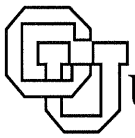
HARVEST
USER'S MANUAL

Darren R. Hardy and Michael F. Schwartz

CU-CS-743-94



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

HARVEST USER'S MANUAL

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

Version 1.0
October 1994
Last Revised on November 3, 1994

University of Colorado at Boulder

Technical Report CU-CS-743-94
Department of Computer Science
Campus Box 430
University of Colorado
Boulder, Colorado 80309

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 R. Hardy (U. Colorado) is a Professional Research Assistant on the project.

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), and two equipment grants from Sun Microsystems' Collaborative Research Program. 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.

Copyright

Contents

1 Introduction to Harvest 1

2 Subsystem Overview 2

3 Getting and Installing the Harvest Software 4

4 Making Basic Use of Harvest 5
- 4.1 Running a Gatherer . . . 5
- 4.2 Running a Broker . . . 8
- 4.3 Querying a Broker . . . 9
- 4.4 Running a Cache . . . 12

5 Advanced Features of Harvest 14
- 5.1 Customizing a Gatherer . . . 14
  - 5.1.1 Customizing the type recognition step . . . 15
  - 5.1.2 Customizing the candidate selection step . . . 15
  - 5.1.3 Customizing the presentation unnesting step . . . 15
  - 5.1.4 Customizing the summarizing step . . . 15
  - 5.1.5 Setting variables in the Gatherer configuration file . . . 16
- 5.2 Incorporating manually generated information into a Gatherer . . . 16
- 5.3 Tips on gatherering locally versus remotely . . . 18
- 5.4 Administrating a Broker . . . 18
- 5.5 Tuning Glimpse indexing in the Broker . . . 20
- 5.6 Using different index/search engines with the Broker . . . 20
- 5.7 Running a Cache hierarchy . . . 22
- 5.8 Using the Cache's remote instrumentation interface . . . 23
- 5.9 Running a Replicator . . . 23

6 References 24

A The Summary Object Interchange Format (SOIF) 25
- A.1 Formal description of SOIF . . . 27
- A.2 List of common SOIF attribute names . . . 27
- A.3 Using the SOIF processing software . . . 28

B Essence Summarizer Actions 30

C Gatherer Examples 31
- C.1 Example 1 - A simple Gatherer . . . 31
- C.2 Example 2 - Incorporating manually generated information . . . 32
- C.3 Example 3 - Customizing type recognition and candidate selection . . . 34
- C.4 Example 4 - Customizing type recognition and summarizing . . . 34

D The Broker's Query Manager and Collector Interfaces 37
- D.1 Query Manager interface description . . . 37
- D.2 Collector interface description . . . 38
- D.3 World Wide Web interface to the Broker . . . 38

Index 40

1 Introduction to Harvest

HARVEST is an information discovery and access system [4]. It addresses three critical problems to help users reap the growing collection of information accessible via the World Wide Web [2]. First, it provides an efficient and flexible means of indexing widely distributed information, to support resource discovery. Second, it provides network-adaptive means of caching and replicating heavily accessed information, to prevent bottlenecks. Third, it provides support for accessing and manipulating complex data.

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 6,600, network traffic by a factor of 59, and index space requirements by a factor of 43 when building indexes, compared with previous systems, such as Archie, WAIS, and the World Wide Web Worm[1] [3]. 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 (e.g., all documents with a certain regular expression in the title field).

What this manual covers

This manual details how to install and use the Harvest software. Section 2 gives a brief overview of the Harvest system components. Section 3 describes how to obtain and install the software. Section 4 describes how to make basic use of Harvest for building structured content indexes of Internet information. Section 5 describes how to use more advanced features of Harvest, such as customizing the indexing process. The Appendices provide detailed specifications and examples.

This manual is available online in HTML[2] or PostScript[3]. Other information about Harvest (including a discussion of the motivation, system architecture, demonstrations, and papers about the system) is available from the Harvest Home Page at *http://harvest.cs.colorado.edu/*.

[1] `http://www.cs.colorado.edu/home/mcbryan/WWWW.html`
[2] `http://harvest.cs.colorado.edu/harvest/user-manual/`
[3] `http://harvest.cs.colorado.edu/harvest/user-manual.ps`

2 Subsystem Overview

As illustrated in Figure 1, Harvest consists of a *Gatherer*, *Broker*, *Replicator*, and an object *Cache*. The *Gatherer* 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* retrieves this indexing information from the Gatherers and provides a query interface to the body of gathered information. The *Replicator* can efficiently replicate Brokers around the Internet. Finally, users can efficiently retrieve located information through the *Cache*.

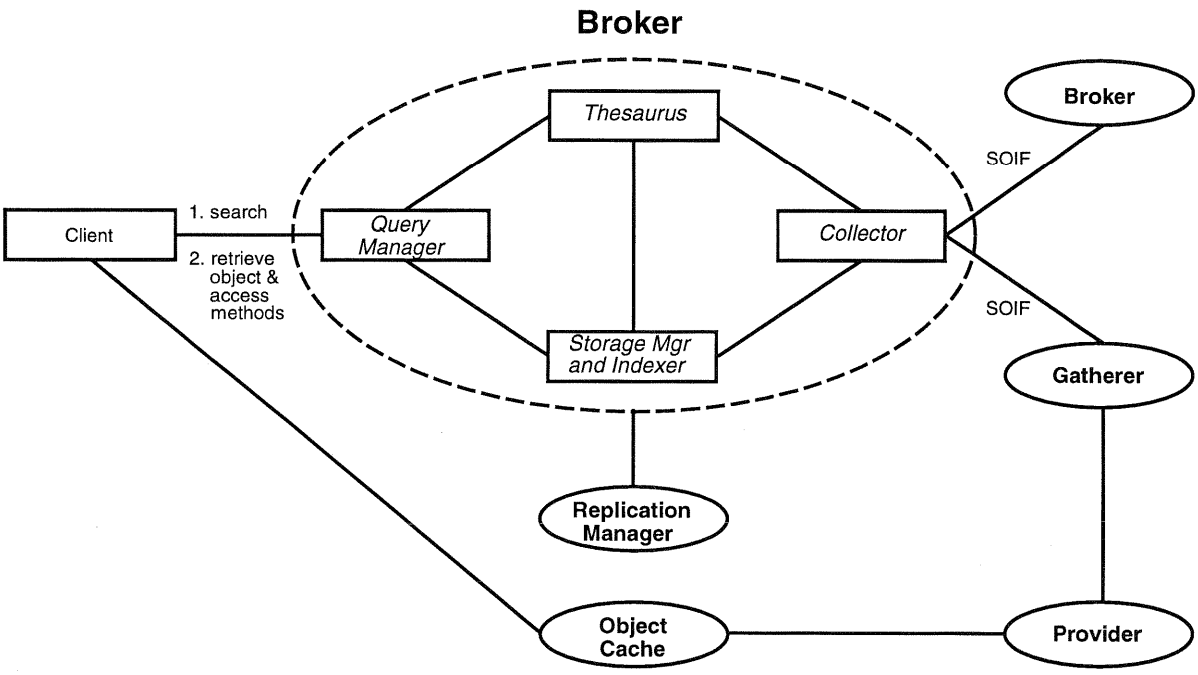


Figure 1: Harvest Software Components

The Harvest software distribution contains a large amount of functionality, in approximately 100,000 lines of code. You don't need to install all of the software to have a useful system. Three common configurations are:

Running a Gatherer plus a Broker will provide a Mosaic accessible, content-based, structured index of a set of information resources.

Running a Gatherer alone will export content-based indexing information about resources from which you gather. Other sites can then build indexes of those resources at much lower server and network costs, compared with remote sites building indexes by retrieving each resource through conventional protocols like FTP, Gopher, or HTTP.

Running a Broker alone will provide a customized index of information gathered by other sites. Other Harvest servers that provide indexing information can be found by searching the Harvest Server Registry[4].

We recommend that you start by running a Gatherer plus a Broker. If your Broker becomes so popular that it creates bottlenecks, you can run a Replicator. You may also want to run an object cache (see Section 4.4), to reduce network traffic for popular data.

[4] `http://harvest.cs.colorado.edu/brokers/hsr/`

Combining Gatherers and Brokers

As illustrated in Figure 2, Harvest Gatherers and Brokers can be combined in various ways. Running a Gatherer remotely allows Harvest to interoperate (via conventional protocols like FTP, Gopher, or HTTP) with Provider sites that are not running Harvest Gatherers. 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.

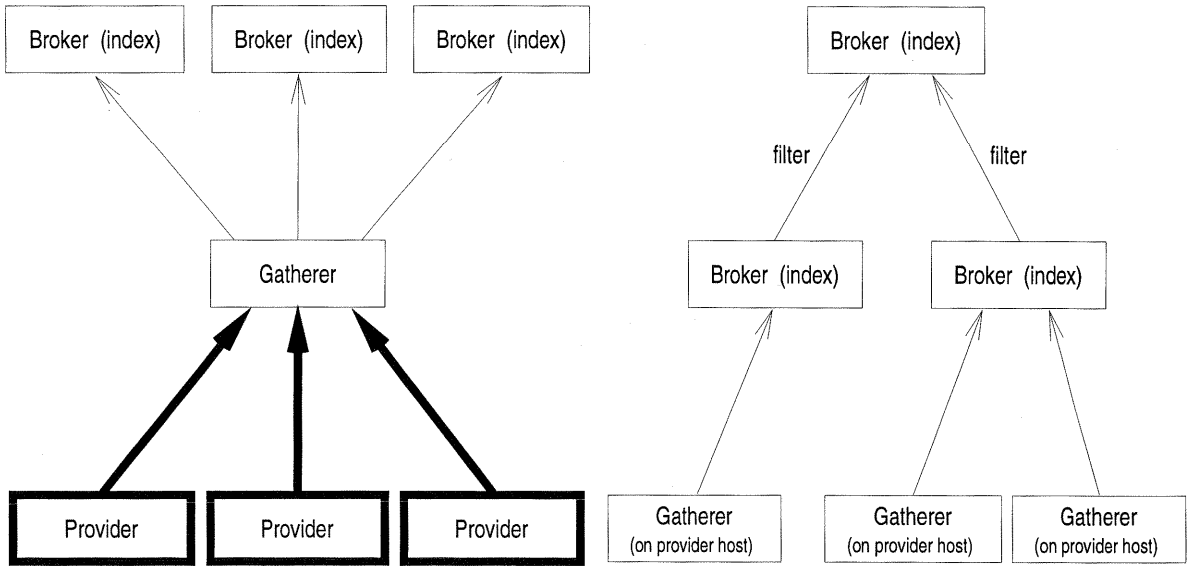


Figure 2: Harvest Configuration Options

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.

For examples of Harvest Gatherers and Brokers, you are welcome to browse the demonstration Brokers[5] and Gatherers[6] at the University of Colorado.

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

3 Getting and Installing the Harvest Software

You can get the Harvest software from our FTP server[7].

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* [8] Do *not* use a common directory like */usr/local* as the prefix, because Harvest installs about 75 (small) programs. Type `make reconfigure` after you've changed the prefix variable, to propagate your change throughout the distribution. If desired, you may edit *src/common/include/config.h* before compiling to change various Harvest compile-time limits and variables.

To build and install the entire Harvest system, type:

```
% make
% make install
```

You may see some compiler warning messages, which you can ignore. Building the entire Harvest system will take about 30 minutes on a DEC Alpha or on newer Sun SPARCstations, and almost an hour on older machines. Including documentation and the compiled source tree, the Harvest distribution takes approximately 25 megabytes of space.

If you have any problems or comments, please email *harvest-dvl@cs.colorado.edu.*

Installing individual Harvest components

To build and install 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 and install the Harvest Gatherer, type:

```
% cd src
% vi Makefile                    (if you want to change the prefix)
% make reconfigure               (if you changed the prefix)
% make gatherer
% make install-gatherer
```

Supported platforms

Harvest was developed using version 2.5.8 of the GNU C compiler on DEC Alphas running OSF/1 v2.0, and on Sun SPARCs running SunOS 4.1. Harvest has also been ported to Solaris 2.3. At present we are concentrating our efforts on supporting OSF/1, SunOS 4.1, and Solaris 2.3. We may eventually support other operating systems (like HP-UX, Ultrix, etc.), but have no immediate plans to do so. If you port Harvest to a new system, please notify us via email at *harvest-dvl@cs.colorado.edu.*

Harvest is written in both C and Perl, so you need to have Perl 4.0 installed. Harvest also uses the GNU compression software *gzip.* The source for the GNU C compiler, *gzip*, and Perl are available from GNU software servers[9].

[7] `ftp://ftp.cs.colorado.edu/pub/distribs/harvest/`

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

[9] `ftp://ftp.gnu.ai.mit.edu/pub/gnu/`

4 Making Basic Use of Harvest

The following subsections introduce the basics of using the Harvest Gatherer, Broker, and Cache. At this point we assume you have already installed the needed software (see Section 3).

Once you have successfully built a Harvest Gatherer, Broker, or Cache, please register your server with the Harvest Server Registry using our registration page[10].

4.1 Running a Gatherer

The Gatherer retrieves information resources using a variety of standard access methods (FTP, Gopher, HTTP, 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 structured indexing information that the Gatherer collects is represented as a list of attribute-value pairs using the Summary Object Interchange Format (see Appendix A). Several example Gatherers are provided with the Harvest software distribution (see Appendix C).

To run a basic Gatherer, you need only list the Uniform Resource Locators (URLs) [1] 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 5.1.5 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://rd.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 would be an HTML document that contains embedded links to other URLs. The Gatherer recursively extracts only the links that point to data that resides on the same server as the original RootNode URL, to prevent accidentally crossing server boundaries. If you want to gather from multiple servers, you must explicitly list them in the configuration file.

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, the Gatherer will only expand a RootNode into a *limited* number of LeafNodes. You

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

can set this limit with the variable *MAX_ENUM* in *src/common/include/config.h.* In a future release of Harvest we will make it possible to set this limit more flexibly.

Note: Harvest does not typically operate as a "robot", 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[11] compliant, and sends the *User-Agent* and *From* request fields to HTTP servers for accountability. Nonetheless, in a future release of Harvest we will integrate some of Koster's suggested guidelines[12] into the Gatherer.

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 your-configuration-file.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[13] the indexing information automatically to Brokers and other clients. To view the exported indexing information, you can use the `gather` client program, as shown below:

```
% gather localhost 8000 | more
```

Controlling access to the Gatherer's database

You can use the *gatherd.cf* file 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
```

Periodic gathering

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 periodically update the data (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`, you should create a script like the following:

```
#!/bin/csh -f
#
#  RunGatherer - Runs the ATT 800 Gatherer (from cron)
#
set GathererAdmin = hardy@cs.colorado.edu
set path = (/rd/bruno/gatherers/att800/bin /usr/local/harvest/bin $path)

cd /rd/bruno/gatherers/att800
Gatherer att800.cf

ls -l . data | Mail -s "ATT 800 finished at `date`" $GathererAdmin
```

[11] `http://info.cern.ch/hypertext/WWW/Protocols/HTTP/HTTP2.html`

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

[13] The Gatherer leaves the `gatherd` daemon running in the background to export the database.

You probably also should create and add something like the following script to your */etc/rc.local* file, so the Gatherer is started after the machine reboots:

```
#!/bin/csh -f
#
# RunGatherd - starts up the gatherd process (from /etc/rc.local)
#
set path = (/usr/local/harvest/bin $path)
gatherd -dir /rd/bruno/gatherers/att800/data 8001
```

The Gatherer maintains a local disk cache of files it gathered to reduce network traffic from periodic gathering or from restarting aborted gathering attempts. [14] 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.

Note that, when used in conjunction with `cron`, the Gatherer provides a more powerful data "mirroring" facility than the often-used `mirror`[15] 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.

[14] The Gatherer uses its own cache rather than the Harvest Cache (see Section 5.7) because gathering isn't likely to exhibit much locality, and would hence affect the Cache's performance.

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

4.2 Running a Broker

The Broker retrieves indexing information from Gatherers or other Brokers, incrementally indexes the collected information, and provides a query interface to it. We have developed a WWW interface to the Broker's query interface so that many Internet users can easily use Harvest Brokers.

The Broker (as suggested in Figure 1) 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 [12] (both freeWAIS[16] and WAIS, Inc.'s commercial version[17]) and Nebula [5] will also work with the Broker (see Section 5.6). The Broker supports a flexible administrative interface (see Section 5.4).

To run a Broker that allows users to query it via a WWW interface (see Appendix D.3 for detailed information about how the WWW interface is implemented), you will need to install and configure an HTTP server. We use the NCSA HTTP software, which you can obtain from their FTP server[18]. Since installing an HTTP server can sometimes be difficult, you might also want to look at the list of *Frequently Asked Questions*[19] on the subject. Once you have an HTTP server configured and you have installed the Harvest Broker software (see Section 3), you need to install the programs for the WWW interface to the Broker (which sends user queries to a Broker, generates a result set in SOIF format, and formats the results for display using HTML viewers; an example formatted SOIF object can be viewed here[20]). Change into the *harvest/src/broker/misc* directory, which contains the source code to the WWW interface to the Broker. Edit the *Makefile* and change the *HTTPD_ROOT* variable (if needed) to the location of where you have installed the HTTP server [21]. Finally, type `make httpd-install` to compile and install the programs for the WWW interface, as well as to create a sample Broker with which you can experiment. At this point, the Broker software and the WWW interface are installed.

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. Finally, to start your Broker, use the `RunBroker` program that `CreateBroker` generates. There are a number of ways you can customize or tune the Broker, discussed in Sections 5.5 and 5.6.

Note: if you get compilation errors while compiling the *lex.yy.c* (generated by `flex`) or *y.tab.c* (generated by `bison`) files in the Broker, verify that you have GNU flex version 2.4.7 and GNU bison version 1.22. You can get the bison[22] code and the flex[23] code by FTP.

[16] `ftp://ftp.cnidr.org/pub/NIDR.tools/freewais/`

[17] `http://www.wais.com/`

[18] `ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/current/`

[19] `http://sunsite.unc.edu/boutell/faq/www_faq.html#provide`

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

[21] Typically, the Broker and httpd run on the same machine. You can run Broker on a different machine than httpd, 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 httpd. You can NFS mount those files or manually copy them over. You'll also need to change the Broker's *query.html* to point to the host that is running the Broker.

[22] `ftp://ftp.gnu.ai.mit.edu/pub/gnu/bison-1.22.tar.gz`

[23] `ftp://ftp.gnu.ai.mit.edu/pub/gnu/flex-2.4.7.tar.gz`

4.3 Querying a Broker

The Harvest Broker can handle many types of queries. 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[24], which supports:

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

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

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

A structured query has the form:

```
tag-name : value
```

where *tag-name* is a Content Summary attribute name, and *value* is the search value within the attribute. If you click on a Content Summary, you will see what attributes are available for a particular Broker. A list of common attributes is shown in Appendix A.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.*"
```

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

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.*

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 senstive. 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[25].)

- `^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."

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

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

4.4 Running a Cache

The Harvest object Cache allows users to retrieve objects quickly and efficiently, often avoiding the need to cross the Internet. The Cache consists of the main server daemon `cached`, some server programs for accessing FTP data, and some optional management and client tools. The FTP programs arose because of the complexity of FTP – while we access Gopher and HTTP using C code built in to `cached`, we access FTP using an external program (`ftpget.pl`), which uses three Perl library files (discussed below).

The Cache uses Proxy access. This is a standard mechanism understood by several available clients (e.g., NCSA Mosaic, netscape, and Lynx) that provides a means of redirecting access for FTP, Gopher, HTTP, etc. through an intermediary process. Often this is used to redirect access through a filtering program on a firewall, providing access control to internal network resources. Proxy access is transparent to the user.

To run the Cache server `cached`:

1. Install `cached` and `ftpget.pl` in a directory located in your path (e.g., */usr/local/bin*). Install the files *chat2.pl*, *ftp.pl*, and *socket.ph* into a library directory, and set the pointer to them in `ftpget.pl`[26].

2. Modify *harvest/src/cache/server/cached.conf* for your site, and then install it as */etc/cached.conf* on your system (or specify the actual location in the `cached` command line, as shown below). At a minimum, you should edit the settings of *cache_dir*, *cache_log*, *cache_access_log*, and *cache_mgr*. 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 *cache_host* variables (see Section 5.7.). Note that `cached` uses three port numbers for the ASCII, UDP, and TCP protocol interfaces. Example values are shown in the provided *cached.conf* file.

3. Run `cached` (or use the `RunCache` program included in the Harvest distribution)[27]. `cached` does *not* need to be run as root. The command line arguments for `cached` are:

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

-e	Print debug message to stderr.
-h	Print help message.
-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.

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.

[26] Note that these paths are set up automatically when you install from the entire Harvest distribution (using GNU autoconf), but you need to set things up manually if you choose to install a pre-compiled binary version of `cached`.

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

During the development of the Cache, we discovered a bug in NCSA Mosaic version 2.4. The standard version of NCSA Mosaic will work correctly for the majority of normal use; however, our patch[28] to Mosaic is required if you want Gopher menus to work correctly. This patch also contains an improved *Reload* button, with which you may force re-retrieval of objects through the cache (i.e., for stale data).

Note: if you find the cache doesn't work with FTP URLs, it may be that 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 your *cached.conf* file). You can verify that `ftpget.pl` works by running:

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

[28] `http://excalibur.usc.edu/dist/`

5 Advanced Features of Harvest

5.1 Customizing a Gatherer

The Harvest Gatherer's actions are defined by a set of configuration and utility ("lib/") files, and a corresponding set of executable ("bin/") programs pointed to by some of the configuration files. In the basic use of Harvest described in Section 4.1, the location of the *lib* directory defaults to the *$(prefix)/lib* directory specified in the top-level Makefile (*/usr/local/harvest* by default), and the *bin* programs are found in the user's PATH.

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 *$(prefix)/lib/*.cf* and *$(prefix)/lib/magic* into your *lib* directory. The details about what each of these files does are described in the subsections 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 5.1.5):

```
bin/    data/    lib/    log.errors    tmp/    log.gatherer

bin:
RunGatherd      RunGatherer

data:
All-Templates.gz        PRODUCTION.gdbm         gatherd.cf
INDEX.gdbm              WORKING.gdbm            gatherd.log

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

tmp:
cache-liburl/
```

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 files in the above *data* directory other than *gatherd.cf* (which you can create to support access control, as described in Section 4.1) are created automatically when you run the Gatherer, as are the files and directories in the *tmp* directory. The files in the above *bin* directory are the boot and cron and scripts, described in Section 4.1. Other files will need to be created in this *bin* directory when you customize the Gatherer, as described in the subsections below. The files in the *lib* directory shown above include the file *GathName.cf*, which you should name according to the function of your Gatherer (e.g., "ColoradoWWW.cf" for gathering data from all the WWW files in Colorado). The other files in the above *lib* directory are described briefly in the following table:

File or Directory	Description
GathName.cf	Specifies URLs and Gatherer configuration variables
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

The Gatherer uses a subsystem called Essence [10, 11] to support customized information extraction. Essence allows the Gatherer to collect indexing information easily from a large amount of resources. In

a nutshell, Essence can determine the type of data pointed to by a URL (e.g., PostScript vs. HTML) [29], "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 to the data to generate a content summary. Users can customize each of these aspects, although a basic set of defaults covers many cases reasonably well. The default summarizing actions are listed in Appendix B. We discuss each of the customizable steps in the subsections below.

5.1.1 Customizing the type recognition step

Essence recognizes types in three ways: by file naming heuristics, by URL naming heuristics, and by locating identifying data with 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.

5.1.2 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 5.1.1) 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.

5.1.3 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.

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.

5.1.4 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

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

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 A.3 for information on how to use the SOIF library to write a summarizer). 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). Look in *harvest/src/gatherer/summarize* for many examples of UNIX summarizers.

5.1.5 Setting variables in the Gatherer configuration file

The Gatherer configuration 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 4.1 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
Top-Directory	Top-level directory for the Gatherer.
Data-Directory	Directory where GDBM database is written.
Errorlog-File	File for logging errors.
Essence-Options	Any extra options to pass to Essence.
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-Port	Port number for gatherd.
Gatherer-Version	Version string for the Gatherer.
Log-File	File for logging progress.
Lib-Directory	Directory where configuration files live.
Working-Directory	Directory for tmp files and *local disk cache.*

Essence-Options is particularly useful, as it lets you customize basic aspects of the Gatherer easily; for example, by specifying `--full-text` you can cause the Gatherer to do full text summarizing rather than the more terse selected text default.

5.2 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 5.1.4). 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, some simple database management tools are in provided in *harvest/src/gatherer/db/gdbmutil*.

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

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

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

...
```

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

Next, run the `mktemplate` and `template2db` programs to generate SOIF and then GDBM versions of these data:

```
% mktemplate annotations [annotations2 ...]
% template2db annotations.gdbm annotations [annotations2 ...]
```

(You can have several files containing the annotations, and generate a singe GDBM database from the above commands.)

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
}
```

5.3 Tips on gatherering locally versus remotely

Our measurements indicate that gathering data via the local file system causes 6.99 times less CPU load than gathering via FTP, primarily because of all the UNIX forking required to gather information via FTP. Similar reductions are possible for gathering locally vs. via Gopher or HTTP.

For large collections (e.g., archive sites containing many thousands of files), it makes sense to try to gather via the local file system. Unfortunately, at this time there is no perfect solution. If you use "file://" URLs you can gather via the local file system, but remote users will not be able to retrieve the documents remotely via Mosaic by clicking on the document URL in the Broker's search results.

In time we will improve the gatherer to allow "ftp://" and other URLs that gather via the local file system when that optimization is possible. In the mean time, if CPU load is more important than allowing remote Mosaic access via the Broker search result links, you can specify "file://' URLs in your Gatherer configuration file so that the Gatherer goes through the local file system. Then, manually translate the URLs on the first line of each template in the Gatherer's database from the file URLs to the proper URL (i.e., FTP). This is a little tricky, but if you want to save the server load by going through the local file system to retrieve the data then it might be worth the effort. You can pipeline `gdbmprint`, `template2db`, and your own sed or Perl script to make the translation.

5.4 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 Appendix D 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 Appendix D.2 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 done through 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:

Set Variable: will set the value of any variable listed in the *broker.conf* file. It takes two parameters: the name of a configuration variable and the new value for the variable.

Add log entry: starts the logging of information about particular types of events. The one parameter is the name of an event type. Currently, event types are as follows:

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.

Add object list: adds multiple objects to the Broker. The parameter is a list of file names (relative to the Broker) that should be added to the Broker.

Close log file: flushes all accumulated log information and close the current log file. No parameters.

Flush log to disk: will flush all accumulated log information to the current log file. No parameters.

Index changes: will cause the Broker to perform an incremental indexing, to index objects that have been added to the Broker recently. No parameters.

Index corpus: reindexes the entire object database. No parameters.

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

Remove expired objects: starts a garbage collection of the registry to remove any object with an expired Time-to-Live. No parameters.

Remove log entry: stops the logging of information about particular types of events. See the *Add log entry* command for a list of events.

Remove object by query: removes all objects that match a particular query. The parameter is a Broker query without any flags, similar to the queries used for bulk transfer.

Remove object list: removes several objects from the Broker. The parameter is a list of SOIF object file names.

Restart server: forces the Broker to reread the registry and the configuration files. This does not actually kill the Broker process. No parameters.

Shutdown server: cleanly stops the Broker. No parameters.

Start collection: begins the Broker's collections, using information in its Collection.conf file. No parameters.

5.5 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.

Since search speed is usually more important than index space in popular Internet servers, the default for Harvest is to use medium-size Glimpse indexes. This is specified by using the *-b* option to `glimpseindex` in the source file *harvest/src/broker/Glimpse/index.c*. This tells `glimpseindex` to generate an index that points to the exact locations of all occurrences of all words. If you want to experiment with the time/space tradeoff, you can try the *-o* option for smaller size. This tells `glimpseindex` to generate an index that points to the files where each word occurs, but not the location within the file; at search time individual files are searched directly. If you specify neither the *-b* nor *-o* option, `glimpseindex` will generate an index that for each word points to a block of files where that word occurs. This provides the most space-efficient (because the pointers are smaller and words that appear many times are stored only once) and slowest search time arrangement.

Another useful option is *-n*, which tells `glimpseindex` to index numbers as well as text. This is useful when searching for dates or other identifying numbers, but it may make the index very large if there are many numbers. The default is not to index numbers.

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).

In a future version of Harvest we will make it possible to set the `glimpseindex` time/space tradeoff and number indexing option from a configuration file, rather than requiring you to modify the source code. Also, in the future we will provide support for `glimpseserver`, which allows indexes to be read into a process and kept in memory, avoiding the added cost of reading the index and starting a large process for each search.

`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[30].

5.6 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 backend, or a Latent Semantic Indexing [9] backend.

At present we distribute code to support an interface to both the public and the commercial WAIS index/search engines, Glimpse, and Nebula [5] [31].

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

[31] 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 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. In time we will make this latter part of the manual more complete.

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

To integrate WAIS with the Broker, follow the instructions at the top of *harvest/src/broker/-Makefile*, and build the new Broker. `CreateBroker` will ask you 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.

Integrating a new index/search back-end into the Broker

To add a new index/search backend to the Broker, you must define twelve routines for indexing, querying, and administering the Broker/indexer interface. These routines collectively define a standard for object indexing, index consistency maintenance, and querying. Depending on the system you are trying to integrate, some of these functions will likely be null calls, and much of the functionality might reside in a few of the other calls.

The code for this interface is in *harvest/src/broker/index.c* and *harvest/src/broker/index.h.* In the distribution these files are symbolic links to the .c and .h files for either Glimpse or WAIS. If you want to define a new indexing interface, create a new subdirectory and set the symbolic links accordingly. You can start with the skeleton files in *harvest/src/broker/Skeleton/*. If you create the routines for a system we do not currently support and are willing to provide those routines to us (possibly with copyright restrictions), please email *harvest-dvl@cs.colorado.edu.*

We discuss each of the routines below. More details about the Broker design and implementation are available in William Camargo's thesis [6]. The functions that define the *indexing* interface between the Broker and the indexer:

`int IND_Index_Start()`
Prepare for indexing a stream of objects.

`int IND_Index_Flush()`
Finish indexing a stream of objects and flush updates.

`int IND_New_Object(reg_t *entry)`
Index a new object from its registry entry.

`int IND_Destroy_Obj`
Remove an object from the indexer.

`int IND_Index_Full()`
Completely reindex all objects.

`int IND_Index_Incremental()`
do a full incremental update of the index.

This Broker/indexer interface is designed to support both object-at-a-time (incremental) and batch (non-incremental) indexers. An indexing session begins with a call to *IND_Index_Start*, where you can call initialization routines for your indexer. For each update, the *Collector* (a part of the Broker) calls

[32] 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.

IND_New_Object or *IND_Destroy_Object*. A batch indexer should just queue the request, whereas an object-at-a-time indexer can use the call to update the object index. When a stream of updates is finished, the Collector calls *IND_Flush* to process any queued updates. Note that if the Broker fails before an index is flushed, updates may be lost. To overcome any inconsistency in the database, the Broker forces a garbage collection that removes and reindexes all objects. For more details about the Collector interface, see Appendix D.2.

The Broker supports configuration through the *broker.conf* configuration file and the *administrative* interface. For more details about the administrative interface, see Section 5.4. An indexer can be configured through two routines:

`int IND_initialize()`
Initialize interface to indexer; called when the Broker is initialized.

`int IND_config(char *value, char *tag)`
Set the value of indexer specific variables; called when the Broker configuration file is loaded and when a variable is set through the administrative interface.

The most complicated routines for most indexers are the *query processing* routines. Since most indexers have different query languages, the Broker translates a query into an intermediate form, which the Broker/indexer interface then translates into an indexer-specific query. The results of the query are then analyzed and a list of UIDs is returned. The following routines define the query interface to the indexer:

`int IND_do_query(qlist *q, int socket, int type, int time)`
Process a query. Three types of queries may be processed as specified by the type parameter: USER, BULK, and DELETE. For BULK queries used by the Broker-to-Broker interface, the time parameter restricts the objects that can be returned. The socket is passed to object display routines for sending results.

`void IND_Init_Flags()`
Initialize indexer-specific query parser flags.

`void IND_Set_Flags(char *tag, char *value)`
Set indexer-specific query parser flag.

5.7 Running a Cache hierarchy

The Harvest Cache supports a hierarchal arrangement of Caches [3]. 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  tcp_port
#
cache_host bigserver.usc.edu   parent   3128 3130 3129
cache_host littleguy2.usc.edu  neighbor 3128 3130 3129
cache_host littleguy3.usc.edu  neighbor 3128 3130 3129
```

5.8 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 is implemented using Tcl/Tk, so you must have that software installed to use the instrumentation interface. (You might try doing "which tclsh" to locate your local Tcl code, in case it happens to be already installed in your path).

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

5.9 Running a Replicator

The Replicator replicates Broker data around the Internet to help distribute Broker server load, and to increase availability and query performance. The Replicator uses the *mirrord* system as a basis for replicating the Brokers. Mirrord is an Internet file system replication service that replicates (or mirrors) file system hierarchies at any number of locations. A "master copy" mirroring scheme is used with all master copies being replicated into a common pool. This approach allows different sites to maintain different collections of files but the resulting archive is a compilation of all of the sites. Mirrord itself is layered atop a hierarchical, flooding update-based group communication subsystem called *floodd* [7].

To run a Replicator, follow the instructions in *harvest/src/replicator/README.*

6 References

[1] 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.*

[2] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The world-wide web. *Communications of the ACM*, 37(8):76–82, August 1994.

[3] 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. Available from *ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.FullTR.ps.Z.*

[4] 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*, Chicago, Illinois, Oct. 1994. Available from *ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.Conf.ps.Z.*

[5] 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.*

[6] W. G. Camargo. The harvest broker. Master's thesis, Department of Computer Science, Pennsylvania State University, 1994. Available from *ftp://grand.central.org/afs/transarc.com/public/camargo/broker.ps.*

[7] 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.*

[8] P. Deutsch and A. Emtage. *Publishing Information on the Internet with Anonymous FTP.* Bunyip Information Systems Inc., May 1994. IETF IAFA Working Group. Available from *ftp://nri.reston.va.us/internet-drafts/draft-ietf-iiir-publishing-01.txt.*

[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. Submitted for publication. 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] 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.*

[13] L. Lamport. *LaTeX: A Document Prepartion System.* Addison Wesley, Reading, Massachusetts, second edition, 1994.

[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] 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.*

A The Summary Object Interchange Format (SOIF)

Harvest Gatherers and Brokers communicate using a an attribute-value stream protocol called the Summary Object Interchange Format (SOIF), an example of which is available here[33]. 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 4.3.

To see an example of a SOIF summary stream, you can run the gather client program, as discussed in Section 4.1. When you do, you'll see output like this:

```
@DELETE { }
@REFRESH { }
@UPDATE {
@FILE { ftp://ecrc.de/pub/ECRC_tech_reports/reports/ECRC-93-10.ps.Z
Time-to-Live{7}:    9676800
Last-Modification-Time{9}:  774988159
Refresh-Rate{7}:    2419200
Gatherer-Name{50}:  Computer Science Technical Reports - Selected Text
Gatherer-Host{21}:  bruno.cs.colorado.edu
Gatherer-Version{3}:    0.3
Type{10}:   Compressed
Update-Time{9}: 774988159
File-Size{6}:   164373
MD5{32}:    43193942d4d53f5a8e4a7b4bcff7a415
Embed<1>-Nested-Filename{13}:   ECRC-93-10.ps
Embed<1>-Type{10}:  PostScript
Embed<1>-File-Size{6}:  428233
Embed<1>-MD5{32}:   84c123582c3d0754a39a78a7e2fb6d23
Embed<1>-Keywords{105}:
technical report ECRC{93{10
Polymorphic Sorts and Types for
Concurrent Functional Programs
Bent Thomsen

}

@FILE { ftp://cml.rice.edu/pub/reports/9404.ps.Z
Time-to-Live{7}:    9676800
Last-Modification-Time{9}:  772872313
Refresh-Rate{7}:    2419200
Gatherer-Name{50}:  Computer Science Technical Reports - Selected Text
Gatherer-Host{22}:  powell.cs.colorado.edu
Gatherer-Version{3}:    1.0
Type{10}:   Compressed
File-Size{6}:   240015
Update-Time{9}: 772872313
MD5{32}:    1712ce5a973cfbb0508b405d6fef1669
Embed<1>-Nested-Filename{7}:    9404.ps
Embed<1>-Type{10}:  PostScript
Embed<1>-File-Size{6}:  488770
Embed<1>-MD5{32}:   84b748dbdda572a1fb1d9c3f67a2dda9
Embed<1>-Keywords{5135}:    /dsp/local/papers/spletter94/spletter94.dvi
```

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

```
Submitted to: IEEE SP. Letters - May 1994
NONLINEAR WA VELET PROCESSING FOR
ENHANCEMENT OF IMAGES
J.E. Odegard, M. Lang, H. Guo, R.A. Gopinath, C.S. Burrus
Department of Electrical and Computer Engineering,
Rice University, Houston, TX-77251
CML TR94-04
May 1994

NONLINEAR WA VELET PROCESSING FOR ENHANCEMENT OF IMAGES
J.E. Odegard, M. Lang, H. Guo, R.A. Gopinath, C.S. Burrus
Department of Electrical and Computer Engineering,
Rice University, Houston, TX-77251
CML TR94-04
May 1994
Abstract
In this note we apply some recent results on nonlinear wavelet analysis
to image processing. In particular we illustrate how the (soft)
thresholding algorithm due to Donoho [2] can successfully be used to
remove speckle in SAR imagery. Furthermore, we also show that transform
coding artifacts, such as blocking in the JPEG algorithm, can be removed
to achieve a perceptually improved image by postprocessing the
decompressed image.
EDICS: SPL 6.2
Contact Address:
Jan Erik Odegard
Electrical and Computer Engineering - MS 366
Rice University,
Houston, TX-77251-1892
Phone: (713) 527-8101 x3508
FAX: (713) 524-5237
email: odegard@rice.edu

1 Introduction
We consider the problem of noise reduction by nonlinear wavelet processing.
In particular we focus on two applications of the recently developed theory related
to wavelet (soft) thresholding [2]. The model

[...rest deleted...]
```

The "@DELETE", "@REFRESH", and "@UPDATE" commands are part of the Broker's Collector interface (described in Section D.2), which provides an additional command level on top of SOIF. Currently, only the @UPDATE section is implemented. Within the @UPDATE section you can see individual SOIF objects, each of which contains a type, a Uniform Resource Locator (URL) [1], and a list of byte-count delimited field name – field value pairs. Because the fields are byte-count delimited, they can contain arbitrary binary data. Note also that SOIF allows *Embed* fields, corresponding to layers of unnesting when summarizing objects (unnesting from a Compressed PostScript to PostScript file above).

SOIF is based on a combination of the Internet Anonymous FTP Archives (IAFA) IETF Working Group templates [8] and BibTeX [13]. Unlike IAFA templates, SOIF templates support streams of objects, and attribute values with arbitrary content (spanning multiple lines and containing non-ASCII characters).

In time we will make a specification for SOIF (and all of Harvest) available, which defines a set of mandatory and recommended attributes for Harvest system components. For example, attributes for a Broker describe the server's administrator, location, software version, and the type of objects it contains.

A.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>

A.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.
MD5	MD5 16-byte checksum of the object.
Refresh-Rate	How often the Broker attempts to update the content summary.
Time-to-Live	The time at which the content summary is no longer valid.
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 Appendix B.
Update-Time	The time that Gatherer updated the content summary for the object.
URL-References	Any URL references present within HTML objects.

A.3 Using the SOIF processing software

This section provides programmers with an introduction to using the SOIF library.

The Harvest system comes with a library interface to SOIF processing in *harvest/src/common/-template*. To compile a program using the SOIF library, use the loader options `-ltemplate -lutil`. The SOIF processing library provides an interface for both parsing and printing SOIF objects, as well as modifying SOIF objects. The SOIF objects are represented as linked lists of attribute-value pairs. Functions are also provided to manipulate these linked lists. The *print_template_body( )* function is useful for writing Essence summarizers. The following is a partial list of the functions supplied in the library:

`void init_parse_template_file(FILE *input_file)`
Parses a SOIF template taking input from a file.

`void init_parse_template_string(char *input_string, int size)`
Parses a SOIF template taking input from a memory buffer.

`Template *parse_template()`
Parses a SOIF template and returns a Template structure.

`void finish_parse_template()`
Cleans up after `parse_template()`.

`int is_parse_end_of_input()`
Returns non-zero if the parsing routine has no more data to process.

`Buffer *init_print_template(FILE *output_file)`
Print SOIF template to memory buffer or to a file if *fp* is not NULL. Returns NULL if printing to a file; otherwise returns a pointer to the Buffer where the data is stored.

`void print_template(Template *t)`
Prints a SOIF template into a file or into a buffer. Must call an init_print routine before, and the finish_print routine after.

`void print_template_header(Template *t)`
Prints a SOIF template header into a file or into a buffer.

`void print_template_body(Template *t)`
Prints a SOIF template body into a file or into a buffer.

`void print_template_trailer(Template *t)`
Prints a SOIF template trailer into a file or into a buffer.

`void finish_print_template()`
Clean up after `print_template()`.

`void add_AVList(AVList *list, char *attr, char *value, int vsize)`
Adds an attribute-value pair to the given attribute-value pair list.

`AVPair *extract_AVPair(AVList *list, char *attr)`
Searches for the given attribute in the AVList. Does a case insensitive match on the attributes. Returns NULL on error; otherwise returns the matching AVPair.

Example using Harvest SOIF processing software

The following example reads SOIF objects from stdin, parses them into the Attribute-Value pair list, adds an Attribute-Value pair to the template's list, and finally prints the modified SOIF template to stdout:

```
/*
 *  print-template - Reads in templates from stdin and prints it to stdout.
 *
 *  Darren Hardy, University of Colorado - Boulder, February 1994
 */
#include <stdio.h>
#include <string.h>
#include "util.h"
#include "template.h"

static void add_print_time(t)
Template *t;
{
        char buf[BUFSIZ];

        sprintf(buf, "%d", time(NULL));
        add_AVList(t->list, "Print-Time", buf, strlen(buf));
}

int main(argc, argv)
int argc;
char *argv[];
{
        Template *template;
        Buffer *b;

        init_parse_template_file(stdin);                /* Initialize parse */
        while (template = parse_template()) {           /* Read next Template */
                add_print_time(template);               /* Add new Attribute-Value */
                b = init_print_template(NULL);          /* Initialize print */
                print_template(template);               /* Print Template to Buffer */
                fwrite(b->data, 1, b->length, stdout);  /* Buffer to stdout */
                finish_print_template();                /* Clean up */
                free_template(template);                /* Clean up */
        }
        finish_parse_template();                        /* Clean up */
        exit(0);
}
```

B Essence Summarizer Actions

The following table provides a brief reference for how documents are summarized depending on their type. These actions can be customized, as discussed in Section 5.1. Some summarizers are implemented as UNIX programs while others are expressed as regular expressions; see Appendix C.4 for more information about how to write a summarizer.

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 RawText summarizer on extracted ASCII text
FAQ, README	Extract all words in file
Font	Extract comments
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	Invoke the RawText summarizer on extracted ASCII text. Note: extracts PostScript differently depending on which package generated the PostScript (troff, TEX, etc.), providing much better results than other PostScript extractors.
RawText	Extract first 100 lines plus first sentence of each remaining paragraph
RCS	Extract RCS-supplied summary
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 RawText summarizer on extracted ASCII text
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.

C Gatherer Examples

The following examples install into */usr/local/harvest/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. For example,

```
% set path = (/usr/local/harvest/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 4.1).

To run this example, type:

```
% cd /usr/local/harvest/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 5.1.5). 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://rd.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{625}:
ftp://ftp.cs.colorado.edu/pub/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://rd.cs.colorado.edu/~schwartz/Home.html
http://rd.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://rd.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 30 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://rd.cs.colorado.edu/harvest/*. Then, for each LeafNode given to the Gatherer, it generated a content summary for it as in the above example summary for *http://rd.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 [15] 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 A).

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[34] of our LSM Gatherer and Broker is available.

To run this example, type:

```
% cd /usr/local/harvest/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 5.1.5 for help on writing configuration files). The library directory contains the typing and candidate selection customizations for Essence. In this example, we've only customized the candidate selection step. *lib/stoplist.cf* defines the types that Essence should not index. This example uses an empty *stoplist.cf* file to direct Essence to index all files.

The Gatherer retrieves each of the LeafNode URLs, which are all Linux Software Map files from the Linux FTP archive *tsx-11.mit.edu*. The Gatherer recognizes that a ".lsm" file is *LSM* type because of the naming heuristic present in *lib/byname.cf*. The *LSM* type is a "nested" type as specified in the Essence

[34] `http://harvest.cs.colorado.edu/brokers/lsm/query.html`

source code[35]. Exploder programs (named `TypeName.unnest`) are run on nested types rather than the usual summarizers. The `LSM.unnest` program is the standard exploder program that takes an *LSM* file and generates one or more corresponding SOIF objects. When the Gatherer finishes, it contains one or more corresponding SOIF objects for the software described within each *LSM* file.

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

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

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

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

```
@FILE { ftp://tsx-11.mit.edu/pub/linux/docs/linux-doc-project/man-pages-1.4.lsm
Time-to-Live{7}:        9676800
Last-Modification-Time{9}:      781931042
Refresh-Rate{7}:        2419200
Gatherer-Name{25}:      Example Gatherer Number 2
Gatherer-Host{22}:      powell.cs.colorado.edu
Gatherer-Version{3}:    0.4
Type{3}:        LSM
Update-Time{9}: 781931042
File-Size{3}:   848
MD5{32}:        67377f3ea214ab680892c82906081caf
}

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

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

[35]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.

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[36] 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 5.1). This Gatherer recognizes World Wide Web home pages, and is configured to only collect indexing information from these home pages.

To run this example, type:

```
% cd /usr/local/harvest/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[37] that uses a similar customization to collect structured indexing information from over 11,000 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 5.1. 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.

[36] `http://harvest.cs.colorado.edu/brokers/pcindex/query.html`
[37] `http://harvest.cs.colorado.edu/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 (see Appendix A.3 for further information on how to write a program that can produce SOIF). 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 /usr/local/harvest/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
```

D The Broker's Query Manager and Collector Interfaces

This section details the Broker query interface and how we implemented the WWW interface to the Broker. It also describes the Collector interface.

D.1 Query Manager interface description

All communication with the Broker is parsed according to the same grammar. This includes user queries, administrative commands, and bulk transfers. A bulk transfer occurs when a Brokers sends a collection request to another Broker. Both user queries and administrative commands are sent by users via the WWW interface. All keywords in the interface grammar are marked with a '#' as the first character. The grammar is as follows:

MESSAGE	⟶	USER-MESSAGE | ADMIN-MESSAGE | BULK-MESSAGE
USER-MESSAGE	⟶	#USER FLAGS #END QUERY
BULK-MESSAGE	⟶	#BULK FLAGS #END QUERY
ADMIN-MESSAGE	⟶	#ADMIN ADMIN-PASSWD ADMIN-COMMAND
ADMIN-PASSWD	⟶	#password ID
ADMIN-COMMAND	⟶	#set ID ID | #collection |
		#clean | #full-index |
		#incremental-index | #open-log |
		#close-log | #flush-log |
		#add-log ID | #rem-log ID |
		#add-object ID | #rem-object-path ID |
		#rem-object-query QUERY
		#restart | #shutdown
QUERY	⟶	#allb | EXPRESSION
EXPRESSION	⟶	RELATION-OP | RELATION-OP LOGICAL-OP EXPRESSION
RELATION-OP	⟶	PRIMARY | not PRIMARY
PRIMARY	⟶	CLAUSE | (EXPRESSION)
CLAUSE	⟶	ID | ID SELECT-OP ID
SELECT-OP	⟶	exact | regexp | less-than | greater-than
LOGICAL-OP	⟶	and | or | except
FLAGS	⟶	FLAG FLAGS | FLAG
FLAG	⟶	shortflag | #index ID | #index ID ID

More information about the Collector operations and corresponding Gatherer commands is available in *harvest/src/gatherer/gatherd/NOTES*.

D.2 Collector interface description

The Broker retrieves indexing information from Gatherers or other Brokers through its *Collector* interface. A list of collection points is specified in the *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 following is an example list of collection points, which incrementally collects compressed information from 3 Gatherers, and collects information from 1 Broker using a query filter:

```
#
#  Collection.conf - Harvest Broker Collection Configuration File
#
bruno.cs.colorado.edu 8511 3 --
canopus.cse.psu.edu   8321 3 --
excalibur.usc.edu     8321 3 --
bruno.cs.colorado.edu 8507 6 --Harvest
```

D.3 World Wide Web interface to the Broker

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[38] (CGI), consists of the following:

- HTML files that use Forms[39] 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.

[38] `http://hoohoo.ncsa.uiuc.edu/cgi/overview.html`
[39] `http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-forms/overview.html`

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.

CGI programs

When you install the WWW interface (see Section 4.2), a few programs are installed into your HTTP server's *cgi-bin* directory:

`BrokerQuery`
: 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.

`DisplayObject`
: This program displays the content summaries from the Broker. It takes the SOIF file from the Broker, formats it in HTML, then returns the HTML to the user. This HTML page 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`
: 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 */brokers* directory when you install the broker (see Section 4.2):

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.

Index

`--allowlist`, 15
`--full-text`, 16
`<LeafNodes>`, *see* Gatherer, LeafNodes
`<RootNodes>`, *see* Gatherer, RootNodes

admin.html file, 18, 38
adminhelp.html file, 38
Annotating gathered information, *see* Manually generated information
Archie, 1
Audio summarizing, 29
autoconf, 4
automatic database, 17

Bibliographic summarizing, 29
bin directory, 14
Binary summarizing, 29
Broker, 8–11
  administration, 18–20, 22
    admin.html file, 38
    adminhelp.html file, 38
  BULK update interface, 22
  collector interface, 37
  creating
    `CreateBroker` program, 8
  incremental vs. batch indexing, 21
  indexing interface
    `IND_Destroy_Obj` routine, 21
    `IND_Index_Flush` routine, 21
    `IND_Index_Full` routine, 21
    `IND_Index_Incremental` routine, 21
    `IND_Index_Start` routine, 21
    `IND_Init_Flags` routine, 22
    `IND_New_Object` routine, 21
    `IND_Set_Flags` routine, 22
    `IND_config` routine, 22
    `IND_do_query` routine, 22
    `IND_initialize` routine, 22
  integrating index/search back-end, 21
  query manager interface, 36
  query processing, 22
  querying, 9
    approximate matches, 10
    Boolean combinations of keywords, 9
    case sensitivity, 10
    default query settings, 11
    examples, 9
    HTML interface, 38
    match count limits, 9
    matched lines vs. entire records, 9
    multiple word matching, 10
    options selected by menus or buttons, 10
    partial word matching, 10
    query.html file, 38
    queryhelp.html file, 38
    regular expressions, 9, 10
    result set presentation, 10
    Simple keyword, 10
    structured queries, 9
    whole word matching, 10
  using different index/search engines, 20
  using WAIS as an indexer, 21
  WWW interface, 37
broker.conf file, 18
`BrokerAdmin` program, 38
`BrokerQuery` program, 38
bycontent.cf file, 14
byname.cf file, 14
byurl.cf file, 14

C summarizing, 29
Cache, 12–13
  `cached` program, 12
  `CachedLynx` program, 12
  `CachedMosaic` program, 12
  `ftpget.pl` program, 12, 13
  NCSA Mosaic bug, 12
  proxy interface, 12
  remote instrumentation interface, 22
  `RunCache` program, 12
  running a hierarchy, 22
cache-liburl directory, 14
`cached` program, 23
CHeader summarizing, 29
`cleandb` program, 16
Commercial WAIS, 8
Common Gateway Interface (CGI), 37
Configuration, 4
`CreateBroker` program, 8, 18, 21, 38
`cron` program, 6, 14
Customizing, *see* Gatherer, customizing

data directory, 14
`DisplayObject` program, 38
Dvi summarizing, 29

Essence, 14, 16, 27, 31, 33
  candidate selection, 15
  configuration files
    bycontent.cf file, 14

byname.cf file, 14
byurl.cf file, 14
magic file, 14
quick-sum.cf file, 14
stoplist.cf file, 14
presentation unnesting, 15
summarizer actions, 29
summarizing, 15
type recognition, 15
using programs to summarize, 34
using regular expressions to summarize, 33

FAQ summarizing, 29
`file` program, 15
Font summarizing, 29
freeWAIS, 8
Full text indexing, 16

`gather` program, 6, 24
`gatherd` program, 6
gatherd.cf file, 14
gatherd.log file, 14
Gatherer, 5–7
access control, 6
customizing, 14–16
candidate selection, 15
configuration file, 16
presentation unnesting, 15, 32
summarizing, 15
type recognition, 15
Essence, 14–16
examples, 30–35
customizing type recognition and candidate selection, 33
customizing type recognition and summarizing, 33
incorporating manually generated information, 31
simple gatherer, 30
`Gatherer` program, 6
LeafNodes, 5, 16
periodic gathering, 6
RootNodes, 5, 16
enumeration limits, 5
tools
`cleandb` program, 16
`gdbmprint` program, 18
`mergedb` program, 16
`mkindex` program, 16
`mktemplate` program, 16
`rmbinary` program, 16
`RunGatherd` program, 7
`RunGatherer` program, 6
`template2db` program, 16, 18
`wrapit` program, 16
`Gatherer` program, 6
GathName.cf file, 14
`gdbmprint` program, 18
Glimpse, 8, 9, 20, 21
tuning, 20
`glimpseindex` program, 20
`glimpseserver` program, 20
`gunzip` program, 15

Hierarchical cache, *see* Cache, running a hierarchy
HTTP v1.0 compliant, 6

INDEX.gdbm file, 14
Installation, 4
individual components, 4
supported platforms, 4
Internet Research Task Force, i
IRTF, *see* Internet Research Task Force

lib directory, 14
local gatherering, 18
localization, 4
log.errors file, 14
log.gatherer file, 14

magic file, 14
Mail summarizing, 29
Makefile summarizing, 29
ManPage summarizing, 29
manual database, 17
Manually generated information, 16, 31
MAX_ENUM #define, 5
`mergedb` program, 16
`mirror` program, 7
`mkindex` program, 16
`mktemplate` program, 16

Nebula, 8, 20
News summarizing, 29

Object summarizing, 29

Patch summarizing, 29
PC archive indexer, 33
Perl summarizing, 29
Platforms, 4
PostScript summarizing, 29
prefix Make variable, 4
PRODUCTION.gdbm file, 14, 17

query.html file, 38

queryhelp.html file, 38
quick-sum.cf file, 14

RawText summarizing, 29
RCS summarizing, 29
README summarizing, 29
ReferBibliographic example summarizer, 34
Regular expressions, *see* Broker, querying, regular expressions
remote gatherering, 18
Replicator, 23
`rmbinary` program, 16
Robots, 6
`RunBroker` program, 8
`RunGatherd` program, 7, 14
`RunGatherer` program, 6, 14

ShellScript summarizing, 29
SOIF, *see* Summary Object Interchange Format
soifhelp.html file, 38
SourceDistribution summarizing, 29
Spelling errors, *see* Broker, querying, approximate matches
stoplist.cf file, 14
Summary Object Interchange Format, 5, 7, 8, 15, 16, 19, 24–28, 30, 38
  common attribute names, 26
  formal description, 26
  processing software, 27
    example, 28
    `add_AVList` routine, 27
    `extract_AVPair` routine, 27
    `finish_parse_template` routine, 27
    `finish_print_template` routine, 27
    `init_parse_template_file` routine, 27
    `init_parse_template_string` routine, 27
    `init_print_template` routine, 27
    `is_parse_end_of_input` routine, 27
    `parse_template` routine, 27
    `print_template_body` routine, 27
    `print_template_header` routine, 27
    `print_template_trailer` routine, 27
    `print_template` routine, 27
  `wrapit` program, 16
SymbolicLink summarizing, 29

`template2db` program, 16, 18
Tex summarizing, 29
tmp directory, 14
Troff summarizing, 29

Unrecognized summarizing, 29

WAIS, 1, 8, *see* Broker, using WAIS as an indexer
`waisindex` program, 21
`waisparse` program, 21
`waissearch` program, 21
`waisserver` program, 21
WORKING.gdbm file, 14, 17
World Wide Web Worm, 1
`wrapit` program, 16
WWW Home pages indexer, 33
WWWW, *see* World Wide Web Worm