

# About libferris

## Table of contents

1 News Flash (2007).....	2
2 What is the libferris project?.....	2

## 1. News Flash (2007)

- Read the [developer's blog](#) for a closer look at news and upcoming features of libferris.
- Upcoming feature: mounting dbus! The ability to control the desktop as a filesystem.
- Recent feature: rsync integration.

```
$ rsync -avzX \  
  postgresql://localhost/mydb/table1 \  
  pg://backserv/mydb/table1
```

- **Fedora 7 yum repository** for libferris is now [available!](#). Other distributions available on "kind request".
- Want to upload an image file to 23hq or flickr? No worries;

```
ferriscp *.jpg 23hq://myusername/upload  
ferriscp gphoto://mycamera/*.jpg flickr://myusername/upload
```

Or mount via ferrisfuse and directly save from the gimp ;-) For more info, see the [linux.com](#) article.

- See how to use libferris and **XQuery** at [xml.com](#).
- Read the latest Linux Journal [article](#) and [the previous ones](#). See also the Linux Format [article](#).
- Need to leave the machine but not your current audio?

```
ferriscp -av amarok://playlist/*.ogg /my-portable-player  
ferriscp -av xmms://playlist/ /my-portable-player
```

- A custom version of redland is available from the sf.net download page which includes support for db4 database environments. It is highly recommended to use this version of redland with libferris for improved stability.
- Key: 72094a4d267ed4b7c7a33792d68d6902

## 2. What is the libferris project?

In non technical terms libferris makes the file system and other hierarchical storage systems easier to use. For the geeks out there, libferris is a virtual file system (VFS) that runs in the user address space. The FAQ contains entries related to installation, configuration and the usage of libferris.

As of July 2005 libferris can mount many interesting things ranging from a filesystem from your local Linux kernel through to LDAP, Evolution, PostgreSQL, dbXML, and RDF. To get

an impression of the current capabilities of libferris mounting see the plugins/context directory of the latest release. New things to mount are always being added :)

Other than mounting things as a filesystem, the other core concept of libferris is extraction of interesting metadata from your libferris filesystems. 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. Though some consider the last point of purely academic interest the end result is that you can add metadata to \*all\* libferris objects even those you only have read access too, for example, you can attach emblems to this 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.

Late in 2004 [extensive support](#) for both fulltext and metadata indexing was added to libferris. This means you can supply queries against the contents or metadata of any libferris accessible object and have the results returned as a virtual filesystem. With the above mentioned metadata available for searching, finding your files can be done in many different ways instead of being forced to generate fixed directory trees using part of a file collections semantics as directory names. The metadata and virtual filesystem play together here allowing you to geospatially tag both your digital pictures, trip plans, and relevant websites and recall these objects in a single virtual directory no matter what their path or URL may be.

There is also a Samba VFS module which allows you to expose a libferris filesystem as a Samba share. Kfsmd uses the inotify kernel interface to allow libferris to watch changes made to your kernel filesystem by non libferris applications and update its indexes appropriately. [Ferriscreate](#) provides a command line and GTK+2 application for creating "new files" with libferris. With this you can create a new db4 database, dbXML database or fulltext index just as easily as you can make a regular file.

The ego filemanager is a GTK+2 interface built on top of libferris. It provides GTK [treeview](#), [gevas/edje](#) and gecko based interfaces and makes extensive use of libferris' clients to provide its functionality.

If you have a project you wish to use libferris with and want extensions made don't hesitate to contact one of the developers to arrange consulting.

For the geeks out there, libferris is a virtual file system (VFS) that runs in the user address

space. At the moment libferris is a shared object that each application can dynamically link to in order to see the file system through a nicer abstraction.

New additions to the XML module allow for data to be converted from one format to another by the VFS for you. To copy data to an XML file:

```
fcreate --create-type=xml --rdn=2.xml root-element=fred /tmp
gfcf -av Makefile.am /tmp/2.xml/fred
```

To copy data to a db4 file

```
fcreate --create-type=db4 --rdn=2.db /tmp
gfcf -av Makefile.am /tmp/2.db
```

Ferris presents a C++ interface that makes heavy use of the STL and IOStreams. Currently ferris has two main internal abstractions: Context and Attribute. A context is much like a traditional file or directory in a file system, the major differences being that a context can have both byte content (like a file) and subcontexts (like a directory). An attribute is a chunk of metadata about a context. Contexts can have many attributes. Some attributes may be large, for example a base 64 encoded version of the context's content (133% context size). On the other hand an attribute can be small, for example the file size is exposed as an attribute.

Access to all contexts and attributes is performed by first requesting either an IStream or IOStream for that context or attribute. In this way the same context/attribute can be open many times at the same time, just like normal kernel based IO.

Ferris uses Loki from "Modern C++ Design" by Alexandrescu. Most objects use automatic garbage collection based on the SmartPtr<> template class from Loki. Where possible objects in ferris use a FerrisRefCounted policy to provide COM like intrusive reference counting. This style is used for Context, Attribute and special wrappers of IOStreams that are provided. IOStreams are wrapped to provide a more flexible API than could be offered using references to IOStreams. There are also new stream classes provided, for example NullStream and LimitingStream. Templates are provided to make SmartPtr<>s to standard IOStreams act just like the underlying stream would, for example, one can have SmartPtr<> ss; ss >> stringObj; and does not have to dereference the SmartPtr<> to use standard IOStreams extractors or inserters.

Ferris uses GModule from glib2 to dynamically load both context and attribute classes at run-time. This way resources are conserved until they are needed. The native file system context is statically linked to ferris at present. When loading either context's or attribute classes ferris uses a double dispatch factory method. Put simply this means that for each plugin there are two libraries, one that tells ferris if the main one really needs to be loaded or not. Using this scheme ferris can load all the meta factory classes at any time and use these

very small meta factories to check if the main factory can create objects that are going to be useful. This scheme is of great use for attribute classes. Attribute classes take a context and can "generate" attributes from the context. An example of this sort of class would be a MD5 or Base64 attribute. Both can be generated from the base context. More interesting attributes are PCM audio and RGBA-32bpp image data. By using the double dispatch factory ferris can handle a great deal of attribute generators and load them on demand.

Ferris currently can decode mp3, read id3 tags, decode many image formats and break some animation formats into frames. This makes ferris a solid starting point for multimedia applications.

Ferris will automatically mount sub file systems for you. Examples of a sub file system include a Berkeley database or XML file. For example it is possible to read a context such as /tmp/myxml.xml/mynode. Using this automatic mounting the differences between storage formats effectively disappear. To a ferris enabled application loading data from a native disk file, a Berkeley database, and XML file, or mbox file appear to be the same. This allows the user of the application to choose the correct storage for the data at hand.

It is planned to move to a microkernel architecture in Version 2.1 of ferris. I choose 2.1 so that ferris does not fall into version 2 syndrome :)