

A virtual filesystem on steroids: Mount anything, index and search it

Ben Martin

September, 2005

Abstract

This paper describes the libferris virtual filesystem. Coverage includes the major abstractions, how filesystems are indexed, how to search indexes and how to mount parts of libferris as Samba shares.

1 Introduction

The libferris virtual filesystem [2] has always sought to push the boundaries of what a filesystem should do in terms of what can be mounted and what metadata is available for files. Over the past 5 years it has expanded its capabilities from mounting more traditional things such as tar.gz, ssh, digital cameras, IPC primitives to be able to mount various ISAM files including: db4, tdb, edb, eet, gdbm, various relational databases including: odbc, mysql, postgresql, various servers such as: Http, Ftp, LDAP, Evolution, RDF graphs aswell as XML files and Sleepycat's dbXML.

Recently support for indexing filesystem data using any combination of Lucene, ODBC, TSearch2, xapian, LDAP and PostgreSQL has been added with the ability to query these backends for matching files. Matches are naturally presented as a virtual filesystem.

A Samba VFS module was created to enable legacy clients to take advantage of libferris. This allows parts of a libferris system to be exported as Samba shares.

To watch what legacy clients are doing to an in kernel filesystem behind libferris' back a kernel filesystem monitoring daemon (kfsmd) was created. Kfsmd takes advantage of the kernel inotify interface and logs file change metadata to either Berkeley db files or a PostgreSQL database.

When you are installing libferris don't forget to run

```
ferris-first-time-user --setup-defaults
```

followed by `cc/capplets/logging/ferris-capplet-logging` and make sure your debug levels are set to None or Emergency using the "all" slider. The libferris code assumes that the user's `~/ferris` directory has been setup using the above commands.

2 How libferris sees the world

The two primary abstractions in libferris are the Context and the Extended Attribute. A Context can be thought of as a superclass of a file or directory. In libferris there

is less of a distinction between a file and a directory with the ability for a file to behave like a directory if it is treated like one. For example, if you try to read a tar.gz file then libferris will automatically mount that as a filesystem and list the contents of the archive as a virtual filesystem.

The Extended Attribute (EA) interface can be thought of as a similar concept to the Linux Kernel's EA interface. That is, arbitrary Key-Value data is attached to files and directories. This EA concept was extended early on in libferris to allow the value for an attribute to be derived from the content of a file. This means that simple things like width and height of an image file become first class metadata citizens along with a file's size and modification time. The limits on what metadata is available extend far beyond image metadata to include XMP, EXIF, music ID tags, geospatial tags, rpm metadata, SELinux integration, partially ordered emblem categories and arbitrary personal RDF stores of metadata.

The concept of emblems is supported in one form or another by many systems other than libferris. In libferris, emblems allow you to tag your files and directories into logical groups. For example, I can create a `PhD` emblem to tag my thesis and papers with. The emblem system in libferris allows you to place an ordering on the emblems themselves. For example, I can say that on my PC the `Hamburg` emblem relates to a more specialized concept than the `Germany` emblem and as such I can place `Hamburg` as a child emblem of `Germany`. The ordering is a partial order to allow me to also put `LinuxKongress05` as a child of `Hamburg` if I like.

For EA values which are to be stored they can be either stored using the operating system kernel's native disk EA, into a personal RDF store or can effect the underlying file in a manner as to make the EA value seem to be stored. An example of the latter is that writing an image width EA will scale an image file so that when you subsequently read the image width you will get the value you just wrote. This extends naturally to other situations such as when mounting X Window, changing the x or y EA for a Window will move that window.

Allowing EA to be stored in a personal RDF file allows you to add metadata to any libferris object, even those for which you only have read access. For example, you can attach emblems or comments to the linux-kongress website just as you would a normal file.

The metadata interface gives all metadata from file size to digital signature status information equal standing. As such you can sort a directory by any metadata just as easily as you would `ls -Sh` to sort by file size. Sorting on multiple metadata values is also supported in libferris, you can easily sort your files by mimetype, then image width, then modification time with all three pieces of metadata contributing to the final directory ordering.

An interesting EA for all files is the "content" EA which is equivalent to the file's byte contents. Exposing the file itself through the EA interface means that any information about a file can be obtained via the same interface.

The code to mount different filesystem types aswell as the code handling the extraction of interesting metadata is loaded via a plugin system using shared libraries. The metadata extraction occurs through so called EA generators. An EA generator can also rely on metadata provided by other EA generators. This presents a non formal inference engine inside the metadata handling of libferris.

For example, the "human-language" EA uses the "as-text" EA to get at the file's text content. The "as-text" EA generator knows how to extract the text contents of a PDF file or a man page. The human-language EA generator is not interested in how the plain text is obtained but only on working out what language a chunk of plain text is in.

As the extraction of plain text is somewhat more complicated than other EA generators it has a plugin system of its own.

There are three other major plugin types: creation, EA indexers and Fulltext indexers. The creation plugins allow one to create simple files, image files etc as well as EA and fulltext indexes. Also, context sensitive creation is supported. For example, one can create new SQL views given a URL to a mounted relational database.

An EA indexer plugin allows the EA for a collection of files to be placed into an index structure for resolution of EA queries. A Fulltext indexer operates on the “as-text” EA to index the plain text content of files. The Fulltext indexers operate on the words (terms) of the text employing case folding, word stemming and other text indexing techniques. Fulltext indexing techniques are different enough from EA indexing techniques to merit distinct plugin systems.

Some of these index plugins also support a custom query language usually by passing queries to a third party library which is handling the indexing. For example, the xapian fulltext index plugin allows you to pass specific xapian queries directly to the backend.

Resolution of fulltext and EA queries returns the results as a virtual filesystem.

3 A tour of the libferris root:// URL

In the following outputs I will be formatting the output for display in this document. Normally the output of some commands can be wider than that which can be shown in a document such as this. At times I’ll only show partial results where listing an entire directory would consume much space for little gain.

3.1 Virtual filesystems

There are many top level URL schemes which libferris can handle. The `root://` scheme was added to libferris to give a single top level root directory. In the context of the `/` directory of a UNIX machine `root://` is the parent of `/`. You can see the UNIX `/` as `root://file/`. That is, many of the URL schemes live as subdirectories of `root://` in libferris.

There are two major types of filesystem plugin for libferris: top level and overmounting. An overmounting virtual filesystem plugin is one which requires an existing filesystem in order to work. For example, the `tar.gz` mounting filesystem in libferris is an overmounting filesystem because it requires an archive file in order to provide its filesystem. On the other hand, a top level filesystem is one which operates independently of any other file. For example, the `http://` filesystem does not require any file as input to its filesystem.

Many top level filesystems are exposed as subdirectories of `root://`. Overmount filesystem plugins make no sense as a top level scheme, for example, the filesystem that handles mounting an XML file requires an XML file to work. Overmount plugins are implicitly used by libferris when you attempt an operation like reading an XML file as a directory.

Your `root://` filesystem will vary depending on which plugins you have installed. For a specific build of libferris part of the `root://` is shown in Figure 1.

To see all the filesystem plugins that your build of libferris knows about examine the `context://` filesystem. Shown in Figure 2 is the context filesystem with the `branchfs` filesystems compacted to allow the table to fit the page. Note that it

```

$$ fls root://
...
evolution file ftp http ipc ldap
mysql odbc postgresql xwin ...

```

Figure 1: Part of the listing of the root:// filesystem.

```

applications db ferrisdev http postgresql
apps dvd ffilter icons rdf
bookmarks eaq file ipc root
branches eaquery file-clipboard ldap rpm
branchfs-... eet ftp libnativembox schema
branchfs-... emblemquery fulltextboolean mime selectionfactory
branchfs-... etagere fulltextquery myrdf socket
branchfs-... events gdbm mysql ssh
branchfs-... evolution gnomeicons news tdb
camera external gpg odbc xml
context fca gphoto pg xwin

```

Figure 2: context:// filesystem.

makes no sense to list some of these directories as they are for overmount plugins. Overmount plugins are listed under the `context://` filesystem to allow you to see a listing of all the plugins libferris knows about on your system.

Some of the directories in Figure 2 are synonyms for the same filesystem existing only to provide a shorter URL for command line interaction. For example, `apps://` and `applications://` point to the same filesystem.

As we explore these filesystems I'll use the `ferrisls` command which mimics the `coreutils ls(1)` command. Instead of using the `-l` long listing option, we'll be using the `-0` (zero) recommended-`ea` option. This operates in much the same way as `-l` though it will ask the filesystem itself what EA are most interesting for the user to see.

The `apps://` filesystem exposes the executable files which libferris knows about on your system. This is mainly used in the `ego` file manager to execute applications. For the `apps://` URL the `-0` option chooses to list the following EA: `name`, `ferris-exe`, `ferris-scheme`, `ferris-iconname`, `ferris-ignore-selection`, `ferris-handles-urls`, `ferris-opens-many`. An example showing the `apps://` filesystem is in Figure 3.

There are some other filesystems which are mainly exploited through the `ego` file manager at current. For example, the `bookmarks://` filesystem is a directory hierarchy for storing your personal bookmarks.

The `branches://` and `branchfs-...` filesystems allow many filesystems to be attached to a single file. For example, when you are reading a directory `foo` it already has a list of children objects which reside in that directory. You may have many emblems attached to the directory, and you might wish to see those emblem associations as a filesystem so you can use the standard command line tools to examine which emblems are attached to the directory. With the `branchfs-medallions` filesystem you can see the emblems which are attached to the directory as a filesystem. One of the more handy branch filesystems is the `branchfs-parents` which allows you to choose where the `..` directory will take you when there are many par-

```

$$ fls -Oh apps://
  name      ferris-exe  ferris-iconname  handles-urls  opens-many
  XEmacs    emacs        .../xemacs.png  0              0
  feh       feh -FZ      .../feh.png     0              0
  ...

```

Figure 3: apps:// filesystem.

```

$$ fls evolution://localhost
contacts mail
$$ fls -O evolution://localhost/contacts/system/
...
witme-ferris  witme-ferris@lists.sourceforge.net
...

```

Figure 4: Mounting Evolution.

ents. Such multiparent cases arise in normal filesystems through the use of symbolic links.

The `branchfs-attributes` turns the EA attached to a file into a filesystem. By doing this you can use `ferrisls` and all of its sorting and filtering to decide which EA you are interested in viewing. Viewing and verifying digital signatures is done with details in the `branchfs-gpg-signatures` filesystem. This shows you who signed something and when and many details for the signature. If you are just interested in the validity of a digital signature relative to your GPG trust setup, simply view the `has-valid-signature` EA for the file.

The `camera://` filesystem mounts `gphoto2` to allow you to `ferrisls` and `ferriscp` your digital photos from your camera. The synonym for this filesystem `gphoto` also exists.

Indexed Sequential Access Method (ISAM) files such as Berkeley db4, tdb, gdbm and eet are handled via the plugins of the same name in the `context://` filesystem listing.

The `eaq` and `eaquery` filesystems are the almost the same. They allow you to pose queries against your default EA index and see the matching files as a filesystem. The `eaq` filesystem matches the URL `eaquery://filter`. The `fulltextquery` filesystem allows you submit queries for your fulltext index inside a URL to view the result set.

The ordering on your emblems described in Section 2 is available as a filesystem through the `etagere://` filesystem. You can also “mv” your emblems around in this filesystem to create the parent child relations. For example:

```
ferrismv etagere://Hamburg etagere://Germany/Hamburg
```

The `evolution://` filesystem allows you to mount your Evolution mail client. Although not shown in Figure 4 your mail subdirectory will give access to your mail folders as shown by Evolution. Using this filesystem it is no longer necessary to save attachments to temporary files to access them from ferris aware systems. A transcript of a session mounting Evolution is shown in Figure 4.

The `external` filesystem handles filesystems which require execution of commands in order to perform their work. For example the zip archive viewing that

```

$$ ferrisls -lh show-ea=size,name,content \
~/sample-oo-writer.odt/content.xml/ \
office:document-content/office:body/office:text

```

size	name	content
0	office:forms	
18	text:p	Paragraph number 1
0	text:p-1	
116	text:p-2	This is the second paragraph ...
0	text:p-3	
39	text:p-4	And in summary, this is really the end.
0	text:p=5	
0	text:sequence-decls	

Figure 5: Mounting an Open Office document. Note that the URL being listed should all be on one line.

allows libferris to mount Open Office documents uses **external** to ask unzip what the contents of an office document are. Although the external filesystem creates overhead of fork and exec it does allow some things to be mounted where there is no easy shared library to handle the data format. Mounting an Open Office document is shown in Figure 5.

The **fca://** filesystem is the gateway to using Formal Concept Analysis (FCA) [4] on your filesystem indexes. The use of FCA allows searching and navigation to be seamlessly combined allowing the user to switch between either styles of navigation.

Some device like supplements are exposed in **ferrisdev://** such as **ferrisdev://generators/uuid** which will generate a new UUID. This is a small filesystem mainly of internal use at current to allow libferris applications to access some handy features without having to detect how to do things on the native system.

The strings defining how you want your views to be filtered and what results you are seeking in an EA index query are specified in a **ffilter** syntax. Using the **ffilter** filesystem libferris can internally mount valid strings in this syntax as a filesystem. Currently this is done using a PCCTS parser generator. As such other parsable data can also be turned into a filesystem given a suitable grammar file for PCCTS.

There is a persistent file copy and paste setup in libferris. This file operations clipboard uses the **file-clipboard** filesystem. This provides history and rollback operations for file manipulations across libferris clients.

SysV semaphores and shared-memory are mounted under **ipc://**.

The **xml** filesystem is responsible for mounting XML files as a filesystem. It should be noted that the XML file to be mounted can be inside another libferris filesystem. For example, when mounting an Open Office document you are mounting an XML file which itself is inside a zip archive file.

libferris can also go the other way and give you an Xerces-C DOM for any libferris filesystem. This virtual DOM is created on demand so as to not exhaust system resources. Using the virtual DOM you can easily use XSLT to create views of your filesystem or perform other manipulations. See **apps/xml** in your libferris tarball for some examples of using libferris with XSLT and CGI.

Mounting your X Window is done via the **xwin://** filesystem. This gives access to your Window objects aswell as being able to mount Klipper on KDE desktops. For klipper you can easily **ls**, **cat** and **cp** your past clipboard interactions and overwriting the top clipboard element is effectively a clipboard copy. The window

```

$$ fls rpm://
Amusements      Development      Libraries      ...
$$ fls rpm://Amusements
Games Graphics
$$ fls rpm://Amusements/Games
Maelstrom-3.0.6-8                jumpnbump-1.50-1.1.fc3.rf
chromium-0.9.12-25                kdegames-3.4.2-0.fc4.2
...
$$ fls rpm://Amusements/Games/kdegames-3.4.2-0.fc4.2
usr
$$ fls rpm://Amusements/Games/kdegames-3.4.2-0.fc4.2/usr/bin
atlantik      kbounce      klickety ...

```

Figure 6: Exploring your filesystem from your package manager's perspective.

mounting allows you to see where your windows are in terms of x,y offsets as well as other interesting data. If you write to the x,y EA then the X Window will be moved to that offset. This is an example of having writable EA where the value written effects the system in such a way as the next read of that EA will give the value written.

The `rpm://` filesystem presents your filesystem in terms of the packages you have installed. As can be seen from Figure 6 when you dig into an installed RPM package you are presented with a view of your main filesystem which is limited to only the files which belong to the RPM package you are viewing.

Metadata in the form of RDF is handled through the `myrdf://` and `rdf` filesystems. The former allows access to a personal RDF store whereas the latter allows one to overmount an RDF/XML or RDF/DB file as a filesystem. Using the `myrdf` store behind the scenes allows `libferris` to let applications store metadata through the EA interface without concern about if the underlying filesystem supports writing EA or even knows what EA are. For example, with `myrdf` I can attach an emblem to a single tuple in a mounted relational database. In such a case the database will not allow me to attach arbitrary metadata to a tuple let alone update such metadata. This also helps in situations where one is mounting a Samba, NFS or native kernel filesystem which does not support EA.

There are three main plugins to support mounting relational databases. These provide the `mysql`, `odbc`, `pg` and `postgresql` filesystems. The latter two are synonyms for the PostgreSQL plugin.

Mounting databases allows you to explore the database server, its databases and their tables and views. In Figure 7 I am viewing a database created with statistics for a Real Time Strategy game. The final command using the `--xml` option for `ferris` will export each tuple in XML format. Instead of embedding the username and password in the URL `libferris` elects to store this information in configuration files. This is a trade off where the risk of accidentally copy and pasting a URL with embedded user credentials is minimized at the expense of having a central store of available credentials and mappings to where to use each credential. For many common URLs inline authentication information is also supported.

One can also mount an LDAP server using the `ldap://` URL.

The `schema://` filesystem exports the schema that `libferris` uses to provide type information for its EA. The types are based on the XSD XML schema system with extensions to support 32 and 64 bit information and show explicit types for things

```

$$ fls pg://localhost
aom kernel_filesystem_monitor_daemon_postgresql lj
publicidx wordnet21
$$ fls pg://localhost/aom/
aom cda cda_cfi
$$ fls --xml pg://localhost/aom/aom
<ferrisls>
<ferrisls url="pg:///localhost/aom/aom" name="aom" >
  <context unit="Villager" costf="50" costw="0"
    costg="0" costfv="0" pop="1" hackd="8" pierced="6"
    crushd="0" range="12" hp="65" hacka="25" piercea="35"
    crusha="99" speed="3.8" lineos="14" prodtm="14"
    favbnty="1.08" id="1" __diagram0__="" name="1"
    primary-key="id" />
...

```

Figure 7: Mounting a PostgreSQL database as a filesystem.

```

$$ fls schema://xsd/attributes
attributedomain boolean double intlist string unknown
binary decimal float schema stringlist
$$ fls schema://xsd/attributes/decimal/integer
long pixelcount unsigned width32 width64

```

Figure 8: Mounting your libferris schema.

like `time.t` epoch values. Part of the schema filesystem is shown in Figure 8.

The `file://`, `http://`, `ftp://` filesystems I'm sure you're all aware of.

And that concludes our quick overview of some of the libferris virtual filesystems available.

3.2 Extended Attributes

For an overview of some of the EA available from libferris see

<http://witme.sf.net/libferris.web/research/eadescriptions.html>

Many of the EA provided by libferris have obvious names. For example, the `md5` EA will compute the MD5 checksum of a file when it is read. The `size`, `mtime`, `name` and all give the obvious data from `stat(2)`. EA like `mimetype`, `is-audio-object`, `is-image-object` etc provide insight into the type of a file. Emblem associations are shown through the “`emblem:`” namespace. For example if you have the emblem `Hamburg` attached to a file then reading the “`emblem:has-Hamburg`” EA will result in the value `1`.

The code responsible for the EA in libferris lives either in the main libferris library or as a plugin at `plugins/eagenerators`. As mentioned above there is also a plugin system to support the “`as-text`” EA which allows a plugin to nominate which files it can convert into plain text.

The plugin system uses two libraries per EA generator. A factory library to determine which files a plugin is interested in and another library to actually extract and handle the EA for supported files. The factory library is statically linked into

libferris to save startup time. The main library, which will be referred to as the plugin is loaded on demand.

A plugin can nominate if it is going to support new attribute creation or not via the `supportsCreateForContext` method. The `isDynamic` method can be used to tell libferris that it can not cache the results of the plugin because new EA may become available for the file. A plugin can also nominate what priority it should have. This allows many plugins to exist for the same operation and when a higher priority plugin might fail for a given file a lower priority plugin can then be tried. For example, the storage of EA using the kernel EA interface might be a higher priority than the storage of EA in `myrdf`. When a new attribute is to be created the native kernel EA plugin will be thus tried first and if the underlying filesystem is not of the correct type then the RDF plugin will get a chance to store the EA.

4 Index manipulations

An article appeared in the Linux Journal describing indexing with libferris [3]. This section provides similar information, some more terse and some more technical.

An EA or fulltext index lives in a directory as far as libferris is concerned. The directory itself may only exist to store the metadata that a plugin requires to access that EA or fulltext index. For example, the PostgreSQL EA index module will store a small db4 file in its index directory telling the plugin where your PostgreSQL database is and what username and database to connect to etc.

You can have as many EA and fulltext indexes as you like. To access a non default index simply use the `-P` option to the indexing or querying command with the path to your alternate index. The defaults are in `~/.ferris/ea-index` and `~/.ferris/full-text-index`.

In the paper the term `findex` is used to refer to either an EA index or a fulltext index to avoid ambiguity with for example a B-Tree index in a relational database.

4.1 Creating an index

Creating an `findex` is done using either the `fcreate` or `gfcreate` tool from the `ferriscreate` [1] package. The difference between the two clients is that `gfcreate` presents a GTK+2 interface for interactive creation whereas `fcreate` is a command line tool which expects all the information required for the object creation to be supplied as command line arguments.

The below discussion shows the use of the TSearch2 and PostgreSQL EA `findex` modules. These require some database setup work in order to be smoothly usable by non database administrators. Other `findex` modules such as the `xapian` module require no such setup work in order to use.

To allow non database administrators to use TSearch2 you have to setup a template database which already includes the `contrib/tsearch2.sql` PostgreSQL sql file. This is because TSearch2 requires C functions to be added to the SQL environment which is an operation that non administrators can not normally perform. The simplest way to use the PostgreSQL `findex` modules is to have a template PostgreSQL database which includes the: TSearch2, `btree_gist`, `int array` and `plpgsql` abilities and use this template in creating your initial postgres database.

This can be done with the `apps/ferris-setup-template-findex-database.sh` command which is shown in Figure 9. Using this command is the recommended way to setup a template database because new PostgreSQL `contrib` modules may also be used in future versions of libferris.

```
PGCONTRIB=/usr/share/pgsql/contrib

echo create database ferrisftxtemplate | psql
cat ${PGCONTRIB}/tsearch2.sql | psql ferrisftxtemplate
cat ${PGCONTRIB}/dbsize.sql | psql ferrisftxtemplate
cat ${PGCONTRIB}/btree_gist.sql | psql ferrisftxtemplate
cat ${PGCONTRIB}/_int.sql | psql ferrisftxtemplate
createlang -d ferrisftxtemplate plpgsql
```

Figure 9: The easiest way to setup a machine to be able to use the libferris PostgreSQL index plugins is to create a template database using the command `ferris-setup-template-index-database.sh`.

To create a fulltext index using the `gfcreate` tool we first create a directory for the index to live in and then invoke `gfcreate` with the path for the index. You can also empty the `./ferris/full-text-index` directory and use this path if you wish to recreate your default index.

```
$$ mkdir /tmp/my-new-index
$$ gfcreate /tmp/my-new-index
```

The GUI for `gfcreate` shows the major MIME types in the leftmost tab, with a misc tab for things that can be created and that are considered distinct from MIME. After selecting misc, all the available index formats are shown in a second level of tabs. In Figure 10, I've chosen to create a TSearch2 PostgreSQL full-text index.

We use our existing PostgreSQL template database in the `gfcreate` shown in Figure 10 which has already got the TSearch2 PostgreSQL functions setup for it. An alternative is to directly setup your database with `psql`, import the TSearch2 sql file for it, name it in the `dbname` parameter and check `db-exists` to tell libferris not to try to create the database but to simply use the existing one with the given name. Using a template database is the recommended way as its the simplest.

If you are using the TSearch2 and PostgreSQL EA index combination then you can use the same PostgreSQL database for both indexes. Shown in Figure 11 is the creation of a new EA index which will use the same PostgreSQL database as we used above. Note that I've selected the `db-exists` option so libferris won't try to recreate the database but instead use the existing one. It is advantageous to create the fulltext index first if you have a fulltext template database and then check the `db-exists` and create your EA index pointing at the same database.

As can be seen from Figure 11 there are many more options for the EA index creation than for the TSearch fulltext index. Many of these options have to do with the trade off between indexing everything (and thus being slow to index each file) and indexing nothing (and thus being fast to add a file to the index). For example, the md5 EA requires each file to be fully read in order to calculate the checksum. If you are not intending to search for files based on their MD5 you can speed up indexing by not storing this attribute. The two parameters `attributes-not-to-index` and `attributes-not-to-index-regex` allow you to specify which EA are not interesting and thus should not be added to your EA index.

The defaults represent a midpoint between speed of adding to the index and richness of the resulting EA index for queries. These defaults for these three parameters can be overridden by running the `cc/caplets/index/ferris-caplet-index` tool,

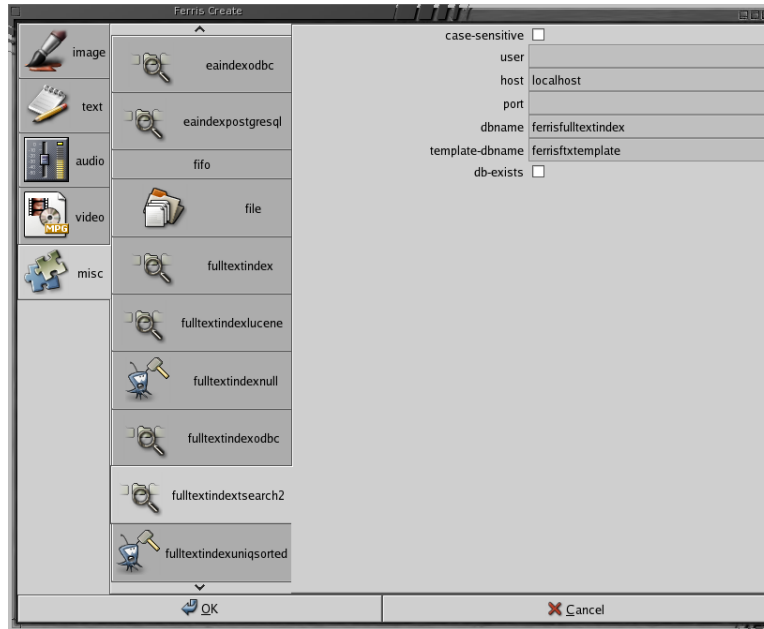


Figure 10: Creating a new TSearch2 fulltext index with gfccreate.

which sets the defaults for new index creation and alters your `~/ferris` indexes for future file indexing.

The schema of the relational database used for the PostgreSQL EA findex allows attributes to be stored either normalized in many tables or directly inline into the “docmap” or document map table. This trade off is similar to storing `stat(2)` information directly in the inode of a file, only with libferris you can choose what other EA you want to roll into the inode. If you are planning to query a handful of EA on a regular basis it might be a good idea to throw away some disk space and inline those EA directly into the docmap by adding them to this parameter.

The `columns-to-bitf` is a fairly advanced argument and for the most part should be left at its default. This allows you to specify which boolean EA you want to pack into a single SQL bit varying field. This gives you a similar boost to the `extra-columns-to-inline-in-docmap` parameter described above though a collection of binary attributes packed into a single bit varying will save space.

As an Extended Attribute can have an arbitrary size you can set a cutoff as to how much data a single EA can provide for a file with the `max-value-size-to-index` parameter.

A final touch can be added to indexes created using the above two modules. As they are both using the same backing PostgreSQL database they can be logically linked by libferris aswell using the command shown in Figure 12. Currently only the PostgreSQL EA findex and TSearch2 findex can be logically attached like this. This limitation will be removed in a future release.

4.2 Populating an index

The `findexadd` and `feaindexadd` commands are used to populate fulltext and EA indexes respectively. Both are non interactive command line tools.



Figure 11: Creating a new EA index using PostgreSQL with gcreate.

```

$$ feaindex-attach-fulltext-index \
  -P ~/path/to/ea-index \
  -F ~/path/to/full-text-index

```

Figure 12: Attaching a fulltext index to an EA index.

Other than the `-P` option to specify the index location these tools operate in some ways similarly to the `ls` command. For `findindexadd` you specify the files to index either as command line arguments or through a file using the `-f` option. The `-f` option accepts input in the form of a single file URL per line. As well as these options the `feaindexadd` command has a `-R` option to recurse into any URLs given on the command line.

One has to pay careful attention to what they are indexing with `libferris`. This is due to the ability of `libferris` to implicitly mount file formats which it knows about. For example, you may find that indexing a particular directory is taking a while, upon investigation you may find that this is due to `libferris` automatically mounting many archive files in your directory in order to index the archive contents.

Usage of the `-f` option gives explicit control over what level of detail you are interested in indexing. If `foo.tar.gz` is listed with `-f` then the files within the archive are not examined to be indexed as well. Of course you can choose to explicitly list the contents of an archive to have its internals indexed if you wish.

For simple usage a `find` followed by an `feaindexadd` will be the right choice.

```

$$ find ~ >|/tmp/filelist
$$ feaindexadd -f /tmp/filelist
$$ findindexadd -f /tmp/filelist

```

In the above I have elected to use both the `feaindexadd` and `findindexadd` com-

```

$$ feaindexquery -P /tmp/ea-index '(width>=640)'
Found 34 matches at the following locations:
file:///usr/share/backgrounds/images/dewdrop_leaf.jpg
...
$$ feaindexquery -P /tmp/ea-index '(size>=100k)'
Found 42 matches at the following locations:
file:///usr/share/backgrounds/images/dewdrop_leaf.jpg
...
$$ feaindexquery -P /tmp/ea-index \
'(&(width<=800)(size>=100k))'
Found 19 matches at the following locations:
file:///usr/.../images/space/apollo08_earthrise.jpg
...

```

Figure 13: Simple queries against your indexes.

```

$$ feaindexquery -P ~/path/to/ea-index \
(&(ferris-fulltext-search==alice wonderland)(size>=30k))

```

Figure 14: Performing an EA and fulltext query at the same time.

mands to give me both a fulltext and EA index for these files. I've not given a `-P` option here so both `indexadd` commands will use their respective default indexes in `~/ferris`.

4.3 Querying an index

There are many interfaces to your indexes. You can use the tools `findexquery`, `feaindexquery` or `ferrisls` for starters. The first two tools were explicitly created with index searching in mind. The `findexquery` tool supports passing in explicit fulltext index query formats. If you know that an index at location `~/my-docs-idx` is an xapian index then you can use the `--xapian` option to pass an xapian format fulltext query and have the result displayed.

A filesystem was created in `libferris` to allow index queries to be resolved and have virtual directories created with the resulting index matches. This allows most tools to directly post index queries, for example using the `ferrisls` tool.

The EA index query syntax is based on “The String Representation of LDAP Search Filters” as described in RFC 2254. This is a simple syntax, providing a small set of comparative operators to make lvalue operator rvalue terms and a means to combine these terms with Boolean and (&), or (|) and not (!) operations. All terms are contained in parentheses, with operators preceding their arguments. The operators are kept simple: `==` for equality, `<=` and `>=` for value ranges and `=~` for regex matches.

Some simple queries are shown in Figure 13.

We can also use the EA index which we created in Section 4.1 and perform both fulltext and EA queries at the same time because we logically linked the two databases. This is shown in Figure 14.

Explained in the LinuxJournal article [3] but and not repeated here is the ability for an EA index to store many versions of metadata for a file and allow you to query for files which had metadata matching the query at a given time.

```

$$ feaindexquery \
'(&(width>=1600)(ferris-current-time<=1 year ago))'

```

Figure 15: Querying for files which had matching metadata at a given time.

```

$$ fls -lh 'eaq://(size>=50k)'
-rw-r----- ben      ben      149.9k  05 Aug 25 13:44 alice13a.txt
-rw-r----- ben      ben      87.0k   05 Aug 25 13:44 crc.txt
...

```

Figure 16: Generating a virtual filesystem showing the results of an EA findex query. The URL is single quoted to stop the shell from attacking it.

A simple example would be looking for files which used to be large images and may have been overwritten with smaller images recently as shown in Figure 15. The full power of the logical connectives can be used on the time constraints as well.

The filesystems `eaq://` and `eaquery://` mentioned in Section 3.1 can be used by any libferris clients which operate on directories. An example of an EA findex query being used with ferrisls is shown in Figure 16.

5 Mounting libferris with Samba

There are many situations where one might wish to access libferris from legacy applications. With this in mind a Samba module was created which allows you to export part of a libferris filesystem as a Samba share.

A simple example would be a Samba share for a part of an XML file that you have mounted. This XML file may itself be inside of an archive like a zip file for an Open Office document. A more featureful example would be a Samba share showing the result of an findex query.

As usual see the README in the FerrisSambaModule for specifics on how to compile, install and setup your module. You will need a source tree for the version of Samba you are intending to use the FerrisSambaModule with. For recent versions (3.0.20+) of Samba you will only have to use the `--with-samba-source=` option for configure to tell it where to find your Samba source tree. The source tree is needed because compiling the module requires header files from Samba which are not installed by default. There are other implementation details which for the sake of consistency you should also see the “Setting up libferris” section at the bottom of FerrisSambaModule’s README file.

A simple share using libferris would be setup using the share configuration shown in Figure 17. Although this seems at first glance like a normal share on a kernel disk it does give you the ability to automount with libferris as shown in Figure 18.

There is an important implementation detail to mounting queries against an findex; A query may very well return two files with the same name and different URLs. As the query results are presented by default in a single directory, two files can not have an identical filename. Currently two solutions are implemented: one makes each file’s URL its file name, the other forces file names to be unique by appending synthetic version numbers to file names which conflict. A possible future extension would be prepending enough of a matching file’s URL to its name to make it unique.

```
[ferristest]
  comment = Testing ferris samba module
  path = /tmp/play
  writable = yes
  vfs objects = libferrissamba
  valid users = foo bar
```

Figure 17: A simple Samba share using libferris to provide the filesystem.

```
$$ cd /mnt;
$$ umount t2;
$$ smbmount //localhost/ferristest t2 -o username=saruchan,password=y
$$ ls -l t2
...
drwxr-xr-x  1 saruchan saruchan    310 Oct 24  1940 t.xml
-rwxr-xr-x  1 saruchan saruchan  91749 Aug 30  2004 v.pdf
...
$$ ll t2/t.xml/company/
total 2.5K
-rwxr-xr-x  1 saruchan saruchan 17 Oct 24  1940 category*
-rwxr-xr-x  1 saruchan saruchan 26 Oct 24  1940 developedBy*
drwxr-xr-x  1 saruchan saruchan  6 Oct 24  1940 nester/
-rwxr-xr-x  1 saruchan saruchan  8 Oct 24  1940 product*
-rwxr-xr-x  1 saruchan saruchan  8 Oct 24  1940 productx*
$$ cd t2/t.xml/company
$$ cat product
Xerces-C
```

Figure 18: Viewing the contents of an XML file as a filesystem using Samba+libferris.

When mounting with Samba it doesn't like having URLs as filenames very much. Thus one should use the `eaquery://filter-shortnames/` URL prefix for mounting index queries. This index uses the policy that files have their original name with possible version numbers to allow many files of the same name to exist in the same directory.

We also take advantage of the `realpath` module argument to pass the URL to use to the libferris module. This means that the many strange URLs that libferris is quite happy to handle can be passed directly in without potentially confusing the Samba server. The Samba configuration for mounting an EA index is shown in Figure 19. Interaction with such a share proceeds as expected as shown in Figure 20.

6 Future directions

The libferris suite will continue to expand what data sources it can mount as well as what metadata it can extract from its data sources.

Non mainline kernel extensions to notify to allow information about which process is modifying which files will allow for better time based searching in future builds. For example, seeing a list of the 20 next files which were modified by the

```
[ferriseaq]
  comment = A filesystem from a query
  path = "/"
  writable = no
  vfs objects = libferrissamba:"realpath=eaquery://filter-shortnames/(size>=30k)"
  valid users = foo bar
```

Figure 19: A simple Samba share using libferris to provide the filesystem.

```
$$ cd /mnt;
$$ umount t2;
$$ smbmount //localhost/ferriseaq t2 -o username=saruchan,password=y
$$ ls -l t2
total 1.9M
-rwxr-x---  1 root root 150K Aug 25 13:44 alice13a.txt*
-rwxr-x---  1 root root  48K Aug 25 13:44 boysw10.txt*
-rwxr-x---  1 root root  88K Aug 25 13:44 crc.txt*
...
$$ cd t2
$$ vi alice13a.txt
```

Figure 20: Viewing the results of an EA index query as a filesystem through Samba.

same application as the current one. Or for a given file, a list of the 20 next files modified by any application on the same virtual desktop.

The two indexing methods using PostgreSQL above also form the base indexing for the use of Formal Concept Analysis on filesystems. The use of FCA on such large and heterogeneous data sources is still the subject of heavy research.

References

- [1] Design of file creation support in libferris, <http://witme.sourceforge.net/libferris.web/ferriscreate.paper2001/index.html>. Visited Sep 2003.
- [2] libferris, <http://witme.sourceforge.net/libferris.web/>. Visited Oct 03.
- [3] libferris in the linux journal, <http://www.linuxjournal.com/article/7771>. Visited Jan 05.
- [4] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis — Mathematical Foundations*. Springer-Verlag, Berlin Heidelberg, 1999.