

Linguist's Search Engine Administrator's Manual

Aaron Elkiss

August 25, 2005

1 Introduction

This document describes the basic architecture of the Linguist's Search Engine and describes the configuration files and Web-based administration tools. It also includes a short troubleshooting section. For information on using the Linguist's Search Engine, see the Getting Started Guide and the User's Guide. For information on how to install the Linguist's Search Engine, see the Installation Guide. For more information on the database, technical functions and overview of the code of the Linguist's Search Engine, see the Developer's Guide.

2 System Overview

This section describes the fundamental objects of the Linguist's Search Engine. For more details about their function and representation, see the Developer's Guide.

2.1 Collections, Documents and Sentences

The basic objects in the Linguist's Search Engine are *collections*, *documents*, and *sentences*.

Collections are groups of documents which can be searched as a unit. Collections have one or more components. Collection components are simply a way to group multiple logically different sets of data into a single collection. For example, a collection might consist of the results of a group of related web searches, and each collection component contain the results of one of those searches.

Documents are represented as a sequence of sentences and can also have arbitrary metadata applied to them. Sentences, within the framework of the Linguist's Search Engine, are an abstract object which can have any kind of data attached to them.

2.2 Annotators and Annotations

The concept of an *annotation* is at the heart of the Linguist's Search Engine. An annotation is simply a piece of data attached to a document or a sentence. It could be a URI or other citation for a document, a sequence of part of speech tags or a parse tree for a sentence, or even an arbitrary piece of HTML to display along with a sentence. Annotation types are configurable via the `annot.ini` (see section 3.1) file and the Web annotation type management tool (see section 6).

An annotation, quite naturally, is produced by an *annotator*. There are two kinds of annotators just as there are two kinds of annotations - *document annotators* and *sentence annotators*. An annotator takes certain inputs - for example the tokenized version of a sentence - and produces output - for example the part of speech tag sequence of a sentence. The type of an annotator is determined by what it produces as output - document annotators produce annotation that applies to the entire document; sentence annotators produce annotation that applies to a single sentence. Sentence annotators can take document as well as sentence input - for example suppose you had the genre of the document and wished to choose a parser model to use based on the genre. You could use the genre as well as the sentence body as input to the parser. Likewise, document annotators can take sentence input as well - in this case the document annotator gets the annotation of all sentences in the document as input.

There is one thing missing - currently there is no provision for an annotator that produces multiple sentence annotation outputs. This would be useful for a wide class of annotations - converting from raw document text to sentences, for example, or models for sentence annotations that have features that depend on document context.

Annotators must be provided as perl modules that contain a function called `do_sentence` (for sentence annotators) or `do_document` (for document annotators). The actual annotator need not be implemented in perl - the LSE is provided with sample interfaces to annotators in C++ and Java as well as pure perl annotators. The simplest way to interface is usually to use the `Expect` perl module, although in some cases external code can be called directly from Perl. See the LSE Developer Guide for the exact format of input and output parameters.

The Perl module for an annotator as well as any other configuration parameters needed for an annotator are specified in `annot.ini` (see section 3.1). Input parameters are specified via the annotation type management Web tool (see section 6), which is a front-end to configuration tables in the database. See the LSE Developer Guide for more information about the database tables for annotation type management.

Once the initial document or sentence information is loaded into the database, the configuration settings determine what new annotations should be produced. Each time an annotation is added, the annotation input configuration is consulted to determine if any new annotations can be produced. If so, the type

of annotation as well as the sentence (or document) is noted in the *todo table*. Annotators then check out sentences or documents to work on from the *todo table*, new annotations are create, and the cycle repeats until the newly-added annotation enables no other annotation types.

Each collection component / annotation type pair has an associated *job ID*. Thus, a job comprises all sentences or documents requiring a particular kind of annotation that belong to a particular collection component. Each job has an associated priority. When an annotator looks for sentences or documents to work on, it takes the job with the highest priority. Priority is calculated based on time since the collection component was created (the longer the time elapsed, the higher the priority) and the number of completed annotations for the job (the more finished, the lower the priority). The priority system tries to ensure that one particularly large collection component does not hog all available resources.

2.3 Queries, Indices and Web Services

The heart of the Web interface for the Linguist's Search Engine is the query page. This page begins by using a SOAP (web service) front-end to a parser (configured via `service.ini` (see section 3.6)) to parse example sentences. When a query is issued, it is dispatched to the *Query Mistress* along with the collection and annotation type (e.g. English parse from Charniak's parser, or Chinese parse from the Stanford parser). The *Query Mistress* looks up the appropriate index for the given collection and annotation type in the database and dispatches the query to a handler for the particular index. Index types can be configured via `query.ini` (see section 3.5), and if no index exists for a particular collection, one can be built on-the-fly using a default handler, also configured in `query.ini`.

3 Configuration Files

There are six configuration files that control the behavior of the Linguist's Search Engine backend. Among other things, the database configuration, annotation types, and machines to run annotators on can all be configured. This section gives an overview of what is configurable in each of the files. For examples and more details about how to configure each particular item, consult the provided sample configuration files.

3.1 Annotation Configuration

`annot.ini` contains configuration information for each sentence and document annotation type as well as settings that apply to all annotation types.

3.1.1 [general] Section

The [general] section contains three parameters that apply to all annotators.

- **batchsize** is the number of sentences an annotator should check out at one time. The way annotators work is to retrieve a number of sentences that require some particular annotation - this parameter controls the size of each batch. A higher number means that an annotator can go longer without contacting the database, but setting the number too high may mean that each database access takes too long and can cause parallelism to suffer, since only one annotator can check out sentences from the database at a time. The reason for this is that the todo table must be locked to ensure that two annotators don't try to check out the same sentences.
- **sleep** controls how long the annotator should pause when it is running in PAUSE mode or after an error.
- **logpath** specifies the complete path to the directory annotators should create logfiles in. The filename for the log is *annotname.hostname.pid* where *annotname* is the name of the annotator, *hostname* is the short hostname of the host the annotator is running on, and *pid* is the UNIX process ID of the annotator.

3.1.2 [job_priority] Section

The [job_priority] section contains settings for the job scheduler. **count_div** specifies the number of sentences that must be annotated for the collection component to lose one priority point; **age_div** specifies the number of hours to have elapsed since the creation of the collection component to gain one priority point.

3.1.3 Individual Annotation Type Sections

Each annotation type section has one required key, **module**, which specifies the perl module containing the **do_sentence** (for sentence annotators) or **do_document** (for document annotators) function. See the developer's guide for more information about writing new annotators.

Other parameters are specific to the particular annotation type. See the configuration file and documentation for each annotation type for more information.

To add a new annotation type, add a section for it in **annot.ini** and configure it via the Web interface (see section 6).

3.2 Offensive Word Filter

`filterwords.txt` contains a list of words, one per line. If the “Filter offensive content” checkbox is selected in the query options, then results with URLs containing any of the words as a substring and sentences containing any of the words as a complete word will display “Filtered URL” and “Filtered Result” instead of the actual result.

3.3 General / Miscellaneous Configuration

`general.ini` contains various general and miscellaneous settings.

3.3.1 [database] Section

The [database] section contains connection parameters for the main LSE database. `dsn` specifies the Perl DBI driver, hostname, database name and other connection parameters. See the documentation for DBI module and the particular database driver. `dbtype` should always be set to `Postgres` as no other database types are currently supported. `dbuser` and `dbpass` are the database username and password, if they are required – if not they can simply be omitted.

3.3.2 [graph_annot] Section

The [graph_annot] section contains a single parameter, `font`, which specifies the complete path to the TrueType font to use when drawing trees on the sentence annotation display page.

3.3.3 [morphology] Section

The [morphology] section contains a single parameter, `tables`, which specifies the complete path to the Berkeley DB databases which are automatically generated from the XTAG morphological database during installation (see the Installation Guide)

3.3.4 [tgrep] Section

The [tgrep] section contains configuration parameters for the `tgrep2` tree search tool. `tgrep` specifies the full path to the `tgrep2` executable and `tgrep_wrap` specifies the full path to the `tgrep2-wrap` wrapper script.

3.3.5 [qbe] Section

The [qbe] section contains various parameters for the tree editor applet. The various `applet_` parameters specify the attributes of the `<applet>` object on the query page; `applet_expand_param` and `applet_wordnet_param` specify the complete URL to the morphological and WordNet expansion perl scripts `expandWord.pl` and `wnTest.pl`.

3.3.6 [wordnet] Section

The [wordnet] section contains one parameter, `path`, which specifies the full path to the BerkeleyDB databases required by `Lingua::WordNet`.

3.4 Preprocessing Configuration

`preprocess.ini` contains settings for preprocessing documents and sentences. It is used by the `dbfilelist` annotator that takes documents downloaded from the Web and adds them to the database as well as the `load-xml.pl` script (see section 4.1), which loads sentences directly into the database from a specially-formatted XML file. It specifies a chain of filters that convert the document from its initial representation into sentences and documents, adds initial annotations such as tokenization, filters out some sentences or documents, and adds the documents and sentences to the database. The [filters] section specifies which of these filters to run; other sections in the file provide additional per-module configuration. There is extensive documentation in the configuration file itself; refer to it to determine how to create new preprocessor configurations.

3.5 Query Configuration

`query.ini` contains settings related to the processing of queries.

3.5.1 [providers] Section

The [providers] section lists each provider that can be used to process queries. A provider is a perl subroutine that accepts a query and various parameters relating to the source the query should be run on, and returns results, warnings and errors in a particular format – see the developer guide for more information. The `collection_indices` table in the database lists the provider for each collection that has been indexed.

The `provider` key names one provider. Each provider must have its own section later on in the file. The `defaultprovider` names the provider to be used by default if a collection has no other indices. The provider named in `defaultprovider` must be one of those listed in the `provider` keys.

3.5.2 [ranker] Section

The [ranker] section lists functions that take a list of results from a provider function and return a ranked list of results. The ranking may be trivial (i.e. results are returned in an arbitrary order). There should be a key (e.g. `charniak`, `zh_parse`) for each annotation type whose results should be ranked.

3.5.3 Individual Provider Sections

Each provider named in the [providers] section must have its own subsection. Each of these sections must have a `command` key, which lists the perl subroutine to run and the parameters to pass to it. Parameters starting with `$` are interpolated using several special variables in addition to the additional keys in the section for the provider.

There are six special variables: `$col_name`, `$col_userid`, `$atype`, `$location`, `$indexid` and `$searchstring`. `$col_name` is the collection name, `$col_userid` is the user who owns the collection, or 'public', `$atype` is the annotation type to search (for example 'charniak' or 'zh_parse'), `$location` is the location of the index (if one was specified in the `collection_indices` table) and `$searchstring` is the search provided by the user. `$indexid` is the unique identifier for the index in the database; it can be used in addition to the location as a way of finding information used for the index. In addition each provider may have additional variables which are interpolated using the values in this configuration file by default, but may be overridden by the user.

In addition, there can be an `indexingcommand` key which specifies the shell (not perl) command to run to build a new index which can be searched using that provider. It should take three command-line arguments - the collection userid, collection name, and the annotation type to index (e.g. `charniak`, `zh_parse`).

3.6 Annotator and SOAP Service Configuration

`service.ini` defines the SOAP (Web-based) services that should be run and the machines they should run on. It also specifies what annotators (defined in `annot.ini`) should be run on which machines.

3.6.1 [general] Section

The [general] section contains settings for multiple services and annotators.

The `uri` key lists the URI prefix for each SOAP service. Since this is a URI, not a URL, the prefix serves to identify rather than locate the service and so is somewhat arbitrary.

The `logpath` key contains the full path to the directory where each instance of a service or annotator should redirect its output. This path must be directly

accessible from any host the service or annotator is running on. Logs are named in the format *servname.pid.hostname* where *servname* is the service or annotator name, *pid* is the UNIX process ID, and *hostname* is the name of the host the service or annotator is running on.

3.6.2 [soap] Section

The [soap] section lists all SOAP services that will run. Each SOAP service must have a **service** entry; the value of the **service** entry is the name of the section in the configuration file containing the rest of the parameters for the SOAP service.

3.6.3 SOAP Service Sections

Each SOAP service listed in the [soap] section must have its own configuration section. Each of these sections has the same available keys.

The **module** and **host** parameters specify what to export and where to run the HOST service. **module** is the perl module whose functions to export via SOAP. **host** lists the hosts to run this SOAP service on (see section 3.6.6).

There are also three parameters for monitoring the SOAP service. They specify the expected output of the service for a given input. **monitor_action** specifies the subroutine in the specified module to run when monitoring the SOAP service **monitor_params** specifies the parameters to the function. This will be `eval()`'ed, so complex structures can be passed. **monitor_result** specifies a snippet of perl code to check that the result from the SOAP service is okay. `$result` will be the return value from **monitor_action**.

3.6.4 [annotators] Section

The [annotators] section lists all annotators that will run. Each annotator must have a **service** entry; the value of the **service** entry is the name of the section in the configuration file with the machines the annotator runs on and the perl subroutine to run. The annotator types are NOT the same as the annotation types listed in `annot.ini` - annotators can produce different annotation types at different times in their life cycle. For example, **sentence_generic** annotators are responsible for all kinds of sentence annotations, **document_generic** annotators are responsible for document annotations, and **wwwdownload** and **dbfilelist** annotators do Web searches and perform the initial processing for documents pulled off the Web.

The **timeout** parameter specifies how long an annotator can go without checking in before it is killed and restarted. Each time an annotator checks in it sees which job it should run and whether or not there is a new command. It also updates a timestamp. When the monitoring process runs it checks the age of

the timestamp. If it is older than the specified interval then the process is killed and a new annotator is started.

3.6.5 Annotators Sections

Each annotator type specified in the `[annotators]` section must have its own configuration section listing the perl module to load (in the `module`) key and subroutine to run (in the `sub` key; this must be the fully-qualified subroutine name) as well as the host or hosts to run the annotator on (see section 3.6.6).

3.6.6 Host Specification and `[hostvar]` section

Both the individual annotator and individual SOAP service sections must list which hosts the annotator or service will run on. The format of the `host` key is *hostname.numinstances*, where *hostname* is the name of the host as defined in the `[hostvars]` section and *numinstances* is the number of instances to start on that host (you might want to run more than one if the host has multiple processors)

The `hostvars` section is somewhat freeform. The only required keys are `perl` and `ssh`, which define the command to run to actually start a service or annotator. In this command the `$host` variable will be replaced with the hostname. If different perl interpreters are running on different machines, different variables can be interpolated into the `perl` key based on the hostname. In the example configuration file, the `perl ${"varname"}` construct is used - first `$host` is interpolated into `perl.$host`, then the actual `perl.hostname` key is interpolated. Each key defines a perl variable and is run through perl's `eval` command. Keys can reference keys appearing before them in the configuration file, but not keys appearing after.

4 Collection Management

There are several tools, both web-based and command-line, for creating and managing collections.

4.1 Collection Creation and Deletion

Collections can be created manually by using the `load-xml.pl` script. The input to this script is an XML file containing the sentences to add. The `load-xml.pl` script also takes an argument, which is the preprocessing configuration file (see section 3.4). The path to the configuration file is relative to the LSE configuration directory.

With the `preprocess-xml.ini` file, the expected XML format is as follows:

```

<?xml version="1.0" encoding="UTF-8"?>

<collection name="some_name_for_collection" userid="collection_owner">
<document uri="someuri">
  <sentence>
    <body>This is the sentence body.</body>
    <tokens>this is the tokenized sentence body .</body>
  </sentence>
</document>
<document uri="anotheruri">
  ...
</document>
  ...
</collection>

```

Other document annotations can appear as attributes on the `<document>` element and other sentence annotations can appear as children of the `<sentence>` element.

This preprocessing configuration does require sentences to be identified ahead of time. However, it should be straightforward to adapt the preprocessing configuration and the `LSE::Sentence::XML` module to handle XML with either raw HTML or raw text. See the preprocessing configuration files and the perldoc for the individual perl modules involved.

For example, if you had the following XML in a file called `sample.xml`:

```

<?xml version="1.0" encoding="UTF-8"?>

<collection name="sample" userid="public">
<document
uri="http://www.nytimes.com/2005/07/04/international/asia/04cnd-afghan.html">
  <sentence>
    <body>U.S. Disputes Afghan Claims That Second Commando Was Located</body>
  </sentence>
  <sentence>
    <body>KABUL, Afghanistan, July 4</body>
  </sentence>
  <sentence>
    <body>An Afghan government official said today that an upswing in Taliban
violence would not derail parliamentary elections, and the United States
acknowledged that a bombing raid in eastern Afghanistan last week had killed
civilians.</body>
  </sentence>
  <sentence>
    <body>Also Monday, an Afghan provincial governor said that a second member
of an American special operations unit had been located, and that Afghan forces

```

```
were attempting to reach him.</body>
  </sentence>
</document>
</collection>
```

you could load it by using the following command:

```
perl -w load-xml.pl preprocess-xml.ini < sample.xml
```

There are a couple of gotchas with this process. First, if a collection already exists, it will not be overwritten to or added to. It would be straightforward to add an option that would make `LSE::Sentence::XML` append to collections that already exist. Also, if collection loading is terminated early either due to user intervention (i.e. pressing ctrl-C or otherwise stopping the process) or because of a problem accessing the database or parsing the XML (due to invalid characters or markup) then the partially-loaded collection will still be in the database. You should delete the collection entirely before attempting to reload the data.

An example of adding to an existing collection:

```
perl -w load-xml.pl preprocess-xml.ini -n < sample_2.xml
```

Administrators can delete public collections and all other collections in the Collection Management administrative tool. Administrators can also delete individual collection components in the Collection Component List tool.

Also note that if the username for a collection is set to `public` then it will be visible to all users of the system.

4.2 Collection Indexing

The `/admin/show_collections_table.pl` script displays all collections and all available indices for collections. You must be logged into the Linguist's Search Engine and in the `admin` group (see section 7) to use this tool. It can be accessed through the Administration Tools link from the LSE front page.

For each existing index, it will display the index type, annotation type, and index location in the `oldindex` column. To delete the index, click `delete`. This may take some time to complete.

To create a new index, select the index type and the annotation type. Not all index types are applicable for all annotation types. The `rist` index type can index any data that is formatted as LISP S-expressions, e.g. Penn Treebank-style parses, such as `charniak` and `zh_parse`. This calls the command listed in `query.ini` (see section 3.5.3). Currently, this command is started in the

background and the location of error output can be difficult to determine, so if the indexing process seems to fail the best solution is to run the indexing command manually.

Also note that if the indexing process fails, the partially-complete index may not show up in the database. To delete the failed index, delete the index files from disk. The location is automatically determined by the indexing command; consult it for more information.

4.3 Collection Export

Collections can be exported in the same XML format as they are by running the `dump-xml.pl` script. See the perldoc for `dump-xml.pl` for more information.

4.4 Access Control

Any user that can access the database can run `load-xml.pl` and `dump-xml.pl`. Access control for PostgreSQL is configured in `pg_hba.conf` - see [the chapter on Client Authentication in the PostgreSQL documentation](#). By allowing users to use `load-xml.pl` and `dump-xml.pl`, they can use the LSE for large-scale automatic annotation projects, for example parsing an entire year's worth of newspaper data or large collections of web pages.

5 Service and Annotator Management

Services and annotators can be managed through the `/admin/show_services_table.pl` and `/admin/show_annotators_table.pl` scripts respectively. You must be logged into the Linguist's Search Engine and in the `admin` group (see section 7) to use these tools. They can be accessed through the Administration Tools link from the LSE front page.

Both services and annotators can be stopped by clicking the "Stop" button. New services and annotators can be started by selecting the type of service or annotator to start, entering the machine to start it on, and clicking "Start". The machine must be configured in `service.ini` (see section 3.6.6).

In addition, you can issue new commands for annotators. You can set the command for individual annotators by selecting the new command in the "New Command" column and clicking "Set New Command" or set the command for all annotators in the same fashion under the main table (this is useful for pausing or stopping all annotators)

The following commands are available:

- **PRIORITY** - for `sentence_generic` and `document_generic` annotators, works on the job with the highest priority

- **PAUSE** - The annotator pauses for 30 seconds and then checks if there is a new command; if not, it pauses again and repeats.
- **SHUTDOWN** - Causes the annotator to stop and remove its entry from the annotators table
- **WWWDOWNLOAD** - Only applicable for the `wwwdownload` annotator - perform Web searches and download their results
- **DBFILELIST** - Only applicable for the `dbfilelist` annotator - process downloaded results of Web searches into sentences and add them to the appropriate collection

6 Annotation Type Management

You can view a summary of all annotation types with the `/admin/show_annot_types.pl` script. You must be logged into the Linguist's Search Engine and in the `admin` group to use this tool.

This tool displays an overview of all annotation types - first all sentence annotation types, then all document annotation types, and then all annotation types mentioned in the `annot.ini` (see section 3.1) configuration file but not configured in the database.

The information for each annotation type shows the description, the sentence and document inputs, and the settings in `annot.ini` for that annotation, if there are any.

To add a new annotation type, first create an entry for it in `annot.ini` and restart the web server. Then at the bottom of the `show_annot_types.pl` script you can choose which type of annotation (document or sentence) the new type is, enter a description, and choose the sentence and document inputs. If you do not choose any sentence or document inputs then annotations of that type will not be generated automatically.

To change the inputs for an existing annotation type, first find the annotation type in the list. If there are inputs currently listed for the annotation type, first click 'Deactivate'. Then choose the new input types and click 'Activate.' Deactivating an annotation type will also prevent it from being generated automatically, and activating will cause it to be generated automatically if the sentence has the required input annotations.

7 User Management

Users can be listed and edited with the `/admin/list.pl` script. You must be logged into the Linguist's Search Engine and in the `admin` group to use this tool.

To sort based on the value of a particular field, click the column heading you wish to sort on.

To edit a user, click on the “Edit” link. You can change the users’s name, email address, and other information, as well as reset the password. To automatically generate a new password for the user and email it to him or her, click the “forgot password” button.

You can change the group a user is in by editing the “Access” field. Currently only `public` and `admin` have a meaning - normal users should be set as `public` and those who should have access to the administration tools should be set as `admin`.

8 Monitoring

There are several cron jobs that run automatically to make sure all components of the Linguist’s Search Engine are running. The `LSE::Monitor` cron job makes sure that all services and annotators are running, and the `LSE::JobControl` cron jobs update priorities for annotation jobs and make sure no to-do items are marked as checked-out without actually being worked on by an annotator. Two other cron jobs delete unused result sets from the result cache and periodically run maintenance on the database.

It is also recommended that you run monitoring software on a computer separate from the main Linguist’s Search Engine installation. A simple monitoring package called `monitor.pl` is included with the Linguist’s Search Engine.

It uses a simple configuration file to choose what services to monitor. The first section defines variables that can be used later on in the file, for example the path and any arguments to netcat:

```
netcat = /usr/bin/nc -w 30 lse.umiacs.umd.edu
```

This variable can then be referenced as `$netcat` later on.

The remainder of the file consists of named sections which each contain a command and the expected output.

For example, it can check that a particular service is running:

```
[lse ssh]
command = 'echo 'SSH-1.99-Monitor' | $netcat -w 1 ssh'
resultbegin = SSH-1.99-OpenSSH_3.7.1p2
```

This just connects to `lse.umiacs.umd.edu` (as defined in the `$netcat` variable) and connects to the `ssh` port, and makes sure that the version string from the SSH server is printed.

You can also run remote commands, assuming you have RSA authentication set up:

```
[lse load]
```

```
command = '/usr/bin/ssh -o \  
UserKnownHostsFile=/nfshomes/lse/.ssh/known_hosts_lse -2 -l lse -i \  
/fs/LSE/lse_project/lse_ssh/id_rsa lse.umiacs.umd.edu perl -w \  
monitor/checkload.pl 3 2 2' \  

```

```
resultbegin = Load under specified threshold
```

Here we have a perl script on `lse.umiacs.umd.edu` called `checkload.pl` that prints “Load under specified threshold” if the load average is not above 3 for the past minute and 2 for each of the past 5 and 15 minutes. `monitor.pl` comes with three such scripts - one to check disk space, one to check load average, and one to check free memory. See each script for details.

If the expected output from a command does not match, an email will be sent to the address specified by the `mailto` key with the actual output. An email is sent each time a service fails, so if the monitoring runs frequently this can generate quite a lot of email.

8.1 Example Crontab

This `crontab` is just an example. The installation process should automatically set up the `crontab` for you, but you may need to correct paths to perl or psq or otherwise tweak it if your cron is not the same cron as the one RedHat uses (Vixie’s cron).

The first `crontab` should be run by the user owning the LSE processes on the machine on which the database server resides. It runs the all the cron jobs except for the external monitoring.

```
PERL5LIB=/fs/LSE/lse_project/lse/lse:/fs/LSE/lse_project/perl  
  
*/15 * * * * /fs/LSE/lse_project/lse/lse/cleancache.sh > /dev/null  
  
* * * * * /usr/local/stow/perl-5.8.5/bin/perl -w \  
-MLSE::Database::JobControl \  
-e "LSE::Database::JobControl::updateJobPriorities();" \  
> /dev/null  
  
0 */2 * * * /usr/local/stow/perl-5.8.5/bin/perl -w \  
-MLSE::Database::JobControl \  
-e "LSE::Database::JobControl::clearInUse();" \  
> /dev/null
```

```
* /5 * * * * /usr/local/stow/perl-5.8.5/bin/perl -w \  
-MLSE::Monitor -e "LSE::Monitor::monitor();" \  
> /dev/null
```

```
30 3 * * * /export/lse/postgresql-8.0.3/bin/psql \  
-d lse -c "vacuum analyze" \  
> /dev/null
```

The second should run on a separate machine; this is the external monitoring process:

```
3,7,9,11,13,17,19,21,23,27,29,31,33,37,39,41,43,47,49,51,53 * * * * \  
/usr/bin/perl /fs/LSE/lse_project/monitoring/monitor.pl \  
/fs/LSE/lse_project/monitoring/lse.cfg > /dev/null
```

9 Troubleshooting

The Linguist's Search Engine is a large and complex system and many things can go wrong. The best diagnostic technique is simply to examine the server logs and look for the first error after things stopped working.

As common issues are discovered, descriptions and solutions will be added to this guide.

9.1 BerkeleyDB

One frequent problem after installing the Linguist's Search Engine is that various components may be linked against different versions of the BerkeleyDB library. This can result in difficult-to-diagnose server errors. See the Installation Guide for more information on how to diagnose and repair this problem.

9.2 Database Connection Issues

One common problem with new installations is the inability to connect to the LSE database from remote hosts. The appropriate places to make changes are in `postgresql.conf` and `pg_hba.conf`. Make sure in `postgresql.conf` that `postgresql` is listening on all IP addresses and not just `localhost` (127.0.0.1), and make sure in `pg_hba.conf` that the machine you are trying to connect from is in fact allowed to connect. See the PostgreSQL documentation for more information.

9.3 Paused Annotators

Sometimes annotators may remain paused for an extended time after the job control cron job that clears in-use todo items runs, either because the cron job unexpectedly terminates early or for other reasons. The best solution is simply to stop the annotators and let the monitoring cron job restart them.

9.4 Runaway Annotators, Stalled Processes, Etc.

A more serious problem is if large numbers of annotators are spawned. This may happen when some annotator stops responding while holding various database locks. The best solution is to stop the web server, force postgresql to shut down immediately, kill all LSE-related processes on all machines, restart the database, then restart the web server. This can be accomplished with the following commands:

```
apachectl stop
pg_ctl stop -D $PGDATA -m i
# now kill all LSE processes
pg_ctl start -D $PGDATA
apachectl start
```

where `$PGDATA` is the path to the PostgreSQL data directory. Also, make sure you're using the right versions of `apachectl` and `pg_ctl` (i.e. the versions specifically installed for LSE)

This solution is also appropriate if there are other runaway processes or if the web server stops responding.