding Harvest support for some of the more powerful features in the commercial WAIS engine.

[43] `http://www.verity.com/`

[44] `ftp://ftp.cs.colorado.edu/distribs/harvest/components/Verity/`

[45] The Geographic Resources Analysis Support System package is from the US Army Corps of Engineers. See `http://www.cecer.army.mil/grass/GRASS.main.html`.

[46] `http://www.nwi.fws.gov/maps_ftp.html`

vector map consists of a set of Minimum Bounding Rectangles (MBRs). There is one MBR for each SOIF template.

We provided two ways to query this database. The first was to find all MBRs which intersected another named area. We used US States as the named areas. This allowed the user to find all wetlands data for Alabama, for example. Additionally, an HTTP "imagemap" was set up so that the user could click inside a state to generate the query.

The second query provided was to find all datafiles within two arc-degrees of a given coordinate. Again an HTTP imagemap was used to generate the coordinate request. The image coordinates were converted to lat/lon and given to a custom-written GRASS program. This program located all MBRs with centers within two arc-degrees of the given coordinate.

A demonstration of this Broker is available[47].

The "`index.c`" for using GRASS as an indexer is distributed with the Version 1.3 source code. Numerous support programs and scripts used with our demo broker are available[48]. The gatherer is also available[49].

Note that the provided support code will not work "out-of-the-box." It was quickly assembled as a demonstration. In order to make it work you need to have good working knowledge of both Harvest and GRASS.

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 No.	Remote Process	Description	Compressed
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 collection	No
6	Broker	Full collection based on a query	No
7	Broker	Incremental collection 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:

[47] `http://harvest.cs.colorado.edu/Harvest/brokers/grass/`

[48] `http://harvest.cs.colorado.edu/Harvest/brokers/grass/Support.tar.gz`

[49] `http://harvest.cs.colorado.edu/Harvest/gatherers/grass-nwi/`

INDEXER	FLAG	DESCRIPTION
All	`#desc`	Show Description Lines
Glimpse	`#index case insensitive`	Case insensitive
Glimpse	`#index case sensitive`	Case sensitive
Glimpse	`#index error N`	Allow N errors
Glimpse	`#index matchword`	Matches on word boundaries
Glimpse	`#index maxresult N`	Allow max of N results
Glimpse	`#opaque`	Show matched lines
WAIS	`#index maxresult N`	Allow max of N results
WAIS	`#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).

Symptom I just upgraded to Glimpse 3.0, and searches fail.

Solution The pre-3.0 indexes are incompatible with the 3.0 indexes. You need to reindex your data using Glimpse-3.0.

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 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:

(a) Run your Gatherers on the same host.

(b) remove the duplicate URLs in a customized version of the *BrokerQuery* 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 4 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). One way this happens is if you use a replica site that is running a different engine than you're used to (e.g., the Brokers at town.hall.org use commercial WAIS, while the Brokers at harvest.cs.colorado.edu use Glimpse). If you are setting up your own Broker (rather than using someone else's Broker), see Section 5.8 for details on how to switch to other index/search engines. Or, it could be that your `BrokerQuery.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.

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/srm.conf* file, when running the NCSA `httpd`. (The 'ScriptAlias' and 'Alias' setup is described in the *INSTRUCTIONS* file in the Harvest software distribution.)

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. For example, we have performance problems on our demonstration machine (harvest.cs.colorado.edu) because we run over a dozen Brokers there. If several users issue queries to different Brokers at once, quite a bit of paging results, and performance degrades noticeably. 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 64 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` (see Section 6.3) 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

`BrokerQuery.pl` UNIX process per query, and inefficiently time sliced by the UNIX kernel. With an accelerator the `BrokerQuery.pl` 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.

6 The Object Cache

6.1 Overview

The Object Cache allows users to retrieve FTP, Gopher, and HTTP data quickly and efficiently, often avoiding the need to cross the Internet. The Harvest cache is more than an order of magnitude faster than the CERN cache[50] and other popular Internet caches, because it never forks for WWW and Gopher access, is implemented with non-blocking I/O, keeps meta data and especially hot objects cached in RAM, caches DNS lookups, supports non-blocking DNS lookups, and implements negative caching both of objects and of DNS lookups. A technical paper is available that discusses the Harvest cache's design, implementation, and performance [7].

The Cache can be run in two different modes: as a proxy object cache, or as an httpd *accelerator*. In this section we discuss the use as a proxy cache; we discuss the httpd accelerator in Section 6.3.

The Cache consists of a main server program `cached`, a Domain Naming System lookup caching server program `dnsserver`, a Perl program for retrieving FTP data, and some optional management and client tools. The FTP program arose because of FTP complexities—while we retrieve Gopher and HTTP data from across the Internet using C code built in to `cached`, we retrieve remote FTP data using an external program (`ftpget`), which uses three Perl library files (discussed below). Once the FTP data have been loaded into the local cached copies, subsequent accesses are performed without running these external programs.

When `cached` starts up, it spawns three `dnsserver` processes, each of which can perform a single, blocking Domain Naming System (DNS) lookup. This reduces the amount of time the cache waits for DNS lookups. The number of `dnsserver` processes to use can be changed in the *cached.conf* file. Future versions may implement non-blocking DNS queries inside the `cached` process and eliminate the need for `dnsserver`.

Another big change with version 1.3 is that objects cached to disk are persistent. Upon restart `cached` now spends some time reloading metadata about on-disk objects. Currently, the cache does not make use of the HTTP "If-Modified-Since GET" feature. Cached objects are removed when they expire. A future release will support the conditional GET.

By default, the cache sends an electronic mail message to *cache_tracker@cs.colorado.edu*, to help us keep track of where caches are running in the Internet. The message lists only the host name, IP address, and port number. You can disable this message by changing the *mail_trace* configuration variable in the *cached.conf* file.

6.2 Basic setup

To set up the Cache server `cached`:

1. Verify your installation. *$HARVEST_HOME/bin* should contain the `cached` program. *$HARVEST_HOME/lib* should contain the `ftpget.pl` program and the Perl library files: *chat2.pl, ftp.pl*, and *socket.ph*. *$HARVEST_HOME/lib/cache* should contain the `client` and `dnsserver` programs.

2. Modify *$HARVEST_HOME/lib/cache/cached.conf* for your site; you may use the `-f` flag to cached to specify the location of your *cached.conf*). In *cached.conf*, any line beginning with a "#" character is ignored. If a variable itself is commented out (or not included in the configuration file), it is given a default value, as specified below.

   At a minimum, you should edit the settings of *cache_dir, cache_log, cache_access_log*, and *cache_mgr*. You can also set the *cache_ftp_program* and *cache_dns_program* variables to point to where your `ftpget` and `dnsserver` programs are located (respectively), if they aren't in directories that are in your path. You can also change other variables, such as per-object timeouts. (Each of the variables is documented in comments in the *cached.conf* file supplied in the Harvest distribution.) In particular, you can specify the topology of neighbor and parent caches, using one or more

[50] `http://www.w3.org/hypertext/WWW/Daemon/Status.html`

cache_host variables (see Section 6.7.). Note that `cached` uses two port numbers for the ASCII and UDP protocol interfaces. Example values are shown in the provided *cached.conf* file. Also, note that cached will always append to the log files (e.g., they're never truncated).

3. Run `cached` (or use the (preferred) `RunCache` program included in the Harvest distribution)[51]. `cached` does *not* need to be run as root. The command line arguments for `cached` are detailed in Appendix A.2.

6.3 Using the Cache as an httpd accelerator

The *httpd-accelerator* is a specially configured Harvest cache that intercepts incoming HTTP requests, on average doubling your performance. Requests that miss and dynamically evaluated queries are forwarded to your httpd for evaluation. Requests that hit the accelerator are serviced blazingly quickly because the Harvest cache is implemented much more efficiently than existing HTTP implementations.

The httpd-accelerator is compatible with both the CERN and the NCSA implementations of httpd. For more information, see the distribution package[52].

Note that it is best not to run a single Cache process as both an httpd-accelerator and a proxy cache, since these two modes will have different working sets. You will get better performance by running two separate caches on separate machines. However, for compatability with how administrators are accustomed to running other servers (such as CERN) that provide both proxy and Web serving capability, we allow the Cache to be run as both a proxy and an accelerator if you sent the *httpd_accel_with_proxy* variable to "on" inside your *cached.conf* configuration file.

6.4 Using the Cache's access control

The Cache support IP-based access control lists for both the proxy interface and the management interface.

The access control lists will allow or deny clients based on IP number matching. The order of the rules is important, they are read and checked sequentially. The keyword *all* will match all IP numbers. For example, to allow *only* hosts on the Class C subnet 128.125.51.0 and the Class B subnet 128.126.0.0 to access the proxy interface to the Cache, specify:

```
access_allow    128.125.51.0
access_allow    128.126.0.0
access_deny     all
```

To allow or deny access to the proxy interface, use the *access_allow* and *access_deny* tags in *cached.conf*. To allow or deny access to the management interface, use the *manager_access_allow* and *manager_access_deny* tags in *cached.conf*.

6.5 Using the Cache's remote instrumentation interface

The Cache provides a remote instrumentation interface, allowing you to gather statistics about many caches from a single graphical client. The instrumentation interface has two user interfaces: one is implemented using WWW, and the other is implemented using Tcl/Tk [16].

To run the WWW-based Cache Manager: open the URL */Harvest/cgi-bin/cachemgr.cgi* on your local HTTP server. You will receive a HTML form in which you can specify the hostname and port on which your Cache is running, and the type of information you wish to receive. After you've entered this information, click on the "Submit" button to send the request to the Cache.

The instrumentation interface also allows caches to be shutdown remotely. To provide a measure of security, we use a simple password mechanism. For this purpose, you should add the user "cache" to

[51] You should edit one of your */etc/rc** files so the Cache is automatically started when your machine boots up.

[52] `http://harvest.cs.colorado.edu/harvest/httpd_accel.html`

your */etc/passwd* file (or the passwd ypmap for *NIS*). cached will check the given password with this account when a shutdown is requested.

6.6 Setting up WWW clients to use the Cache

Users can retrieve objects through the Cache by setting three environment variables before running NCSA Mosaic (version 2.4) or Lynx. The easiest way is to use the provided CachedMosaic or CachedLynx script. These scripts simplify migrating many users to the Cache without changing each user's local environment. To do this, change each script to contain the name of the machine on which you are running a Cache. Then, rename the standard Mosaic and Lynx programs, and change CachedMosaic or CachedLynx to use these new names. Rename the scripts to the executable names that users would normally use to access the standard Mosaic or Lynx programs (e.g., xmosaic and lynx). Finally, copy the scripts in the path where the standard Mosaic and Lynx normally go.

The UNIX Netscape client uses the proxy environment variables, but there are no environment variables for PCs or Macs. What's more, it is difficult to set up "wrapper" programs (as described above) for the UNIX Netscape client, because Netscape will write its initialization file (*/.netscape-preferences*) with an empty set of PROXY variables the first time a user saves options – causing the proxy pointers inherited by the wrapper programs to be ignored. Therefore, to use the Cache from Netscape, the best approach is to set proxy pointers via the "Options/Preferences/Proxies" menu.

6.7 Running a Cache hierarchy

To reduce wide-area network bandwidth demand and to reduce the load on HTTP servers around the Web, Harvest caches resolve misses through other caches higher in a cache hierarchy, as illustrated in Figure 3. For example, several of the Harvest project members are running caches on their home workstations, configured as children of caches running in laboratories at their universities. Each cache in the hierarchy independently decides whether to fetch the reference from the object's home site or from the Cache or caches above it in the hierarchy.

The cache resolution algorithm also distinguishes *parent* from *neighbor* caches. A parent cache is a cache higher up the hierarchy; a neighbor cache is one at the same level in the hierarchy, provided to distribute cache server load. When a cache receives a request for a URL that misses, it uses UDP "pings" to try to determine which of the neighbor caches, parent caches, or object home site can satisfy the request most efficiently.

Note that, as the hierarchy deepens, the root caches become responsible for more and more clients. For this reason, we recommend that the hierarchy terminate at the first place in the regional or backbone network where bandwidth is plentiful.

To place your Cache in a hierarchy, use the *cache_host* variable in the *cached.conf* to specify the parent and neighbor nodes. For example, the following *cached.conf* file on *littleguy1.usc.edu* configures its cache to retrieve data from one parent cache and two neighbor caches:

```
#
#  cached.conf - On the host: littleguy1.usc.edu
#
#  Format is: hostname  type  ascii_port  udp_port
#
cache_host bigserver.usc.edu   parent   3128 3130
cache_host littleguy2.usc.edu  neighbor 3128 3130
cache_host littleguy3.usc.edu  neighbor 3128 3130
```

Note: earlier Versions of the Cache supported a tcp_port. That is no longer supported starting in Version 1.3.

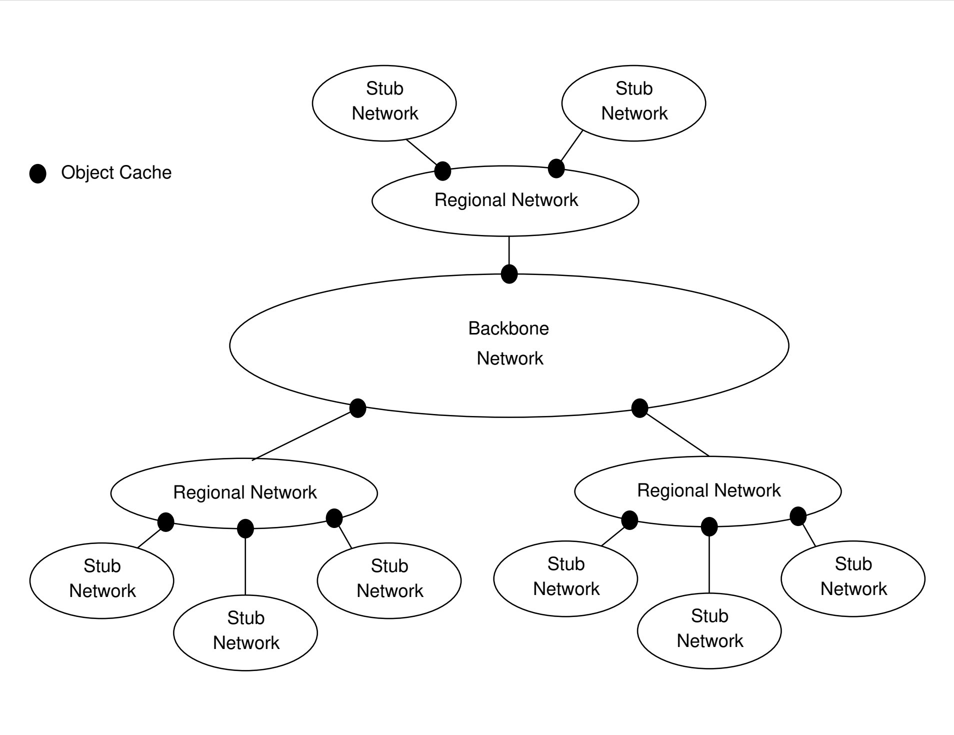

Figure 3: Hierarchical Cache Arrangement

6.8 Using multiple disks with the Cache

The *cache_dir* variable (see Section 6.2) only allows you to specify one directory into which cached objects are stored. If you have large caching data requirements (e.g., for scientific data access or if you are running a high-level cache through which many children Caches resolve misses), you may need to have the Cache store data onto several disks. In a future version of Harvest we will provide a way for users to to list multiple *cache_dir*'s and have the Cache cycle through all of them when storing data. For now, you can use one of two "tricks" to store cached data on multiple disks.

If you are running under SunOS, the easiest method is to create a memory-based file system using the SunOS *tmpfs* interface, create a directory on this file system, and then specify this directory in the Harvest *cache_dir* configuration variable. Next, use the SunOS `swapon` command to specify additional disks that are used as backing store for the *tmpfs* specified file system. See the SunOS manpages for *tmpfs* and `swapon` for more details.

If you are not using SunOS, you can achieve multiple disk usage with the cache using the following hack: Once the cache is running, remove part of its caching subdirectory structure (the numbered directories under the Cache's *data* directory), and replace it with one that is spread over multiple file systems. For example, you could do something like this:

```
% mkdir /my/other/big/disk/{5,6,7,8,9}{0,1,2,3,4,5,6,7,8,9}
% cached &
% sleep 5
% cd /tmp/cache
% rm -rf {5,6,7,8,9}?
% ln -s /my/other/big/disk/{5,6,7,8,9}? .
```

6.9 Details of Cache operation

6.9.1 Cache access protocols

The cache supports three access protocols: *tcp*, (called *encapsulating* in [7]), *udp* (called *connectionless* in [7]), and *ascii* (called *proxy-http* in [7]). The *tcp* protocol encapsulates cache-to-cache data exchanges to permit end-to-end error detection via checksums and, eventually, digital signatures. This protocol exchanges an object's remaining Time-To-Live (TTL). The cache uses the *udp* protocol to implement the parent-child resolution protocol. This protocol also permits caches to exchange small objects without establishing a TCP connection, for efficiency. While the *tcp* and *udp* protocols both support end-to-end reliability, the *ascii* protocol is the supported by most Web browsers (e.g., NCSA Mosaic, Netscape, and Lynx), in so-called proxy mode. In that arrangement, clients request objects via one of the standard information access protocols (FTP, Gopher, or HTTP) from a cache process. The term "proxy" arose because the mechanism was primarily designed to allow clients to interact with the WWW from behind a firewall gateway. Proxy access is transparent to the user.

In Harvest v1.3, we added an unsupported, user-contributed patch to the Cache which allows WAIS proxy access. To enable this functionality, set the *wais_relay* variable in your *cached.conf* to the host and port on which your WAIS server is running.

6.9.2 Cacheable objects

It does not make sense to cache certain types of objects. For example, it doesn't make sense to cache queries, since the result of a query might change each time for some services. The following rules are used to determine which objects **not** to cache:

1. Any HTTP URLs that do not use the 'GET' REQUEST_METHOD
2. Any HTTP URLs that use a CGI program denoted by a "?" character in the URL, or "`/cgi-bin/`" in the URL.
3. Any HTTP URLs with the *Authorization* tag in the MIME request header (this is used as a response to "401" messages when an HTTP server asks for a password before retrieving a file).
4. Any "401" HTTP responses.

6.9.3 Unique object naming

A URL does *not* name an object uniquely; the URL *plus* the MIME header issued with the request uniquely identify an object. For example, a WWW server may return a text version of a postscript object if the client's browser is not able to view postscript. We believe that this capability is not used widely, and currently the cache does not insist that the request MIME headers match when a request hits the cache. However, the cache does record the MIME header used to fetch each object.

6.9.4 Cache consistency

The Cache employs TTL-based cache consistency, patterned after the Internet's Domain Naming System [15]. Because of this, the Cache can return stale data. Unfortunately, HTTP, Gopher, and FTP provide neither a means for owners to specify TTLs, nor a protocol to pass TTLs from servers to caches and clients [53]. Hence, when the Harvest cache fetches an object from the object's home, it is forced to assign a default TTL. When the TTL expires, the cache discards the object. When a cache fetches an object from a parent or neighbor, it inherits the parent's remaining TTL. Version 1.3 includes support for setting TTLs as a percent of an object's lifetime. Different TTL parameters can be specified to match different URL regular expressions.

[53] Netscape Communications Corp. is promoting active documents, which is the needed standard.

For measurements about the Cache's consistency mechanism (and other mechanisms with which we have experimented), see [7].

Periodically, the Cache will run a garbage collection routine which will purge all objects from the cache which have expired TTLs. You can adjust the frequency of the garbage collection by changing the *clean_rate* value in *cached.conf*. By default, the Cache will purge expired objects every 30 minutes.

6.9.5 Negative caching and DNS caching

To reduce the costs of repeated failures (e.g., from erroneously looping clients), we implemented two forms of negative caching. First, when a DNS lookup failure occurs, we cache the negative result for one hour. Second, when an object retrieval failure occurs, we cache the negative result for a parameterized period of time, settable via the *negative_ttl* variable in the *cached.conf* file. The default value is 5 minutes.

The cache also implements a cache of successful DNS lookups. The timeout for this cache is "hard-coded" at one day.

6.9.6 Security and privacy implications

WWW browsers support various authorization mechanisms, all encoded in MIME headers exchanged between browser and server. The *basic* authorization mechanism involves clear-text exchange of passwords. For protection from eavesdropping, a *Public Key* authorization mechanism is available. Here, the server announces its own public key in clear-text, but the rest of the exchange is encrypted for privacy. This mechanism is vulnerable to IP-spoofing, where a phony server can masquerade as the desired server, but the mechanism is otherwise invulnerable to eavesdroppers. Finally, for those who want both privacy and authentication, a *PGP* based mechanism is available, where public key exchange is done externally.

For example, a *basic* authentication exchange follows the following dialog:

```
Client: GET <URL> HTTP/1.0
Server: HTTP/1.0 401 Unauthorized -- authentication failed
Client: GET <URL> HTTP/1.0
        Authorization: <7-bit-encoded name:password>
Server: One of:
          Reply (authorized and authenticated)
          401 Unauthorized (not authorized)
          403 Forbidden (not authenticated)
          404 Not Found
```

Note that the basic and public key schemes offer roughly the same degree of security as Internet rlogin. Their authentication relies on client IP addresses, which can be spoofed, and they assume that intruders do not masquerade as real servers. Their authorization relies on user names and passwords, which can be snooped.

When a server passes a `401 Unauthorized` message to a cache, the cache forwards it back to the client and purges the URL from the cache. The client browser, using the desired security model, prompts for a username and password, and reissues the GET URL with the authentication and authorization encoded in the request MIME header. The cache detects the authorization-related MIME header, treats it as any other kind of non-cacheable object, returns the retrieved document to the client, but otherwise purges all records of the object. Note that under the clear-text *basic* authorization model, anyone, including the cache, could snoop the authorization data. Hence, the cache does not weaken this already weak model. Under the *Public Key* or *PGP* based models, neither the cache nor other eavesdroppers can interpret the authentication data.

Proxy-caching defeats IP address-based authentication, since the requests appear to come from the cache's IP address, rather than the client's. However, since IP addresses can be spoofed, we consider this liability an asset of sorts. Proxy-caching does not prevent servers from encrypting or applying digital signature to their documents, although encryption disables caching.

As a final issue, unless Web objects are digitally signed, an unscrupulous system administrator could insert invalid data into his proxy-cache. You have to trust the people who run your caches, just as you must trust the people who run your DNS servers, packet switches, and route servers.

6.9.7 Summary: object caching "flow chart"

To summarize and fill in missing details about the above discussion, the Cache works as follows:

- When the Cache starts, it writes its process identifier to the file *cached.pid* in the same directory as where *cached.conf* resides. This can be useful if you want to write programs that monitor the status of the cache.
- When a request is made for an object currently in the cache, it is returned to the client and noted in the cache log files, and noted in the cache's "Least Recently Used" (LRU) chain.
- When a request is made for an object not currently in the cache, the cache retrieves the object and simultaneously feeds bytes to the client and stores the object into both VM and disk. If the object turns out to be bigger than the parameterized *cache_mem* setting (from the *cached.conf* file) or if it's bigger than the maximum object size for that access type (also specified in the *cached.conf* file), the object is rejected, and an error is returned to the user (rather than returning the object). In a future version of Harvest we will allow proxying for objects that are too large, passing them to the client but removing them from VM and disk.
- The cache performs LRU replacement in both VM and disk, but with different size limits, as specified in *cache_mem* and *cache_swap*. Any object in VM will also be on disk, but not necessarily vice versa.
- When the amount of RAM used by the hot-object VM cache reaches *cache_mem_high* % of the specified *cache_mem* size, the cache starts throwing objects out of the VM cache, until they reach *cache_mem_low* % of the specified *cache_mem* size. (These values are set in the *cached.conf* file.) Objects evicted from the VM cache stay in the disk cache until their cache-assigned TTL expires, they are evicted by the disk cache replacement policy, or the user manually evicts them by clicking the browser's "reload" button.

  The high/low water mark mechanism is intended to let the Cache cut back on memory usage when it starts using too much memory, so it can avoid constantly bringing in objects and then having to evict them. There's no high/low water mark on disk.

6.10 Meanings of log files

The Cache logs information to four files, the locations for the first three of which are specified from the *cached.conf* file. (The final file is always stored in the file *log* in the cache's data directory):

1. *cache_access_log*: this is Cache access logfile, which logs the hits and misses. The format of this file is one line per access, with the following fields:
   (a) timestamp (in RFC 850 date format)
   (b) retrieved URL
   (c) client's IP address
   (d) object size (only non-zero when there's a hit)
   (e) hit/miss indicator. This field encodes the access method (TCP, UDP, and our binary protocol), HIT/MISS, and occasionally a reason (e.g., TCP_MISS_TTL means it was a miss because the object had timed out). Note: prior to Harvest Version 1.2, the client's IP address was not logged to this file.

The *cache_access_log* also can be written in the httpd common logfile format[54] which is used by many HTTP servers. To enable, set *emulate_httpd_log* in your *cached.conf* file to *on*. By default, the Cache will write this log file in the httpd common logfile format.

2. *cache_log*: this file contains logs of error messages, such as errors binding to sockets that are already in use (e.g., because you restarted the cache too quickly; see Section 6.11).
3. *cache_hierarchy_log*: this file logs the parent/neighbor relationships for each hierarchical resolution that occurs (e.g., if an object is faulted from the parent cache, this file will indicate that fact).
4. *cache_dir/log*: The cache stores objects on disk as numbered files inside of numbered directories. The *cache_dir/log* file stores the mapping between URL and numbered file.

Note that all logs are appended (rather than recreated whenever the cache starts up).

6.11 Troubleshooting

Symptom The cache **refuses to start**, because the port number is tied up.

Solution Usually this means that you killed the cached process recently, and the UNIX operating system hasn't timed out the port number yet. Wait a minute and try starting cached again. Finally, it may be that your *cache_dns_program* variable in the *cached.conf* file is incorrectly set.

Symptom Once I try to retrieve an object from a site that is down, when I try again the **cache immediately tells me it is down**, seemingly without trying again.

Solution This happens because the Harvest cache employs *negative caching* for failed Domain Naming System (DNS) lookups and object retrievals. In other words, after a failure occurs, it will not try again for a configurable period of time (the default is 5 minutes; it can be set from the *negative_ttl* variable in your *cached.conf* file). Note that your browser's *Reload* button will have no effect on the negative DNS cache.

Symptom The *Reload* button doesn't cause stale images and other data to be reloaded.

Solution This is a shortcoming of Web browsers – they don't give you a way to "point" the *Reload* button at inlined images, etc.

Symptom When I get a temporary name lookup failure, the cache thinks the object still can't be looked up when I try again later.

Solution For performance and scaling reasons, we cache *negative* results from DNS lookups for 5 minutes (and, the *Reload* button has no effect on the negative DNS cache). Note that this is not a configuration file settable parameter (i.e., you would have to change the source code to change it).

As a work-around, the Harvest Cache includes a little program called "client" that will let you reload stale objects as follows:

```
Usage: client [-nr] [-c hostname] [-p port] url
       -n don't print object to stdout.
       -r force reload.
       -c get object from hostname default is localhost.
       -p use port as a cached port number.  Default is 3128.
```

So, if you know the URL of the inlined image you can reload it. This is not a very good solution (e.g., you need to view the source of the object to find the URL that needs to be reloaded, and then run 'client' from the shell to do the reload), but there's not much we can do to make it more convenient as long as the Web browsers don't provide better support for reloads.

[54] http://www.w3.org/hypertext/WWW/Daemon/User/Config/Logging.html#common-logfile-format

Solution A source code patch is available in the *contrib/Mosaic-2.4+cached-fixes/* directory, under the top-level directory of each Harvest software distribution sites[55], for an improved *Reload* button for NCSA Mosaic and Lynx. This patch will allow you to force re-retrieval of objects through the cache. Note that the *Reload* button already provides this functionality in Netscape.

Symptom The cache **fails with FTP URLs**.

Solution This is usually because you do not have the `ftpget.pl` program installed properly. Verify that `ftpget.pl` is in your PATH when you execute `cached` (or that the `ftpget.pl` program is explicitly listed in the *cache_ftp_program* variable in your *cached.conf* file). You can verify that `ftpget.pl` works by running:

```
% ftpget.pl - ftp.dec.com / I anonymous harvest-user@
```

Symptom **Gopher menus** retrieved through the cache appear **incorrectly formatted** from within my WWW browser.

Solution This is an alignment bug common to at least NCSA Mosaic version 2.4, Spyglass Mosaic, Netscape 1.0, and Lynx. We have informed Netscape Communications Inc., Spyglass Inc, and NCSA about this problem. In the mean time, you can repair the problem with a patch to Mosaic, which we make available in the *contrib/Mosaic-2.4+cached-fixes/* directory under the top-level directory of each Harvest software distribution sites[56]. Or, you can disable Gopher caching by unsetting the Gopher proxy environment variable (or the Gopher proxy preference setting from within Netscape).

Symptom My cached process uses up **more memory** than what I specified in the *cache_mem* variable in *cached.conf*.

Solution The *cache_mem* variable specifies how much memory is to be used by the cache's *hot object* RAM cache (see [7] for technical details). When the cache reaches this level of memory consumption it will not use any more data for hot objects, but it will continue to store meta data for cached objects (including those stored only on disk), at about 80-110 bytes per object. The memory size will thus continue to grow until the disk cache becomes full. In a future version of Harvest we may move meta data for cold objects to disk, and make *cache_mem* into a hard limit on the memory size of the cached process. Note that you can control how much memory is used by adjusting the *cache_mem_high* and *cache_mem_low* variables in *cached.conf*. When the amount of memory reaches the level specified in *cache_mem_high*, the cache discards hot objects from the RAM cache until usage reaches the level specified in *cache_mem_low*.

Symptom The cache aborts because it **can't bind** to the ASCII port.

Solution Check to see that no other process is using this port. If not, the port will become available after a short wait, typically less than 5 minutes. As background, we elected to enable port locking to prevent people from accidentally starting two cache processes. As a consequence, sometimes you can't bind to a port while TCP is going through the 3-way handshake necessary to terminate a previous connection.

Symptom The cache **quits silently**.

Solution The cache attempts to write fatal errors into /dev/console; if it cannot, it will exit silently when it encounters fatal errors. Also check that you've created the cache's working directory and that this directory is owned by the effective user id on which your cache is running.

Symptom I changed an object, but the cache keeps **returning the old version**.

[55] `http://harvest.cs.colorado.edu/harvest/gettingsoftware.html`
[56] `http://harvest.cs.colorado.edu/harvest/gettingsoftware.html`

Solution Try reloading the cache with the “reload” button on your Mosaic or Netscape client. The cache implements negative caching for 5 minutes, so you may have to wait up to 5 minutes to fetch an object if you fetched it once before the object was created. If the object is an image, note that Mosaic can't issue a reload. If you really want to flush an image, use netscape or the cache's reload program.

Symptom I launched the **cache or httpd-accelerator**, and they **don't work right**.

Solution Did you setenv HARVEST_HOME?

Symptom I tried to install the **httpd-accelerator**, but it **won't** let me **bind** to port 80.

Solution You have to start the httpd-accelerator as root if you plan to bind to a port numbered lower than 1024.

Symptom The cache/httpd-accelerator **works for everything but FTP**.

Solution Since the cache changes effective UID to “nobody” or “daemon”, check that your Perl libraries are all set to permission “r-xr-xr-x”.

7 The Replicator

7.1 Overview

The Harvest replicator will distribute copies of a Broker's database to replicas running throughout the Internet. Replication distributes the server load on a Broker, improving query performance and availability, and the replicator attempts to minimize network traffic and server workload when propagating updates.

The replicator manages a single directory tree of files. One site must be designated as the *master copy*. Updates to the master copy propagate to all other replicas and the master copy eventually overwrites any local changes made at individual replicas. It is possible to configure a replicated collection so that a different *master copies* manages separate sub-trees, to distribute the responsibility of (gathering and) managing a large collection. Each replicated collection is exported through a (single or possibly hierarchically nested) *replication group*. When a replica joins a replication group, it begins to fill with data. The right to join a replication group is managed by an access control list. If a replication group grows to hundreds or thousands of members, a new group can be created to ease management. This arrangement is illustrated in Figure 4.

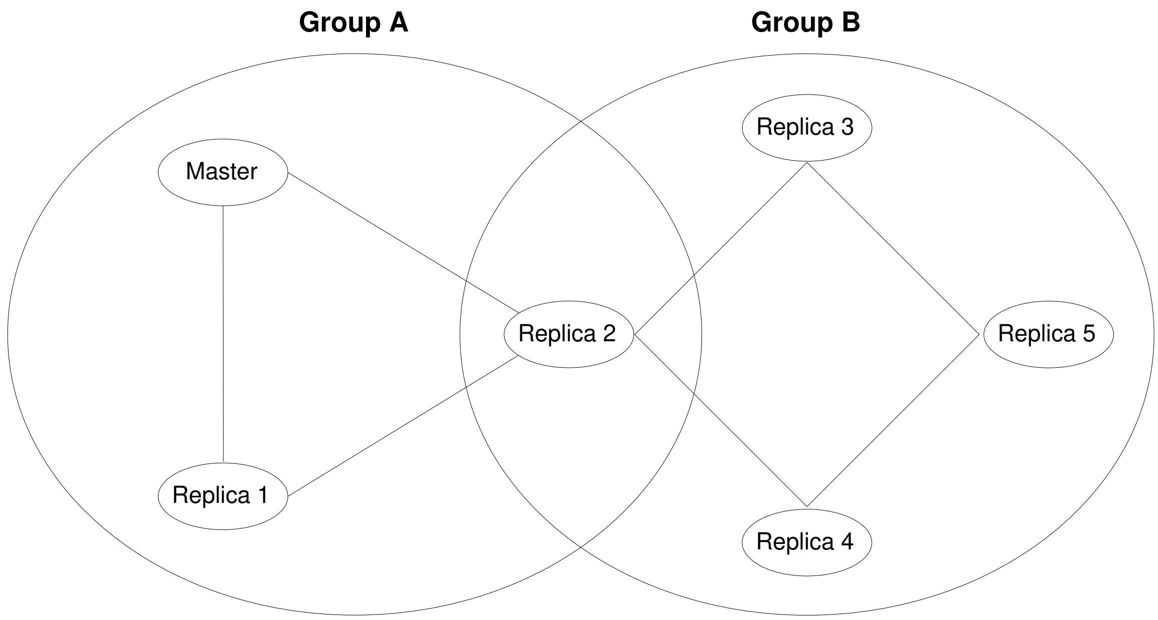

Figure 4: Replicator System Overview

The Harvest replicator consists of four components:

1. We use the `FTP mirror`[57] system to check file dates and sizes and to perform the actual file transfers between replicas.
2. `Mirrord` generates configuration files that tell `FTP mirror` where to retrieve data, based on a recent bandwidth and delay estimates between group members.
3. `Floodd` periodically performs bandwidth and delay measurements among members of a replication group, for use by `mirrord`. A `floodd` group master computes the "logical update topology" for the group, which depends on the bandwidth and delay estimates between group members.

[57] `ftp://src.doc.ic.ac.uk/packages/mirror/mirror.tar.gz`

4. `Archived` distributes updates of the *Version* file, which is used to determine when to run `FTP mirror`. This reduces the frequency with which `FTP mirror` runs, improving performance. The *Version* file is updated each time the Harvest Gatherer runs.

The replication system design is discussed in more depth in [8].

7.2 Basic setup

Since the `FTP mirror` system uses anonymous FTP, you need to set up anonymous FTP access on the machine where you will run a replicator. This is fairly straightforward, usually involving creating an account and home directory for user "ftp". See the `ftpd` manual page for details.

To run a Replicator, retrieve the replica distribution[58] (it's also included in the Harvest source distribution in the *src/replicator* directory), and then do:

```
% tar xf replica_distribution.tar
```

This will create the files *mirrord.tar.gz* and *CreateReplica*. Execute `CreateReplica` and follow the prompts. The default answers to the *CreateReplica* installation script will create a replica of Harvest's www-home-pages Broker[59]. We suggest you start by replicating the www-home-pages Broker before you create your own replicated Broker.

At the end of running `CreateReplica` you will be given the URL for a page that allows you to control and monitor the status of *floodd* and *mirrord*. For example, from this page you can force logical topology updates, check the time of next synchronization between *mirrord*'s, and view replication group membership and bandwidth estimates.

Note that it takes a while before data starts filling in your replica, because `floodd` needs a chance to run and compute bandwidths, and `mirrord` needs a chance to run and generate the `FTP mirror` configuration file needed to pull over the data. Typically data will start arriving 20-30 minutes after a new replica is first created. Note also that if you force an update before this time (using the `floodd` control/status HTML page), data will start arriving sooner, but may be retrieved from a suboptimal neighbor (e.g., across a slow network link).

The replicator allows you to tell it which files to replicate, allowing it to be used in more general settings than just for replicating Brokers. For the case of Brokers, however, the files you want to replicate are *admin/Registry* and the *objects/* directory. The other files are either set up locally by the site administrator, or regenerated from the replicated files (e.g., the index files are generated from the replicated objects).

In a future version of Harvest we will integrate the replica creation mechanism into the `RunHarvest` command.

CreateReplica usage line

The `CreateReplica` program is used as follows:

```
-h      This message.
-m      Creating replica for master site.
        For this case, it won't create broker, assume broker is already there.
-l      Specify parameters locally without querying  a remote site.
-g url  Get paramters from mirrord home page of a remote site.
        (-g http://harvest.cs.colorado.edu:4100/parameters)
```

Note: at present we have partially-automated support for managing port numbers when you create a new replication group. When you run `CreateReplica -m` you'll need to specify port numbers for the master. After that, you can send email to *mirrord@catarina.usc.edu* listing your port numbers and the

[58] `ftp://catarina.usc.edu/pub/broker/replica_distribution.tar`

[59] `http://harvest.cs.colorado.edu/brokers/www-home-pages/`

name of your replication group, and we'll register them in a database that is checked by slave sites when they run `CreateReplica`. Alternatively, you can simply provide the port numbers manually to each slave site (e.g., by sending email to the administrators at each site, containing the port numbers to use). In a future version of Harvest we will make the master/slave registration mechanism more automated, similar to how HSR registrations work (see Section 3.7).

7.3 Customizations

There are a number of variables set in the various *lib/* files. Most of the ones you would want to change are settable from the `CreateBroker` script. One useful file to know about is *lib/default-ignore*. This file lists regular expressions for names of files to avoid replicating. The *lib/default-ignore* file included in the Harvest distribution contains some regular expressions used to avoid files such as temporary copies of editor sessions, core dumps, and log files.

Another useful customization is to create an access control list. To do this, edit *group.conf*, to point to a file listing class A/B/C network numbers to allow or deny. An example follows:

```
;;
;; A paranoid group that only talks to known hosts.
;;
(:group-define
 (:group-name"group1")
 (:allow-site "128.125.0.0")          ; allow access from USC
 (:allow-site "128.138.0.0"))         ; allow access from UColo - Boulder

;;
;; A permissive group that disallows known trouble sites.
;;
(:group-define
 (:group-name "group2")
 (:deny-site "18.0.0.0"))             ; deny access to MIT hackers!
```

7.4 Distributing the load among replicas

Once you have a set of replicas set up, you need to arrange to have queries routed among them. In Harvest Version 1.0 we allowed users to select a query from the Broker's *query.html* page. Starting with Harvest Version 1.1 we made the load sharing automatic and transparent to the user, by interposing a process that redirects queries to one of the replica set, chosen at random. This process runs as a CGI program called *Query.cgi*, which is created by the `CreateBroker` command (which is run by the `RunHarvest` command). The *Query.cgi* program lives in the same directory as *query.html*, and redirects users to one of the *query.html*'s for that Broker.

By default, *Query.cgi* is never used, and *query.html* is used as the Broker's Home Page. If you run a replicated Broker, you need to do the following:

1. Modify the configuration file so your `httpd` treats all .cgi files (rather than just those in the cgi-bin directory) as CGI programs (if your `httpd` doesn't support this, then include the Broker directory as one of the valid CGI script directories). For example, in NCSA `httpd`, you need to add this line to your *srm.conf*:

   ```
   AddType application/x-httpd-cgi .cgi
   ```

2. Change the Broker-Home-Page in *broker.conf* to point to *Query.cgi* instead of *query.html*. Then, advertise this URL as the Broker's point of entry.

As replica Brokers are added, you need to edit *Query.cgi* to include the new *query.html* URLs. For the URL "http://whatever/Harvest/brokers/foo/Query.cgi" people can see the full list of replicas using the URL "http://whatever/Harvest/brokers/foo/Query.cgi?list".

7.5 Troubleshooting

Symptom *CreateReplica* compiles and starts *floodd* and *mirrord*, but I **can't see if anything is working**.

Solution Check your *floodd* status page[60] to see if your *floodd* is getting estimates from the other sites. Check your *mirrord* status page[61] to see if your *mirrord* is running. Note: the port numbers listed here assume you selected the default port numbers when you ran *CreateReplica*. If not, adjust accordingly.

Symptom The *mirrord* and *floodd* pages **never get updated**.

Solution The WWW does not support automatic updating of cached client's pages. You need to use your browser's reload button to get fresh information.

Symptom Under Mosaic, the *floodd* and *mirrord* pages **appear truncated**.

Solution This can occur because Mosaic's long MIME header causes *floodd* and *mirrord* to issue a TCP RST, which confuses proxy caches. Either disable your cache or query *floodd* and *mirrord* with netscape.

Symptom Both *floodd* and mirrord daemons are running, but **no data are arriving**.

Solution Click *synchronize* on your *mirrord* status page[62]. See if you see a "notification in progress" message.

Symptom I get "notification" messages, but still **no data show up**.

Solution Kill *mirrord* and restart it as "../bin/mirrord -llog" from inside your replica directory. This will create file replica/log, with extended trace information.

[60] `http://localhost:9500/`
[61] `http://localhost:4000/`
[62] `http://localhost:4000/`

8 References

[1] The Government Information Locator Service (GILS). Technical report, May 1994. Available from *http://info.er.usgs.gov/public/gils/gilsdoc.html.*

[2] T. Berners-Lee. *RFC 1630: Universal Resource Identifiers in WWW.* CERN, June 1994. IETF URI Working Group. Available from *ftp://ftp.internic.net/rfc/rfc1630.txt.*

[3] T. Berners-Lee, L. Masinter, and M. McCahill. *RFC 1738: Uniform Resource Locators (URL).* CERN, December 1994. IETF URI Working Group. Available from *ftp://ftp.internic.net/rfc/-rfc1738.txt.*

[4] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, Boulder, Colorado, Aug. 1994. Submitted for publication. Available from *ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.FullTR.ps.Z.*

[5] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The Harvest information discovery and access system. In *Proceedings of the Second International WWW Conference '94: Mosaic and the Web*, pages 763–771, Chicago, Illinois, Oct. 1994. Available from *ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.Conf.ps.Z.*

[6] C. M. Bowman, C. Dharap, M. Baruah, B. Camargo, and S. Potti. A file system for information management. *Proceedings of the Conference on Intelligent Information Management Systems*, June 1994. Pre-publication version available from *ftp://ftp.cse.psu.edu/pub/bowman/doc/iims.ps.Z.*

[7] A. Chankhunthod, P. B. Danzig, C. Neerdales, M. F. Schwartz, and K. J. Worrell. A hierarchical Internet object cache. Technical Report CU-CS-766-95, Department of Computer Science, University of Colorado, Boulder, Colorado, March 1994. Submitted for publication. Available from *ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/HarvestCache.ps.Z.*

[8] P. Danzig, K. Obraczka, D. DeLucia, and N. Alam. Massively replicating services in autonomously managed wide-area internetworks. Technical report, Department of Computer Science, University of Southern California, Jan. 1994. Available from *ftp://catarina.usc.edu/pub/kobraczk/ToN.ps.Z.*

[9] S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, and R. Harshman. Using latent semantic analysis to improve access to textual information. *Proceedings of the CHI '88*, 1993.

[10] D. R. Hardy and M. F. Schwartz. Customized information extraction as a basis for resource discovery. Technical Report CU-CS-707-94, Department of Computer Science, University of Colorado, Boulder, Colorado, Mar. 1994. To appear, *ACM Transactions on Computer Systems.* Pre-publication version available from *ftp://ftp.cs.colorado.edu/pub/cs/techreports/-schwartz/Essence.Jour.ps.Z* .

[11] D. R. Hardy and M. F. Schwartz. Essence: A resource discovery system based on semantic file indexing. *Proceedings of the USENIX Winter Conference*, pages 361–374, January 1993. Pre-publication version available from *ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Essence.Conf.ps.Z.*

[12] ISO. Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML). Technical report, International Organization for Standardization, 1986.

[13] B. Kahle and A. Medlar. An information system for corporate users: Wide Area Information Servers. *ConneXions - The Interoperability Report*, 5(11):2–9, November 1991. Available from *ftp://think.com/wais/wais-corporate-paper.text.*

[14] U. Manber and S. Wu. Glimpse: A tool to search through entire file systems. *Proceedings of the USENIX Winter Conference*, pages 23–32, January 1994. Pre-publication version available from *ftp://cs.arizona.edu/reports/1993/TR93-34.ps.Z.*

[15] P. Mockapetris. Domain names - concepts and facilities. RFC 1034, November 1987.

[16] J. K. Ousterhout. An x11 toolkit based on the tcl language. *Proceedings of the USENIX Winter Conference*, January 1991.

[17] R. L. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm.* MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992. Available from *ftp://ftp.internic.net/rfc/rfc1321.txt.*

[18] E. van Herwijnen. *Practical SGML.* Kluwer Academic Publishers, 1994. Second Edition.

[19] C. Weider and P. Faltstrom. The WHOIS++ directory service. *ConneXions - The Interoperability Report*, 8(12):20–24, December 1994.

A Programs and layout of the installed Harvest software

A.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*:

```
COPYRIGHT           README          brokers/        lib/
ChangeLog           RunHarvest*     cgi-bin/
INSTRUCTIONS.html   bin/            gatherers/
```

COPYRIGHT contains the copyright restrictions for Harvest. *ChangeLog* contains a detailed account of the history of changes to the Harvest software. *INSTRUCTIONS.html* contains the instructions on how to install and run the Harvest software. This HTML file is also maintained on-line[63]. *README* contains the README file from the Harvest source distribution. `RunHarvest` is the script used to create and run Harvest servers (see Section 3.7). The `RunHarvest` program is not in the *bin* directory so that it can be run without requiring the user to set their *HARVEST_HOME* environment variable first, and to make it stand out because it is the primary (and simplest) way to start up the system. `RunHarvest` has the same command line syntax as `Harvest` below.

A.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.
-expert        Most flexible Q&A.  No defaults used. (default)
-glimpse       Use Glimpse for the Broker. (default)
-wais          Use WAIS for the Broker.
-dumbtty       Dumb TTY mode (default).
-cleantty      Clean TTY mode - clears screen, etc.
-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.
```

`RunCache`
: Useful script to continually run the Harvest Cache.
Usage: `RunCache [cachedir]`

[63] `http://harvest.cs.colorado.edu/harvest/INSTRUCTIONS.html`

`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]`

`cached`

The Harvest Cache daemon. This program is run by the `RunCache` script.

Usage: `cached [-ehs] [-f config-file] [-d debug-level] [-[apu] port]`

```
-e                Print debug message to stderr.
-h                Print help message.
-R                Do not set REUSEADDR on port.
-s                Disable syslog output.
-f config-file    Use given config-file instead of /etc/cached.conf.
-d debug-level    Use given debug-level, prints messages to stderr.
-a port-number    Specify ASCII port number.  Defaults to 3128.
-p port-number    Specify TCP (binary) port number.  Defaults to 3129.
-u port-number    Specify UDP port number. Defaults to 3130.
```

`gather`

The client interface to the Gatherer.

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

A.3 *$HARVEST_HOME/brokers*

The *$HARVEST_HOME/brokers* directory contains 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*.

A.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).

A.5 *$HARVEST_HOME/gatherers*

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

A.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

Perl 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`

newsget.pl
: Perl program used to retrieve USENET articles and group summaries from NNTP servers.
Usage: `newsget.pl localfile news-URL`

soif.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`

A.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. Mainly used by the Replicator after a Broker has been updated.
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]`

`glimpse, glimpseindex, glimpseserver`
: The Glimpse indexing and search system as described in Section 5.

`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`

A.8 *$HARVEST_HOME/lib/cache*

The *$HARVEST_HOME/lib/cache* directory contains contains a simple testing client and some routines needed by the Tcl-based cache manager, as follows:

`CachedLynx, CachedMosaic`
: Wrapper programs to help migrate users to use the Harvest cache, as described in Section 6.

cached.conf
: Sample configuration file for `cached`, as described in Section 6.

client
Client interface to the Harvest Cache.
Usage: `client [-nr] [-c hostname] [-p port] url`

```
-n don't print object to stdout.
-r force reload.
-c get object from hostname default is localhost.
-p use port as a cached port number.  Default is 3128.
```

dnsserver
Daemon used by the Harvest Cache to do non-blocking DNS queries.
Usage: `dnsserver`

A.9 *$HARVEST_HOME/lib/gatherer*

The *$HARVEST_HOME/lib/gatherer* directory contains the default summarizers described in Section 4.5.1, plus various utility programs needed by the summarizers and the Gatherer, as follows:

***.sum**
Essence summarizers as discussed in Section 4.5.3.

***.unnest**
Essence presentation unnesters or exploders as discussed in Section 4.5.4 and Appendix C.2.

***2soif**
Programs used by Essence presentation unnesters or exploders to convert files into SOIF streams.

bycontent.cf, byname.cf, byurl.cf, magic, stoplist.cf, quick-sum.cf
Essence configuration files as described in Section 4.5.4.

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, mkcompressed, mkgathererstats.pl, mkindex, and 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; **mkcompressed** generates the compressed cache *All-Templates.gz* file; **mkgathererstats.pl** generates the *INFO.soif* statistics file; **mkindex** generates the cache of timestamps; and **rmbinary** removes binary data from a GDBM database.

deroff, detex, dvi2tty, extract-perl-procs, extract-urls, get-include-files, print-c-comments, ps2txt, ps2txt-2.1, pstext, skim, and unshar
Programs to support various summarizers.

dbcheck, enum, fileenum, ftpenum, ftpenum.pl, gopherenum, httpenum, newsenum, prepurls, and staturl
Programs used to perform the RootNode enumeration for the Gatherer as described in Section 4.3.

dbcheck checks a URL to see if it has changed since the last time it was gathered; enum peforms a RootNode enumeration on the given URLs; fileenum peforms a RootNode enumeration on "file" URLs; ftpenum calls ftpenum.pl to peform a RootNode enumeration on "ftp" URLs; gopherenum peforms a RootNode enumeration on "gopher" URLs; httpenum peforms a RootNode enumeration on "http" URLs; newsenum peforms a RootNode enumeration on "news" URLs; prepurls is a wrapper program used to pipe Gatherer and essence together; staturl retrieves LeafNode URLs so that dbcheck can determine if the URL has been modified or not. All of these programs are internal to Gatherer.

essence

The Essence content extraction system as described in Section 4.5.4.

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

```
--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
```

extractdb, print-attr

Prints the value of the given attribute for each SOIF object stored in the given GDBM database. print-attr uses stdin rather than *GDBM-file*.

Usage: `extractdb GDBM-file 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, print-template`
Program to generate valid SOIF based on a more easily editable SOIF-like format (e.g., SOIF without the byte counts). `print-template` can be used to "normalize" a SOIF stream; it reads a stream of SOIF templates from stdin, parses them, then writes a SOIF stream to stdout.
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]`

B 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 here[64]. 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.

B.1 Formal description of SOIF

The SOIF Grammar is as follows:

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

[64] `http://harvest.cs.colorado.edu/Harvest/cgi-bin/DisplayObject.cgi?object=harvest/soif-example`

B.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:

ATTRIBUTE	DESCRIPTION
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. (e.g., 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.
Keywords	Searchable keywords extracted from the object.
Last-Modification-Time	The time that the object was last modified. Defaults to 0.
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, PC-Compressed, Patch, Perl, PostScript, RCS, README, RFC, SCCS, ShellArchive, Tar, Tcl, Tex, Text, Troff, Uuencoded, WaisSource. For information about the default Essence summarizer actions for these types, see Section 4.5.1.
Update-Time	The time that the summary object was last updated. REQUIRED field, no default.
URL-References	Any URL references present within HTML objects.

C Gatherer Examples

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

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)
```

C.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).

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). 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 [17] and its size (*File-Size*) in bytes are also added to the content summary.

C.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 Appendix B).

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. A demo[65] of our LSM Gatherer and Broker is available.

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 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.

[65] http://harvest.cs.colorado.edu/Harvest/brokers/lsm/query.html

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[66]. 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
```

[66] The *harvest/src/gatherer/essence/unnest.c* file contains the definitions of nested types. To specify that a type is nested, follow the directions at the top of the *unnest.c* file.

```
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. We have a demo[67] available via the Web.

C.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 4.5.4). 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 Appendix C.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
```

We have a demo[68] that uses a similar customization to collect structured indexing information from over 20,000 Home Pages around the Web.

C.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 4.5.4. 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 C.2 and C.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.

[67] `http://harvest.cs.colorado.edu/Harvest/brokers/pcindex/query.html`

[68] `http://harvest.cs.colorado.edu/Harvest/brokers/www-home-pages/query.html`

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.

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"
```

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
```

Index