# FIOSys
## (How FIOSys Works)

A Software Developer's Tool that helps other Software Developers bring their products to market faster, with more advanced features, and added value so that their products are more competitive.

**By**:Jon Fernandez
**Date**:6/1/2013
www.fiosys.net

# Table of Contents

# I. How FIOSys works

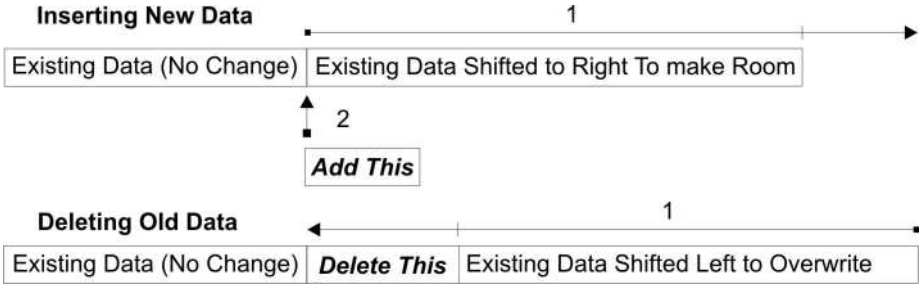
## FIOSys Provides Advanced File IO Services that Applications need.

**Right now, for a 3rd party developer to create from scratch an application that uses even simple file handling, requires a significant amount of time spent up front building the basic tools that go into handling that application's files (input/output and in-file organization).** The goal of FIOSys is to provide these lower level services from day one of development, freeing the developer to concentrate on the actual problem that the developer is trying to solve (*building the better mousetrap*).

Of course **every programming language** allows you to **open and close** a file and perform **reads and writes**. But there are **numerous other tasks** that go into **storing data in a file in a reliable and sensible fashion**. Here is the basic list of services FIOSys provides the developer to solve these needs starting on day one.

### *A. Intelligent Insertion and Deletion of data*

**Inserting into and deletion of data from a file is one of the most time intensive jobs a computer performs.** Think of a 1 gigabyte file, where you want to periodically insert or delete small amounts of data into that file (add new database records, spreadsheet cells …). On average, this job requires a massive 1/2 gigabyte read and write of the data just ahead of where the file is being updated. **For an insertion, data must be shifted to the right to make room for the new data. For a deletion, data must be shifted to the left to overwrite the data being deleted.** This entire process is a time consuming and massive strain on the slowest part of the computer process (disk file IO).




There are various ways that this process can be improved. **FIOSys does this in a way that takes just slightly more time than it takes to write that amount of data just by itself.** So to insert or delete a 1K amount of data would take about the amount of time it takes to write 2K of data. This is many magnitudes better than pushing 1/2 a gigabyte of data around inside our 1 gigabyte file every time an insert/delete needs to occur.

Other methods that developers have used involve storing all new data at the end of a file where it's quick and easy. Complex methods must then be used to index or reference this data in a pseudo sequential fashion, or mark data for deletion at a later time when it can be done in bulk. This works but is cumbersome, prone to errors, and counter-intuitive.

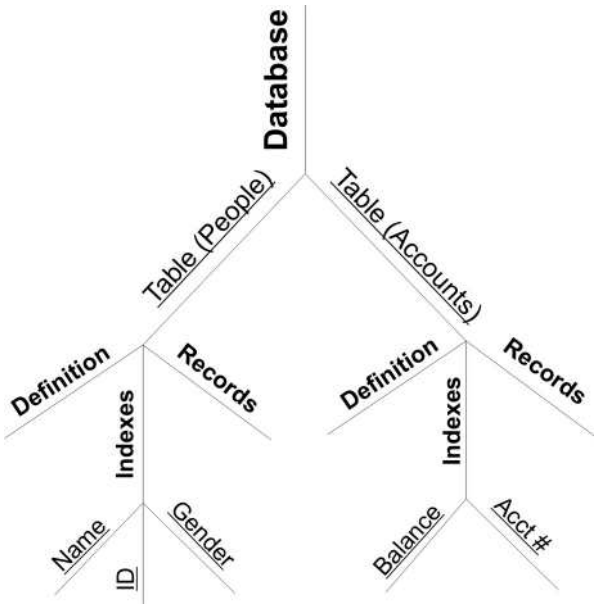
*As a programmer, the idea that I can insert data right in the middle of a file right where it **logically goes**, or **delete data at will without some form of marking or caching**, is ideal!*

## *B. Treelike File Organization*

When the first database applications were developed, a single database might have consisted of **multiple files**, one file for the database definition, one for the actual data records, and one for each index. If the need was for a multiple table database like modern databases usually are, the number of files in the database would increase proportionally.

**Of course having so many separate files like this is inconvenient and risk prone** for many reasons: the number of files open, keeping the files consistent with each other, the ability to backup or recover a complete file system, the ability to transport (Email, download, hand someone a CD ...) a complete file system.

**FIOSys allows an application to organize a file into separate branches of data in any configuration desired.** A fully multibranched tree organization is maintained in every single file. A **major branch** of a file could be a **database**, with **subbranches** off of the database **representing each table** in that database. Each **table branch** would have **additional subbranches** for the table **definition, data records, indexes, stored procedures** …




Even simple files transferred over the internet might want separate branches for a definition of the file, any stored images or text, different versions of the text (HTML, XML, Plain text …).

**It is a major benefit for a developer to easily manage in <u>one file</u>, all the pieces of data that encompass the complete set of information that application needs.** It is also a huge undertaking to create this capability from scratch. To add data into one part of a file (new data records, indexes …), while deleting data from another part of the file, with each region growing and shrinking in a way that affects all the other parts of the file around it - and to do this reliably and efficiently without corrupting the file in the process - is a formidable undertaking.

## *C. Preimaged, Incorruptible Files*

**The power goes out or an application crashes.** What is the state of your bank's financial transactions, or the article you were writing, and all the last 4 hours of changes that have been made to your application's data. Even if your IT department is good and backups are made every 1/2 hour, a system crash at the wrong time will almost always leave your data files in an unfinished, scrambled, and usually corrupt state. Maybe days, months, years of data are at risk. And **how often have you or your IT department been doing backups <u>and</u> checking the integrity of the backup process.** Have you ever heard this conversation from techs after a serious system crash (*we have*)?

*The backup was running correctly when we first set up the system, but it turns out there was a problem we didn't realize was there until the crash occurred and the system hasn't been backing up correctly for **months!***

**FIOSys provides a built-in preimaging capability** so that short of the drive needle scraping across the surface of the disk, files cannot be corrupted. Built in Preimaging is used to store changes in a temporary file. Once it is time to write changes back to the original file, the data is still in the preimage file so that if the power goes out, upon startup, the changes to the file will be rewritten completely and safely from the preimaged data so that nothing is corrupted or lost.

**FIOSys also provides a more intelligent Preimaging capability than the earlier attempts at doing this**, which essentially **used to double the time** it took to do file IO (write to the preimage file and flush/close the file, then write the same data to the actual file …). Tests on our preimaging mechanism shows that preimaged vs. non preimaged file IO adds about a 25% overhead when preimaging is used.

## *D. Transaction Processing*

Let's say a developer is writing an accounting application for a bank and the system must ensure that when a customer's checking account makes a payment, the corresponding account that receives the payment is also credited **at the same time**. It would be a real problem if Susie Q's checking account got debited, but the system crashed and the mortgage payment she just made never got credited. Money just disappeared into the air and Susie got a late notice, added fees, and her credit rating is now impaired.

**FIOSys provides built in transaction processing**, so that all of a transaction is recorded or none of it is recorded. FIOSys is told when a full transaction is complete, and all changes to all the involved files are then written at once (and preimaged beforehand of course).

## *E. Multiuser Collision Handling*

Many applications allow multiple individuals to affect stored data at the same time. Network Databases, airline reservation systems ... must all allow record locking so that 2 individuals can access the same data without overwriting each other's changes (2 plane reservations aren't made for the same seat at the same time).

**FIOSys automates collision handling allowing locking of portions of a file** depending on who got to that part of the file first. There are many permutations available with this to decide who controls the data and what occurs during a collision.

## *F. User Rights System*

An Admin system is included that allows owners to assign differing Rights (Read, Write, Build ...) to different Users. Rights can be assigned down to the Tree level, so different branches of a file's tree structure can be restricted to specific Users.

## *G. Balanced Binary Processing*

FIOSys employs a sophisticated balanced binary tree mechanism for its internal processes including:

1. **RAM based buffering of all IO -** What has been read once can be found instantly, with very little IO going back to disk unless file updates are being committed.

2. **Collision handling between different Users -** Multiple User, File, Tree regions are maintained in a quick lookup mechanism to handle user control and collision handling when to Users try to access the same data in a file.

## *H. Self Testing*

One of the features we are most proud of is the **built-in self-testing capability** of our system. Every task FIOSys performs, from simple data comparisons to binary tree handling to buffered file IO to multiple user collision conditions, has rigorous tests developed to challenge each part of the system. We strive to anticipate every process and eventuality that can occur. At any point during our own development, we can run a complete suite of tests to ensure that the system still operates correctly. We are committed to having at least 1/4 of all our development time spent maintaining the self-testing capacity of the system (so far actually about 1/3).

What does this provide other than we sleep better at night because of it? **It means that every part of the system is rigorously tested through each and every version of development.** No system goes out that doesn't pass the same test that every previous version of the system passed. When a new problem arises (a bug is found), we look first at why the self-testing didn't work, develop a new test that exposes the condition that brought about that bug, and that test is now part of the improved self-testing system.

In addition, we will save various test data files created at each stage of FIOSys development so that these files can be tested against newer FIOSys versions. This ensures that FIOSys behaves correctly over the years with all legacy applications.

**Again, this means that every version of our system <u>is as or more reliable</u> than any previous version of the system.**

## I. Disk or RAM Based file capability

FIOSys Files can either be disk or RAM based. There is no difference between how the two operate. RAM based files are potentially 1,000 times faster than disk files and are useful for temporary files that need to process quickly and where preimaging or survival of the data isn't important.

Consider a high speed testing environment (an atom smasher, a multiuser test suite …) where real-time data needs to be processed and analyzed quickly. There might be a battery backup to ensure data integrity if the power goes down, or the actual survival of the data isn't necessary to run the test (if the power goes down, the atom smasher is down too).

Using a RAM based file is also useful during application testing where a developer wants to challenge their application with a lot of data to verify program logic, but the actual survival of the test data isn't important.

## J. Cross Platform Compatibility

FIOSys is fully written in C, so that any Operating System with a C compiler can use it (which should be all the major OSs). We are dedicated to FIOSys not being an extension of one platform over another. Current development uses the Windows Visual Studio IDE, but all significant code is native C and OS independent.

The benefit with this is that any file can be handed back and forth between different Operating Systems. This would be especially critical with Web applications that use data files, as the server handing off a file could be Unix, and the client browser receiving the file might be Windows.

FIOSys will also be little\big endian neutral. The means to accomplish this is simple to do though the capability won't be fully implemented until we port to a big endian system.

## II. Links

**Summary** - www.fiosys.net/fiosys/attachments/summary.pdf

A one page Summary about the Benefits and Status of the FIOSys project.

**Purpose** - www.fiosys.net/fiosys/attachments/planpurpose.pdf

A roughly eight page document on the primary Benefits (purpose) of developing FIOSys.