Inside FileFlex

by David Gewirtz

Component Software Corporation

Table of Contents

About FileFlex: The Relational Database Inside Multimedia

> "Component's FileFlex embedded relational database has given our multimedia developers the ability to embed standalone database functionality within their CD-ROM and kiosk development projects," said Joe Dunn, Macromedia's vice president of product management.
>
> "Component's acquisition of the ODBC technology now makes it possible for highly interactive multimedia productions to exchange information with databases throughout the entire corporation."

Component Software Corporation is the leading supplier of embedded database solutions for multimedia productions. FileFlex has had an interesting history. Initially developed by Dan Shafer and Dennis Chen at GUI, Inc. of Redwood City, the product (then called "Dashboard") was licensed to Symmetry Software for distribution. Ultimately never shipped by Symmetry, the product (now called "FileFlex") was licensed to Hyperpress Publishing and shipped in 1991. Component acquired Hyperpress (and along with it, FileFlex marketing rights) in 1992 and has been publishing FileFlex ever since. Then in September 1993, Component's founder David Gewirtz acquired all rights and ownership to the product including its source code. Since then, David Gewirtz has taken the product through numerous major and minor revisions and improvements. Recently, Component acquired ODBC technology from Solis and has released a new, separate ODBC Gateway product that connects authoring environments with enterprise-wide databases.

In addition to being the author of FileFlex, Component Software founder David Gewirtz is the author of The Flexible Enterprise (published by John Wiley & Sons) and Lotus Notes Revealed! (published by Prima). He is also the Editor in Chief of Workspace for Lotus Notes and Insider for Lotus cc:Mail, both published by The Cobb Group, division of Ziff Davis. Gewirtz is an assistant professor of computer science at Mercer College in Princeton.

Must Read: FileFlex Support

- FileFlex Headquarters on the Web
- The FileFlex Mailing List
- Online Service Support Forums
- How to Get Technical Support

FileFlex is a highly technical product targeted at solely technical users who want to embed relational database technology within their own applications.

As part of creating FileFlex, I started developing a regular dialog with FileFlex users via electronic mail. I was able to provide regular update status reports to all members of our e-mail mailing list and discuss issues relating to the product's features and functionality. This became an incredibly powerful tool for getting customer feedback and helped those on the list be more informed as the development progressed.

To continue this dialog, **I STRONGLY RECOMMEND** you subscribe to our free mailing list services as soon as possible.

FileFlex Headquarters on the Web

FileFlex Headquarters, our main location for all FileFlex informaiton and support, is on our World Wide Web site for FileFlex users. This site contains the latest versions, archives of technical and marketing documents, mailings, and all sorts of other incredibly useful information. Component on the Web is located at:

```
http://www.component-net.com
```

This site is the first place you should look for any upgrade or free bug-fix update information, as well as answers to commonly asked questions. Remember, the very latest version of this manual is available here for browsing and downloading.

The FileFlex Mailing List

FileFlex users have access to a free "listserv" feature. This service is available to both full FileFlex customers and FileFlex Lite users. I hope you find this cool new communications tool useful. Listserv is an automated mail list manager that lets you get regular information about FileFlex and talk with other FileFlex embedded database users. The two new automated lists are:

```
fileflex-announce
fileflex-talk
```

FileFlex-Announce

If you have any interest in FileFlex you should subscribe to fileflex-announce. Whenever we have important information to announce (anything from new version information to important tips and problem workarounds), we'll post them through fileflex-announce.

In order to receive mail from fileflex-announce, send an e-mail message to listserv@netcom.com. In the first line of the message (not the subject!), include the following:

```
subscribe fileflex-announce
```

FileFlex-Talk

The fileflex-talk list is an "open" list. The purpose of fileflex-talk is to provide a mechanism for FileFlex users and fans to converse among themselves. This list is unmoderated and unedited. Anything you send to fileflex-talk will be replicated and mailed to all of the subscribers on the list. In turn, anything any other subscriber mails to the list will be sent to you (assuming you've subscribed).

Fileflex-talk is intended to provide FileFlex users with a way to share great ideas, get help on sticky problems, and work together to explore new uses of the program. So send messages and reply to messages. Get together and talk, have fun, and share great stuff!

In order to receive mail from fileflex-talk, send an e-mail message to listserv@netcom.com. In the first line of the message (not the subject!), include the following:

```
subscribe fileflex-talk
```

In return for this short effort on your part, you'll be the first to know about upcoming releases, and-more importantly-be involved in issues and decisions relating to features, functions, and policies for FileFlex. And you won't be getting information filtered by some sort of marketing flunky-you'll be talking directly with the developer.

See you on the list!

Online Service Support Forums

If you don't have access to Internet FTP, don't dispair. We've made access available to you as well. You can look for FileFlex files and other goodies in the following locations:

- CompuServe: Macromedia Forum (GO MACROMEDIA)
- HyperCard Forum (GO MACHYPER)
- Component Section (GO COMPONENT)

How to Get Technical Support

**Users of FileFlex Lite must contact
Macromedia at 415-252-9080 (Fax 415-703-0924)
for technical support.**

**Please do not call Component.
We're just not equipped to accept
technical support calls from all of
Macromedia's customers.**

Technical support is available free of charge (unless you become a raging pain in the ... but that's another story) directly from the author. We don't normally charge for support. We also don't normally write or debug your applications for you. Please note that our preferred support method is electronic mail. We generally turn around e-mail support questions within a day, but make no guarantee for phone or fax-based support queries. If you do call, you'll be asked for your download password (on your disk) to confirm you're a customer of the full version before you'll be able to get technical support from us.

Remember, if you're a FileFlex Lite user, you can subscribe to the listserv and visit our website for free, but you must contact Macromedia for support at the phone number listed above.

The absolute best way to get support is via E-mail to fileflex@component-net.com.

1. Introducing FileFlex

- FileFlex 2.0 Features and Benefits
- FileFlex Lite Edition
- FileFlex Professional Edition
- FileFlex System Capacities
- The Difference Between FileFlex and a Full DBMS
- Interfacing FileFlex to Host Environments
- FileFlex/ODBC Gateway

FileFlex is the ragingly-fast, cross-platform, relational database engine for multimedia, designed for creating CD-ROMs, kiosks, and interactive multimedia projects. FileFlex is fully xBASE/dBASE compatible. FileFlex and FoxPro (and other xBASE-compatible software) can directly exchange files without conversion. FileFlex requires no special drivers and only 100K of RAM--allowing you to add relational database capabilities to their projects at virtually no cost in system resources. FileFlex also includes dynamic, on-the-fly office-quality encryption and full-text search and retrieval capabilities. FileFlex Lite is bundled with Director 5, allowing Director users to add database capabilities to their multimedia productions.

FileFlex 2.0 Features and Benefits

FileFlex 2.0 features include:

- **FileFlex is cross-platform:** Director users don't have to rewrite FileFlex Lingo source code, it's the same on both Macintosh and Windows! You can use the same data files on Macintosh and Windows!

- **Extremely high-performance:** FileFlex uses high-performance data management algorithms to retrieve data in any size databases. Properly used, FileFlex can retrieve any record in a database containing in excess of a billion records in well under a second on even the slowest machines.

- **Optimized for the host environments:** FileFlex has been optimized with 32-bit native versions for Windows 95, Windows NT, and the PowerPC processor on Mac OS. Native 16-bit versions are also available for Macintosh 68K and Windows 3.1. A "fat" binary version is available.

- **Add database features to your projects:** Director, Authorware and other tools can give you complete interface flexibility. And now with FileFlex, you've got all the speed, power, and capability of a relational database as well. FileFlex works behind the scenes, completely hidden from the user.

- **Fully-relational database:** FileFlex is fully relational. You may have up to 20 database files open at once. You can index files on key fields and relate the indexes, permitting instant access to related information.

- **Elegant integration into your environment:** Grab a single record or any desired group of records. You can drop retrieved data into your application's native fields or global variables. FileFlex gives you transparent, totally non-instrusive access to thousands or millions of records.

- **Easier update management:** Because your data is stored externally from your project, updating users with new versions of your project is much easier.

- **Simple and complex searches are possible:** Use logical expressions to find information in a database file. A logical expression can consist of constants, field names, and up to 12 functions joined together by operators.

- **Standard files supported, FoxPro compatible:** FileFlex supports the use and updating of ragingly fast index files. FileFlex can directly read all xBASE-compatible data files and can dynamically reindex as data changes. FileFlex can directly exchange files without any conversion required.

- **New, advanced features:** FileFlex introduces dynamic, on-the-fly office-quality encryption/decryption and simple full-text search and retrieval.

FileFlex is a very tight code library. It is distributed as a loadable code module (a DLL under Windows and a code resource or shared library on the Macintosh). As a result, there's no need for interapplication communications and you don't have to worry about special installations for alternate applications. FileFlex also does not require any special drivers. Once you install the one code module, FileFlex is available.

FileFlex Lite Edition

When you get Director 5 from Macromedia, you'll also get a copy of FileFlex Lite. FileFlex Lite is a complete version of FileFlex with only one limitation: you can only access the first 1,000 records of a database. If you want access to the full professional edition of FileFlex, you can purchase it directly from Component at www.component-net.com.

Users of FileFlex Lite must contact
Macromedia at 415-252-9080 (Fax 415-703-0924)
for technical support.

Please do not call Component.
We're just not equipped to accept

technical support calls from all of Macromedia's customers.

FileFlex Professional Edition

When you buy the full FileFlex from Component Software, you're getting the Professional Edition of the product. You can tell you've got the professional edition because your DBVersion() command will return a "2.0P" rather than a "2.0L" in the version string.

The professional edition is the complete software, including the ability to access an unlimited number of records, documentation, detailed technical information, technical support, access to E-mail feedback channels, and development tools. This "whole product" is called the FileFlex professional edition and if you're reading this document and have purchased FileFlex from Component, you own the professional edition.

You are licensed to run FileFlex on one personal computer per copy of FileFlex professional edition purchased. If there is more than one person in your organization developing using FileFlex, please purchase a professional edition for each person.

FileFlex System Capacities

- Max records per data file (professional edition): 1,073,741,823
  FileFlex Lite (bundled with Macromedia Director) supports up to 1,000 records
- Max characters per record: 4,000
- Max fields per record: 128
- Max characters per field: 254 / memo fields hold up to 32K
- Digits of precision in numeric computations: 16
- Maximum characters per character string: 254 / memo fields up to 32K
- Maximum characters per index key: 100
- Maximum memory variables: limited to memory only
- Limits on number of files open: constrained by available memory

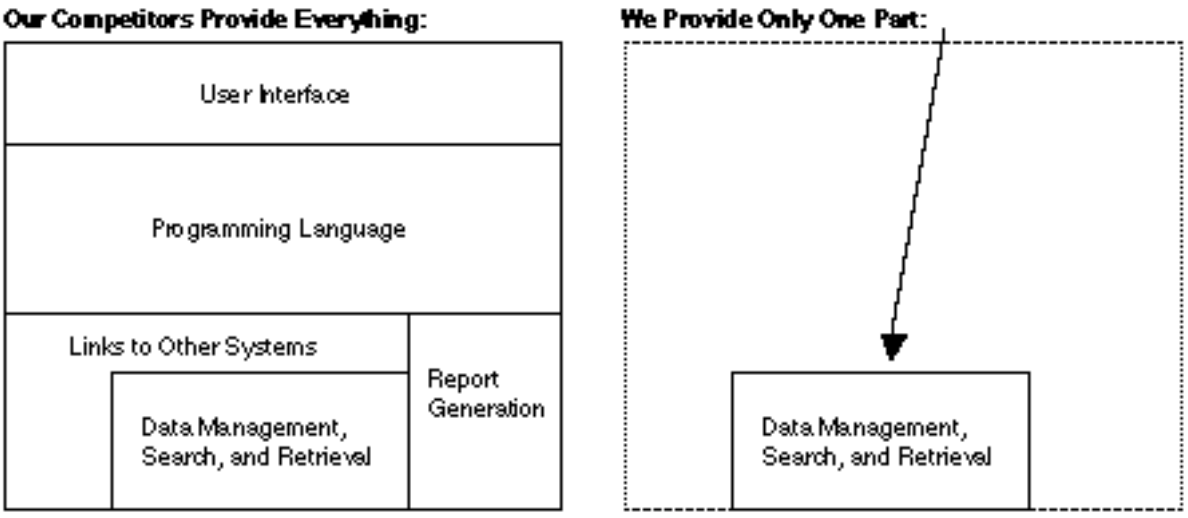
The Difference Between FileFlex and a Full DBMS

FileFlex is a database product in a market unbelievably overcrowded with database tools and dominated by some of the largest software companies-including behemoths Microsoft and Oracle. These are not competitors you want to meet even in broad daylight in a deserted alley!

Yet FileFlex is amazingly successful. Why? Is it because it's a better product than those produced by the big boys? Is it because our marketing is vastly superior? Is it because we're entrenched in some industry-critical installations?

If you answered "No" to all of these, you'd be correct. In fact, the "Yes" answer applies to our competitors. Had we attempted to compete with them on their terms, we would have been squashed like a bug. And justifiably so. FileFlex has neither the engineering technology nor the marketing muscle to compete in the big leagues. But it is a highly successful farm league player.

The big products (names like FoxPro, dBASE, Oracle, Ingres, Informix, Access, 4th Dimension, FileMaker Pro) are general purpose database products. They contain all the features you need to build whatever you want. Not only do they manage the data, but they have tools for building the interface (how the computer interacts with you), reports, and controlling operation (usually a built in programming language). FileFlex has none of that. FileFlex just manages data. The picture below shows how much more of the database "solution" our competitors provide:



So, how do we compete? We have different customers. The big market in databases is in full solutions (i.e., accounting systems, order processing, banking). But there's also a market (much smaller!) that just needs to get at and retrieve data. The buyers in this market are programmers and developers themselves and their desire is to embed data access technology inside their own products and solutions (without becoming beholden to some large monolithic technology provider).

The overall database market is in the billions. The total market for embedded databases is way under $50 million. Very few players are in the embedded market, so we're able to make money. Many of the big database systems, while most effective in the hands of programmers, can still be used by almost anyone. But to use FileFlex, you must be a bona-fide, card-carrying propeller-head. No non-nerds need apply.

If you haven't guessed already, nichemanship (playing to market niches) is an obstacle avoidance strategy that works. When it comes to databases, our competitors were after the mainstream, so we went after developers.

Interfacing FileFlex to Host Environments

In addition to the normal stuff that you get in a programmer's manual, the purpose of this section is to help you understand some of the design decisions that went on behind creating FileFlex' cross-platform architecture.

Development Environment	Native Operating System/Architecture			
	MacOS/PPC	MacOS/68K	Win95/NT	Win3.1
Director 5	Yes	Yes	Yes	Yes
Director 4	Emulated	Yes	16-bit	Yes
Authorware 3	Emulated	Yes	Planned	Planned
HyperCard 2.3	Emulated	Yes		
SuperCard 2.5	Emulated	Yes	Planned	Planned
Metrowerks C	Planned	Planned		

I've been down this path before with a number of other projects and products, both for Component and other companies. I know that a straight port can often result in a working piece of code for one version and unmaintainable code for versions later on. This happens especially if you have two different sets of source code. Changes made to one somehow have to be properly rippled back to the other and, 99 times out of 100, some critical sets of changes just don't make it. This is one very good reason why it often takes much longer for a piece of software on one platform to follow a piece of software on the other. I was unwilling to incur the overhead cost and pain of maintaining lots of different sets of source code for each of the products that FileFlex runs inside. From that desire came the first major decision and major architectural change in FileFlex, the need for a common source code model.

Common Source Code Model

Briefly, a common source code model means that there is one body of source code used to generate all of the individual different versions of the product. As a result, a change that's made in a source file doesn't have to be duplicated in an identical source file on another system. For any given piece of code, there is only one individual source code file.

This architecture necessitated the breaking up of FileFlex internal structure in such a way that it allowed for two moving targets. The first moving target, at the bottom end of FileFlex, is the operating system interface. This would include the various function and toolbox calls used to do things like access files, check timing, move record pointers around, and so forth. The series of routines that control this needed to be rebuilt into, for all intents and purposes, a virtual machine or a Virtual Operating System Layer. In other words, at some point in FileFlex a function call is made to access, say, disk data. That function call, however, knows to tell the operating system layer on the Macintosh to issue a toolbox call and under Windows to issue a different kind of File I/O routine.

But there was another important requirement, and that was that this Virtual Operating System Layer must not incur a speed hit, and must not increase the size of the resulting binary or memory requirements in any measurable way. As a result the Virtual Operating System Layer is

controlled entirely at compile time by setting what are called #ifdefs. #Ifdefs, when set one way, cause the compiler to compile one bank of source code and, when set another way, cause the compiler to compile a different bank of source code.

On top of the operating system layer are the main guts of FileFlex and that is common across not only all operating systems but all development environments of which today we support a moderate amount and in the future I intend to support a much more significant amount.

The existence of all these development environments necessitated the creation of the third layer, the top layer of FileFlex: the Development Environment Interface Layer. This layer, too, has to change with the different development environments, especially since FileFlex must live inside them. Virtually every environment has its external API interface that must be supported.

Every one of these software development environments will have a different plug-in interface, and FileFlex had to be rearchitected in some way that supporting different interfaces did not require a vast rewrite of the product. Otherwise, it just wasn't going to happen. So, the Development Environment Interface Layer was created to do some degree of standardization between each of these development interfaces. This, too, was controlled by #ifdefs, allowing each interface to be created at compile time without incurring a performance or resource allocation hit during runtime.

Common FileFlex API

Another fundamental decision was to require FileFlex to have a common API in any development environment in which it was used. It struck me as worthless if you learn to use FileFlex in Authorware and then you moved to Director and you had to learn a whole new set of commands. Or if you moved from Director on the Mac to Director under Windows and the commands behaved differently. ach environment may call its externals in a slightly different manner. Obviously, Authorware (which is a visual programming environment) will call its externals in a manner substantially different from the way Lingo calls its externals.

In any case, if you know FileFlex's commands, calling conventions, and syntax in one environment, you will know them for all across all platforms. This is particularly important in the case of cross-platform development environments like Director. An absolutely essential development goal was that you could move projects from Director on the Macintosh to Director on Windows and vice versa, without having to rewrite all your code. There are obviously some exceptions. There are some minor exceptions for text handling and there are, of course, exceptions for file path handling. So, for example, you would separate items in a path with colons on a Macintosh and with backslashes under Windows. However, those changes are minor compared to having to rewrite all of your functions when you move it over from Mac to Windows. The same architectural goal is true of any other cross platform development environment.

Common Data File Format

Another requirement was that the data file that you use in any given development environment can be used in any other given development environment without having to go through a translation process. That's why we use the DBF standard with FileFlex to begin with, because we want to make sure that we support an industry standard data file. And it makes no sense if you decide to migrate your application from Director to Authorware or from SuperCard to Director, that you need to do a massive conversion on your data file. Again, this is especially true in the case of going from Mac to Windows. I talked to numerous users who are building CD-ROMs, and they're building a hybrid CD-ROM where they'll have a Macintosh-executable and a Windows-executable, and two or three hundred megabytes of data files. The last thing they want to do is duplicate those data files simply to support a Windows format and a Mac format. We allow you to share the data file format between each platform that FileFlex will operate under. One caveat: this does not necessarily mean that you can access the same data file at exactly the same time from multiple platforms or from multiple processes. FileFlex does not currently support multi- user access, although that's certainly something that we're exploring for future releases.

Pretty Ambitious Goals

To summarize, I had some pretty ambitious goals for this port of FileFlex. One, I wanted to make sure that we used a common source code model so I wouldn't have to spend an enormous effort synchronizing lots and lots of source files. Two, we architecturally divided FileFlex up into 3 layers: the Virtual Operating System Layer, the FileFlex Layer, and the Development Environment Interface Layer. This modular structure allows us to tune FileFlex for each of the host environments it needs to operate under. Further, the modularity could have no cost in terms of performance or resources, and so we needed to develop a compile time environment that would allow that to happen. Three, the application programming environment from the point of view of developers using FileFlex had to be identical from platform to platform, so no extra learning curve needed to be incurred. Fourth, there had to be a commonality of source code across platforms. This is so if you write code in one development environment on the Macintosh, you can move it to an identical development environment under Windows without having to rewrite all your code. And fifth, with the ability to use shared data files so that you didn't have to store twice the amount of data if you happened to want to build a hybrid CD.

The final requirement for FileFlex was that your decision to use it could not increase your overhead substantially. As a result, FileFlex consists of one single DLL or code resource file. When you distribute FileFlex to your customer, you only need to be concerned about moving that one file into a directory that you choose. FileFlex does not require you to have tons of files that need to be moved to lots of different directories in order to operate, as so many other Windows products do require. I consistently tried to make sure that FileFlex didn't turn into a pain in the butt.

Cross Platform Considerations

Today, FileFlex for Windows only runs inside of Macromedia Director. I have near term intentions to support both Authorware and SuperCard under Windows, but I have not yet completed these ports.

There is actually only one other significant cross platform consideration that you need to worry about other than your own personal religious operating system preferences. That concern is line termination in text and how it differs between operating systems. Briefly, the Macintosh terminates each line with a newline, and under Windows the convention is that each line is terminated with a carriage return and line feed. This doesn't become an issue in data storage in FileFlex except in memo fields which have hard carriage returns.

In fact, this actually isn't a consideration if you are moving data only inside of Director, because Director on both Macintosh and Windows uses the Macintosh format newline end of line character on both environments. As a result, if you write something that you store into a memo field in FileFlex on the Macintosh and you move it over to Windows and you run it in Director under Windows with FileFlex for Windows you will be able to read that memo field with no discernible difference.

The place where there is some concern is bringing in externally created DBF data files. This would happen, for example, if you created a data file in FoxPro under Windows and in the memo field had a number of carriage returns that were hard typed into the memo field. When you read that data in FileFlex and you moved it into Director, Director would have some ugly characters dropped into place, because Director would be looking for Mac formatted text data and FileFlex would have delivered the PC formatted data instead. Obviously, the same is true if you take data that you store in FileFlex that you want to read in FoxPro, FoxPro would get confused because it would have new lines, but not carriage returns.

To get around this problem, FileFlex, both on the Macintosh and in Windows, has a data file function called DBConvertCRLF. The purpose of DBConvertCRLF is to take a chunk of Macintosh text and convert it to a format that external Windows environment tools would expect, or take a chunk that comes from an external Windows format tool, and convert it to a form that would be suitable for use in the Macintosh. This applies to users who aren't even doing cross platform development, but who simply want to read PC data files that have memo fields. Now you can read a PC format data file with the memo field. You can read that data into a variable, you can run it through DBConvertCRLF, and out the other end will come a memo field that you can be comfortable displaying in a field on either a HyperCard stack or a Director project or whatever. Then, you can go the reverse. You can make your changes and you can send it back through DBConvertCRLF and store it back out to a file so that PC users can understand it without needing weird conversions.

FileFlex/ODBC Gateway

Component Software Corporation recently announced the acquisition of the Solis ODBC gateway from Solis Development, Inc. Renamed the FileFlex/ODBC Gateway, the new product provides a connection between Macromedia's multimedia authoring tools Director and Authorware and enterprise-class database systems like Oracle, Sybase, and Informix. The FileFlex/ODBC Gateway is a separate product from FileFlex. Usually they will not be used together.

FileFlex itself is a relational database engine that reads and writes dBASE III format .DBF files. By setting a compatibility mode in FoxPro, you'll be able to directly read data within FileFlex (which itself is an extension that adds features to Director).

Advantages are access to the same data that FoxPro reads with a minimal overhead in terms of drivers. Disadvantages are that Director and FoxPro can not have the data file open at the same time and that FileFlex does not provide multi-user access. FileFlex also doesn't handle the pure FoxPro format files. Another advantage is that runtime distribution of the FileFlex components are pretty inexpensive.

The FileFlex/ODBC Gateway adds Open Database Connectivity (ODBC) functionality to Director and Authorware. But you also need to have the ODBC drivers (standard under Windows, additional cost for Mac from a company called InterSolv). Because of the drivers and all the additional software, RAM requirements and system overhead are considerably more than the straight FileFlex solution. The FileFlex/ODBC Gateway allows multi-user access, allows (assuming your driver talks it) you to talk to many different databases in their optimized file formats, and also lets you talk to other database formats. But the cost structure is such that you're probably not going to distribute a solution using the FileFlex/ODBC Gateway for consumer CD-ROMS.

FileFlex is a great CD-ROM/kiosk solution. The FileFlex/ODBC Gateway is more of an enterprise oriented solution for access to large corporate databases.

"Component's FileFlex embedded relational database has given our multimedia developers the ability to embed standalone database functionality within their CD-ROM and kiosk development projects," said Joe Dunn, Macromedia's vice president of product management. "Component's acquisition of the ODBC technology now makes it possible for highly interactive multimedia productions to exchange information with databases throughout the entire corporation."

The new FileFlex/ODBC Gateway is a very important link in the chain connecting multimedia content to corporate information warehouses. The FileFlex/ODBC Gateway makes Director and Authorware ODBC-compliant. This means that projects built in these multimedia authoring tools can share information via SQL with corporate databases Sybase, Oracle, and Informix as well as PC-level databases like FoxPro, dBase, and Clipper.

Taken together, FileFlex and the FileFlex/ODBC Gateway provide a full range of embedded database solutions. FileFlex itself is a complete, self-contained database engine. It reads and writes the dBASE III/xBASE/DBF database format and is ideally suited for self-contained projects like CD-ROMs and kiosks. Many widely available commercial CD-ROMs use FileFlex behind the scenes to manage data.

By contrast, the FileFlex/ODBC Gateway provides access to databases anywhere in the corporation. A typical enterprise-wide multimedia database system using ODBC requires the multimedia tool (i.e., Director or Authorware), the FileFlex/ODBC Gateway, one or more ODBC drivers, and a corporate database engine like Sybase or Oracle.

Both FileFlex and the FileFlex/ODBC Gateway are available for Windows and Macintosh. Component Software's FileFlex database engine and the former Solis product are the two primary products providing database functionality across platforms and within Macromedia's authoring environments. This acquisition makes Component the leading supplier of embedded database solutions for multimedia productions. Feel free to contact Component or check our web site at www.component-net.com to get the latest information on the FileFlex/ODBC Gateway.

2. FileFlex Release Notes

FileFlex 2.0 Release Notes

The features and functions in FileFlex 2.0 are the result of fabulous support from the engineering team at Macromedia. With Director 5, Macromedia has substantially improved the way Director supports add-in products. Since the very earliest glimmer of the new architecture, called Macromedia Open Archictecture (MOA), we've been working with Macromedia to tightly integrate FileFlex with that new architecture.

FileFlex Lite Licensed by Macromedia for Director 5

FileFlex has been licensed by Macromedia for inclusion in Director 5. Everyone who gets a copy of Director 5 will also find a special version of FileFlex on the disk. We've built a special version of FileFlex, called FileFlex Lite, for Director customers. The FileFlex Lite edition includes all the capabilities of the full, professional edition of FileFlex, except that it will only access up to the first 1,000 records of the database. FileFlex Lite users, of course, can always upgrade to the full professional edition.

Native 32-bit Versions for PowerMac, Windows 95, and Windows NT

FileFlex has been redesigned to be 32-bit native for the PowerPC processor, Windows 95, and Windows NT. Because FileFlex makes use of 32-bit architecture enhancements of the host environment, the 32-bit versions of the FileFlex engine are available only as Director 5 Xtras.

Massive Performance Improvements

The performance increase between the 16-bit version under Windows 3.1 and the 32-bit version under Windows 95 is nothing short of incredible. FileFlex has always been engineered first and foremost for speed. But by rebuilding FileFlex to run natively under high-performance environments, FileFlex is now even faster.

Very Tight Integration with Director 5

Because of the support we received from Macromedia, we've been able to very tightly integrate FileFlex into Director 5. FileFlex for Director 5 is now implemented as an Xtra. To use FileFlex, all you need to do is drop it into the Xtras folder and the list of Lingo commands is expanded to include all of FileFlex' database functions. FileFlex, especially in the "G" operator of DBGetCurrRecVal now has much better awareness of Director's native data types.

FileFlex Tool Xtras

FileFlex now includes a number of utility programs that make using the product easier. Each of these utility programs (in particular the Database Designer and Verification Suite) are available at authoring time from within Director. FileFlex adds a number of Tool Xtras available from Director's Xtras menu that launch these various utilities.

FileFlex Database Designer

The FileFlex Database Designer makes it much easier to create databases and index files, as well as browse existing database files. A simple drag-and-drop interface is all it takes to create new databases and index files.

FileFlex Verification Suite

FileFlex now includes the FileFlex Verification Test Suite. This set of tests exercises the FileFlex engine and confirms that everything works properly. It was essential during development of FileFlex and is available to you to verify FileFlex has been installed properly and for your exploration of FileFlex functionality.

All New Documentation in HTML Format

We're happy to say that we get very few complaints about FileFlex. However, the one negative we've consistently gotten was for our documentation, which was somewhat patched together from various versions. With FileFlex 2.0, that all changes. The FileFlex documentation has been totally rewritten from the ground up, into more than 23 chapters of detailed information, tips, and reference material. The new documentation is supplied in HTML format, so it is interactive and hypertext-browsable from your favorite web browser (we recommend Netscape Navigator).

The entire new FileFlex manual is also on our website at www.component-net.com and is fully web browsable. We believe we are one of the very first companies to publicly post our complete documentation for web browsing. An important note: The very latest version of the documentation (reflecting any changes we make based on user feedback) will always be available for browsing at www.component-net.com and for downloading.

Finally, for those people who don't have access to a web browser (why???), we've included a version of the manual readable in Microsoft Word. You can also purchase a printed copy of the manual directly from Component Software for a nominal fee.

New Indexing Functions

FileFlex 2.0 adds a number of new indexing functions. These include DBIndexExpr (returns the index expression used to create an index file), DBListIndexFields (returns a delimited list of fields indexed by a given index file), and DBBuildSeekExpr (returns a properly formatted seek string, ideal for complex or multi-field index searches).

More New FileFlex Functions

FileFlex 2.0 also adds DBDatabaseExists (determines if a database file exists where you expect it) and DBMaxRecs (returns the maximum number of records your database can access). In addition to these functions, FileFlex 2.0 adds the DBGetGlobal and DBSetGlobal functions, designed primarily for internal FileFlex performance testing.

Renamed DBQuery Function

There has often been confusion between the various search functions in FileFlex. In particular, there was confusion between DBLocate and DBSeek, both of which seemed to imply finding something. It turns out DBSeek is properly named because it does a "seek" (a commonly used and very precise computer science term), directly locating data in an indexed database. But DBLocate is really the query interface to the database. Since that's the case, we've renamed DBLocate to DBQuery in FileFlex 2.0. If you have an existing database using DBLocate, we've still provided the wrapper scripts, but they'll go away in the next release in favor of just DBQuery.

Compiler Versions

FileFlex 2.0 has been created using CodeWarrior 7 on the Macintosh and Visual C++ 1.5 for Windows 3.1 and Visual C++ 2.0 for Windows 95 and Windows NT.

FileFlex 1.7.1 to 1.7.4 Fix Notes

FileFlex v1.7.4 incorporates bug fixes for both Macintosh and Windows versions of software and applies to the SDK, Runtime, and Demonstration variants of the product.

FileFlex 1.7.4 Bug Fix Notes

FileFlex 1.7.4 fixes two small problems:

- Fixed bug in Mac and Windows versions where DBSum() didn't calculate the sum correctly. Modified the code to use long doubles, accounting for the possibility of rather large expressions.

- Modified KeyMaker to check for first 3 characters of version, so that incremental upgrades can still work with the KeyMaker.

FileFlex 1.7.3 Bug Fix Notes

FileFlex 1.7.3 was released as the 1.7.3b6 release of FileFlex for Macintosh and Windows. The beta release will expire in 30 days and contains only the DLL (Windows) and the XFCN (Macintosh), rather than the full product file set. The problems fixed in 1.7.3b6 have been incorporated into FileFlex 1.7.4 and above.

Details about Macintosh File Sharing resolution

I fixed a bug in the Mac version where DBUse returned a -120 (e.g., can't open file error) when file sharing was turned on and the file was locked. I changed the behavior so that now FileFlex first attempts to open the file as read/write. If any open attempt fails, it attempts to open the file in exclusive read/only mode.

Details about Windows File Access resolution

I fixed a bug in the Windows version (but also reflected back into Mac version) that generated -400 (file lock) errors when DBUse attempted to open files. Users were able to work around the lock error using SHARE.EXE, but it meant that they'd need to install SHARE. This problem was found mostly on networked Windows machines. There was also a problem with users accessing files on locked volumes. I fixed this problem by attempting to open the file first as read/write, then as read-only.

This was a very difficult bug to reproduce. Readers of fileflex-talk will recall some message traffic about both the -400 error and the locked volume error, but because this only occurred on certain configurations (mostly networks), it proved difficult to test and debug.

Note: DBCloseAll() may error out on locked files. When working on locked volumes, be sure to DBClose each individual file.

FileFlex 1.7.2 Bug Fix Notes

FileFlex for Macintosh v1.7 has a moderately serious bug. This note details the bug and describes v1.7.2, the bug fix release. Windows users will not experience this problem.

Multiple Open Database Bug Details

In FileFlex for Macintosh v1.7, customers reported problems accessing multiple open data files. Here's what's going on:

- The DBSelect() function is used to set an internal FileFlex variable that determines the database that's currently active. In FileFlex 1.7, DBSelect() correctly sets the variable.

- Various internal FileFlex functions access record data based on internal buffers. To determine which buffer to read, and the appropriate offset into the buffer, they perform a mathematical operation based on the current database variable and other factors.

- This mathematical operation worked correctly when the current database variable was zero (indicating the first open database). As a result, when you open a single database and access it's data, everything works fine.

- However, the math operation did not work correctly when computing the offset into subsequent buffers. As a result, when you attempted to read data from a second or subsequent data file, you recieved the field width information and field contents information from the first data file. Clearly, this was not proper behavior.

It is important to note that this bug does not damage the data files in any way. It just simply returns data from the wrong file buffer.

Bug Fix v1.7.2 Details

FileFlex for Macintosh v1.7.2 fixes this bug. The internal routine correctly computes the proper buffer offset, and correctly returns the width and contents of records, regardless of which data file is being accessed.

This bug does not impact Windows users. There is no other difference between v1.7 and v1.7.2 other than this bug fix.

FileFlex 1.7.1 Bug Fix Notes

FileFlex for Windows v1.7 has a moderately serious bug. This note details the bug and describes v1.7.1, the bug fix release. Macintosh users will not experience this problem.

DBCount() Bug Details

In FileFlex for Windows v1.7, customers reported a problem getting accurate results from the DBCount call for databases with more than 32,000 records. Here's what's going on:

- Calling DBCount() on a large database (say, 100,000 records) generates an internal function call.

- The internal FileFlex function correctly accesses the database and correctly returns a 32-bit data value containing the proper number of records (in this example, 100,000).

- The number of records, stored in a unsigned 32-bit integer needs to be converted to string data for the scripting language.

- The string conversion routine erroneously treats the count integer as a 16-bit value instead of a 32-bit value. It therefore chops off any bits of the count value over 16-bits, sends that to the string conversion routine, and returns that to the calling scripting language.

- If the number of records in a database is less than 32,767, then the count value fits into the 16-bit value and the DBCount() is returned correctly. If the number of records is greater, the result from DBCount() is unpredictable.

It is important to note that this bug does not damage the data files in any way. It just simply returns the count value incorrectly.

Bug Fix v1.7.1 Details

FileFlex for Windows v1.7.1 fixes this bug. The internal routine correctly receives the 32-bit count value, keeps it as an unsigned 32-bit count value when passing to the string conversion routine, and correctly returns the number of records in the database, regardless of the number of records.

This bug does not impact Macintosh users. There is no other difference between v1.7 and v1.7.1 other than this bug fix.

FileFlex 1.7 Release Notes

The features in FileFlex 1.7 are the direct result of ongoing discussions with FileFlex users. This is a great example of the value of two-way communication between software designers and users. Thanks for all the feedback!

Cross-Platform Product, New Windows Product!

FileFlex is now a cross-platform product. As in earlier releases, FileFlex on the Macintosh is implemented as an XFCN that operates in Director, HyperCard, SuperCard, Authorware, and more. The big change is that FileFlex has now also been implemented under Windows as a DLL for Windows 3.1.

Note: FileFlex for Windows v1.7 only runs under Director for Windows. Future versions of FileFlex will add support for additional development tools.

Common API

A major design criteria for the cross-platform FileFlex was transparent conversion between platforms. Users developing in either Director for Macintosh or Director for Windows and using the FileFlex API will be able to move their movies to the other platform without having to rewrite any of the FileFlex Lingo code (with the exception of file directory path specifications).

Common File Format

A related design requirement was common database files across platforms. This is especially important for hybrid CD-ROM developers who want to only have one set of database files usable by either Macintosh or Windows binaries. FileFlex DBF data files require absolutely no modification or conversion and can be read, created, and updated by both FileFlex for Windows and FileFlex for Macintosh.

Massive Architectural Improvements

To support multiple platforms, FileFlex has been heavily rearchitected, resulting in faster, more reliable performance and greater ease in porting to additional environments.

New Compiler Versions

FileFlex for Macintosh has been ported from Think C 7.0.3 to Metrowerks CodeWarrior6, providing substantial performance and reliability improvements. FileFlex for Windows has been implemented in Visual C++ 1.5 under Windows 3.1.

New Cross-Platform Functions

FileFlex 1.7 adds new cross-platform support functions. DBInitPlatform and DBClosePlatform provide a common source code initialization environment across platforms. DBPlatform returns a string describing the platform currently running. DBConvertCRLF provides text conversion support for those special instances where the line-termination needs to be customized for either Macintosh or Windows.

12-15% Overall Improvement in Speed

FileFlex, overall, is 12-15% faster across all function calls. Your scripts that make repeated, looped calls to FileFlex will see a substantial speed boost.

Better Support for Director Users

FileFlex 1.7 now provides better support for Director users. New features include a full set of descriptive FileFlex DBxxx wrapper scripts (so you no longer need to remember the direct FileFlex function code!), a Director-based FileFlex Test Bench, and a VIDEO.DIR tutorial created by a generous FileFlex customer.

Single Code Resource Module

In prior FileFlex releases (on the Macintosh), the FileFlex engine was actually delivered as a set of six code resources: an XFCN and five CODE resources. There were two major disadvantages

to this. The first was that the resources were often installed incorrectly, causing nasty problems. The second was that each call to FileFlex resulted in the loading and unloading of six resources. FileFlex 1.7 resolves this under both Macintosh and Windows. FileFlex 1.7 implements the database engine as a single resource (an XFCN on Macintosh and a DLL on Windows). This provides increased reliability and performance improvements because FileFlex 1.7 now loads 1/6 of the number of resources it did previously.

Updated Online Guide

We're keeping with plan and updating the online guide to reflect new features, answers to commonly asked questions, and new things we've learned about using FileFlex. Plus, this version adds completely new chapters. Be sure to scan the new guide carefully for information you need!

Fixed Bugs

Fixed numerous minor bugs and a rather nasty one where retrieving data from a database with the direct global option didn't deallocate a handle and thereby caused a creeping memory leak.

FileFlex 1.6 Release Notes

The features in FileFlex 1.6 are the direct result of online discussions with FileFlex users. This is a great example of the value of two-way communication between software manufacturers and users. Thanks for all the feedback!

Compiler Version

FileFlex has been ported from Think C 6.0.1 to Think C 7.0.3. This was not nearly the porting effort required to move from FileFlex 1.3.7 (written with Think C 4.0) to FileFlex 1.5 (written in Think C 6.0.1). Even so, you should know that we're very carefully watching the action in the compiler business and are considering moving to Metrowerks for future versions. Note: We've done more PowerMac testing and while FileFlex 1.6 is still not available in native mode, we've seen very good performance in emulation.

Direct Global Variable to Database Interface

FileFlex 1.6 adds a new "G" parameter to DBGetCurrRecVal and DBWriteRec. Rather than writing to/from card or background fields, FileFlex can now directly read from and write to global variables of corresponding names. This is great for both HyperCard users and other developers (especially those using WindowScript and Director).

Text Search and Retrieval from Memo Fields

FileFlex 1.6 now provides a DBFindMemo function that searches through memo fields for matching strings. For maximum speed, searching is done in physical record order.

Dynamic, On-the-Fly Encryption

FileFlex 1.6 now provides you with a whole host of ragingly fast integrated encryption capabilties. The idea behind dynamic encryption is that the data is plain-text in HyperCard (or whatever) but en-route to or from a FileFlex data file the data is encrypted and decrypted. The core encryption technology is far from "spook-proof". Encryption will prevent casual users from gaining access to data, but won't stop determined hackers from cracking the code. Because of this, FileFlex isn't subject to export restrictions. The commands DBGetMemo, DBWriteMemo, DBGetFieldByName, DBGetFieldByNum, DBGetCurrRecVal, and DBWriteRec all have new "E" (for Encrypt) or "D" (for Decrypt) optional parameters.

Standalone Encryption Functions

The FileFlex 1.6 engine now also provides standalone DBEncrypt and DBDecrypt functions that allow you encrypt any data whether or not you choose to store it in the database.

Path Specification Option added to DBUse, DBUseIndex

This feature is particularly important to runtime licensees who expect to install FileFlex data files on new volumes (for example, off a CD-ROM and onto a hard disk). In version 1.5, the runtime FileFlex decrypted the sole parameter provided by DBUse and DBUseIndex and used that decrypted string as the file and path specification to be opened. FileFlex 1.6 adds a new "pathSpec" parameter that can be provided in plain-text, allowing your program to customize the path on-the-fly during operation or installation. The runtime engine would then append the decrypted root database file name to the the end of the plain-text path and open the file. Outside of the runtime version, FileFlex simply appends the database file name to the path spec and opens the file...this is good for testing and for some clarity of design architecture. Note: We have stopped shipping 1.5 runtimes. If you expect to license a FileFlex runtime, make sure you've upgraded and tested with FileFlex 1.6.

Updated Online Guide

We're keeping with plan and updating the online guide to reflect new features, answers to commonly asked questions, and new things we've learned about using FileFlex. Plus, this version adds two completely new chapters: the FileFlex Encryption Reference and a complete list of FileFlex result codes! Be sure to scan the new guide carefully for information you need!

FileFlex 1.5 Release Notes

Welcome to FileFlex 1.5! This release has been a long time coming and are very, very grateful for the patience, support, and understanding of our customers and beta testers. There have been a lot of changes in FileFlex between version 1.3.7 (the last release) and version 1.5, not the least of which are the developers. My name is David Gewirtz and I am now the owner and developer of FileFlex. I purchased the full rights to the FileFlex product from GUI, Inc. and for the past six months have been developing the version you now have in your hands. I hope you like it. There'll be even more to come over the next months and years.

New Architecture

FileFlex has been substantially rewritten. It has been ported from Think C 4.0 to Think C 6.0.1. Floating point has been completely redesigned. FileFlex now supports Apple's SANE numerics package, which should make it work nicely on Macintosh models both with and without an FPU, as well as in emulation mode on the Power Macs. Note: We do intend to do a Power Mac version of FileFlex sometime in the future, but we are dependent on the compiler writers to deliver a stable compiler before we can begin. In any case, since FileFlex is disk intensive, not calculation intensive, you should see reasonable performance in emulation.

The Dreaded '040 Virtual Memory Requirement Is Gone

Yes, Virginia, you can now turn off Virtual Memory. Redesigning FileFlex so it no longer requires you to turn on VM when running on '040 machines has been the principle design goal of this release. You can now happily run FileFlex on Quadra's, Centri's, and '040 accellerators in any memory mode you prefer.

10% Overall Improvement in Speed

FileFlex, overall, is about 10% faster. Index files are checked faster, calculations (particular floating point-related) are faster, DBLocate is faster, and context switching is faster.

Works in Any Application that Supports XCMDs

FileFlex 1.5 and above, using the Direct XFCN Interface, will now work in any application or development tool that supports XCMDs. This means that users of tools like Director, Authorware, and others now have a method of managing data with their multimedia productions.

Native FoxPro Compatibility

FileFlex can directly exchange its files with FoxPro without any conversion required. If you follow the guidelines in this Online Guide, FileFlex and FoxPro can easily and instantly exchange files in both directions.

New FileFlex XFCN Naming Method

For historical reasons, the actual XFCN used to do all the database management has been called XDashboardDB. Since we are now supporting direct access to the XFCN, it makes sense to name it more appropriately. The new FileFlex 1.5 XFCN (and all the associated wrapper code) is called "FileFlex". To make sure this doesn't cause all sorts of problems when you first install this upgrade, we've also included an identical version of FileFlex 1.5 with the XFCN named "XDashboardDB". This can be found in the "XDashboardDB Version" folder on your distribution disk. Note: XDashboardDB naming will no longer be supported after this release or in runtime licensing, so get used to using the FileFlex named wrappers and XFCN. XDashboardDB is no longer provided in FileFlex 1.6 and above!

New Direct XFCN Interface

While a few adventurous souls have always been able to call the FileFlex XFCN's directly, it was never supported or documented. With FileFlex 15's use in non-HyperTalk environments, it becomes necessary to support directly calling the XFCN. The Direct XFCN Interface is fully documented (including design considerations and function numbers), in this Online Guide.

New 66 Page Online Guide to FileFlex

We've been listening to customer requests for improved documentation. While time considerations didn't permit us to create a new manual set for this release, we are including an all-new 66 page Online Guide to FileFlex, containing all sorts of useful information. This Online Guide is fully searchable and printable.

Improved Runtime Licensing

Prior to Version 1.5, licensing of FileFlex was a hassle. There was no guarantee that people using FileFlex resources embedded in developers' products wouldn't pull the XFCN and use it on their own. As a result, licensing was costly and restricted. With FileFlex 1.5, that has all changed. You can purchase a license to distribute FileFlex to your users for a nominal fee: $100 per product title you're distributing. This fee is the same whether you're distributing it internally, for profit in a consumer product, as a shareware product, or even giving it away. When you are granted a distribution license, you'll get a special runtime version of FileFlex that gives you all the capabilities you need, yet prevents users from extracting and re-using the FileFlex resources.

3. Interacting with FileFlex

- Wrapper Scripts
- FileFlex Has Its Own Data Files
- xBASE Standard Data File Formats
- FileFlex Data Types
- Managing Multimedia Data
- How Does "Relational" Work?
- Calling FileFlex

FileFlex is a relational database engine. It's functionality is designed to store, retrieve, search, and sort text-based information. Unlike most database engines, which are standalone applications containing other features including user-interface builders and report writers, FileFlex only manages data. Also, unlike most other database engines, FileFlex is designed to be embedded within your application, not reside as an external application called by your application or communicated with via interprocess communications.

This embedded nature of FileFlex provides a great deal of benefit to the developer of multimedia applications:

- No external drivers are required
- Nothing needs to be installed in the System folder or the Windows directory
- No other applications need to be purchased
- Exchanging data can happen at function-call speeds, rather than interprocess communications speeds (much slower)
- The memory footprint is much lower (about 150K) rather than upwards of 8MB for standalone database engines
- FileFlex can be programmed in your existing host language (i.e., Lingo, HyperTalk) rather than in a specialized database language. FileFlex simply extends your hosts language, adding more commands.

A principle design decision was that FileFlex uses each host application's "plug-in" architecture. This allows the host application to talk to FileFlex with no penalty for speed and allows FileFlex to be very tightly integrated into the programming environment of the host application. For example, in Director 5, the FileFlex engine exists as a Lingo Xtra. Just drop it into the Xtras folder and full relational database capability is available.

Wrapper Scripts

When you install FileFlex, your development environment talks with the FileFlex plug-in via "Wrapper Scripts". Wrapper scripts are very short handlers written in your host language's scripting language. Here's a sample wrapper script for Director 5, written in Lingo:

```
on DBSelect dbID
    DBCheckActive
    return FileFlex("5",string(dbID))
end DBSelect
```

In the above example, the DBSelect function is defined. You'll learn later that DBSelect switches the current database from one open database file to another. The wrapper script does two simple functions. First it calls DBCheckActive, itself a wrapper that determines if the FileFlex engine has been properly initialized. Second, it calls the FileFlex engine plug-in with the function code "5" and with an ID number contained in the variable dbID.

The reason this is done is that the FileFlex engine is actually one solid code module. The FileFlex code knows which of it's forty-some-odd functions to perform based on which code number is specified. So, if FileFlex is passed a code 3, then it performs a DBUse to open a database. If it's passed a code 30, it knows to write a record to the database.

But programming by remembering all those numerical codes is difficult and hard to debug. It's much easier to remember descriptively named commands. These are the commands in the wrapper scripts, such as DBSelect above.

There is one other reason why the wrapper scripts are so important: You can place a breakpoint inside the wrapper. In this way, you can tell exactly what data you're passing to the database engine through the function's parameters, prior to actually invoking the engine. This debugging tool can often be an invaluable way to make sure you know what you're telling FileFlex to do.

FileFlex Has Its Own Data Files

FileFlex data files (the database) are external to your project . This is in direct contrast to how Director stores data. When you build a Director movie the "normal" way, you defined various text cast members within the movie. The data itself is stored in the cast members, and as you add data, the movie size grows.

There are some amazing advantages to having the data stored separately from the controlling application.

The first, of course, is speed. Text-based searches using FileFlex'ultra-high speed indexing technology allow you to go anywhere in the database virtually instantly. Because the data is structured specifically around this search mechanism, everything is optimized for speed.

The next key advantage to using your development environment as a front-end to data is how easy it is to change "views" or "layouts" on the data file. If you've used FileMaker, you know that you can have a bunch of different layouts for the same data fields. The layout is simply the way

you look at the data. For example, you might have a purchase order system with fields such as P.O. number, vendor name and address, a list of items, expected date, and a total dollar amount. However, depending on the level of detail interest you have in the order, you might want to view the data different ways: with all the details, or just a summary of the information. FileFlex allows you to work the same way. Rather than having to switch your entire cast or stack, FileFlex lets you instantly jump between databases and sort orders without incurring any overhead of disk access.

Lastly, it is far easier to deliver updates to users of your software if the program or application is separate from the data. There's no need to try to save off the old data and read it into the new, updated project (or update all the scripts of the old movie). You just delete the old movie and replace it with the new. The data file is untouched.

xBASE Standard Data File Formats

When you create a FileFlex database, you're going to create one, two, or three different types of data files. They are:

- **DBF:** The industry standard xBASE/dBASE III data file format. This format can be read by virtually any program that can read database file formats.

- **DBT:** If you define memo fields (large, unstructured text fields), you'll create a DBT file as well as a DBF file. The xBASE definition states that memos are stored in these external files.

- **NDX:** When you create a FileFlex index, you'll create one NDX file for each sort order. NDX files are also an xBASE standard format.

The value of using these file formats is that they are extremely fast and extremely common. While they're the native format for FileFlex, virtually any database engine can read and write the xBASE standard. Even products that aren't databases, like Word 6 and Excel, can both read and export into the DBF format.

One word of caution though: the xBASE standard, to which FileFlex adheres religiously, is the dBASE III file format. If you choose to export data from another application (say Access), be sure to specify the dBASE III format. If you select native, accellerated FoxPro format or dBASE IV format, FileFlex will refuse to recognize the file.

FileFlex Data Types

FileFlex supports the following native data types:

- **Character:** Fixed-size ASCII fields, containing up to 256 characters;
- **Numeric:** Fixed-size, fixed-point numeric values;

- **Logical:** True/False values;
- **Date:** Full date information, including century values of the form YYYYMMDD
- **Memo:** Free-form, variable size text containers. Memo fields can store up to 32K.

Managing Multimedia Data

FileFlex does not internally store "blob" data like graphics files, QuickTime movies, sounds, etc. This doesn't mean you can't or shouldn't use FileFlex to manage multimedia data. Far from it. Instead, you should store the images or video clips as individual files on disk, perhaps assigning them to specified folders or directories. Then, store the location, path, or some identifying characteristic inside a FileFlex text or memo field. When you want to access the image, do a search in FileFlex, grab the path from a character field, and just import the image into Director.

There are strong advantages to this approach. First, you're able to use your native image editing tools to modify the files. Second, you can replace images individually, rather than having to reinstall a complete database. And third, as you add images, you can span disk volumes or store certain images or multimedia components easily on different parts of your network.

Finally, by not storing multimedia data within the data file, the database files are substantially faster and smaller.

How Does "Relational" Work?

FileFlex is a fully relational database system. This means you can "connect" or "relate" data together. The best way to describe how this works is with our ever-popular mail list example.

Imagine a mailing list that consists of two "databases" or .DBF files. Here are the fields for the two databases:

```
NAME AND ADDRESS DBF                KEYWORD DBF

Customer ID <----------> Customer ID
Name                                      Keyword
Company
Street
City
State
Zip
Country
```

As a software publisher, we'd use the database for two purposes: doing direct mailings to new prospects, and tracking our registered customers.

Whenever a customer buys one of our products and returns the registration card, we enter his address information into the NAD (name and address) database and the product purchased in the

keyword database. Since a customer might have purchased more than one product from us at varying times, there will probably be more than one entry in the keyword database.

Later, the customer calls us for technical support. Using the "view" that allows us to call up the customer's information on the screen, we ask the customer for his last name and zip code (which is how the data is indexed). Our program does the following:

- Selects the NAD database (DBUse);
- Selects the last+zip index and does a seek on that combination;
- It throws the appropriate address information into the appropriate card fields;
- Next, the program gets the customer ID from the customer's NAD record;
- Selects the Keyword database;
- Selects the Customer ID index and seeks on this customer's ID;
- Fills a list field with the products the customer's purchased by grabbing the keyword from subsequent records until the ID number changes.

That's one use of the data. Here's another: Let's assume we've completed an upgrade to FileFlex. We want to do a mailing to our customers to give them the option of upgrading. Here's what our program does:

- Selects the keyword database;
- Selects the index on the keyword;
- Seeks on the first occurance of the FileFlex keyword;
- Repeats through until the keyword is no longer FileFlex;

    a. Gets the customer ID from the keyword field
    b. Selects the NAD database;
    c. Selects the NAD database index for customer ID;
    d. Does a seek on customer ID;
    e. Retrieves and prints a label;

Calling FileFlex

Calling FileFlex is a simple matter of executing a function call. Here's an example that opens the database file "STARS.DBF":

```
put DBUse("STARS.DBF") into dbResult
```

As described earlier, DBUse is a wrapper that tells the FileFlex engine to open a database file. Every FileFlex function returns a value, which should be placed into a variable and then tested. In the above example, the result of the DBUse command is placed into the variable dbResult.

When you call most FileFlex functions, you should check the value returned by the program to be sure that no error occurred in the process. The mechanism by which FileFlex notifies you of

an error is simple. If the value returned by the function call is less than zero, then an error occurred. The FileFlex Result Code Reference lists all the possible error codes.

To handle errors in your scripts, then, you must simply:

- Execute a FileFlex function and place it's result into a variable;
- Check the value of this variable after each FileFlex function call;
- Respond appropriately

Here is a simple example of dealing with a potential FileFlex error condition:

```
on mouseUp
    put DBUse("VIDEO") into dbResult
    if dbResult < 0 then
        beep
        exit
    end if
    -- continues other normal processing
end mouseUp
```

This is a good general-purpose approach to the problem. However, you may wish to get more specific, as this expanded version of the above handler indicates:

```
on mouseUp
    put DBUse("VIDEO") into dbResult
    if dbResult = -120 then
        alert "Can't find VIDEO file"
        exit
        end if
    if dbResult = -200 then
    alert "Database file is bad"
    exit
    end if
    -- continue other processing
end mouseUp
```

If other errors could occur but you don't need to provide special handling for them, you can add another if-then clause to deal with those as well.

Six FileFlex functions can return positive results that indicate some special situation has arisen. When you use these functions, you should know that a return value of 1 means that the beginning of the file has been reached and a return value of 3 means the end of the file has been reached. .

- DBTop (return of 3 means file is empty)
- DBBottom (return of 3 means file is empty)
- DBSkip
- DBSeek
- DBQuery
- DBFindMemo

Here is an example of dealing with these conditions (the DBSkip call moves the pointer to the next record in the database):

```
on mouseUp
    put DBSkip(1) into dbResult
    if dbResult = 3 then
        alert "End of file reached"
        exit
    end if
    -- continues normal processing
end mouseUp
```

4. FileFlex and Director 5

- Database Access in Director
- Designing Databases for Use in Director Productions
- Using FileFlex with Director 5
- FileFlex Adds Database Commands to Lingo
- Cross Platform Considerations
- Installing FileFlex Files
- Upgrading from Earlier FileFlex Releases
- A Sample Lingo Session with FileFlex

Glenn Picher, a leading Director developer, recently wrote about FileFlex and Director for the February '96 issue of the Lingo User's Journal. The description was so clear that we asked for and got permission reprint portions of the review below:

Database Access in Director

When a multimedia production grows in scope to include a large volume of data, Director's built-in Lingo facilities for managing data can be pushed beyond their limits. Director's text cast members (in which you might store list data) have a 32K maximum on the Mac, and the FileIO XObject has problems reading in text files of more than 64K on Windows. There are also performance problems in searching large lists with Lingo. Symbol searching is fast on both the Mac and Windows platforms, but on Windows, a 64K limit on symbol data and list size effectively force you to use much slower string-based searches in smaller sublists. Windows string searches are considerably faster than on the Mac, probably due to the Intel x86 family of processors' one-opcode string instructions; but even on Windows, string searches can be slow with large volumes of information. Such Lingo-coded database schemes also can tie up a good deal of memory, which can be a problem on low-horsepower machines.

FileFlex from Component Software is a good solution that extends the volume of information possible in cross-platform Director multimedia productions, while kicking access speed into overdrive and simplifying the creation of data.

FileFlex is a database "back end" that your end users never see. You can design an interface "front end" any way you please with the Director cast members you're used to, then use Lingo to tie into the FileFlex back end for database functions. Since FileFlex uses compiled code, rather than interpreted Lingo, performance is in an altogether different league.

Rather than storing database information in Director cast members or in its own proprietary file format, FileFlex is designed to read and write standard dBASE-format files, supported by

FoxPro, FileMaker Pro, and many other database applications on both Mac and Windows. Significantly, this means your data can be created, checked and updated independently of your Director development efforts. This can free the time and brainspace of a Director-saavy Lingo programmer while someone else massages the actual database content. Thus even a client who might know nothing at all about multimedia production, but knows a database program even better than you do, can join the production team in a meaningful way .

This approach gives you the power of a full-featured database application while creating your title, and even opens the door to useful authoring-time automation such as AppleScript droplets in combination with FileMaker Pro. This can be especially helpful in converting existing databases for use in multimedia productions. Since you'd only use a full-featured database program while you're authoring, you'd only need to license FileFlex-- small, fast and relatively cheap-- for end users. FileFlex itself has a very small footprint, both on disk and while active in memory. Alternatively, you can reconfigure and update databases with FileFlex using Lingo, without any database program.

FileFlex supports over a billion records in a single database file, allows access to multiple databases, and supports multiple independent index files (for speed in searching) for each database. Practically speaking, you'll never push these limits with Director. FileFlex databases don't store pictures, sounds or digital video movies per se, but could be used to keep track of internally stored castmembers, or even externally stored files that Director can import on-the-fly using Lingo's "the filename of cast" property or "importFileInto" keyword.

The software development kit for FileFlex is licensed separately from a distributable runtime version. The only difference is a not-too-onerous copy protection encryption scheme built into the runtime version which does not affect performance in any way. The runtime version only opens database filenames that have been pre-encrypted at authoring time.

The SDK and runtimes are licensed separately for each platform. The SDK lists for $195, but can be had for $119 plus $8 shipping. Run-time licenses are $100 per title plus $8 shipping. In short, it'll cost you $470 to license FileFlex for a crossplatform project. If your client will be reselling the product, they'll need to own their own SDK and runtime licenses.

Designing Databases for Use in Director Productions

How would you design a database and use it in a Director production? The database contents are most easily created and maintained in a full-featured database program such as FoxPro or FileMaker Pro. You'd first define the structure of the database according to the needs of your multimedia presentation.

dBASE files are defined by the "fields" they contain. A field is akin to a variable name in Lingo; each contains one discrete piece of information. For example, suppose you wanted to present a timeline in a Director project. For each event in the timeline, you'd need to record the date, a short description of the event, the name of a picture to be displayed when the event is viewed, a more lengthy narrative description of the event, and perhaps cross-references to subjects

contained elsewhere in the project to which the event is relevant. Using FoxPro, you could use the Catalog Manager to define a table containing the fields "DATE", "DESCRIBE", "PICTURE", "FULLTEXT" and "SUBJECTS".

Unlike Director variables, database fields are generally defined in advance with a fixed size to represent a particular type of data, such as a date, a string or an integer. But it's also possible to use a free-form "memo" field when the size of the data isn't known in advance or varies widely. In this example, the "FULLTEXT" field would likely be a memo field, since a thorough narrative desciption of an event might take a sentence or a few pages. Memo fields can store up to 32K of text or other data.

Each event would be entered into the FoxPro table as a separate "record", which groups together data for all the defined fields of a particular event in the timeline. The table could contain any number of individual event records. The table can then be exported as a dBASE file with memo information, which actually generates two files. For example, the FoxPro "Events" table would be exported to "Events.dbf" (which contains the definition of the fields in the database, followed by all the individual event records) and to "Events.dbt" (which contains all the memo fields used in the database). These files are then used by the FileFlex externals within Director.

When your Director project first starts up, you need to set up FileFlex before actually opening any database files. In the startMovie handler, for instance, you would use the DBInitPlatform() FileFlex handler to locate and open the FileFlex engine. Once the externals are loaded and available, you initialize FileFlex with the DBOpenSession() handler. This sets up some global variables that FileFlex needs internally. Finally, you're ready to open a database.

The DBUse() handler opens a database file. Its syntax is variable. You can pass a full pathname, or a filename followed by the full pathname to its containing folder. For example, DBUse("events", the pathName) will open the .dbf database file created by FoxPro in the example above, assuming it's located in the same folder as the currently running Director movie. Since projects running from a CD or run on both platforms can't know the users' hard drive names in advance at authoring time, this two-parameter usage is much preferred. It's also important to use the two-parameter syntax for any project in which you will distribute the runtime version of FileFlex, since all filenames must encrypted at authoring time. The separate pathname needn't be encrypted.

If the database opened by DBUse() contains memo fields, its matching .dbt file is also implicitly opened. Since FileFlex allows more than one database to be open at a time, the final step is to call DBSelect(), supplying the database number returned by the DBUse() handler. Now you're ready to access records in the database.

At this point, your Director interface might display a splash screen, let the user choose where to navigate to, and ultimately end up in the timeline screen. You might provide button controls for selecting a particular year, zooming the scale of the timeline, or scrolling the timeline from left to right. Given the users' choices, you'd then want to extract events from the appropriate time period from the database and display them on the screen.

It's beyond the scope of this article to provide a detailed implementation of this timeline example. However, you might use some of the following FileFlex calls to extract information to the screen:

-- DBTop(), DBBottom(), DBGo(), DBSkip() and DBLocate() to select particular event records.

-- DBCount() to determine how many records are available

-- DBGetCurrRecVal("G") to set Lingo global variables (using the same names as the database's fields) to the values defined by the current record.

--DBFindMemo() to locate records with memo fields containing certain words.

--DBUseIndex(), DBCreateIndex(), DBSelectIndex(), or DBCheckIndex() to work with "index" files, which are a very quick way to sort and locate information. FileFlex uses .ndx formatted index files, rather than FoxPro's .idx format; however, this isn't a problem, since FileFlex can create its own .ndx files from FoxPro .dbf files.

When your Director project quits, you'll want to be sure to use DBClose() and DBClosePlatform() to free FileFlex's open files and resources.

Again, we'd like to thank Glenn and Tab Julius (publisher of the Lingo User's Journal) for permission to use the above article.

Using FileFlex with Director 5

Director has some amazing interactive media presentation capabilities. It would be an ideal front-end development tool except for one problem: it has no data storage mechanism. FileFlex solves that problem.

The Director application and multimedia production take up the vast majority of system resources. FileFlex, embedded in within the Director movie file and using only 100K, can search, sort, save, and retrieve information in industry-compatible database files external to the Director movie files.

FileFlex Adds Database Commands to Lingo

Think of FileFlex as an extension to the Lingo scripting language (this, of course, means you've got to be conversant with Lingo before you'll be able to use FileFlex). FileFlex provides no interface tools (not even alerts!), so it doesn't get in the way of your carefully crafted multimedia presentation. Instead, FileFlex provides a broad set of information storage, searching, access, encryption and management functions. FileFlex also provides ragingly-fast dynamic, on-the-fly encryption and full text search and retrieval.

The easiest way to make use of FileFlex within Director 5 is to drop the FileFlex Lingo Xtra into the Xtras folder where your Director application is located. Alternatively, you can use the OpenXLib Lingo command to provide access to the FileFlex code resources. Once you've done this, FileFlex API calls behave like any other Lingo functions.

Cross Platform Considerations

Director users can embed database capabilities in their movies and be confident they'll be usable under both Macintosh and Windows. There are only a few minor concerns you'll need to take into account when using FileFlex across platforms:

- Path specifications to your database will be different. The Macintosh uses colons (:) to separate folders, while Windows uses backslashes (\) to do the same. If you choose to open a database with a fully-realized path, you'll need to change your code as in the example below:

```
if DBPlatform() is "FFWIN" then
    put DBUse("C:\DEV\DBHOUSE\NAMES.DBF") into dbID
else
    put DBUse("Windy City:Data Warehouse:Names.dbf") into dbID
endif
```

- If you're using memo field data that comes from an external database (not entered in Director), you may need to be concerned with line-termination. The Macintosh terminates each line with a newline, and under Windows the convention is that each line is terminated with a carriage return and line feed. FileFlex v1.7 provides a DBConvertCRLF function that helps manage this inconsistency.

That's it. Otherwise, your Lingo calls to FileFlex will be the same and the data files will be the same. We designed the Windows version specifically to make cross-platform porting as effortless as possible.

Installing FileFlex Files

FileFlex is very simple to install. All you need to do is copy the files into their appropriate locations as described in the next few paragraphs.

FileFlex takes full advantage of the Director 5 Xtra interface. This means that there are optimized versions of FileFlex for Macintosh 68K, Macintosh PPC, Windows 3.1, Windows 95, and Windows NT.

FileFlex Engine Files

The following files represent the actual Lingo Engine Xtra for Director 5:

- Macintosh
  - **FileFlex Engine:** The "fat binary" version of FileFlex that can be used with PowerMacs and 68K Macintosh systems.

- Windows
  - **FFDIR5.X32:** The FileFlex Xtra for 32-bit Windows 95 and Windows NT
  - **FFDIR5.X16:** The FileFlex Lingo Xtra for 16-bit Windows 3.1

Note: Users of the bundled Lite version will have files called FileFlex Engine Xtra Lite, FFDIR5LT.X32, and FFDIR5LT.X16, respectively.

FileFlex Tool Xtra Files

FileFlex also includes a number of tool Xtras that make accessing and manipulating FileFlex data more effective. These include:

- Macintosh
  - **FileFlex About Tool:** The "fat binary" version of the FileFlex tool Xtra that launches the about dialog for use with PowerMacs and 68K Macintosh systems.
  - **FileFlex Test Bench Tool:** The "fat binary" version of the FileFlex tool Xtra that launches the FileFlex Test Bench for use with PowerMacs and 68K Macintosh systems.
  - **FileFlex Deign Tool:** The "fat binary" version of the FileFlex tool Xtra that launches the FileFlex Database Designer for use with PowerMacs and 68K Macintosh systems.
  - **FileFlex About Tool:** The "fat binary" version of the FileFlex tool Xtra that launches the FileFlex Validation Suite for use with PowerMacs and 68K Macintosh systems.

- Windows
  - **FFABOUT.X16:** The FileFlex tool Xtra that launches the about dialog for FileFlex for 16-bit Windows 3.1
  - **FFABOUT.X32:** The FileFlex tool Xtra that launches the about dialog for FileFlex for 32-bit Windows 95 and Windows NT
  - **FFBENCH.X16:** The FileFlex tool Xtra that launches the FileFlex Test Bench for 16-bit Windows 3.1
  - **FFBENCH.X32:** The FileFlex tool Xtra that launches the FileFlex Test Bench for 32-bit Windows 95 and Windows NT
  - **FFDESIGN.X16:** The FileFlex tool Xtra that launches the FileFlex Database Designer for 16-bit Windows 3.1
  - **FFDESIGN.X32:** The FileFlex tool Xtra that launches the FileFlex Database Designer for 32-bit Windows 95 and Windows NT
  - **FFVERIFY.X16:** The FileFlex tool Xtra that launches the FileFlex Validation Suite for 16-bit Windows 3.1
  - **FFVERIFY.X32:** The FileFlex tool Xtra that launches the FileFlex Validation Suite for 32-bit Windows 95 and Windows NT

FileFlex Movie Utility Files

FileFlex includes a number of Director movies that are used as utilities to FileFlex. These utilities should be launched using the tool Xtras described above, but they can also be launched by selecting Open from the Director File menu (not recommended). Each of these files has a dash ('-') in front of it's name. This is so Director 5 doesn't display the name of the movie in the Xtras menu.

- Macintosh
  - **-About FileFlex:** The FileFlex about dialog. This also displays the current version and platform supported.
  - **-Database Designer:** The FileFlex Database designer is used to create database files, create index files, and browse populated databases.
  - **-Validation Suite:** The FileFlex Validation Suite is used to verify the correct functioning of the FileFlex engine. It runs each FileFlex routine and checks to be sure it operates successfully. It's used here as part of the development process but it can also be used by you to verify that FileFlex is properly loading.
  - **-Test Bench:** This is an unsupported movie document that's useful for tinkering with FileFlex.

- Windows
  - **-FFABOU.DIR:** The FileFlex about dialog. This also displays the current version and platform supported.
  - **-FFDESIG.DIR:** The FileFlex Database designer is used to create database files, create index files, and browse populated databases.
  - **-FFVERIF.DIR:** The FileFlex Validation Suite is used to verify the correct functioning of the FileFlex engine. It runs each FileFlex routine and checks to be sure it operates successfully. It's used here as part of the development process but it can also be used by you to verify that FileFlex is properly loading.
  - **-FFBENCH.DIR:** This is an unsupported movie document that's useful for tinkering with FileFlex.

Finally, FileFlex includes a test data folder containing the VIDEO.DBF and VIDEO.DBT data files.

Note: If you want to modify these files, MAKE A COPY and place them outside the Test Data folder. The FileFlex Validation Suite relies on the files in the Test Data folder being exactly as shipped from the Factory. Also, after running the validation suite, you may wish to remove from the Test Data folder all of the various temporary DBF, DBT, and NDX files other than VIDEO.DBF and VIDEO.DBT created by the validation suite.

FileFlex Folder/File Structure--Macintosh

Here's how your files should be arranged in their folders on the Macintosh:

- The Director 5 Application

- Xtras
  - FileFlex Xtras
    - FileFlex Engine
    - FileFlex About Tool
    - FileFlex Design Tool
    - FileFlex Test Bench Tool
    - FileFlex Verification Tool
    - -About FileFlex
    - -Validation Suite
    - -Database Designer
    - -Test Bench
    - Test Data
      - VIDEO.DBF
      - VIDEO.DBT

FileFlex Folder/File Structure--Windows 3.1

Here's how your files should be arranged in their folders on Windows 3.1:

- The Director 5 Application
- XTRAS
  - FFXTRAS
    - FFDIR5.X16
    - FFABOUT.X16
    - FFDESIGN.X16
    - FFBENCH.X16
    - FFVERIFY.X16
    - -FFABOU.DIR
    - -FFVERIF.DIR
    - -FFDESIG.DIR
    - -FFBENCH.DIR
    - TESTDATA
      - VIDEO.DBF
      - VIDEO.DBT

FileFlex Folder/File Structure--Windows 95 and Windows NT

Here's how your files should be arranged in their folders on Windows 95 and Windows NT:

- The Director 5 Application
- XTRAS
  - FFXTRAS
    - FFDIR5.X32
    - FFABOUT.X32
    - FFDESIGN.X32
    - FFBENCH.X32

- FFVERIFY.X32
- -FFABOU.DIR
- -FFVERIF.DIR
- -FFDESIG.DIR
- -FFBENCH.DIR
- TESTDATA
  - VIDEO.DBF
  - VIDEO.DBT

Color Palette Considerations

The various utility movies all use the "System - Mac" 8-bit color palette and when invoked by the tool Xtras, are displayed as movies in a window (MIAW). There are some color concerns here because your movie may not (and probably won't) have the same color palettes as the utility movies. When each of the utility movies displays, it will attempt to change the palette to the "System - Mac" palette so the movie graphics look correct. When it exits, it will tell the stage to return the color palette back to it's original form. This will work fine if your movie is running. If it is stopped, however, the palette will not revert until you rewind or otherwise reset your movie to show your palette.

Upgrading from Earlier FileFlex Releases

If you're upgrading from an earlier FileFlex release, there are some things you should be aware of:

- The FileFlex Engine is now a Director Xtra, not an XFCN or a DLL. While we ship an XFCN with FileFlex (professional edition), it's solely for use by Director 4 users who can't yet migrate to Director 5 and for users of XFCN compatible environments. Likewise, the XObject DLL version of the engine is for use with Windows Director 4 users. We STRONGLY recommend migrating to Director 5.

- The FileFlex wrapper scripts have changed--for the better. They are now identical on both the Macintosh and Windows platforms, making porting between platforms even easier. Make sure you update your wrapper scripts.

- You should no longer use OpenXlib. Instead, be sure to place the engine inside the Xtras folder, preferably in the FileFlex Xtras folder or FFXtras folder as described above.

- FileFlex is no longer a code resource. If you've embedded a copy of FileFlex inside your movie file, remove it.

A Sample Lingo Session with FileFlex

What follows is a Lingo session (from the Director Message Window) showing some basic FileFlex interaction with Director:

```
--    Welcome to MacroMind Director
--
put DBOpenSession()
-- "0"
put DBUse("Video.dbf")
-- "1"
put DBSelect("1")
-- "0"
put DBListFields()
-- "12
TITLE,C,30,0
DESCRIPT,M,10,0
RATING,C,4,0
TYPE,C,10,0
DATE_ARRIV,D,8,0
AVAILABLE,L,1,0
COST_RENT,N,6,2
COST_BUY,N,6,2
TIMES_RENT,N,5,0
NUM_SOLD,N,5,0
FORMAT,C,3,0
LENGTH,N,3,0
"
--
put DBGetCurrRecVal("L ")
-- "1
N
TITLE,C,AMADEUS
DESCRIPT,M
RATING,C,PG
TYPE,C,Drama
DATE_ARRIV,D,19871225
AVAILABLE,L,F
COST_RENT,N,1.00
COST_BUY,N,1.00
TIMES_RENT,N,245
NUM_SOLD,N,125
FORMAT,C,1B
LENGTH,N,120
"
put DBGo("15")
-- "0"
put DBGetCurrRecVal("L ")
-- "15
N
TITLE,C,THE FOG
DESCRIPT,M
RATING,C,R
TYPE,C,Suspense
DATE_ARRIV,D,19840831
```

```
AVAILABLE,L,T
COST_RENT,N,2.99
COST_BUY,N,29.95
TIMES_RENT,N,4478
NUM_SOLD,N,2348
FORMAT,C,VB
LENGTH,N,94
"
```

Of course, you'll want to embed your Lingo code within buttons and pretty screens.

FileFlex is an amazingly versatile tool. Combine it with the visual power of Director and there's no limit to the cool projects you'll be able to create.

5. Using FileFlex with FoxPro

- Embedded Database for Multimedia
- Transparent Access to PC Data
- Programming in the Language of Your Choice
- FoxPro <-> FileFlex DBF File Compatibility
- FoxPro <-> FileFlex Memo File Compatibility
- Text-Only Memo Fields
- Create Your Databases in FileFlex
- SET COMPATIBLE FOXPLUS
- Converting to FileFlex Memo Files
- Index File Compatibility

One of FileFlex' most valuable features is it's compatibility with FoxPro from Microsoft. The pairing of these two products can give you some amazingly powerful capabilities.

FoxPro is an impressively full-featured relational database system. It has code-compatible counterparts available on Macintosh, DOS and Windows, and of course, is supported by the huge Microsoft development, sales, and support organization. FoxPro provides all the capabilities you'd ever want in a relational database product including interface development, reporting, multiuser capability, and a library of thousands of applications built using it's own dBASE-derived programming language. FoxPro even supports AppleEvents, allowing you to create AppleScript scripts to drive interaction with the database and other Macintosh applications.

This awesome power, however, comes at a considerable price. FoxPro is distributed on eight high-density floppy disks which, when loaded on your computer, use up to 23 megabytes of disk storage. The program itself takes over 3 megabytes of disk space and needs a minimum of a 5 megabyte partition to run.

FileFlex and FoxPro are studies in contrast. While FoxPro takes an enormous amount of disk space (up to 23 megabytes), FileFlex takes about140K. While FoxPro requires a 5 megabyte RAM partition, the FileFlex embedded database takes about 100K of RAM.

Of course, FileFlex doesn't provide nearly the capabilities that FoxPro does. FileFlex doesn't have a report writer, it doesn't have an interface builder (in fact, FileFlex has no interface features of any kind), and it doesn't have all advanced database features that FoxPro has.

Yet FileFlex plays a very important role in your development strategy. That's because you can embed full relational database functionality inside your applications with a RAM cost of about 100K. You don't have to worry whether your users have FoxPro, nor do you need to worry about whether they've installed all 23 megabytes or have all that extra RAM to let it run.

Fundamentally, FileFlex allows you to add relational database capabilities to your application at virtually no cost in system resources and without any special awareness on the part of your users. FoxPro compatibility opens huge vistas of capabilities to your application. That's because FoxPro becomes an add-on Power Tool for your application. Let me give you a few examples:

Embedded Database for Multimedia

You've developed an information kiosk for a local bank in Director. Users can interact with the kiosk, getting high-quality multimedia presentations on the bank's services. They can also enter information about themselves and the services they're most interested in.

The Director application and your production take up the vast majority of system resources. FileFlex, embedded in your Director production and using only 100K, captures customer information and saves it in database files on the kiosk's local hard disk.

Later, some bank executive decides he wants to see a complex report about the customer's who've used the kiosk - and he wants to see it weekly. Clearly not a job for FileFlex but a very easy job for FoxPro. On a separate computer (it might even be an analyst's Windows or DOS machine!), a quick report is built in FoxPro that uses the data saved off in FileFlex. Instant report. No data conversion. No hassles. The full power of the high-end FoxPro relational database where it's needed and the streamlined behind- the-scenes capabilities of FileFlex where it is need.

Transparent Access to PC Data

You've built a wonderful internal office management system in Authorware. It does everything you need and fits everyone's needs perfectly. That's because it's been developed and refined regularly for the past four years.

But now you want a new feature. You want to subscribe to a service that provides all the names, addresses, and phones of everyone in the US - on a CD-ROM and updated monthly. Unfortunately, the CD-ROMs are in dBASE format, intended to be used with a FoxPro database running on a DOS PC.

With FileFlex, you're able to access that data from within your custom application - without being forced to recode it to run completely in FoxPro.

There's also a reverse benefit here as well. You sell your own "house" mailing list through a list broker. Ever two or three months, you send them an updated list of names. But - you guessed it! - they're a PC house and they need data readable by FoxPro. No problem. Use FileFlex to maintain the names inside your Authorware application. When it's time to send the mail list broker a new list, just copy the file and send it off to them. No format conversion.

Programming in the Language of Your Choice

Many people don't like programming in the old dBASE programming language. Modern languages and tools are often object-oriented or targetted to accomplishing specific sorts of tasks (like Lingo for creating multimedia). Even if you need to generate data for use by FoxPro or other dBASE systems, why should you be forced to program in the old dBASE or FoxPro language?

With FileFlex you're not forced to use a language you don't like. Use FileFlex within the development language of your choice and you'll still create files in the format you need.

In each of these examples, your application was in use. FileFlex was behind the scenes and totally invisible to your users. FoxPro was available to help with support and analysis, but it wasn't an integral part of your application. Your application retained its full identity, yet got access to a powerful new tool for support and analysis.

That's the power of the cross-compatibility between FileFlex and FoxPro.

FoxPro <-> FileFlex DBF File Compatibility

Both FoxPro and FileFlex use the industry standard DBF file format (also called xBASE format) as their native file formats.

Both FileFlex and FoxPro can read each other's DBF files without any special conversion or transformation.

There are some minor exceptions. FileFlex does not support the following FoxPro field data types:

- Picture
- General
- Floating (although FileFlex does support a fixed-point numeric format that interchanges with FoxPro)

When you design your database, make sure you avoid these field types if you want full data interchange between FoxPro and FileFlex.

FoxPro <-> FileFlex Memo File Compatibility

Both FoxPro and FileFlex have memo fields. These are large, non-indexed fields containing up to 32K of free text. In both products, memo fields are stored in files separate from the DBF file. However, FoxPro's native memo field file format is called FPT, while FileFlex's is the more

industry standard DBT format. The DBT format is used in dBASE, xBASE, and even in FoxBase+, FoxPro's predecessor.

FoxPro can read and write FileFlex DBT files. FileFlex cannot read and write FoxPro FPT files.

There are a couple of simple steps to make sure your memo files can be interchanged between FoxPro and FileFlex.

Text-Only Memo Fields

Neither FoxBase nor FileFlex can support binary information stored within memo fields. If you want to be sure of reliable data interchange, don't store binary data in memo fields.

Create Your Databases in FileFlex

Whenever possible, create the database file using FileFlex. If you define a memo field, FileFlex will automatically create the proper DBT file.

SET COMPATIBLE FOXPLUS

When in FoxPro, before you access a FileFlex file, use the SET COMPATIBLE command. To make sure you're staying compatible with FileFlex data, issue the following command in your FoxPro application or type it into the command window:

```
SET COMPATIBLE FOXPLUS
```

Converting to FileFlex Memo Files

When you create a database within FoxPro, it automatically creates the FPT format memo files, even if you've used SET COMPATIBLE FOXPLUS. But you can easily export your FoxPro-created file to a FileFlex compatible version. Once you've exported it to the FileFlex DBT format version, you can then read and write that format from within both FoxPro and FileFlex.

To convert a FoxPro table containing memo fields from FTP format to DBT format, you'll need to execute the COPY TO FoxPro command with TYPE FOXPLUS.

Note: FoxPro also offers a fancy Copy To dialog from the menu. For reasons known only to Microsoft, they left the FOXPLUS format option off the Copy To dialog. You'll need to execute it as a text command.

Index File Compatibility

FileFlex uses the xBASE standard NDX format. FoxPro does not. It uses it's own proprietary index format. If you change data in one program and then use it in another, you may want to make sure you check and update indexes.

There is a perverse twist on this concept:

Often databases are distributed in xBASE compatible format. One example of this is US Department of Census data distributed on CD-ROM. The data comes on the CD-ROM, as do the pre-built NDX format index files.

If you are a FoxPro user and you wanted to do an indexed lookup, it would be impossible. You'd have to completely rebuild the entire CD-ROM's index.

But FoxPro is one of those programs that will run XCMDs. So, you can load FoxPro with the FileFlex engine...(I told you it was a bit perverse!)...and use the FileFlex DBUseIndex, DBSelectIndex, DBSeek, and finally DBCurrRecNum functions to open the NDX index, search out the data needed by the FoxPro program, and return the record number back to the FoxPro program.

Strange, but true.

6. Using FileFlex in XFCN Environments

- The FileFlex "Wrapper" Mechanism
- General Compatibility Issues
- Using FileFlex with Director 4
- Using FileFlex with HyperCard
- Using FileFlex with Authorware Professional
- Using FileFlex with UserLand Frontier/Aretha
- Using FileFlex with SuperCard 2.x
- Installing XFCN-Based FileFlex

Besides working in Director, the professional version of FileFlex will work with any application that allows you to run XFCNs (on the Macintosh) and Director 4 (on both Macintosh and Windows). There are thirty or more of these applications, including Director 4, Authorware, Frontier, SuperCard and many others.

One of the design goals for FileFlex is that the data set you use (the FileFlex database files) and the instructions you use to control access to the data (the DBxxx instructions) are identical, regardless of which development environment you're in. Most users will generally work in one environment, but there are times that it's necessary to use one development tool for one purpose and another for another purpose. In these instances, FileFlex can come in very handy to make sure you've got common data and don't need to learn another API.

Because each of these applications are very different, their development environments and development requirements vary greatly. As a result, we can't provide you with detailed instructions for each tool. What we can do is provide general techniques and background for using FileFlex in pretty-much any XCMD-compatible application.

The FileFlex "Wrapper" Mechanism

In most environments, when you make a call to FileFlex, you're really executing a script "wrapper" that then calls the actual XFCN code resource. Here's a sample of one such wrapper:

```
Function DBSeek seekExpr
    DBCheckActive
    return FileFlex (13, seekExpr)
End DBSeek
```

In the above wrapper, the actual XFCN call is "FileFlex (13, seekExpr)"; the other lines define the DBSeek function and execute the HyperTalk handler DBCheckActive.

So, if there's only a few lines around the XFCN call, why use wrappers at all?

There are three reasons:

- The wrapper mechanism allows us to define a bunch of easily understandable calls (i.e., DBUse, DBSeek, DBSkip, DBGo) to one XFCN resource. The alternative (which we'll explore below) would have been to require you to memorize the function codes (i.e., 13 for DBSeek).

- The wrappers allow us to add special features without changing the XFCN. In particular, we added DBCheckActive, which makes sure that the FileFlex session has been opened (through use of DBOpenSession).

- The wrapper mechanism allows you to check the parameters you're passing to the XFCN. The number one problem coding FileFlex routines occurs when the developer passes the wrong parameters to the database function. By installing a wrapper, you can put a breakpoint and drop into the debugger just inside the wrapper and before the function call. Way useful.

Wrappers are wonderful tools. But....

...wrappers can't be used if you're developing something in one of the thirty or so products that support XFCNs but we have not supplied wrappers for (e.g., Authorware, UserLand Frontier, Software Ventures Microphone, etc.)

If you use these functions, you'll have to call the XFCN directly. The direct XFCN calls are described in detail in the Direct FileFlex API Interface.

General Compatibility Issues

FileFlex will work well with almost all XCMD-compatible environments. The XCMD standard originated in HyperCard and has since been adopted by a number of other tool vendors. There are a few things to keep in mind, however.

FileFlex requires the GetGlobal and SetGlobal HyperCard XCMD callback to be implemented. This is because FileFlex demands access to two specific globals, **gdbglobalptr1030** and **gdberrorflag**. This is normally not a problem with most tools that provide XCMD support. But if you encounter inconsistent behavior in your development tool, check with the vendor and make sure GetGlobal and SetGlobal callbacks are supported.

If you want to get or send data to FileFlex-and you're not using something that maps field information directly (like HyperCard or SuperCard)-you'll need to avoid the "B" and "C"

parameters. Bring the card information into variables using DBGetFieldByName or DBGetFieldByNum. You can also bring the entire record into a variable with DBGetCurrRecVal("L")-or any non-"B" or non-"C" parameter. Then parse the variable data down into usable chunks.

If your environment supports the SetGlobal and GetGlobal callbacks (most do), you may want to send and retrieve data from global variables using the "G" option. Make sure you understand how your environment deals with globals so you avoid surprises.

Using FileFlex with Director 4

FileFlex is no longer supported for use with Director 4. If you really need it, consider using FileFlex v1.7, which is available for download from the Component web site. We strongly recommend you upgrade to Director 5. Please note that while Director 5 supports the XFCN interface, we've taken special pains to develop high-performance Xtra interfaces to Director and will not be supporting using the XFCN version of FileFlex with Director 5.

Using FileFlex with HyperCard

In HyperCard, FileFlex provides a mechanism through DBWriteRec and DBGetCurrRecVal that provides a one-for-one mapping between like named fields in FileFlex files and HyperCard stacks. To access this feature, you'd use the "B" (for background) or "C" (for card) parameters to tell FileFlex where to send the data.

The professional edition of FileFlex ships with a test bench for use in HyperCard, as well as a "goodies" stack with additional tools.

Using FileFlex with Authorware Professional

In Authorware, it seems that globals retain their value (i.e., they're static). This means the value of gDBActive1030 will not reset to empty when you exit your Authorware production. You must use extra care to set gDBActive1030 to empty when you leave or start your production.

We are working towards a well-supported version of FileFlex for both Authorware Windows and Authorware Macintosh. Be sure to subscribe to the FileFlex listserv and visit the website for announcement details.

Using FileFlex with UserLand Frontier/Aretha

Frontier, from version 2.0 on, supports XCMDs and XFCNs. Frontier treats everything as a function returning a value. FileFlex fits very well into this model. In fact, it's pretty neat to see the Frontier object database interact with the FileFlex databases.

Frontier ships with a special droplet that installs XCMDs and XFCNs directly into the root file. Make sure when you install the FileFlex XCFN that the code resources also follow along. Frontier data elements in the object database are static. As in Authorware, you'll need to be sure to set gDBActive1030 to empty explicitly when leaving Frontier.

Using FileFlex with SuperCard 2.x

FileFlex works very nicely with SuperCard 1.6 and above (we recommend you use SuperCard 2.x and up). Because SuperCard is so much like HyperCard, you can even use the wrapper mechanism and the "B" and "C" parameters to DBGetCurrRecVal and DBWriteRec.

Open the FileFlex stack and copy the stack script into your project script. This will give you access to the wrappers that are so valuable for development and debugging.

SuperCard uses a unique mechanism of storing most resource information in the data fork. Although the FileFlex XFCN can be converted to the data fork format, it expects to find its supporting code resources in the resource fork. Make sure that you have access to the code resources in the resource fork before attempting to run FileFlex from within your SuperCard project.

Since FileFlex consists of an XFCN resource and five CODE resources, things can get confusing. When the XFCN loads, it expects to look into the resource fork for the CODE resources. To make FileFlex function properly, call openResources with the appropriate parameters in your openProject handler and closeResources in the closeProject handler. (Thanks to Andrew Meit for researching this for us!)

Also remember that your resources and data fork quasi-resources will live in different places in standalone projects and editable projects. When you create your own standalone, make sure you've grabbed the appropriate code resources from the SuperCard sharedfile.

Installing XFCN-Based FileFlex

There is no formal installer program for FileFlex. Just copy the files off the distribution disk to your desired location on your hard drive volume of choice.

Referencing FileFlex as an External Function

The very best way to use FileFlex is via your development tool's equivalent to HyperCard's "start using" command. This allows you to call on FileFlex' resources without needing to embed either the wrapper scripts nor the XFCNs within your application-making it MUCH easier to update. Many other development tools have a similar external library reference function (in Director, it's called "openXLib", in SuperCard, it's "open resources").

Installing the XFCN Resources

If you must install FileFlex into your project, you'll need to be sure to install the FileFlex XFCN into your project. This is done using ResEdit or some other resource editor. If you don't have (or don't know how to use) ResEdit, you should use the "start using" method. Just open up the FileFlex file in the resource editor, look for the single FileFlex XFCN, and copy it into your project.

Note: Since version 1.7, FileFlex does not require 'CODE' resources!

Installing the Wrapper Scripts

We've provided wrapper scripts to help you call the FileFlex externals. For HyperCard and SuperCard users, the wrapper scripts are stored in the stack script of the FileFlex stack.

Upgrading FileFlex in an Existing Project

We're hoping that most people using FileFlex are using the "start using" method of accessing the functions. If so, upgrading is a simple matter of deleting the old FileFlex stack and calling on the new one.

Here's what you need to do to clear out the old FileFlex and use the new one (or if you wish to install FileFlex in files of your choosing):

- Make sure you make a backup of your original files.
- (This step is appropriate only if you're upgrading an old FileFlex prior to v1.7) Open ResEdit (all good programmers should have ResEdit!) and open the stack or file containing the original FileFlex resources. Delete both the XDashboardDB XFCN and/or the FileFlex XFCN and the four 'CODE' resources (for version 1.3.x) or the five 'CODE' resources (for version 1.5). Keep this file open. Say goodbye to 'CODE' resources. I think they're finally gone for good!
- (From here on, the instructions are valid if you're either upgrading or installing FileFlex for the first time). Using ResEdit, open the new FileFlex 1.7 file. Copy the FileFlex 1.7 XFCN from the FileFlex file to your original project file (whether a Director movie, HyperCard stack, or whatever). Close and save your original file. Exit ResEdit.
- Install the wrapper scripts
- Open HyperCard and open the FileFlex 1.7 stack script. Copy the entire contents of the stack script to the clipboard. The stack script has been substantially changed since version 1.6. If you've hacked the scripts, you'll need to look carefully at the changes.
- Close the scripts
- Close the FileFlex 1.7 stack script.
- Install the scripts into your project
- Go to your original stack (the one containing the earlier FileFlex wrapper scripts). Select and delete the original FileFlex wrapper scripts. Paste in the new FileFlex 1.7 wrapper scripts. Close and save your script.

That should do it. Have fun!

7. Initiating FileFlex Operations

FileFlex operates as a self-contained little world within the host environment. Each time you make a call to FileFlex, you're rapidly loading and unloading the FileFlex engine. But since FileFlex needs to maintain an awareness of all the database information between calls (when, normally, it would be completely out of the picture), FileFlex maintains it's own internal memory structures and data.

FileFlex also resides as an external code resource to your host environment. Before you can give FileFlex commands, you need to be able to establish communications with FileFlex. In most development environments, this is accomplished by opening a resource or opening an external library. In Director 5, this can be accomplished as easily as dropping the FileFlex Lingo Xtra into the Xtras folder (see the Installing FileFlex Files).

We've defined a special standard function for establishing this communication with FileFlex called DBInitPlatform. This function tells your host environment where to find FileFlex and how it should be called. It often requires you to write or customize a few lines in your native environment's language to open the resource. Details on defining DBInitPlatform are described in the FileFlex API Reference.

Once you've established communications, FileFlex must be initialized the first time it's used and "turned off" when you're finished. When you tell FileFlex to initialize itself, it allocates all the memory data structures it'll need for your database session. You'll need to call DBOpenSession for this to happen.

Warning: FileFlex is NOT reentrant. You may only call DBOpenSession once. Then, you can't call it again unless you've turned FileFlex off by calling DBCloseSession. Attempting to do repeated DBOpenSession or DBCloseSession calls without it's mate will result in very bad things happening.

After you are done using FileFlex, you should clean up your environment by releasing memory used for buffers. To do this, put the following call into the handler called when exiting your movie or project. (It may be placed elsewhere, but this is the most logical place for its use.)
put DBCloseSession() into dbResult This will free all the data structures in use by the FileFlex engine. Failure to do this may cause all sorts of unpleasant problems.

Finally, call DBClosePlatform to deallocate the external resource and turn off communications to FileFlex.

8. Creating Databases

- Using the FileFlex Database Designer
- Executing the DBCreate Function
- Describing the Database

The easiest way to create a new database is to use the click and drag interface of the FileFlex Database Designer. Just launch the FileFlex Database Designer (from Director 5, you'll just select it under the tools Xtra menu).

You can also program database creation. To make it possible for your project to create new database files, you must:

- Build a field to contain the database description as described below.
- Execute the DBCreate function and pass it the appropriate information.

We'll discuss these tasks in reverse order because that's the easiest way to understand them.

Runtime Note: When you purchase the FileFlex runtime, the DBCreate and DBCreateIndex functions are disabled. If you purchase a license to distribute FileFlex with your application, you may not include the DBCreate function in your product. (Our lawyers made us say what follows) Failure to adhere to this requirement voids your license and may result in civil and/or criminal liability for copyright infringement or other violations of law.

Using the FileFlex Database Designer

New in 2.0: FileFlex now includes an interactive tool for creating database files, creating index files, and viewing existing database files. This tool is called the FileFlex Database Designer. You can invoke the FileFlex Database Designer in two ways. If you're using Director 5, just select it from under the tools Xtra menu in Director. If you're not using Director 5, the FileFlex Database Designer is available as a double-clickable application. In either case, once launched, the interface is the same.

The Database Designer interface has three tabs: Create, Index, and Browse. To create a new database, click the Create tab. Likewise, to create an index, click the Index tab and to browse a database, click the Browse tab.

Creating a new database is a matter of simply dragging fields from the field buckets located along the bottom of the Create page into the gray field on the left side of the screen. FileFlex supports five basic data types, each is represented by a bucket.

When you click and drag in the text bucket, a panel will show up by your cursor. Drag that panel into the gray area and release. A dialog box will be displayed that will let you enter the field name, switch between field types, and (if appropriate) select the width of the field.

Keep this up until you've added all the fields you want. If you don't like a given field, just click, hold the mouse, and drag the field panel to the trash. When you're done, just click Make It and a database file will be created.

To browse an already created (and populated) database, select the Browse tab. Open a database file by clicking the button marked Open a Database. Click the left and right arrows to move between records and the up and down arrows to scroll the list of fields.

Executing the DBCreate Function

The DBCreate function requires four parameters. In order, they are:

- the name of the database you wish to create (with optional path name and no file suffix)
- the number of fields in the database
- the name of the container in which the description of the fields can be found
- a Boolean ("true" or "false") that tells FileFlex what to do if it finds a database file of the same name as it attempts to create the new one

All of these parameters are self-explanatory except the last. If the fourth argument passed to the DBCreate function is "true," and FileFlex finds an existing file, it will not overwrite it. Instead, it will abort the database creation process and return an error. If the fourth argument is "false," it will overwrite any existing database file of the same name. (Note that FileFlex appends a suffix, ".DBF," to the name you provide, so it won't overwrite a text or other file of the same name, only a file whose name matches the name you provide as a first argument, with the ".DBF" appended.)

After the database is created, it is automatically opened by FileFlex and is ready to receive data. If you don't want to work with the file immediately, be sure to close it.

If you include any memo ("M") fields, FileFlex will automatically generate a matching file with an extension of ".DBT" to hold the memo field information.

Describing the Database

The container in which you store the description of the database file to be created should be a field, a variable, or any other easily accessible container. The container should contain one line for each field to be created. Each line should contain, in order, these items, separated by commas:

- name of the field
- type of field

- length of field, for string and numeric fields
- number of decimal places for numeric fields

Remember the xBASE field naming rules when naming fields in your database file: keep the names under ten characters, do not include anything but letters, numbers, and the underscore character. If you are unfamiliar with these rules, consult a good book on xBASE programming. The field type will be one of these characters: C (for a character, or string, field), N (for a numeric field), M (for a memo field), L (for a logical, or Boolean, field), or D (for a date field).

Character and numeric require lengths to be supplied. Only numeric fields require the fourth element. For memo, Boolean, and date fields, no length information is required. Supplying it may produce unexpected results in the database file.

Here is an example of a database file description that might appear in a container:

```
NAME,C,24
STREET,C,24
AGE,N,2,0
MARRIED,L
DOB,D
COMMENTS,M
```

Executing DBCreate with this description would produce a database file with six fields: two 24-character string fields; a two-digit, no-decimals age field; a logical field called MARRIED; a date field holding the date of birth and called DOB; and a memo field named COMMENTS.

Example

If we wanted to create a database file called FRIENDS.DBF in the current default path using the fields in the previous section which we had stored in a card field called dbFields, this is how we would write the function call:

```
put dbCreate ("FRIENDS",6,cd fld dbFields,false) into dbResult
```

9. Managing Data Files

- Opening and Selecting Database Files
- Default File Paths
- Navigating in the Database

Opening and Selecting Database Files

To open a database file, call the FileFlex function DBUse, supplying a database file name (including full path name if required) as an argument. Put the result into a variable, because you will need to refer to this file by its returned value later. Be sure not to use the same variable for any two files you to have open at one time.

Just type the following line into your script, changing the name of the database file appropriately.

```
put DBUse("HD:DB Files:Test.DBF") into databaseID
```

Alternatively, you can split the path specification into two parameters: the first the actual file name and the second the path specification. In the runtime, the path specification will remain unchanged but the database file name will be decrypted:

```
put DBUse("Test.DBF","HD:DB Files:") into dbID -- Developer FF
put DBUse("&#R*$*@U","HD:DB Files:") into dbID -- Runtime FF
put DBUse("Test.DBF","C:\DB\DBFiles\") into dbID -- Windows
```

Only one database file can be the current file. You may have numerous database files open at once, but only one of them will be the current database at any one time. The DBSelect function will allow you to choose any open database to act as the current database. To choose a previously opened database as the current file, use the FileFlex function DBSelect. It requires a single argument, the variable into which you put the ID of the database when you opened it with your call to DBUse.

```
put DBSelect(DatabaseID) into dbResult
```

Just as there can only be one current database file, so there can only be one record in that file that is recognized by FileFlex as the "current" record. All operations are performed on or relative to this record. When FileFlex switches between databases, the current record is saved.

There are times when you might want to know the database ID of the currently selected database. For example, you may want to interrupt processing on a file to undertake some special processing on another file and then return to processing the original file. To do so, use the FileFlex function DBCurrDBNum. It returns the ID of the currently selected database. This ID is the same as that returned when you called DBUse for this database.

```
put DBCurrDBNum()
```

When you are finished with a database, you can close the file and reclaim the memory used for its buffers by calling the FileFlex function DBClose and passing the database ID returned by the call to DBUse as an argument. DBClose returns an error code of 0 on successful completion, or an alternative error code if there is a problem.

```
put DBClose(DatabaseID) into dbResult
```

If you have more than one file open and want to close a specific open file, you must first insure that the file you want to close is the current file. Use DBSelect for this purpose. If you have more than one database file open and you are ready to end your work with FileFlex (or with that set of files), you can close them all in one step with the DBCloseAll function. It takes no argument.

```
put DBCloseAll() into dbResult
```

Default File Paths

When using FileFlex, it's often easy to confuse the default working directory with the directory where your files are located. Here's the typical manifestation of the problem:

"I've been trying to open the sample video database file included with FileFlex. When I try opening the file, FileFlex returns a -120 error. What's happening?"

Confusion is understandable. Imagine you've got a script of this form:

```
on startMovie
    put DBOpenSession() into dbResult
    put DBUse("Video") into videoID
end startMovie
```

First off, you should always check the results of DBUse. In this case, DBUse will return an error unless Video.DBF is located in the same folder as your Director application. While it might seem logical that Director and FileFlex would look in the directory where your movie is located, the concept of a current working directory is based on the folder where the application (in this case Director) resides.

To fix the problem, either move your database files to the same folder as Director, or prepend the complete path name (i.e., "Windy City:FileFlex Projects:Video Project:") before the name "Video.DBF". Obviously, Windows users will use backslash (\) instead of colon (:).

Navigating in the Database

To count the number of records in the current database file, you can use the FileFlex DBCount function. Since it operates only on the current database, you must first call the DBSelect function

to make the desired database current if it is not already. The first line below makes the selected database current. The second line will then return the number of records in the database file.

```
put DBSelect(databaseID) into dbResult
put DBCount() into numRecs
```

To find the record number of the current record in the current database file, you can use the FileFlex DBCurrRecNum function. This routine will return the current record number. This routine returns the physical record number. If the logical record order has been changed with DBQuery or indexing, moving one record forward or backward in the database won't correspond directly to adding one or subtracting one from the physical record number.

```
put DBCurrRecNum() into CurrentRecordNumber
```

You can position the current record pointer at any physical record, even with a currently active index, by using the FileFlex DBGo function. It takes one argument, the record number to which you wish to be positioned. It does not retrieve any data.

```
put DBGo(39) into dbResult
```

The above line will move the current record pointer to physical record 39. A "physical" record is usually different from a "logical" record, which refers to the record's number in indexed, or sorted, sequence.

You may sometimes want to get to the top of the database file or to the end of the file to find out the last record number so you can add a new record. FileFlex supplies two functions for these purposes.

To get to the top of the current database file and retrieve the value of the first record in the file, use the DBTop function. This will position the current record pointer to the top record in the database (physical record or, if an index is open, the first record in index order). You can then use DBGetCurrRecVal to retrieve the data in this record into fields or a container as desired.

Type the following lines, changing the name of the container as appropriate, to have FileFlex move to the first record in the current database and put that record's field values into a named container:

```
put DBTop() into dbResult
put DBGetCurrRecVal("A") into contents
```

If you don't know the number of the record you want to access, but you know its position relative to the current one, use the DBSkip function. You can move the current database record pointer forward or backward a specified number of records with the FileFlex DBSkip function. You must supply an integer (positive or negative) describing the number of records to skip. A negative number tells FileFlex to move backward in the file. When it reaches the indicated record, FileFlex leaves the record pointer positioned at the record. You can then use the DBGetCurrRecVal function to retrieve the fields in that record as desired.

```
put DBSkip(23) into skipResult
put DBGetCurrRecVal("A") into dbResult
```

The behavior of DBSkip depends on whether there is an index file open or whether a DBQuery operation is currently active. If an index file is currently in use, DBSkip will skip the number of records in the sequence in which the index sorts the file. If a DBQuery function has been called with an argument other than an empty string, then DBSkip will skip records matching the search criteria defined in the DBQuery function.

10. Reading Data from a Database File

- Retrieving a Field by Name or Number
- Reading a Record into a Container
- Retrieving a Memo Field
- Retrieving a Record into Global Variables
- Reading a Record into a Card

There are several ways you can retrieve information from a database file. These are discussed in the following order:

- a field at a time, by name or number
- a record at a time into a container, (which also requires a separate step to retrieve any memo fields)
- a complete record into global variables
- a record (complete or partial) at a time into a card (HyperCard/SuperCard only)

Remember that FileFlex acts as a "back-end" to your application or multimedia production. As a result, you're going to be dealing with fields of different sorts. FileFlex itself is organized into records and fields, with each field being the smallest chunk of addressable data. Most development environments also have their own internal storage areas, also often called fields. For example, in Director, a text cast member (also referenced as a field) and in HyperCard, there are card and background fields.

Your development environment actually displays it's own fields, but does not display FileFlex fields. As a result, retrieving data is an effort of moving data from FileFlex fields to your host application's fields. Often this takes place via an intermediate step of retrieving the data from a FileFlex field into a variable, processing the variable data, and then placing it appropriately in the host application's display field.

Retrieving a Field by Name or Number

The simplest method of data retrieval in FileFlex is to read a specific field's contents into a variable or container. The contents of a field in the current record may be found using the field's name or number directly, and the FileFlex functions DBGetFieldByName and DBGetFieldByNum. The first requires the name of a database field as an argument; the second requires a number corresponding to a database field number in the current file.

Both of these functions return the specified field contents into the container named in the put command associated with the function call:

```
put DBGetFieldByName("NAME") into field "Employee"
put DBGetFieldByNum(5) into myCount
```

This is an easy way to retrieve data, but it might not be the fastest. If you use either DBGetFieldByName or DBGetFieldByNum, you'll be retrieving one field at a time, for each record. Alternatively, if you use one of the functions described below, you'll be able to get the entire record at a time. A good rule of thumb is to use DBGetFieldByName if you just need one or two fields and you want to explicitly place the value of FileFlex fields into a specific container.

Note: The function DBGetFieldByNum has been part of FileFlex since it was released. It remains for backward compatibility. However, we don't recommend you use it for data retrieval because the potential for an error by misassigning the field number is pretty measurable.

Reading a Record into a Container

The next method of FileFlex data retrieval reads the entire current record (except for memo fields, which are discussed later) into a designated variable or field. You may then use "chunking expressions" to separate these contents into elements for use in your scripts.

You will use the same FileFlex function, DBGetCurrRecVal, to retrieve the entire record's contents, whether you are placing these contents into a container or whether you are using a card layout as a template to decide which fields to retrieve, and where to place their contents. This latter process is discussed below.

DBGetCurrRecVal is a very powerful and complex function. It controls how data is retrieved from FileFlex. The function operates on the current record. The first parameter is an option character that tells FileFlex how the data is to be retrieved. These characters include:

Option Character	Description
G	Retrieve data into matching global variables
C	Retrieve data into matching card fields (HyperCard/SuperCard only)
B	Retrieve data into matching background fields (HyperCard/SuperCard only)
D	Decrypt data as retrieving (used in conjunction with other option characters)
X	Retrieve all fields into data chunk
(any char)	You can use any character other than those

```
above.   We recommend 'X', 'L', or 'A'.
```

To retrieve the current record's contents into a field or variable, you should call the DBGetCurrRecVal function with a single character parameter (we recommend "X"). This operation will place into the designated variable or container the contents of the current record in the following format:

```
Line 1         = Record Number
Line 2         = Delete Flag (Y/N)
Line 3   -Line n              = FieldName, FieldType, FieldValue
```

Lines 3 to the end of the container will each contain the value of a specific field in the database except in the case of a memo field. If a memo field is encountered, its name and type ("M") will be returned but the third item in the line, which normally contains the field's value, will be empty. You can then use the FileFlex DBGetMemo function to retrieve the memo field.

Here is an example, using the DBGetCurrRecVal function, and the contents of a variable called currRecContents after a sample record has been read (notice we use "L" this time):

```
put DBGetCurrRecVal("L") into currRecContents
```

The variable currRecContents will have contents similar to these:

```
7
N
Name,C,Dennis Wight
Salary,N,100000.00
Birth,D,19631030 (YYYYMMDD)
Single,L,T (T/F)
Note,M,
```

In the above example, the seventh field is a memo field called "Note" whose contents must be retrieved separately. DBGetCurrRecVal also has the ability to retrieve data and decrypt it. See the chapter on Office Quality Encryption for details.

Retrieving a Memo Field

If you need to retrieve a memo field, you should use the DBGetMemo function. It takes a single argument, the name of the memo field to be read. Here is a sample script line to retrieve a given memo field from the current record:

```
put DBGetMemo("NOTES") into field "Memo Field"
```

DBGetMemo also has the ability to retrieve data and decrypt it. See Office Quality Encryption for details.

Retrieving a Record into Global Variables

FileFlex has the ability to place the contents of FileFlex data file fields into corresponding global variables. In most development environments, if the global variable doesn't exist, FileFlex will create it. You should note, however, that this is a function of how the development environment works. As a result, it's often good practice to pre-create the global variables (by putting some value into each corresponding variable) or by declaring the variables as global.

Use DBGetCurrRecVal by using the option character "G", as in the following example:

```
put DBGetCurrRecVal("G") into DBResult
```

Reading a Record into a Card

In HyperCard or SuperCard, if you make the call to DBGetCurrRecVal with an argument of "B" or "C", then the function places each field in the FileFlex data file into a field in the card or background, respectively, whose name corresponds exactly to the name of the field in your scripting environment. Using the above example, assume you have background fields called Name, Salary, and Note and make the call as follows:

```
put DBGetCurrRecVal("B") into currRecContents
```

Then the background field called "Name" would have the value "Dennis Wight" in it, the field "Salary" would have "100000.00" and the field "Note" would have the contents of that database memo field. The other values in the database would simply be ignored.

Essentially, this use of the DBGetCurrRecVal function treats the current card's card or background fields (depending on which argument you use) as a template for the retrieval of information from the database. This is a very powerful feature of FileFlex. You could, for example, design a project that consisted of a series of cards, each of which had a different collection of card fields named after database fields, and then allow the user to select any one from a list for examination. Then you can use the navigation commands (go, for instance) in the scripting language to go to that card and execute a single command that is the same for all of the various card formats.

11. Adding and Updating Data

- Changing Field Values from a Container
- Changing a Memo Field from a Container
- Changing a Field's Value from Global Variables
- Updating from a Template Card
- Adding a New Record to the Database
- Multiple Database Update Technique

Changing Field Values from a Container

To update any field or group of fields in a database record from the contents of a single container in the scripting environment, use the FileFlex DBWriteRec function with three arguments.

The first argument is any character other than G,E, B or C (or their lower-case equivalents).

The next two arguments are the database record number and the record's new value. The new field value(s) must be stored in a field or variable with a separate line for each field being updated. Each of these lines will have two comma-delimited values, the first representing the name of the field and the second containing the field's new value.

Unmodified fields (i.e., those whose names do not appear in the container) are left as they were before the update process started. For example, to change the marital status of an employee from single to married and his number of dependents from zero to two, you might store the following information in a variable called newData:

```
MStatus,M
Dependents,2
```

and then write a line something like this:

```
put DBWriteRec("L",12,newData) into dbResult
```

DBWriteRec also has the ability to save data and encrypt it. See the chapter on Office Quality Encryption for details.

Changing a Memo Field from a Container

You may completely replace the contents of a given memo field with the contents of a container (typically a field which the user has edited) by using the FileFlex DBWriteMemo function,

supplying the database field name and its new value. The new value can be text or the name of a container holding the text.

Assume you've had the user provide some additional information for the Notes field in the current database record, and that you've stored the new memo field in a background field called "New Notes." You can update the information in the database with a line like this:

```
put DBWriteMemo("NOTES",field "New Notes") into dbResult
```

DBWriteMemo also has the ability to save data and encrypt it. See the chapter on Office Quality Encryption for details.

Changing a Field's Value from Global Variables

FileFlex has the ability to place the contents of global variables into corresponding FileFlex data file fields. If a matching global variable doesn't exist, the field will not be updating.

Note: This also implies that if the variable exists, it will be written. Therefore, make sure the contents of all applicable global variables are kept up to date.

Use DBWriteRec by using the option character "G", as in the following example:

```
put DBWriteRec("G") into DBResult
```

Updating from a Template Card

If you use HyperCard or SuperCard, you can use this feature. If you name the fields in your scripting environment to correspond with the names of the database fields, you can perform a record update from the contents of a card with a single call to the function DBWriteRec. In this case, the function requires only two arguments rather than the three we needed above.

The first is the letter "C" or "c", if you want FileFlex to take data from the card fields on the card involved, or the letter "B" or "b" if you want FileFlex to take the fields' new values from background fields.

The second is the number of the record to be updated.

FileFlex will only update fields in the database that have corresponding fields in the card involved in the update.

Assume that record 14 of a database file contains these fields and values:

```
Name                C J Nielsen
Salary   65000
Dept                13B
```

```
Jobcode5505.12
```

If you have a card with background fields named Name, Salary, and Jobcode and you put the name "C J Nielsen" into the first field, a value of "67500" into the second, and a value of "3918.2" into the third, you can update the salary and jobcode fields with this command:

```
put DBWriteRec("C",14) into dbResult
```

with the card containing the correct values being the current card at the time of the update. After that command executes, record 14 looks like this:

Name	C J Nielsen
Salary	67500
Dept	13B
Jobcode	3918.2

Adding a New Record to the Database

To add a record to the current database file, you may use any of the commands that update a record (as described above) and supply a record number that is at least one higher than the number of records in the database. You can find out how many records are in the database with the DBCount function. For example, if you want to use DBWriteRec to add a new record to the database from the card fields on the current card, you would write a script like this:

```
on mouseUp
    -- update happens in response to mouse-click on a button
    put DBCount() into numRecs
    add 1 to numRecs
    put DBWriteRec("C" numRecs) into DBResult
end mouseUp
```

It is not essential that you do this, however. If you use an arbitrarily large number for the record number, FileFlex will add the new information to the end of the file, if that number is at least one larger than the total number of records in the file.

Multiple Database Update Technique

You can use the DBWriteRec with the G,B, or C option to update more than one open database file from the same card containing new information. Because DBWriteRec ignores all globals or fields that are invalid for the current database, only those fields that are found in the current database will be udpated when you use one of these command formats.

Thus if you had information stored in globals that belonged in three different databases, you would simply use a DBWriteRec() call for each database in turn, something like this:

```
-- Assume DB opened earlier
put DBSelect(partList) into dbResult
put DBWriteRec("G",32) into dbResult -- updates Record #32

-- Assumes we opened DB earlier
put DBSelect(openItemFile) into dbResult
put DBWriteRec("G",11) into dbResult -- updates Record #11
put DBUse("BACKORDERS") into dbResult -- opens a database

-- get total number of records in file
put DBCount() into numRecs

-- add record at end of file
put DBWriteRec("G",numRecs+1) into dbResult
```

Each of the three databases would be updated with the information in global variables whose names exactly matched fields contained in that database.

12. Deleting Records

- Marking Records for Deletion
- Unmarking Delete-marked Records
- Physically Removing Marked Records
- Physically Deleting Records
- Determining if a Record is Deleted

When a record is deleted by the user, FileFlex does not physically remove the record from the file automatically. Rather, it marks the record as deleted. This allows your program to "undelete" a record.

When you wish to make these deletions irretrievable, you can either use the two-step process of physically removing the marked records and compacting the file to remove unused space or simply zap the records from the file as documented below.

Because of this complexity, FileFlex includes five functions that deal with deleting records:

- DBDeleteRecs
- DBRecallRecs
- DBPack
- DBZapRecs
- DBRecordDeleted

Marking Records for Deletion

The DBDeleteRecs function marks a range of records for deletion. It takes two arguments: a starting record number and an ending record number. The following function call, then, would mark records 9-15 of the current database file for later deletion:

```
put DBDeleteRecs(9,15) into dbResult
```

Unmarking Delete-marked Records

To unmark one or more records previously marked for deletion, use the DBRecallRecs function. Like the DBDeleteRecs function, it takes two arguments defining a range of records previously marked for deletion but not yet deleted. The records are unmarked and will not be deleted by a subsequent use of the DBPack function described below. Here's a sample use of this function:

```
put DBRecallRecs(11,12) into dbResult
```

Physically Removing Marked Records

To remove all records that are currently marked for deletion and pack the file to remove unused space resulting from the deletions, use the FileFlex DBPack function. This function requires no arguments since it operates on all previously marked records. Its use would appear something like this:

```
put DBPack() into dbResult
```

Physically Deleting Records

When you wish to physically remove one or more records from the current database file in one step, use the DBZapRecs function. It requires two arguments that define a range of records to be marked for deletion and deleted. In effect, the DBZapRecs function combines the results of the DBDeleteRecs and DBPack functions into one operation. Here is a sample of its use:

```
put DBZapRecs(9,15) into dbResult
```

Determining if a Record is Deleted

The FileFlex function DBRecordDeleted allows you to determine whether any given record in a database file is marked for deletion. This function requires a record number as an argument and returns "Y" if the record is marked for deletion, or "N" if it is not. For example:

```
put DBRecordDeleted(39) into recDeleted
```

13. Searching by Example (DBQuery)

- Expression Constants
- Field Names
- Intrinsic Functions and Operators
- Constructing Search Expressions
- Clearing a Search Condition

The DBQuery function (formerly DBLocate) allows you to set up complex queries (i.e., this is greater than that and that is less than this). The power in DBQuery is that it allows you to very tightly narrow in on what you're looking for. However, DBQuery operates by iteratively scanning the entire database, which means that it takes much longer to find something that's near the end of the database file than something that's near the beginning. In general, you should use DBQuery when your dataset size is relatively small. For much larger databases, consider turning to indexed searches (DBSeek).

The DBQuery function requires a single argument, which is a search expression string enclosed in quotation marks. This string must evaluate to a logical expression (i.e., interpreting it must lead to a "TRUE" or "FALSE" conclusion). For example, this line:

```
DBQuery("SALES > 50000")
```

will set up a search condition which tells FileFlex that you only want to see records where the value in the database field called "SALES" exceeds $50,000. The DBQuery function will move you to the first matching record, assuming one exists.

The DBQuery logical expression can consist of constants, field names, and functions, joined together by operators. In the example above, there is a field name ("SALES"), a constant (50000) and an operator (">", meaning greater than). The key to using DBQuery is learning to construct a query expression.

Expression Constants

Constants in FileFlex DBQuery expressions can be numeric, character strings, or logical values:

- A numeric constant contains a number (e.g., 65, 7.9, or -2932).

- A character constant is any string of letters, numbers, or special symbols enclosed in single quotation marks (e.g., 'FileFlex', 'comedy', or 'R'). Take considerable care to

surround the DBQuery expression character constants in a single quotation marks, rather than the double quotation marks common the the typical string definition of C, Lingo, or HyperTalk.

- A logical constant is one of these values: .TRUE., .FALSE., .T., or .F. Notice that these constants are surrounded by periods; these are essential for FileFlex to understand your constant as the logical value, rather than a string.

If a field in your database file contains a logical value and you want to test for it, you must be sure to provide the constant correctly. For example, the field called IN_STOCK in a video database file could be designed to contain a TRUE or FALSE value. To locate only videos that are in stock, you could set up the logical expression this way:

```
put DBQuery("IN_STOCK = .T.") into dbResult
```

but if you did this instead:

```
put DBQuery("IN_STOCK = 'T'") into dbResult
```

you would not find any records because FileFlex would look for the expression string 'T' in the field IN_STOCK rather than the logical value ".T." (meaning, of course, "TRUE").

Field Names

You must insure that field names in your DBQuery logical expressions match the database file's field names exactly. As always, case doesn't matter but punctuation does. Thus if the database field is called IN_STOCK and you attempt to define a value for a field called INSTOCK, you will produce an error. Be sure to remember that field names are NOT enclosed in quotes.

Intrinsic Functions and Operators

DBQuery allows you to use certain intrinsic functions (functions built into the expression analyzer) that can aid your search. One of the most useful is UPPER. UPPER converts the string being compared to upper case. By placing UPPER in your query expression, you can be sure that you'll find a string, regardless of the case stored in the database. For example:

```
put DBQuery("UPPER(FIRSTNAME) = 'DAVID'") into searchResult
```

Since we don't know whether the field FIRSTNAME contains "David" or "david" or "DAVID", the UPPER function allows us to find without regard to case. All of the intrinsic functions are defined in the Intrinsic Function Reference.

Constructing Search Expressions

Search expressions are pretty easy to construct. Even so, we get many technical support calls from users who get confused about what goes into the various strings. It gets even more interesting when a user wants to construct a search expression containing the value of a host-environment variable. So let's break it down for those using Lingo or HyperCard.

First, the typical search expression begins with the function call, followed by an open parenthesis. Everything between the open parenthesis and the closing parenthesis is the search expression:

```
                search expression goes below
                ----------------------------
put DBQuery(                                ) into searchResult
```

To your host programming language, the search expression is simply a string. Therefore, you could put the string into a variable and call pass the variable to DBQuery, as in the following example:

```
put "UPPER(FIRSTNAME) = 'DAVID'" into queryString
put DBQuery(queryString) into searchResult
```

Notice that the query string, like most normal strings, is bounded by double quotes and placed into the variable queryString. Again, the host environment only knows this as a string and does not parse the contents of the string.

But what if you wanted to replace the literal 'DAVID' with a string of your own choosing? Let's assume that you wanted to find the record matching the contents of the variable myFirstName. What many people do, and what WILL NOT WORK is the following:

```
-- the following won't work
put "UPPER(FIRSTNAME) = 'myFirstName'" into queryString
put DBQuery(queryString) into searchResult
```

All the above does is ask DBQuery to find a record where the contents of the field FIRSTNAME contains the literal string 'myFirstName'. What you need to do is construct a complex string. This is just pure Lingo or HyperTalk, there's no FileFlex magic here. The easiest way to see this is to construct a string in the following way:

```
-- store the first half of the query expression string
put "UPPER(FIRSTNAME) = '" into firstHalf
-- build the string properly
put firstHalf & myFirstName & "'" into queryString
put DBQuery(queryString) into searchResult
```

Take extra care to notice that the single quote (') is contained in the double quotes and is passed to DBQuery. By making sure that the myFirstName variable is passed outside the double quotes, you're ensuring that it's evaluated by your host environment before it's passed to DBQuery.

Confusion evaluating strings has got to be one of the most common technical support calls. So read and study the descriptions above, learn about strings in your host language, and please make sure you understand how strings work in your host language before calling us up. We'll just tell you to read this section anyway!

Clearing a Search Condition

Once you have set up a search condition with DBLocate, it stays in effect until you invoke another one. DBSkip will follow the database sequence looking for records that match the criterion in the last DBLocate command. This chain is broken by use of the command:

```
DBLocate("")
```

Thereafter, DBSkip will revert to its previous mode of operation (sequential or sort-order movement through the file).

14. Ultra-fast Searching with Indexes

- Index Files Supported
- When Indexes are Updated Automatically
- Opening and Using Index Files
- Finding a Record by Index
- Building a Seek Expression
- Getting Index File Information
- Creating New Indexes and Updating Old Ones
- Multi-Field Indexes
- Intrinsic Functions in Indexes
- You Must Index on a String Data Type

FileFlex excels at index files. Letting FileFlex use indexes is like giving a prized racehorse the chance to just let it all out, throttling up a perfectly tuned race car, or punching a fighter jet into afterburner. When we say that FileFlex can locate any record in a sea of billions of records faster than the blink of an eye, we're talking FileFlex indexes.

Whenever possible, we recommend you use indexed-based searching. Indexes work by algorithm, rather than brute force. FileFlex looks at the string you're search for, does a mathematical calculation that basically tells it how far into the file to move, and boom, it's on the record you need. By contrast, both DBQuery and the full-text search DBFindMemo scan on a record by record basis. This means that if the data you're looking for is at the end of the file, DBQuery and DBFindMemo must individually check all the preceding records prior to finding the match. DBSeek (the interface to indexes) just does a calculation and whammo! You're on the record.

Indexes are not without their price, however. Since indexes can't check every record and rely instead on complex offset calculations, they don't support complex queries. But they're ideal for most queries you'll need. Need to find someone's address? Construct an index combining last name and first name, do a DBSeek, and--poof!--you're on the record. Need to find everyone in the Southeast Region who's booked over $1.25 million and who hasn't gotten a recent raise? Use DBQuery and be prepared to wait a while.

Indexes also take disk space, often quite a lot. Each index is it's own file and that file contains the complete data of the field it's indexing, as well as some overhead space used internally by FileFlex. So, if you've got this handy name and address database and you want to index on last name, followed by first name, you've got one index file. If you want to index based on zipcode, you've got another index file, and so forth. But what the heck. CD-ROMs are big, text is small, and new hard drives are cheap. Use indexes and rejoice in the raw speed!

Note: It is possible to use DBSeek and DBQuery incorrectly and get dog-poor performance. These tools provide you with the capabilities. But it's up to you to design something that works efficiently. Just because you've got a hot database engine is no excuse for poor application design.

Index Files Supported

FileFlex supports the use and updating of dBASE III-compatible index files only. Other index file architectures such as FoxPro indexes cannot be used. However, FileFlex can reindex a file using the dBASE index file structures. This makes it easy to use files which have been indexed using other methods.

When Indexes are Updated Automatically

Any time you have opened one or more index files related to an open database file and you make changes to that database file, FileFlex automatically updates those indexes to reflect the new file contents.

Opening and Using Index Files

Among the functions you may wish to perform on an index file are the following:

- open a specific index file
- check an open index file to be sure it matches the file's contents
- select from among two or more open index files to make one current
- close an open index file

Opening an Index File

Use the FileFlex DBUseIndex function to open an index file for use. Supply the index file's name as an argument. Assign the result of this function to a variable (usually global) because you'll need to refer to the index file's ID in other scripts and handlers. Here's an example of the use of this function:

```
put DBUseIndex("STARS") into StarIndex
```

Note that the name of the index file should include any extension the file might have. In the Macintosh environment, extensions are normally omitted, but in DOS environments, the file extension ".NDX" will generally be used for dBASE III-compatible index files.

We regularly get tech support calls from customers who get index file errors. The most common reason is that the database the index file indexes must be open and selected (DBUse and

DBSelect) before executing a DBUse. One good practice is to do all your DBUse's and DBUseIndex's at the start of your application's life (for instance, in a startMovie handler). Use DBUse to open a data file and assign an ID to a global variable. Then call DBUseIndex repeatedly, opening all the index files for that data file, again assigning the IDs to globals. Repeat this until all your data and index files are open.

Note: Windows 3.1/DOS users may need to check the FILES= option in your CONFIG.SYS to make sure you're allowed to open this many files. If not, you'll need another file opening strategy, like only opening those files in use by a given module.

Checking an Index File for Currency

If you suspect that an index file may not be "in sync" with its database file, you can confirm its currency with the DBCheckIndex function. This function returns an error code of -8 if its contents do not match those of the database file. Otherwise, it returns the normal result code of 0.

This function would be most likely used in conjunction with the DBReindex function described below in a script line that looks like this:

```
if DBCheckIndex(indexID) = -8 then
    put DBReindex(indexID) into temp
end if
```

The value of indexID would, of course, be the result of the previous call to the DBUseIndex function as described above.

Selecting an Open Index File

You must inform FileFlex which of (potentially) several open index files it should use to access the current file with the DBSelectIndex function. Supply the ID of the index file (returned by the DBUseIndex function) as an argument:

```
put DBSelectIndex(StarIndex) into dbResult
```

Only one index file can be active at any time. It dictates the order in which record retrieval takes place. However, all open index files are updated each time a record is added or permanently.

If you're jumping between databases as well as indexes, don't forget to execute a DBSelect for the appropriate database file prior to calling DBSelectIndex on it's associated indexes.

Closing an Index File

Once you have opened an index file for use, you must also close it as part of cleaning up after your application. The FileFlex DBCloseIndex function closes a designated index file:

```
put DBCloseIndex(StarIndex) into dbResult
```

To close an index file use the DBCloseIndex function along with its associated ID (returned by the DBUseIndex function) to close it.

Finding a Record by Index

One of the main reasons for using indexes is to enable the database to find a record by the specific content of a specific field, namely, the one on which the index is based. The other reason is to keep the file in a specific order, or at least to give the appearance of retrieving information in that specific order. Remember that FileFlex never physically reorders your data. Indexes are the FileFlex equivalent of sort...but an equivalent that lets you switch between an unlimited number of sort orders instantly.

To locate a record which matches a specific expression in the indexed field, using the currently open index file, use the FileFlex DBSeek function, supplying a parameter containing the expression to be matched in the indexed field:

```
put DBSeek("Fred Jones") into foundRec
```

The seekExpr should contain the value for which you wish to search the indexed field. If this command fails to find the record (i.e., the search moves beyond the end of the file), the function returns a value of 3. You can then retrieve the contents of the located record.

Once you have found a record that matches the index key given as an argument, subsequent uses of the function will move the current record pointer to succeeding records in the file until a value of 3 is returned, indicating no more records match the criterion.

Building a Seek Expression

Seek expressions can be complex. One important thing to know is that the seek expression must match the exact number of characters as the data stored. So, in the example above with Fred Jones, DBSeek would have indicated an exact match only if the field being searched was defined to contain exactly ten characters and the contents was "Fred Jones".

But what if you're not looking for a name that exactly fits the field width? The short answer is you need to pad out the string to the right number of characters. Let's assume we're searching a name field defined as 15 characters wide. We'd need the following search expression to get an exact match ('~' indicates a space character):

```
put DBSeek("Fred Jones~~~~~") into foundRec
```

New in 2.0: In previous versions of FileFlex, you had to hand-construct a function to pad the string to the appropriate number of spaces. But in FileFlex 2.0, we introduce a new utility function called DBBuildSeekExpr. This function requires as it's parameters the ID of the index to

be searched, as well as one search expression for each field in the index. It returns a constructed search string you can then pass to DBSeek.

Let's assume, for example, that the index we're using has the ID of 1. We'd call the following to get a perfectly constructed seek expression:

```
put DBBuildSeekExpr(1,"Fred Jones") into theExpr
put DBSeek(theExpr) into foundRec
```

DBBuildSeekExpr is very smart. If the index has been defined to convert all data to upper case, the DBBuildSeekExpr will convert your search string to upper case. If you pass a number as a string, it will appropriate pad the spaces and decimal points to be sure that your search string is in the appropriate order. Further, DBBuildSeekExpr will construct complex queries with exactly the right spacing for use in complex searches.

Say, for example, that you created a multi-field index, searching last name, then first name. Using the following code would generate a perfectly balanced and sized search string:

```
put DBBuildSeekExpr(1,"Jones", "Fred") into theExpr
put DBSeek(theExpr) into foundRec
```

Getting Index File Information

New in 2.0: In previous versions of FileFlex, once you created an index, you better darn well name it clearly because there was no way to find out the original index expression. But now, you can get at the index expression in two ways: DBIndexExpr and DBListIndexFields.

DBIndexExpr

New in 2.0: The new DBIndexExpr returns the index expression that was used to create the index file. Make sure the index file in question has been opened, then pass the index ID as the sole parameter to DBIndexExpr. The function will return the actual string used to create the index:

```
put DBIndexExpr(1) into theExpr

-- theExpr contains the following:
UPPER(NAME)+STR(SALARY,6,2)
```

DBListIndexFields

New in 2.0: The new DBListIndexFields takes two parameters, the index file ID and a delimiter and returns a list of fields indexed by the specified index, separated by the delimiter. Here's an example:

```
put DBListIndexFields(1,RETURN) into theResult
-- theResult contains
NAME
SALARY
```

Creating New Indexes and Updating Old Ones

FileFlex includes functions that enable you to create a new index on a database file and to re-index a file whose index may be out of date. Like with creating a database, FileFlex 2.0 includes the FileFlex Database Designer that takes much of the work out of creating index files. Simply launch the Database Designer (in Director 5, it's under the Tools menu) and choose Index. Drag and drop your way into an easily designed index.

Creating a New Index

If you have a database file open and you wish to use programming to index it on a field for which no index is presently available, you can create a new index file with the FileFlex DBCreateIndex function. This function requires four arguments, as explained below, and follows this format:

```
put DBCreateIndex(fileName, indexExpr, unique, safety)
into dbResult
```

The four arguments required are:

- fileName, which is the name of the index file to be created (including a path name if you wish to place it somewhere specific)
- indexExpr, which is the name of the field on which you wish to create the index
- "0", reserved for future use in FileFlex
- safety, which is a logical value that is set to 0 if you want to overwrite an existing file of the same name if one exists, 1 if you wish to be prohibited from doing so

Re-Indexing a File

There are times that an index file and its associated data file can become unsynchronized. This might happen in a mixed-platform database environment, for example, where many people are using FileFlex to access a central database but a database administrator is using Access or FoxPro to maintain the files separately. You can determine if a particular index file is synchronized with the database in use by using the FileFlex DBCheckIndex function described earlier. If you find that it is not synchronized, you can use the FileFlex function DBReindex with the following syntax:

```
put DBReindex(indexID) into dbResult
```

Note that this operation uses an existing and previously opened index file and replaces it with the newly generated index.

Note: From a performance perspective, it's often not wise to do a DBCheckIndex and then a DBReindex if the index is out of sync. DBCheckIndex will scan each record until it finds a mismatch, and this takes time. It's often easier just to run a DBReindex and go for a cup of coffee, go out to lunch, or go home for the night (depending on the size of the database).

Multi-Field Indexes

FileFlex will permit you to index a single file on multiple fields. To create such an index, simply list the names of all the fields you wish to use in the index field group as a single string, with a plus sign concatenating them. Here's an example:

```
put DBCreateIndex("TERRSALE","TERRITORY+SALES",0,0) into dbResult
```

Fields should be listed from primary key to secondary key. Indexing will take place in major-minor order from left to right. In the above example, the file will end up being sorted by the field called TERRITORY and, within TERRITORY, by SALES.

There is no theoretical limit to the number of fields that can be combined into an index provided, of course, that the length of the argument to the DBCreateIndex command does not exceed the limitations of the environment in which you are running FileFlex, usually 256 characters.

Remember, once you create a multi-field index, you're going to want to search for data. The easiest way is to create a multi-field index seek expression using DBBuildSeekExpr as described above.

Note: Multi-field indexes only work on character fields. All fields must be character fields, so use the intrinsic functions to convert appropriately.

Intrinsic Functions in Indexes

To create an index using an intrinsic function, simply include it inside the string defining the index expression and call DBCreateIndex. Here's an example:

```
put DBCreateIndex("TERRSALE","UPPER(SALES)",0,0) into dbResult
```

The above would index the field SALES, but would sort as though all the letters in the SALES field were converted to upper case.

DBCreateIndex supports the following intrinsic functions:

```
DTOC
DEL
```

```
DELETED
RECNO
STR
SUBSTR
UPPER
```

And yes, you can actually do bizarre index expressions like:

```
"UPPER(LAST)+UPPER(FIRST)+STR(AGE)"
```

Intrinsic functions are documented in detail in the Intrinsic Function Reference.

You Must Index on a String Data Type

Because FileFlex was originally designed for use with scripting environments, it expects all its parameters to be strings.

Even so, if you were to index a FileFlex file by a numeric field, the indexing process would work. Problem is, you'd never be able to seek out any data.

That's because FileFlex expects to receive string values when running DBSeek. Indexing on a numeric value would require FileFlex to somehow convert the string it always receives to a numeric value. It doesn't do this. So, if you want to index something in numeric or date order, use the intrinsic functions DTOC and STR.

15. Full-Text Searching in Memo Fields

- Easy Full-Text Search
- Building a Complex Indexed Full-Text Search

While FileFlex character fields are fixed in width at the time of database design, and can store only up to 256 characters each, memo fields have no such restrictions.

FileFlex memo fields hold up to 32K and are allocated in 512-byte increments. As a result, you can comfortably place descriptions, notes, reports, letters, and other long information into FileFlex memo fields.

Easy Full-Text Search

FileFlex provides a simple, yet powerful function that allows you to find text anywhere in a memo field. The DBFindMemo function searches the specified memo field in the current database for the string specified. This is something of a brute-force searching mechanism. FileFlex checks every record, scanning each field of each record for the matching search string:

```
DBFindMemo("NOTES","status on FileFlex") into dbResult
```

Building a Complex Indexed Full-Text Search

If your database is very large, you'll find that DBFindMemo will take some time to find your record. But you can combine a number of FileFlex features together to create a rather sophisticated fully-indexed full-text search.

Note: Unlike normal indexes, this mechanism doesn't dynamically update the full-text index unless the index is rebuilt. Therefore it's most appropriate for use with CD-ROM or unchanging information.

There are two phases to performing a complex indexed full-text search:

- At development time, construct the index itself
- At runtime, search the index for your words

Constructing a Full-Text Index

Before you can build your index, you need to think through the indexing architecture. The most important database is the file where your memo field is located. Let's call this the "memo database". When the user enters a search word, we want to move the record pointer in the memo database to the first matching record.

You'll need to create two additional files: a database file (let's describe it as the "word database") containing the words to be searched and an index into the word database file.

Define the word database with two fields: a fixed width field for each word in the memo file, and a fixed width field containing the physical record number of the record in the memo file containing the word. Alternatively, the you can replace the record number with any other unique key that you can search for...but you'll need an extra index file if you use this technique.

Finally, you'll need a standard index file that uses the fixed word field as the indexed field and the word file as the data file. So, here's what you've got so far:

- **Memo database:** The database containing your memo field. This is where the actual information you're looking for resides;

- **Word database:** The database containing in one field all the words in the memo database memo field, and a field containing a pointer to the appropriate record in the memo database;

- **Index-on-word file:** An index file indexing the word field in the word database (make sure you use the UPPER intrinsic function to reduce ignore case).

Now you have to write a utility routine that follows the following algorithm (remember that this is a routine that you as the developer will run, not one used by the end-user of your software):

```
For every record in the memo database..

    Read the contents of the memo field into a variable

    For every word in the variable...

        Write the word and the current memo database record
        number to a new record in the word database

Make sure the index file is up-to-date with the data in the
word database.
```

At this point, you've got everything you need to do an incredibly fast, indexed full-text search. Let's look at how it'll run in your application to retrieve a value.

Searching Using a Full-Text Index

You get the word to be searched via some form of user input. You then pass that word to a full-text seek routine that does the following:

- Selects the word database (DBSelect)
- Selects the index-on-word index (DBSelectIndex)
- Prepares a seek expression, converting to upper case (DBBuildSeekExpr)
- Does the seek, moving to the first match (DBSeek)
- Grabs the pointer to the memo database (DBGetFieldByName)
- Selects the memo database (DBSelect)
- Moves (DBGo) to the physical record containing the memo field

You can, of course, write the routine to iterate through the list of matching words if you want to build a table of all the records that contain a given word.

16. Office-Quality Encryption

- Encryption Limits
- Standalone Encryption Functions
- Dynamic, On-the-Fly Encryption Functions
- Some Background on Implementation Choices

FileFlex' core encryption technology is "office quality", meaning its good enough for casual protection in an office environment, but far from "spook-proof". FileFlex encryption will prevent casual users from gaining access to data, but won't stop determined hackers from cracking the code.

I originally intended to implement DES but determined that DES wouldn't encrypt in-place and was quite slow. In-place encryption is critical; when you've got a 30-character field, you want to make sure the encrypted data will fit back into it. The new FileFlex encryption routines do on-the-fly, dynamic encryption and are exceptionally fast.

Encryption Limits

While there's no technical reason you can't encrypt everything, FileFlex doesn't support encryption of non-alphanumeric fields. This is because the encrypted data is totally random and you couldn't put the data back into the fields. If you want to encrypt numbers (i.e., salary data), store them in alphanumeric fields. For speed reasons,FileFlex doesn't check to see if you're attempting to encrypt or decrypt from a non-text field. FileFlex will attempt to encrypt or decrypt the data. However, the results are likely to be invalid or unreliable.

In practice, you also shouldn't encrypt any field you intend to search or index. In theory, you could do a DBQuery on encrypted data by doing a search for the encrypted string. But this doesn't seem really viable in practice.

Encryption keys should be printable characters. Encryption using keys that aren't standard characters may be unreliable. I won't accept any encryption bug reports where the key isn't something that can be typed from a keyboard (without using the Option key).

Standalone Encryption (DBEncrypt and DBDecrypt)

The easiest new commands to use are DBEncrypt and DBDecrypt. Both take as arguments a string and a key and return either an error code or an encrypted or decrypted string, as appropriate. In the following sample line, "ardvark" is the key:

```
put DBDecrypt(fld "src","aardvark") into plainText
put DBEncrypt(fld "src","aardvark") into scrambleText
```

Dynamic, On-the-Fly Encryption Functions

The idea behind dynamic encryption is that the data is plain-text in your project but enroute to or from a FileFlex data file the data is encrypted. Dynamic encryption options are extensions to existing FileFlex functions. The extended functions include:

FileFlex Function	Encryption Mode
DBGetCurrRecVal	dynamic decrypt
DBWriteRec	dynamic encrypt
DBGetFieldByName	dynamic decrypt
DBGetFieldByNum	dynamic decrypt
DBGetMemo	dynamic decrypt
DBWriteMemo	dynamic encrypt

DBWriteMemo

Syntax of DBWriteMemo is DBWriteMemo(memoField, memoVal). An optional "E" or "Encrypt" parameter and a key parameter has been added to the DBWriteMemo call so that the encryption syntax is:

```
DBWriteMemo(memoField, memoVal [, encryptFlag, key])
```

Example:

```
put DBWriteMemo("MYMEMO",fld "memo","E","aardvark") ¬
    into theResult
```

Note: Everything inside the brackets "[]" are optional.

DBGetMemo

Same idea as DBWriteMemo. Encryption syntax is:

```
DBGetMemo(memoField [, decryptFlag, key])
```

Example:

```
put DBGetMemo("MYMEMO","D","aardvark") into fld "memo"
```

DBGetFieldByName, DBGetFieldByNum

In both cases, decryption requires additional parameters:

```
DBGetFieldByName(fieldName [, decryptFlag,key])
DBGetFieldByNum(fieldNum [, decryptFlag,key])
```

Example:

```
put DBGetFieldByName("SALARY","D","aardvark") into fld "salary"
put DBGetFieldByNum(8,"D","aardvark") into fld "salary"
```

Some Background on Implementation Choices

When you look at DBGetCurrRecVal, this is where things start to get interesting. Encryption/decryption couldn't be implemented with just an "E" or "D" parameter since that would require you to encrypt or decrypt ALL the fields at once. This is not practical. The easiest "out" for would have been to require you to get encrypted fields one field at a time using DBGetMemo and DBGetFieldByName. This isn't wasn't slick as I'd like and user feedback indicated it would be overly onerous in your development cycle.

The next choice was to specify the whether a field is to be encrypted by prepending some special character in front of a field name, and this way the program would be able to check the first character. If the first character was the special character, the field would be decrypted. The problem with this approach is that you'd have to redefine the whole bloody database schema each time you switched from an encrypted field to an unencrypted field (and vice versa). But it's pretty easy to understand and is marginally elegant.

How Field-List Decryption Works

The third choice was to pass a list of field names to the DBGetCurrRecVal function when decrypting. This is the method that's been implemented in FileFlex. Here's how it works:

Assume the variable "decryptList" contains a comma-delimited list of fields to be decrypted. Here's an example:

```
SALARY,NOTES,AGE
```

To turn on decryption, you'd pass a "D" parameter. (You want to be able to specifically turn on decryption because checking field names will be slower than normal operations--and therefore you don't want it on when not necessary).

Here's an example to bring data back into a card:

```
put DBGetCurrRecVal("GD",decryptList,"aarvark") into theResult
```

"C" for global variables, "D" is the decryptFlag. Here's the syntax:

```
DBGetCurrRecVal("C|B|G[D]" [,decryptFieldList, key])
```

Note: Decryption is not supported for options that return the entire record in a single container.

DBWriteRec (with "G", "B", or "C" parameters)

DBWriteRec can be used to write either from a set of matching named global variables, from card or background fields (SuperCard/HyperCard only) or from a complex container. This segment talks about doing the C,B,G magic.

Again, you'd build a field list. For the purpose of this example, we'll assume the variable encryptList contains the field list:

```
put DBWriteRec("GE",12,encryptList,"aarvark") into theResult
put DBWriteRec("E",12,newData,encryptList,"aardvark") ¬
    into theResult
```

"G" for global variables, "E" to encrypt. 12 is the record to write. EncryptList is the field list and "aarvark" is the key. Syntax is:

```
DBWriteRec("C|B|G[E]",recNum [,encryptList,key])
```

Note: Encryption is not supported for options that save the entire record from a single complex container.

17. FileFlex Information Functions

- Count Records
- Calculate an Average
- Calculate the Sum of a Field

This section outlines three functions that don't fit logically into other places in the manual:

- DBCount
- DBAverage
- DBSum

Count Records with DBCount

Use this function whenever you want to know how many records are in a dBASE file. For example:

```
put DBCount() into howMany
```

Calculate an Average with DBAverage

With this function, you can calculate the average of a designated field's values in all of the records of a database file. The function requires the name of a field as an argument. The field must contain numeric data or an error will result. Here is a typical call to this function:

```
put DBAverage("Salary") into avgSal
```

Calculate the Sum of a Field with DBSum

With the FileFlex DBSum function, you can calculate the total of a designated field's values in all of the records of a database file. The function requires the name of a field as an argument. The field must contain numeric data or an error will result. Here is a typical call to this function:

```
put DBSum("Salary") into bigBucks
```

18. FileFlex API Reference

When developing an application using FileFlex, you'll be talking to the FileFlex engine through an application programming interface (API). This chapter describes each function in detail and should be used as your primary source for how a function is called.

- Functions by Name
- Functions by Category

Functions by Name

DBAverage
DBBottom
DBBuildSeekExpr*
DBCheckIndex
DBClose
DBCloseAll
DBCloseIndex
DBClosePlatform
DBCloseSession
DBCopyright
DBConvertCRLF
DBCount
DBCreate
DBCreateIndex
DBCurrDBNum
DBCurrRecNum
DBDatabaseExists*
DBDecrypt
DBDeleteRecs
DBEncrypt
DBFindMemo
DBGetCurrRecVal
DBGetFieldByName
DBGetFieldByNum
DBGetGlobal*
DBGetMemo
DBGo
DBIndexExpr*
DBInitPlatform
DBListFields
DBListIndexFields*
DBLocate@
DBMaxRecs*
DBOpenSession
DBPack
DBPlatform
DBQuery+
DBRecallRecs
DBRecordDeleted
DBReindex
DBSeek
DBSelect
DBSelectIndex
DBSetGlobal*
DBSkip
DBSum
DBTop
DBUse
DBUseIndex
DBVersion
DBWriteMemo
DBWriteRec
DBZapRecs

* New in 2.0
+ Name changed in 2.0
@ Obsolete in 2.0

Functions by Category

- Initialization Functions
- Creating Your Own Database Files
- Managing Database Files
- Navigating in the Database
- Retrieving Information
- Updating Information

- Using Index Files
- Performing Calculations
- FileFlex Version Information

Initialization Functions

- Initializing the Host Platform (DBInitPlatform)
- Initializing FileFlex (DBOpenSession)
- Closing Down FileFlex (DBCloseSession)
- Closing the Host Platform (DBClosePlatform)

Creating Your Own Database Files

- Creating Databases the Easy Way
- Creating Databases Programatically (DBCreate)

Managing Database Files

- Open a Database File (DBUse)
- Determining if a Database Exists (DBDatabaseExists)
- Select Database (DBSelect)
- Identify Selected Database (DBCurrDBNum)
- Close Database (DBClose and DBCloseAll)
- Getting a List of Fields (DBListFields)

Navigating in the Database

- Count Records (DBCount)
- Current Record Number (DBCurrRecNum)
- Go to a Record (DBGo)
- Move to Top or Bottom (DBTop and DBBottom)
- Skipping Records (DBSkip)

Retrieving Information

- Get Current Record (DBGetCurrRecVal)
- Finding Information In A Record (DBQuery)
- Get Field Contents (DBFieldByName and DBGetFieldByNum)
- Get Memo Field (DBGetMemo)
- Full Text Search in Memo Fields (DBFindMemo)
- Converting Platform Specific End-of-Lines (DBConvertCRLF)
- Is Record Deleted? (DBRecordDeleted)

Updating Information

- Update Record or Field (DBWriteRec)
- Updating a Record Directly From Matching Global Variables
- Updating a Record Directly From Card or Background Fields
- Updating a Record From a Single Container
- Specifying the Record Number to be Updated
- Add New Record (more on DBWriteRec)
- Update Memo Field (DBWriteMemo)
- Deleting Records (DBZapRecs, DBDeleteRecs, DBPack, DBRecallRecs)

Using Index Files

- Creating an Index File (DBCreateIndex)
- Open an Index File (DBUseIndex)
- Use an Index File (DBSelectIndex)
- Close an Index File (DBCloseIndex)
- Seek Specific Record (DBSeek)
- Building a Seek Expression (DBBuildSeekExpr)
- Determining an Index's Expression (DBIndexExpr)
- Determining an Index's Fields (DBListIndexFields)
- Dynamically Updating Indexes
- Checking Index File Integrity (DBCheckIndex)
- Rebuilding the Index File (DBReindex)

Performing Calculations

- Adding theTotal Value of a Field (DBSum)
- Averaging a Field's Value (DBAverage)
- Encrypting and Decrypting Data (DBEncrypt, DBDecrypt)

FileFlex Version Information

- Determining the Current Version of FileFlex (DBVersion)
- Determining the Host Platform (DBPlatform)
- Getting the FileFlex Copyright Message (DBCopyright)
- Determining the Maximum Accessible Records (DBMaxRecs)
- Internal Test Functions (DBGetGlobal, DBSetGlobal)

Initialization Functions

Initializing the Host Platform (DBInitPlatform)

Syntax: DBInitPlatform()

There are very few differences you need to be concerned about when programming in different platforms (i.e., Mac vs. Windows). We've added two standard function calls that allows you to place the bulk of the platform-specific code in one place. When you move your code from Mac to Windows, you'll want to just look at this code.

On the Macintosh, DBInitPlatform is where you place the "OpenXLib" or "start using" command to grant you access to the FileFlex engine as an external function.

- **Director 5:** If you install the Xtra inside the Xtras folder (recommended!), you won't need to use the openXLib reference since Director will automatically open FileFlex when Director loads. When you build a runtime version, if you make sure your installer creates an Xtras folder at the same level as your projector application and put the runtime Xtra into the folder, you'll have the same benefits. **(New in 2.0)**

- **Macintosh XFCN:** If you install the XFCN inside your movie, project, or stack, you won't need to externally reference the XFCN at all. Director 4 users should consider upgrading to Director 5 rather than using this method. Users of FileFlex Lite have not been provided with an XFCN version.

A typical DBInitPlatform definition on the Mac looks like this in Director:

```
on DBInitPlatform
    openXLib "FileFlex"
end DBInitPlatform
```

Under Director 4.0x for Windows, DBInitPlatform both provides access to the FileFlex engine and initializes the FileFlex engine as a Director XObject. Your DBInitPlatform call MUST look like the following:

```
on DBInitPlatform
    global gDBGlobalPtr1030, FFxobj
    put empty into gDBGlobalPtr1030
    openXlib "FFDIR.DLL"   -- < -- see comment below
    set FFxobj = ff(mNew) -- do not tinker with this line!
end DBInitPlatform
```

Director 4.0x Users: There is often confusion about the location of the FileFlex engine. When not provided a fully-realized path specification, both MacOS and Windows looks for the FileFlex engine (FileFlex on the Mac and FFDIR.DLL in Windows) in the current working directory. The current working directory is the directory in which the currently running application is located. Note that your stack or movie is not the application unless you've turned it into a standalone. So, if you want to avoid providing a fully specified pathspec in your start using or OpenXlib, put the FileFlex engine into the same directory/folder as the application that calls it. Better yet, consider upgrading to Director 5 and using the FileFlex Xtra.

Initializing FileFlex (DBOpenSession)

Syntax: DBOpenSession()

Before calling any FileFlex functions, you must notify FileFlex that you intend to use its routines. To do this, put the following call into the handler called most immediately after your project is opened (in Director, this would be the startMovie handler). The variable called dbResult is arbitrarily named; you can use any variable name you like. During debugging of a FileFlex interface, you might even put these results into fields or the Message Box so you can see exactly what is happening.

```
put DBOpenSession() into dbResult
```

Note: For those of you calling the FileFlex XFCN directly (in an unsupported environment like AppWare or Oracle Media Objects) FileFlex expects to set/check a global variable called gDBActive1030. Make sure you set gDBActive1030 to "true" before calling FileFlex(1), the direct XFCN equivalent of DBOpenSession. If gDBActive1030 is already true, then don't call FileFlex(1).

Closing Down FileFlex (DBCloseSession)

Syntax: DBCloseSession()

After you are done using FileFlex, you should clean up your environment by releasing memory used for buffers. To do this, put the following call into the handler called when exiting your movie or project. (It may be placed elsewhere, but this is the most logical place for its use.)

```
put DBCloseSession() into dbResult
```

This will free all the data structures in use by the FileFlex engine. Failure to do this may cause all sorts of unpleasant problems.

Note: For those of you calling the FileFlex XFCN directly (in an unsupported environment like AppWare or Oracle Media Objects) FileFlex expects to set/check a global variable called gDBActive1030. Make sure you set gDBActive1030 to "" (empty) before calling FileFlex(2), the direct XFCN equivalent to DBCloseSession. If gDBActive1030 is already empty, then don't call FileFlex(2).

Closing the Host Platform (DBClosePlatform)

Syntax: DBClosePlatform()

On the Macintosh, DBClosePlatform is where you place the "stop using" or "closeXLib" command to grant you access to the FileFlex engine as an external function.

- **Director 5:** If you install the Xtra inside the Xtras folder, you won't need to use the closeXLib reference since Director will automatically close FileFlex when Director exits. When you build a runtime version, if you make sure your installer creates an Xtras folder at the same level as your projector application and put the runtime Xtra into the folder, you'll have the same benefits. (New in 2.0)

- **Macintosh XFCN:** If you install the XFCN inside your movie, project, or stack, you won't need to externally reference the XFCN at all. Director 4 users should consider upgrading to Director 5 rather than using this method. Users of the Director 5 Special Edition have not been provided with an XFCN version.

A typical DBInitPlatform definition on the Mac looks like this in Director:

```
on DBClosePlatform
    closeXlib "FileFlex"
end DBClosePlatform
```

Under Director 4.0x for Windows, DBClosePlatform both closes access to the FileFlex engine and disposes of the FileFlex engine as a Director XObject. Your DBClosePlatform call MUST look like the following:

```
on DBClosePlatform
    global FFxobj
    put FFxobj(mDispose)
    closeXlib "FFDIR.DLL"
end DBClosePlatform
```

Note: Be sure to only close the host platform after you've closed files and executed a DBCloseSession. Only call DBCloseSession ONCE!

Creating Your Own Database Files

Creating Databases the Easy Way

New in 2.0: The easiest way to create your own FileFlex databases is to use the new FileFlex Database Designer included with FileFlex. See Creating Databases for more details.

Runtime Note: You must pre-create all databases you distribute. You may not include the FileFlex Database Designer itself with your application.

Creating Databases Programatically (DBCreate)

Syntax: DBCreate(<database name>, <# fields>, <field list>, <overwrite flag>)

Hardier souls may choose to use the DBCreate function to generate database files. DBCreate requires four parameters: the name of the database, the number of fields, a container containing a field list, and "true" or "false" (true if you don't want FileFlex to overwrite a data file).

Each line of the field list container is used for a field. Each line contains the following information: name of field, type of field, length, and number of decimal places (for numeric fields).

Here's an example call:

```
put DBCreate("FRIENDS",6, cd fld dbFields,false) into dbResult
```

Runtime Note: You must pre-create all databases you distribute. You may not use the DBCreate call with your application.

Note: See Creating Databases for a more detailed description of creating databases with DBCreate.

Managing Database Files

Open a Database File (DBUse)

Syntax: DBUse(<database name> [, <path spec>])

To open a database file, call the FileFlex function DBUse, supplying a database file name (including full path name if required) as an argument. Put the result into a variable, because you will need to refer to this file by its returned value later. Be sure not to use the same variable for any two files you to have open at one time.

Just type the following line into your script, changing the name of the database file appropriately.

```
put DBUse("HD:DB Files:Test.DBF") into databaseID
```

Alternatively, you can split the path specification into two parameters: the first the actual file name and the second the path specification. In the runtime, the path specification will remain unchanged but the database file name will be decrypted:

```
put DBUse("Test.DBF","HD:DB Files:") into dbID -- Developer FF
put DBUse("&#R*$*@U","HD:DB Files:") into dbID -- Runtime FF
```

(Special thanks to Wally Rutherford for pointing out the need for this feature.)

Determining if a Database Exists (DBDatabaseExists)

Syntax: DBDatabaseExists(<database name> [, <path spec>])

New in 2.0: You can easily determine if the database file you're about to manage is available by calling DBDatabaseExists. You might want to use this function if you're moving data from CD-ROM to a hard drive, or if you want to be sure the CD has been inserted and the database is available. DBDatabase Exists use the same parameters as DBUse (the database name and an optional path specification). If the database is not available, DBDatabaseExists returns a negative result code.

Note: DBDatabaseExists leaves <database name> closed after executing. If the database is available, you'll need to reopen it with DBUse.

Select Database (DBSelect)

Syntax: DBSelect(<database ID>)

Only one database file can be the current file. You may have numerous database files open at once, but only one of them will be the current database at any one time. The DBSelect function will allow you to choose any open database to act as the current database. To choose a previously opened database as the current file, use the FileFlex function DBSelect. It requires a single argument, the variable into which you put the ID of the database when you opened it with your call to DBUse.

```
put DBSelect(DatabaseID) into dbResult
```

Just as there can only be one current database file, so there can only be one record in that file that is recognized by FileFlex as the "current" record. All operations are performed on or relative to this record. When FileFlex switches between databases, the current record is saved.

An example will make this clearer. Let's assume that there are two databases open: a names database (ID 1) and an addresses database (ID 2). You're currently on the 53rd record of the names database. When you were last in the addresses database, the current record pointer pointed to record number 2,302. Now, when you execute a DBSelect(2), which moves you to the addresses database, the addresses database becomes current and the current record is now 2,302. If you then reselect the names database by issuing a DBSelect(1), the names database becomes current and the current record is 53.

Identify Selected Database (DBCurrDBNum)

Syntax: DBCurrDBNum()

There are times when you might want to know the database ID of the currently selected database. For example, you may want to interrupt processing on a file to undertake some special processing on another file and then return to processing the original file. To do so, use the FileFlex function DBCurrDBNum. It returns the ID of the currently selected database. This ID is the same as that returned when you called DBUse for this database.

```
put DBCurrDBNum()
```

Close Database (DBClose and DBCloseAll)

Syntax: DBClose(<database ID>)
Syntax: DBCloseAll()

When you are finished with a database, you can close the file and reclaim the memory used for its buffers by calling the FileFlex function DBClose and passing the database ID returned by the call to DBUse as an argument. DBClose returns an error code of 0 on successful completion, or an alternative error code if there is a problem.

```
put DBClose(DatabaseID) into dbResult
```

If you have more than one file open and want to close a specific open file, you must first insure that the file you want to close is the current file. Use DBSelect for this purpose. If you have more than one database file open and you are ready to end your work with FileFlex (or with that set of files), you can close them all in one step with the DBCloseAll function. It takes no argument.

```
put DBCloseAll() into dbResult
```

Here's an interesting tip: DBCloseSession executes a DBCloseAll prior to evacuating FileFlex from memory. So, while you should as good practice close your database files, FileFlex will clean up after itself--as long as you remember to close the session.

Getting a List of Fields (DBListFields)

Syntax: DBListFields()

You may occasionally wish to examine the fields in a database--their names, the types of data they contain, and their sizes, etc. FileFlex provides a function that allows you to retrieve such information about the currently active database file, if you wish.

You can obtain a list of all of the fields in the current database file with the FileFlex DBListFields function. Since it operates on the current database, you must first call the DBSelect function to make the desired database current if it is not already. The field list is returned in the variable or container you specify. The format for the field list devotes one line to each field, except for the first line, which contains a number indicating the number of fields in the file. Each line of the variable, or container, describes a field as follows: field name, field type, field width, decimal places The field type can be any of the values shown below:

```
ABBREVIATION  FIELD TYPE

    N                   numeric
    C                   character
    M                   memo
    L                   logical
```

```
D                       date
```

The DBListFields function's result should be placed into a variable or field so that you can later use it to relate field names and content types to field numbers.

In the following example, the first line makes the database whose name you supply as a parameter the current database, and the next one puts a list of its fields into a card field called "DBFields".

```
put DBSelect(databaseID) into dbResult
put DBListFields() into field "DBFields"
```

The first line of "DBFields" will contain the number of fields in the file. All subsequent lines will contain each field's name, type, width, decimal places as a comma-delimited list.

From the above example, the field named "DBFields" would contain the following information, assuming this is the structure of the file being examined:

```
Name,C,20,0
Salary,N,10,2
Single,L,1,0
Note,M,10,0
```

Note: Even non-numeric fields contain exactly four items in their field descriptions, including the number of decimal places (which will always be zero for non-numeric fields). Be forewarned that this is NOT the same structure as required by DBCreate.

New in 2.0: FileFlex 2.0 provides the DBListIndexFields function, which provides a similar service for fields placed into an index.

Navigating in the Database

Count Records (DBCount)

Syntax: DBCount()

To count the number of records in the current database file, you can use the FileFlex DBCount function. Since it operates only on the current database, you must first call the DBSelect function to make the desired database current if it is not already. The first line below makes the selected database current. The second line will then return the number of records in the database file.

```
put DBSelect(databaseID) into dbResult
put DBCount() into numRecs
```

Current Record Number (DBCurrRecNum)

Syntax: DBCurrRecNum()

To find the record number of the current record in the current database file, you can use the FileFlex DBCurrRecNum function. This routine will return the current record number. This routine returns the physical record number. If the logical record order has been changed with DBQuery or indexing, moving one record forward or backward in the database won't correspond directly to adding one or subtracting one from the physical record number.

```
put DBCurrRecNum() into CurrentRecordNumber
```

Go to a Record (DBGo)

Syntax: DBGo(<record #>)

You can position the current record pointer at any physical record, even with a currently active index, by using the FileFlex DBGo function. It takes one argument, the record number to which you wish to be positioned. It does not retrieve any data.

```
put DBGo(39) into dbResult
```

The above line will move the current record pointer to physical record 39. A "physical" record is usually different from a "logical" record, which refers to the record's number in indexed, or sorted, sequence.

Move to Top or Bottom (DBTop and DBBottom)

Syntax: DBTop()
Syntax: DBBottom()

You may sometimes want to get to the beginning (top) or to the end (bottom) of a database file. FileFlex supplies two functions for these purposes. DBTop and DBBottom are both subject to the logical record order. If you've got a current index or DBQuery operating, DBTop will take you to the logical first record and DBBottom to the logical last record.

To get to the top of the current database file and retrieve the value of the first record in the file, use the DBTop function. This will position the current record pointer to the top record in the database (physical record or, if an index is open, the first record in index order). You can then use DBGetCurrRecVal to retrieve the data in this record into fields or a container as desired.

Type the following lines, changing the name of the container as appropriate, to have FileFlex move to the first record in the current database and put that record's field values into a named container:

```
put DBTop() into dbResult
put DBGetCurrRecVal("A") into contents
```

Use DBBottom() to get to the last record in the database.

Skipping Records (DBSkip)

Syntax: DBSkip([-]<# records>)

If you don't know the number of the record you want to access, but you know its position relative to the current one, use the DBSkip function. DBSkip is also subject to logical record order. If an index or DBQuery is active, DBSkip will move the offset number of records according to the logical order of the database.

You can move the current database record pointer forward or backward a specified number of records with the FileFlex DBSkip function. You must supply an integer (positive or negative) describing the number of records to skip. A negative number tells FileFlex to move backward in the file. When it reaches the indicated record, FileFlex leaves the record pointer positioned at the record. You can then use the DBGetCurrRecVal function to retrieve the fields in that record as desired.

```
put DBSkip(23) into skipResult
put DBGetCurrRecVal("A") into dbResult
```

Here is how the function would normally be called if you want to skip 10 records from where you are currently positioned in the file:

```
put DBSkip(10) into dbResult
```

Or, for example, you may enter:

```
DBSkip(15) -- to skip forward fifteen records.
DBSkip(-2) -- to skip backward two records.
```

The behavior of DBSkip depends on whether there is an index file open or whether a DBQuery operation is currently active. If an index file is currently in use, DBSkip will skip the number of records in the sequence in which the index sorts the file. If a DBQuery function has been called with an argument other than an empty string, then DBSkip will skip records matching the search criteria defined in the DBQuery function.

Retrieving Information

Get Current Record (DBGetCurrRecVal)

Syntax: DBGetCurrRecVal("A" | "B" | "C" | "G")
Syntax: DBGetCurrRecVal("AD" | "BD" | "CD" | "GD", <decryption key>)

Once you have arrived in your database at the record whose value you wish to retrieve, you can retrieve its contents with the FileFlex DBGetCurrRecVal function.

You can return the contents of all the fields in the current record. After executing the following command, line 1 of the container will contain the record number, line 2 the delete flag, and all subsequent lines the data in the file, with the name, type, and value of each field on a separate line.

```
put DBGetCurrRecVal("A") into field Contents DBGetCurrRecVal
```

does NOT return the values of memo fields in the file if you use this method. You must retrieve each memo field separately using the GetMemo function in that case.

You can retrieve fields from the database file directly into global variables. In most development tools, FileFlex retrieves those database fields which have a global variable of the same name--if a global doesn't exist, it is created. If you use the "G" parameters to place retrieved data into global variables, FileFlex will retrieve any memo field that has a corresponding global variable.

```
put DBGetCurrRecVal("G") into foo
```

Be sure to check how your development environment handles globals. Some environments require globals to be pre-defined, others make globals static, storing data from one session to another. You'll need to know the characteristics of your development system to get the most joy and least surprises from this feature.

You can retrieve the contents of any database record whose number you know by combining the FileFlex function DBGo followed by the use of the DBGetCurrRecVal function. For example, to get the contents of record 23 and put them into their global variables, you would carry out these two lines of code:

```
put DBGo(23) into dbResult
put GetCurrRecVal("G") into foo
```

If you don't know the number of the record you wish to retrieve, but you do know something about its contents, you can locate the record with either DBQuery or DBSeek.

Encryption: This function has encryption/decryption options. See Office Quality Encryption for details.

HyperCard and SuperCard only: If you use the XFCN interface in either HyperCard or SuperCard, you can retrieve only selected fields from the database file into the card fields on the current card. In this case, FileFlex retrieves only those database fields which have a card field of the same name. If you use "C" or "B" parameters (See below) to place retrieved data into the card or background fields of the current card, FileFlex will retrieve any memo field that has a corresponding HyperCard field.

```
put DBGetCurrRecVal("C") into foo
```

You can also retrieve only selected fields from the database into background fields on the current card. In this case, FileFlex retrieves only those database fields which have a bg field of the same name. Note that the variable name "foo" is arbitrary and unused but must be supplied.

```
put DBGetCurrRecVal("B") into foo
```

Finding Information In A Record (DBQuery)

Syntax: DBQuery(<query expression>)

To query a database based on certain criteria and then find the appropriate record, use DBQuery (formerly 'DBLocate'). DBQuery will locate a record based on the contents of one or more database fields for which you do not have an index or for which you do not wish to use the index file for some reason (e.g., you want to perform a complex query), you can use the DBQuery function.

The DBQuery function takes a single argument, which must be either a string or a container that holds a string. The string must evaluate to a logical expression (i.e., one that can be True or False). Generally, these expressions are strings that contain the name of a database field, a comparison operator (=,>,<,<>, etc.) and a value to be compared.

For example, this line:

```
put DBQuery("sales > 50000") into dbResult
```

will set up a search condition which tells FileFlex that you only want to see records where the value in the database field called "sales" exceeds $50,000.

You can combine search terms using logical connections (.AND., .OR., and .NOT.). For example, to find records where sales are more than $50,000 and the customer is located in Michigan, you could use a line like this:

```
put DBQuery("sales > 50000 .AND. state='CA'") into dbResult
```

New in 2.0: This function used to be called 'DBLocate' but since it's designed to perform queries, we changed the name to be more clear. For backwards compatibility, we're still including the DBLocate wrapper script, but it will be eliminated by in the next release (FileFlex 2.1).

See Searching by Example for more details on using DBQuery and Intrinsic Function Reference for details on using FileFlex built-in functions in your query expressions.

Get Field Contents (DBFieldByName and DBGetFieldByNum)

Syntax: DBGetFieldByName(<field name>)
Syntax: DBGetFieldByNumber(<field number>)

You can obtain the value of a single field in the current record if you know the field's name or number using the FileFlex functions DBGetFieldByName and DBGetFieldByNumber. Both functions return the value of the indicated database field into the result variable.

Type the following line, providing your own field name, to retrieve a single named field from a database:

```
put DBGetFieldByName("EMPNAME") into field "Name"
```

Type the following line, providing your own field number, to retrieve a single field from a database:

```
put DBGetFieldByNumber(23) into theName
```

Either of these commands can use a container name as an argument. In that case, FileFlex assumes the container holds the name or number of the field desired.

Encryption: This function has encryption/decryption options. See Office Quality Encryption for details.

Get Memo Field (DBGetMemo)

Syntax: DBGetMemo(<field name>)

Whenever you use the FileFlex record-retrieval function DBGetCurrRecVal to place the values in a database record into a single container rather than global variables or card or background fields, you must separately retrieve any memo fields in that record.

The FileFlex DBGetMemo function will retrieve a named memo field's contents and put that data into the named container. Type the following line, changing the name field as needed, to retrieve a memo field:

```
put DBGetMemo("NOTES") into myMemo
```

If you use DBGetCurrRecVal to retrieve a record list, you can tell that GetCurrRecVal has found a memo field even if you don't know the database file's structure, because any line of the container into which you place the record values that contains a memo field has an "M" as its second item and an empty third item.

If you use DBGetCurrRecVal with the global variable "G" option or card ("C") or background ("B") fields option (HyperCard/SuperCard only) to retrieve database information, you need not retrieve memo fields individually. FileFlex will do this for you automatically.

Encryption: This function has encryption/decryption options. See Office Quality Encryption for details.

Full Text Search in Memo Fields (DBFindMemo)

Syntax: DBFindMemo(<field name>, <search string>)

In earlier versions of FileFlex (and most dBASE/xBASE database implementations), information that goes into a memo field is not searchable. FileFlex now supports a DBFindMemo command that does full text searches of memo fields for specified strings.

DBFindMemo is very fast and reasonably powerful. It does searches in physical record order (meaning that any existing DBLocate, DBSeek, and DBSkip options are ignored). DBFindMemo starts at the current physical record and searches iteratively, record-by-record until it either finds a matching string or the end of the file. Because it starts at the physical record, you may want to turn off any pre-existing searches with DBLocate("") to make sure a DBTop() takes you to the first physical record.

If DBFindMemo succeeds, it returns a 0; if it fails, it returns a 3 (and positions the physical record pointer at the end of the file).

If you use DBFindMemo, you should be aware that it searches until it finds...so that if you've got a huge database and unleash DBFindMemo, it could take some time until the routine returns a result. DBFindMemo does not offer any conditional expressions; if you look for "TITLE=foo", it will search for the string "TITLE=foo". DBFindMemo is extremely literal; it knows not of spaces, words, items, separators or much else. DBFindMemo is not case specific. Searching for "foo" will return successful if "FOO", "foo", "fOO" or even "foodstuffs" is found.

```
put DBFindMemo("NOTES","Component Software") into successFlag
```

The first parameter is the name of your memo field, the second is the string you are searching.

Converting Platform Specific End-of-Lines (DBConvertCRLF)

Syntax: DBConvertCRLF(<data> [, "A" | "M" | "W"])

The Macintosh terminates each line with a newline, and under Windows the convention is that each line is terminated with a carriage return and line feed. If you use memo fields, you may need to be aware of this.

Note: Director on both Macintosh and Windows uses the Macintosh format newline end of line character on both environments. As a result, if you write something that you store into a memo field in FileFlex on the Macintosh and you move it over to Windows and you run it in Director under Windows with FileFlex for Windows you will be able to read that memo field with no discernible difference.

The format of DBConvertCRLF is:

```
put DBConvertCRLF(<data>[, <option>]) into newData
```

DBConvertCRLF is important when you're reading DBF files created outside of this protected environment. DBConvertCRLF takes a container (a variable or a field) and converts the end-of-line characters according to the option parameter. If there is no option parameter (or the option parameter is set to "A" for "Automatic"), DBConvertCRLF converts the data into a form appropriate for the currently executing platform. If the option parameter is set to "M" (for Macintosh), DBConvertCRLF converts the data into newline-terminated lines for use on the Macintosh. If the option parameter is set to "W" (for Windows), DBConvertCRLF converts the data into carriage-return-line-feed-terminated lines for use in Windows.

Is Record Deleted? (DBRecordDeleted)

Syntax: DBRecordDeleted()

Like the original dBASE/xBASE standard, FileFlex allocates a single byte in each record to serve as a deletion flag. When you tell FileFlex to delete a record using DBDeleteRecs, FileFlex doesn't actually remove the data. Rather it sets a flag in the data file that tells you whether the record is marked for deletion. This makes operations very, very fast and also allows you to recover the data later. However, when navigating over records, it's important that you check to see if a record has been marked for deletion to determine whether you want to present the information to the user. To permanently delete a record, see the functions DBPack and DBZapRecs.

You can determine if any specific record has been deleted by using the FileFlex function DBRecordDeleted, supplying the record number as a parameter. FileFlex returns a "Y" if the record's delete flag has been set, "N" otherwise.

Type the following lines, with your own record number, to use this function:

```
put DBGo(29) into dbResult
put DBRecordDeleted() into recDeleted
```

Of course, if you are already positioned at the record you want to identify as deleted or not, you do not need to use the DBGo command.

Updating Information

Update Record or Field (DBWriteRec)

Syntax: DBWriteRec("G" | "GE", <record number>)
Syntax: DBWriteRec("C" | "CE", <record number>)
Syntax: DBWriteRec("B" | "GE", <record number>)
Syntax: DBWriteRec("X" | "L" | "A", <record data>, <record number>)

To update all or part of a database record, you can use the FileFlex function called DBWriteRec. This function takes three arguments.

Encryption: This function has encryption/decryption options. See Office Quality Encryption for details.

Updating a Record Directly From Matching Global Variables

If you pass a "G" as the first argument, you indicate to DBWriteRec that you want to take values from global variables and pass them to the database. The names of the global variables need to correspond to the names of the FileFlex fields.

Note: It is important that you pre-define and fill a complete set of global variables with names corresponding to those in the database. Since globals behave differently between development environments, passing a full set of globals is the best way to make sure everything works the way you want.

Updating a Record Directly From Card or Background Fields

HyperCard/SuperCard only: If you are using the XFCN interface in HyperCard or SuperCard, you can drop data directly into fields. The first argument to DBWriteRec is the container type. "C" indicates that the new values to be inserted into the selected record can be found in card fields on the current card whose names correspond to the names of fields in the database file. "B" indicates that the new values are stored in background fields of the same names as database fields.

Updating a Record From a Single Container

Instead of supplying a "B", "C", or "G" as the first argument, you can pass any other letter code. For example, an "A" means that the values are contained in a separate container (i.e., a variable) whose name is supplied as the third argument. Values are formatted with the field name and contents, separated by a comma. For example, assume that the variable myData contained the following:

```
NAME,David Gewirtz
COMPANY, Component Software
PRODUCT, FileFlex
```

A call to DBWriteRec in the form:

```
put DBWriteRec("X",myData,33) into dbResult
```

would place "David Gewirtz" into the field "NAME", "Component Software" into the field "COMPANY", and "FileFlex" into the field "PRODUCT", all on record 33.

Specifying the Record Number to be Updated

If the "G", "C", or "B" options have been used, the second argument is the record number of the record to be updated. It can be a value, a variable, or the DBCurrRecNum() function. If you've used the single container option, you'll put the record number into the third argument of DBWriteRec.

Add New Record (more on DBWriteRec)

To add a new record to a database file, you can use any of the various approaches to updating an individual record in the database but supply a record number that is at least one number higher than the number of records in the current database. Before you add a record, then, you should use the DBCount function to determine how many records are in the database now, then add 1 to that value. Pass the result to the FileFlex DBWriteRec or DBWriteMemo function as appropriate.

The following lines will set up a variable called newRecNum to contain a value one higher than the number of records in the current database.

```
put DBCount()+1 into numRecs
```

Now you can use this value (which you might want to declare as a global variable for use in other functions) to add records to the file. To add an entire record to a database file in FileFlex, you can choose any of the three above update methods.

Update Memo Field (DBWriteMemo)

Syntax: DBWriteMemo(<field name>, <field data>)

If you use the DBWriteRec function and a named container to update a database record and that database contains one or more memo fields, you must update those fields' contents individually with the DBWriteMemo function. You supply two arguments: a field name and the name of a container that holds the memo field's new contents. Notice that this function operates only on the current record, so you must make the appropriate record current before calling this function.

Type the following line into your script, making changes in the names of the fields and containers as appropriate:

```
put DBWriteMemo("Notes",field "New Notes") into dbResult
```

Encryption: This function has encryption/decryption options. See Office Quality Encryption for details.

Deleting Records (DBZapRecs, DBDeleteRecs, DBPack, DBRecallRecs)

Syntax: DBZapRecs(<start record #>, <end record #>)

Syntax: DBDeleteRecs(<start record #>, <end record #>)
Syntax: DBRecallRecs(<start record #>, <end record #>)
Syntax: DBPack()

You can use either a one-step approach to database record deletion or you can take a two-step approach.

In the one-step approach, you use the FileFlex DBZapRecs function and supply two comma-delimited values to indicate the first and last records to be deleted as arguments. The records are physically deleted and the database compacted. This makes the records unretrievable by any means.

Type the following line, supplying your own record numbers, to carry out this one-step record deletion:

```
put DBZapRecs(9,23) into dbResult
```

Note that even if you want to delete only one record, you must supply two arguments. To delete record 23, then, you would use:

```
put DBZapRecs(23,23) into dbResult
```

The two-step approach marks records for deletion and only handles their physical removal from the database on specific demand. This approach has two primary advantages. First, it can be undone as long as the second step hasn't yet been taken. Second, it is more efficient than the one-step approach since the sometimes time-consuming process of compacting the database only takes place once on demand rather than after each batch of records is deleted.

Type the following line, supplying your own record numbers, to carry out the first step in this two-step deletion:

```
put DBDeleteRecs(9,23) into dbResult
```

If you later change your mind and want to undelete some or all of these records, type the following line, with your own record numbers substituted:

```
put DBRecallRecs(9,23) into dbResult
```

When you are certain that you want to delete all marked records (for example, at the end of a day or processing session), call the FileFlex function DBPack. It takes no arguments and physically removes all marked records from the database, compacting the file when it has done so. This makes recovery of deleted records impossible. It's also relatively slow.

```
put DBPack() into dbResult
```

If the file you are working with has memo fields, it therefore has an associated file (usually with a name that ends in .DBT for reasons of compatibility with other versions of dBASE). Deleting

records that contain memo fields will not clear the memo file of those records. The file space must be reclaimed by a programming loop that reads each record and writes it to a new file. This was not our design decision but is part of the way xBASE systems behave.

If you want to clear the unused space from .DBT files, you're going to have to create a second database and then copy the record information from the first database into the second. This will create an optimized .DBT file. We don't recommend bothering with this unless space is at a premium and time is not.

Using Index Files

Creating an Index File (DBCreateIndex)

Syntax: DBCreateIndex(<index file>, <index expr>, "0", "0" | "1")

Use the DBCreateIndex function to create a brand-new index file. DBCreateIndex expects four parameters: the name of the index file, the index expression (usually the name of the field you wish to index), the string "0", and "0" if you want to overwrite an existing index file and "1" if you don't.

```
put DBCreateIndex("STARS","UPPER(NAME)","0","0") into dbResult
```

Runtime Note: You must pre-create all indexes you distribute. You may not use the DBCreateIndex call or the FileFlex stack itself with your application.

New in 2.0: You can use the FileFlex Database Designer to construct your index files. See Ultra-fast Searching with Indexes to understand more about how indexes work and how to construct index expressions. Also see Intrinsic Function Reference for more details on index expressions.

Open an Index File (DBUseIndex)

Syntax: DBUseIndex(<index file> [, <path spec>])

To open an index file, use the FileFlex DBUseIndex function. Supply the index file's name, including path name if needed, as an argument. Assign the result of this function to a global variable, since you'll need to refer to its database ID in other scripts and handlers.

```
put DBUseIndex("STARS") into starIndex
```

Alternatively, you can split the path specification into two parameters: the first the actual file name and the second the path specification. In the runtime, the path specification will remain unchanged but the database file name will be decrypted:

```
put DBUseIndex("STARS","HD:DB Files:") into ndxID -- Developer FF
put DBUseIndex("&#R*$","HD:DB Files:") into ndxID -- Runtime FF
```

(Special thanks to Wally Rutherford for pointing out the need for this feature.)

Note: Indexes are tied to their original data files (this makes sense because you can't index data in a file that doesn't have the data). As such, you must have the appropriate data file selected with a DBSelect function before calling DBUseIndex. Failure to do so usually results in FileFlex generating an error code, the user calling technical support, and said user being told to read this section again.

Use an Index File (DBSelectIndex)

Syntax: DBSelectIndex(<index ID>)

To use a particular index file, you must first open it with the DBUseIndex function. This function returns a value you should store in a global variable. This global is then supplied as an argument to the SelectIndex function to tell FileFlex to use this index as the current index file.

Assuming that you have previously opened the index file STAR.NDX and have its reference stored in the global variable starIndex, you would make that index current with a line like the one below.

```
put DBSelectIndex(starIndex) into dbResult
```

Note: Indexes are tied to their original data files. As such, you must have the appropriate data file selected with a DBSelect function before calling DBSelectIndex. Failure to do so usually results in FileFlex generating an error code, the user calling technical support, and said user being told to read this section one more time.

Close an Index File (DBCloseIndex)

Syntax: DBCloseIndex(<index ID>)

To close a particular index file, it must already have been opened previously with the DBUseIndex function. This function returns an index ID value you should store in a global variable. This ID value is then supplied as an argument to the DBCloseIndex function to tell FileFlex you no longer need the index file to be available.

Assuming that you have previously opened the index file STAR.NDX and have its reference stored in the global variable starIndex, you would close that index with a line like the one below. (Copy and paste that line, changing the variable name as appropriate) into your own script to close a particular open index.)

```
put DBCloseIndex(starIndex) into dbResult
```

Note: Indexes are tied to their original data files. As such, you must have the appropriate data file selected with a DBSelect function before calling DBCloseIndex. Failure to do so usually results in FileFlex generating an error code and, well, you know the rest.

Seek Specific Record (DBSeek)

Syntax: DBSeek(<seek expression>)

To locate a record that matches a specific expression in the indexed field(s) of your database and retrieve that record's values, you will use the FileFlex DBSeek function. The function takes a single argument, which is the seek expression, (i.e., the value to be searched for in the indexed database file using the current index).

The DBSeek function repositions the current record pointer to the record that matches the criterion in the seek expression or the one immediately following where the record would have been found if no matching expression is located in the currently active index. You can then use the DBGetCurrRecVal function to return the values in that records fields.

You must, of course, make sure that the index file that arranges the file in order by the desired field on which you wish to search is the current index file. Use the DBSelectIndex function for this.

```
put DBSelectIndex(starID) into dbResult
put DBSeek("Marilyn Monroe") into dbResult
```

You are now ready to retrieve the desired field(s) from this record.

Once you have located a record matching your index criterion with DBSeek, you may want to locate other matching records in the file. In that case, you should use DBSkip(1) to move the current record pointer to the next matching record in indexed order.

When the indexed field on which you are performing a DBSeek operation is a character field, DBSeek may locate a record that only partly matches your search criteria. Furthermore, if it does not find a matching record, it positions the record pointer at the next record in the file AFTER where it expected to find the information for which you have instructed it to search.

You can determine which type of result DBSeek has obtained because its return value is:

```
0 to mean an exact match was found
2 to mean a partial match was found
4 to mean no match was found
```

Where an exact match is not important, you need only check for a return value of 4. All other times, you should check for a return value other than 0 and respond accordingly.

A trick often used by FileFlex developers is to do a DBSeek on a partial match (for example, "Appl" as the seek expression to find all the possible "Apple Computer", "Apple Computer, Inc.", "Apple Pie", etc entries). Even though the DBSeek return code is other than zero, by grabbing the search field value using DBGetFieldByName and comparing, it's possible to tell if a record is a near match.

New in 2.0: DBSeek requires that the seek expression be the same width (including blank spaces) as the field definition. So if you define the field CITY to contain ten characters, you should pass "NEW~YORK~~" (the ~'s are used to indicate spaces) to the DBSeek function. FileFlex 2.0 adds a new DBBuildSeekExpr function that can be very helpful in building perfect seek expressions (see below).

Building a Seek Expression (DBBuildSeekExpr)

Syntax: DBBuildSeekExpr(<indexID>, <index expr> [, <index expr> ...])

The DBSeek function works hand-in-hand with the DBCreateIndex function. DBCreateIndex takes an index expression template (i.e., "UPPER(LAST)+UPPER(FIRST)") and creates a new index file. DBSeek uses an index expression (i.e., ("GEWIRTZ DAVID ") and searches the index file. The challenge is that the index expression must precisely match the index template in structure or DBSeek will not perform an exact match.

New in 2.0: Until FileFlex 2.0, the user had to very carefully construct the DBSeek expression by hand, converting character case, padding strings, offsetting decimal values and combining multi-field strings into a properly constructed seek expression. The potential for error was considerable and this caused frustration among new and old users alike. FileFlex 2.0 introduces DBBuildSeekExpr, a helper function that constructs a proper seek expression based on the template in the index file itself. DBBuildSeekExpr properly pads the string width, converts strings to upper case for case-insensitive searches, offsets decimal values appropriately, and combines multiple fields into a single search expression.

Using DBBuildSeekExpr is quite simple. The first parameter is the index ID of an open index file. The second and subsequent parameters correspond to each individual index expression in the index file. For example, let's assume you had an index file declared as "UPPER(LAST) + UPPER(FIRST)". Fields FIRST and LAST are each ten characters wide. Here's what you'd pass to DBBuildSeekExpr if you wanted to find my name:

```
put DBBuildSeekExpr("1","Gewirtz","David") into expr
```

DBBuildSeekExpr would look at the index definition of the index file represented by ID #1 and would place into expr the following string (the ~'s are used to indicate spaces to make it easier for you to read):

```
"GEWIRTZ~~~DAVID~~~~~"
```

Determining an Index's Expression (DBIndexExpr)

Syntax: DBIndexExpr(<indexID>)

New in 2.0: DBIndexExpr will retrieve the expression used to create the index file and return it to you as a string. Continuing on the example above, DBIndexExpr("1") will return "UPPER(LAST) + UPPER(FIRST)".

Determining an Index's Fields (DBListIndexFields)

Syntax: DBListIndexFields(<indexID>, <delim>)

New in 2.0: DBListIndexFields also provides the index expression back to you, but it strips out the intrinsic function calls and returns only the field names. Further, DBListIndexFields requires a delimiter character that lets you specify how you want the fields returned. Here's what DBListIndexExpr("1",",") would return:

```
UPPER,LOWER
```

And here's what DBListIndexExpr("1",RETURN) would return:

```
UPPER
LOWER
```

Dynamically Updating Indexes

Index files are maintained automatically by FileFlex. Whenever you execute a DBWriteRec function call, FileFlex updates all currently open index files for that database. No special programming is needed.

Checking Index File Integrity (DBCheckIndex)

Syntax: DBCheckIndex(<indexID>)
DBCheckIndex will examine an index file to determine if it is up to date and reliable. A value of less than zero indicates that the Index file is suspect.

```
put DBCheckIndex(indexID) into dbResult
```

Note: Indexes are tied to their original data files. As such, you must have the appropriate data file selected with a DBSelect function before calling DBCheckIndex. Failure to do so usually results in FileFlex generating an error code and the user calling technical support.

Rebuilding the Index File (DBReindex)

Syntax: DBReindex(<indexID>)

If your index file is not valid, you can force FileFlex to completely reconstruct the file using DBReindex.

```
put DBReindex(indexID) into dbResult
```

Note: Indexes are tied to their original data files. As such, you must have the appropriate data file selected with a DBSelect function before calling DBReindex. Failure to do so usually results in FileFlex generating an error code and the user calling technical support. Are you beginning to notice a trend here?

Performing Calculations

Adding theTotal Value of a Field (DBSum)

Syntax: DBSum(<field name>)

To calculate the total of all of the numeric contents of a specific database field, use the FileFlex DBSum function. Supply the name of a numeric field and put the result into a container. Copy and paste the following line into your script to handle this task, making appropriate changes in the field and container names:

```
put DBSum("SALARY") into myPayroll
```

Averaging a Field's Value (DBAverage)

Syntax: DBAverage(<field name>)

To calculate the average of all of the numeric contents of a specific database field, use the DBAverage function. Supply the name of a numeric field and put the result into a container. Copy and paste the following line into your script to handle this task, making appropriate changes in the field and container names:

```
put DBAverage("SALARY") into avgSalary
```

Encrypting and Decrypting Data (DBEncrypt, DBDecrypt)

Syntax: DBEncrypt(<string>, <encryption key>)
Syntax: DBDecrypt(<string>, <decryption key>)

DBEncrypt and DBDecrypt are standalone decryption functions provided in the FileFlex engine that allow you to get and retrieve encrypted data. These are ideal for encrypting and decrypting passwords, for example. The first parameter is the string you want to encrypt or decrypt. The second parameter is the encryption key, a string that is used as the seed for the encryption

formula. DBEncrypt returns an encrypted string. DBDecrypt, assuming it gets the right key, returns the decrypted string. Both are case sensitive.

FileFlex Version Information

Determining the Current Version of FileFlex (DBVersion)

Syntax: DBVersion()

You can ask FileFlex to tell you it's version number using the function DBVersion. It will return the numerical version number as a string followed by a status message indicating whether the external is licensed for distribution or not.

```
put DBVersion()
```

New in 2.0: FileFlex now specifies some very useful information as part of the version string. Version strings are now composed of 4 parts: the version number, the capability code (i.e., runtime, demo, sdk), the platform (i.e, 68K, WIN), and the host environment (i.e., Director 5, Authorware, XFCN, etc). Here's how they break down:

```
2.0.0L-PPC/DIR4 (Developer Version - Not Licensed for
-----|   |    |    Distribution)
  |   |   |    |
  |   |   |    |
  |   |   |    The host development environment
  |   |   |
  |   |   The host OS environment
  |   |
  |   Single character, if available, determines edition
  |   of FileFlex: 'L' is lite edition, 'R' is runtime
  |   'D' is demo, 'P' is the professional edition SDK
  |
Three-tier version number. The first number represents
major version, the second minor version, and the third
bug-fix updates.   Bug-fix updates can always be downloaded
for free from the www.component-net.com website.
```

If you want to get the version number only of the version string, you'll need the first five characters:

```
put char 1 to 5 of DBVersion()
```

Determining the Host Platform (DBPlatform)

Syntax: DBPlatform()

You can ask FileFlex to the current platform (i.e., Windows or Mac) by calling DBPlatform:

```
put DBPlatform()
```

FileFlex will return the following strings:

Platform	String Returned
Macintosh 68K	FF68K
Macintosh PPC	FFPPC
Windows	FFWIN

Getting the FileFlex Copyright Message (DBCopyright)

Syntax: DBCopyright()

You can ask FileFlex to report it's copyright message by using the DBCopyright function like this:

```
put DBCopyright()
```

Determining the Maximum Records Accessible (DBMaxRecs)

Syntax: DBMaxRecs()

New in 2.0: FileFlex now ships in a variety of "editions". Our online demo is available free to anyone who wants to see what FileFlex can do. However, it is limited to accessing only the first 100 records of a database file. Similarly, our Lite edition, bundled with Macromedia's Director, enables full database access on databases of 1000 records or less. The DBMaxRecs function allows returns the maximum number of records the engine will allow you to address. The function will return either a numerical value in string form (i.e., "100", "1000") or the string "unlimited". Remember, the professional edition of FileFlex is designed for very large databases; you can access upwards of a billion records.

Internal Test Functions (DBGetGlobal, DBSetGlobal)

Syntax: DBGetGlobal(<globalName>)
Syntax: DBSetGlobal(<globalName>,<value>)

New in 2.0: FileFlex uses the global variables of the host development environment in a number of ways, including setting and retrieving the values of host globals using DBWriteRec and DBGetCurrRecVal. As a result, the internal FileFlex engine must be able to successfully interact with the global variables of the host language. The functions DBGetGlobal and DBSetGlobal are not designed for your use. Rather, they are functions that allow us to test (in the validation suite) whether the global interface between FileFlex and the host globals work successfully. There's really no good reason for you to call DBGetGlobal yourself, for example, because it'll be much

easier for you to just put the global value into a container directly. But, if you're curious how these functions are used, look at the movie scripts in the validation suite.

19. Intrinsic Function Reference

FileFlex defines 12 functions that you can use in DBSeek and DBQuery logical expressions.

Intrinsic Functions for DBSeek and DBQuery

The following functions can be used in either DBSeek or DBQuery expressions:

- CTOD, which takes a character string as an argument and converts it into an internal database date format (e.g., CTOD(19901225), which converts the date Dec. 25, 1990, to internal database date format)

- DEL, which takes no argument and returns "*" if the current record is marked for deletion, a blank otherwise

- DELETED, which returns .TRUE. if the current record is marked for deletion, .FALSE. otherwise

- RECNO, which takes no argument and returns the current record number

- STR, which takes a numeric value, a number indicating length and a number indicating number of decimal places as arguments and converts the numeric value into a character string (e.g., STR(5.7, 4,2) will return "5.70")

- SUBSTR, which extracts a substring of the string supplied as its first argument, beginning at the position specified in the second argument and continuing for the number of characters specified by the third argument (e.g., SUBSTR('ABCDEF',2,3) returns "BCD")

- UPPER, which takes a character value as an argument and converts it to uppercase

Intrinsic Functions for DBQuery Only

The following functions can be used in DBQuery expressions only:

- DATE, which takes no argument and returns the system date from the computer's internal clock

- DTOC, which takes a database date format value as an argument and converts it into the equivalent string (e.g., CTOD(DATE()), which converts today's date into a character string so that if today were Dec. 25, 1990, the result would be "19901225")

- IFF, which takes another logical expression as its first argument, followed by an indication of what should happen if that logical comparison is true, and another indication of what should happen if it is false (see discussion below)

- RECCOUNT, which takes no argument and returns the total number of records in the database (not to be confused with the FileFlex DBCount API function, which permits you to retrieve the record count and do something with it other than use it in a logical expression)

- VAL, which takes a character string and returns its numeric value (e.g., VAL("233") returns 233).

The IFF function described above allows you to nest logical conditions with one DBQuery call. For example, you might design a credit collection and management system with the requirement that each customer file has not only a credit limit, but also a description of when you wish to take some collection action. The credit limit might be stored in a field called LIMIT and the action description in a field called COLLECT. This would enable you to establish different collection policies for each customer. You might use a line something like this:

```
put DBQuery("COLLECT = IFF(LIMIT>10000,'Casual',' '")
    into dbResult
```

This would confine your database review to records where the field COLLECT had a value of "Casual" and the LIMIT field is greater than $10,000. In other words, it would allow you to examine all casual customers with high credit limits.

Logical Operators (DBQuery Only)

Within the DBQuery expression, you can join functions, constants, and field names with logical operators. There are two types of such operators: comparative and connective.

The comparative operators recognized by FileFlex are shown below. Except for the "$" operator, they are the "standard" comparative operators recognized by many major database systems and

programming languages. The "$" operator tests for the presence in one string of another (e.g., "ABC" $ "ABCDEF" returns .TRUE.).

Comparative Operators

SYMBOL	INTERPRETATION
=	equal to
<>	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
$	is contained in

The connective operators recognized by FileFlex are the same as those understood by dBASE: .NOT., .OR., and .AND. Note that, like the .TRUE. and .FALSE. constants mentioned earlier, these operators must be surrounded by periods to avoid their being confused with strings of characters.

These connective operators can be combined with the logical operators above to create some fairly complex search criteria. For example, if you wanted to find all your customers in California or Arizona with low account balances and who had received your catalog, you might end up with an expression like this:

```
DBQuery("(State='CA' .OR. State='AZ') .AND.
    Balance<=1000 .AND. Cat=.T.)")
```

String Constants in DBQuery Expressions

FileFlex defines a special string constant operator used within the DBQuery expressions. The string operator is the single quote (') -- make sure you don't use "smart quotes". When you call DBQuery, you always need to pass it a string containing the search expression. This is string according to the rules of your host development environment. But when you're comparing the state to 'CA', as in the above example, you're passing a DBQuery string to the function. In this case, the string you're asking DBQuery to compare within the expression must be surrounded by single quotes.

20. Result Code Reference

- Positive Result Codes
- Negative Result Codes

Most FileFlex calls return result codes that are either negative or positive. You should check the result code after every call to FileFlex. Generally, positive result codes (zero and up) mean that the function executed successfully. Negative result codes generally mean that something didn't go right with the function call.

One of the most common technical support call is "Gee, my database isn't working." When I ask the caller if he or she checked the result codes after, say, DBUse, I'm told "No, why?" I then go into a long, reasoned discourse that, summarized, goes like this: "If you don't check an earlier result, the later operation isn't guaranteed to work. So, if your DBUse fails (and it often does when you don't specify the file path properly), lots of stuff later in your code will fail. The moral of the story: check result codes."

Positive Result Codes

Positive result codes are different from FileFlex function to FileFlex function. Therefore, the positive result codes are organized by function call:

-- DBBottom ----------
3: File is empty

-- DBFindMemo ----------

0: The search was successful.
3: The search failed and the record pointer is at end of file.

-- DBSeek ----------

0: An exact match was found.
2: A partial match was found.
3: Seek past end of file.
4: No match was found.

Note: A result code of 4 means that the database engine stopped searching when it encountered a data item of a value higher than the search engine. It leaves the record pointer on this value. (Hint: This has some interesting algorithmic potential)

-- DBSkip ----------

0: The skip operation was completed successfully.

1: The skip operation moved before the beginning of the file.
3: The skip operation moved past the end of the file.

-- DBTop ----------

3: File is empty

-- DBUse ----------

0 and up: DBUse returns codes of zero and up if the database file was successfully opened. These codes are the database ID numbers and are used by the DBSelect function.

-- DBUseIndex ----------

0 and up: DBUseIndex returns codes of zero and up if the index file was successfully opened. These codes are the index ID numbers and are used by the DBSelectIndex function.

Negative Result Codes

Negative result codes indicate that some internal error was identified by FileFlex. Here's a list of negative codes by number:

-1 = General program error
-2 = Database is empty
-3 = Index is empty
-4 = No such data field
-5 = No such record
-6 = Field not numeric
-7 = Insufficient number of parameters
-8 = Index file out of date
-100 = File creation error
-120 = File open error
-140 = File read error
-160 = File write error
-180 = File close error
-200 = Bad database file
-240 = Database file not found
-260 = Record length too large
-280 = Record length inconsistent
-310 = Error closing index file
-320 = File is not an index file
-330 = Index file is missing a key
-335 = Index file record error
-340 = Key is not unique
-350 = Key equates to a logical value
-360 = Key length or type has changed or is invalid
-370 = Key length exceeds 100 characters
-380 = Seek attempted with no index file active
-500 = Database not found during expr evaluation
-510 = Attempt to evaluate null expression
-515 = Illegal date in expression

-520 = Expecting ',' or ')' in expression
-530 = Incomplete expression
-540 = Math overflow in expression
-550 = Operator or parameter type mismatch
-560 = Right bracket missing in expression
-570 = Unrecognized function
-575 = Unrecognized operator
-580 = Unrecognized value
-590 = Unterminated string in expression
-610 = Not a memo field
-620 = Not enough memory to convert CRLF
-900 = Out of memory
-950 = Internal error
-960 = No encryption key provided
-961 = No encryption string provided
-962 = Path specification required
-963 = Encryption option not valid
-910 = Attempt to move beyond max allowed records
(100 in demo, 1000 in Lite edition)

21. The Direct FileFlex API Interface

You can directly access the FileFlex engine by passing it an appropriate set of parameters. We always recommend using the wrapper scripts, but our users have requested the full list of function calls for the occassional direct access. So here it is.

FileFlex Parameters

All parameters passed to FileFlex engine must be of the string data type. These include the function call numbers.

FileFlex Calls by Name

```
-- DBAverage ------------------------------------------------------
Direct Call:                FileFlex(22, numFieldName) / FileFlex(19)
FileFlex DB Call:           DBAverage(numFieldName)

-- DBBottom -------------------------------------------------------
Direct Call:                FileFlex(12)
FileFlex DB Call:           DBBottom()

-- DBBuildSeekExpr ------------------------------------------------
Direct Call:                FileFlex(45, indexID, val, ...)
FileFlex DB Call:           DBBuildSeekExpr(indexID, val, ...)

-- DBCheckIndex ---------------------------------------------------
Direct Call:                FileFlex(34, indexID)
FileFlex DB Call:           DBCheckIndex(indexID)

-- DBClose --------------------------------------------------------
Direct Call:                FileFlex(4,dbID)
FileFlex DB Call:           DBClose(dbID)

-- DBCloseAll -----------------------------------------------------
Direct Call:                FileFlex(6)
FileFlex DB Call:           DBCloseAll()

-- DBCloseIndex ---------------------------------------------------
Direct Call:                FileFlex(8,indexID)
FileFlex DB Call:           DBCloseIndex(indexID)
```

```
-- DBClosePlatform ------------------------------------------------
Direct Call:                ** Not available **
FileFlex DB Call:           DBClosePlatform()
```

Note: This is a script you're expected to write or modify from the provided wrapper scripts.

```
-- DBCloseSession -------------------------------------------------
Direct Call:                FileFlex(2)
FileFlex DB Call:           DBCloseSession()
```

Note: FileFlex(2) expects to set/check a global variable called gDBActive1030. Make sure you set gDBActive1030 to "" (empty) before calling FileFlex(2). If gDBActive1030 is already empty, then don't call FileFlex(2).

```
-- DBCopyright ----------------------------------------------------
Direct Call:                FileFlex(999)
FileFlex DB Call:           DBCopyright()

-- DBConvertCRLF --------------------------------------------------
Direct Call:                FileFlex(41,data,option)
FileFlex DB Call:           DBConvertCRLF(data,option)

-- DBCount --------------------------------------------------------
Direct Call:                FileFlex(19)
FileFlex DB Call:           DBCount()

-- DBCreate -------------------------------------------------------
Direct Call:                FileFlex(33, dbName, numFields,
                               fields, safety)
FileFlex DB Call:           DBCreate(dbName, numFields, fields,
                               safety)

-- DBCreateIndex --------------------------------------------------
Direct Call:                FileFlex(31, indexName, indexExpr,
                               unique, safety)
FileFlex DB Call:           DBCreateIndex(indexName, indexExpr,
                               unique, safety)

-- DBCurrDBNum ----------------------------------------------------
Direct Call:                FileFlex(36)
FileFlex DB Call:           DBCurrDBNum()
```

Note: Returns the current db number

```
-- DBCurrRecNum ---------------------------------------------------
Direct Call:                FileFlex(21)
FileFlex DB Call:           DBCurrRecNum()

-- DBDatabaseExists -----------------------------------------------
Direct Call:                FileFlex(43, datafile[, path])
FileFlex DB Call:           DBDatabaseExists(datafile[, path])
```

```
-- DBDecrypt ----------------------------------------------------
Direct Call:                    FileFlex(39, string, key)
FileFlex DB Call:               DBDecrypt(string, key)

-- DBDeleteRecs -------------------------------------------------
Direct Call:                    FileFlex(15, startRec, endRec)
FileFlex DB Call:               DBDeleteRecs(startRec, endRec)

-- DBEncrypt ----------------------------------------------------
Direct Call:                    FileFlex(38, string, key)
FileFlex DB Call:               DBEncrypt(string, key)

-- DBFindMemo ---------------------------------------------------
Direct Call:                    FileFlex(40, memofield, string)
FileFlex DB Call:               DBFindMemo(memofield, string)

-- DBGetCurrRecVal ----------------------------------------------
Direct Call:                    FileFlex(28, containerType)
FileFlex DB Call:               DBGetCurrRecVal(containerType)
```

Note: If containerType is "B" or "C", FileFlex attempts to place its values into HyperCard or SuperCard background or card fields, respectively. This is one of the few places where FileFlex makes direct HyperCard callbacks. If you're not in HyperCard, don't pass "B" or "C" (choose some other character, like "A", instead). If containerType is "G", FileFlex attempts to place its field values into matching global variables, creating them if they don't exist. This function supports decryption. See Office-Quality Encryption for syntax.

```
-- DBGetFieldByName ---------------------------------------------
Direct Call:                    FileFlex(26, fieldName)
FileFlex DB Call:               DBGetFieldByName(fieldName)
```

Note: This function supports decryption. See Office-Quality Encryption for syntax.

```
-- DBGetFieldByNum ----------------------------------------------
Direct Call:                    FileFlex(27, fieldNum)
FileFlex DB Call:               DBGetFieldByNum(fieldNum)
```

Note: This function supports decryption. See Office-Quality Encryption for syntax.

```
-- DBGetGlobal --------------------------------------------------
Direct Call:                    FileFlex(201, globalName)
FileFlex DB Call:               DBGetGlobal(globalName)
-- DBGetMemo ----------------------------------------------------
Direct Call:                    FileFlex(24, memoField)
FileFlex DB Call:               DBGetMemo(memoField)
```

Note: This function supports decryption. See Office-Quality Encryption for syntax.

```
-- DBGo ---------------------------------------------------------
Direct Call:                    FileFlex(10,recNum)
FileFlex DB Call:               DBGo(recNum)
```

```
-- DBIndexExpr ----------------------------------------------------
Direct Call:                    FileFlex(44,indexID)
FileFlex DB Call:               DBIndexExpr(indexID)

-- DBInitPlatform -------------------------------------------------
Direct Call:                    ** Not Available **
FileFlex DB Call:               DBInitPlatform()
```

Note: This is a script you're expected to write or modify from the provided wrapper scripts.

```
-- DBListFields ---------------------------------------------------
Direct Call:                    FileFlex(29)
FileFlex DB Call:               DBListFields()

-- DBLocate (obsolete)--------------------------------------------
Direct Call:                    FileFlex(35, expr)
FileFlex DB Call:               DBLocate(expr)

-- DBListIndexFields ----------------------------------------------
Direct Call:                    FileFlex(46, indexID, delim)
FileFlex DB Call:               DBListIndexFields(indexID, delim)

-- DBMaxRecs ------------------------------------------------------
Direct Call:                    FileFlex(47)
FileFlex DB Call:               DBMaxRecs()

-- DBOpenSession --------------------------------------------------
Direct Call:                    FileFlex(1)
FileFlex DB Call:               DBOpenSession()
```

Note: FileFlex(1) expects to set/check a global variable called gDBActive1030. Make sure you set gDBActive1030 to "true" before calling FileFlex(1). If gDBActive1030 is already true, then don't call FileFlex(1).

```
-- DBPack ---------------------------------------------------------
Direct Call:                    FileFlex(18)
FileFlex DB Call:               DBPack()

-- DBPlatform -----------------------------------------------------
Direct Call:                    FileFlex(42)
FileFlex DB Call:               DBPlatform()

-- DBQuery --------------------------------------------------------
Direct Call:                    FileFlex(35, expr)
FileFlex DB Call:               DBQuery(expr)

-- DBRecallRecs ---------------------------------------------------
Direct Call:                    FileFlex(16, startRec, endRec)
FileFlex DB Call:               DBRecallRecs(startRec, endRec)

-- DBRecordDeleted ------------------------------------------------
```

```
Direct Call:                FileFlex(20, recNum)
FileFlex DB Call:           DBRecordDeleted(recNum)

-- DBReindex ----------------------------------------------------
Direct Call:                FileFlex(32, indexID)
FileFlex DB Call:           DBReindex(indexID)

-- DBSeek -------------------------------------------------------
Direct Call:                FileFlex(13, seekExpr)
FileFlex DB Call:           DBSeek(seekExpr)

-- DBSelect -----------------------------------------------------
Direct Call:                FileFlex(5,dbID)
FileFlex DB Call:           DBSelect(dbID)

-- DBSelectIndex ------------------------------------------------
Direct Call:                FileFlex(9,indexID)
FileFlex DB Call:           DBSelectIndex(indexID)

-- DBSetGlobal --------------------------------------------------
Direct Call:                FileFlex(202,globalName, globalVal)
FileFlex DB Call:           DBSetGlobal(globalName, globalVal)

-- DBSkip -------------------------------------------------------
Direct Call:                FileFlex(14, num)
FileFlex DB Call:           DBSkip(num)

-- DBSum --------------------------------------------------------
Direct Call:                FileFlex(22, numFieldName)
FileFlex DB Call:           DBSum(numFieldName)

-- DBTop --------------------------------------------------------
Direct Call:                FileFlex(11)
FileFlex DB Call:           DBTop()

-- DBUse --------------------------------------------------------
Direct Call:                FileFlex(3,dbName [, pathSpec])
FileFlex DB Call:           DBUse(dbName [, pathSpec])

-- DBUseIndex ---------------------------------------------------
Direct Call:                FileFlex(7,IndexName [,pathSpec])
FileFlex DB Call:           DBUseIndex(IndexName [,pathSpec])

-- DBVersion ----------------------------------------------------
Direct Call:                FileFlex(0)
FileFlex DB Call:           DBVersion()

-- DBWriteMemo --------------------------------------------------
Direct Call:                FileFlex(25, memoField, memoVal)
FileFlex DB Call:           DBWriteMemo(memoField, memoVal)
```

Note: This function supports encryption. See Office-Quality Encryption for syntax.

-- DBWriteRec ----------
Direct Call: FileFlex(30, containerType,recNum [, value])
FileFlex DB Call: DBWriteRec(containerType,recNum [, value])

Note: This function supports encryption. See Office-Quality Encryption for syntax.

-- DBZapRecs ----------
Direct Call: FileFlex(17, startRec, endRec)
FileFlex DB Call: DBZapRecs(startRec, endRec)

FileFlex Calls by Number

-- 0 ----------
Direct Call: FileFlex(0)
FileFlex DB Call: DBVersion()

-- 1 ----------
Direct Call: FileFlex(1)
FileFlex DB Call: DBOpenSession()

Note: FileFlex(1) expects to set/check a global variable called gDBActive1030. Make sure you set gDBActive1030 to "true" before calling FileFlex(1). If gDBActive1030 is already true, then don't call FileFlex(1).

-- 2 ----------
Direct Call: FileFlex(2)
FileFlex DB Call: DBCloseSession()

Note: FileFlex(2) expects to set/check a global variable called gDBActive1030. Make sure you set gDBActive1030 to "" (empty) before calling FileFlex(2). If gDBActive1030 is already empty, then don't call FileFlex(2).

-- 3 ----------
Direct Call: FileFlex(3,dbName [, pathSpec])
FileFlex DB Call: DBUse(dbName [, pathSpec])

-- 4 ----------
Direct Call: FileFlex(4,dbID)
FileFlex DB Call: DBClose(dbID)

-- 5 ----------
Direct Call: FileFlex(5,dbID)
FileFlex DB Call: DBSelect(dbID)

-- 6 ----------
Direct Call: FileFlex(6)
FileFlex DB Call: DBCloseAll()

```
-- 7 ---------------------------------------------------------------
Direct Call:                    FileFlex(7,IndexName [, pathSpec])
FileFlex DB Call:               DBUseIndex(IndexName [, pathSpec])

-- 8 ---------------------------------------------------------------
Direct Call:                    FileFlex(8,indexID)
FileFlex DB Call:               DBCloseIndex(indexID)

-- 9 ---------------------------------------------------------------
Direct Call:                    FileFlex(9,indexID)
FileFlex DB Call:               DBSelectIndex(indexID)

-- 10 --------------------------------------------------------------
Direct Call:                    FileFlex(10,recNum)
FileFlex DB Call:               DBGo(recNum)

-- 11 --------------------------------------------------------------
Direct Call:                    FileFlex(11)
FileFlex DB Call:               DBTop()

-- 12 --------------------------------------------------------------
Direct Call:                    FileFlex(12)
FileFlex DB Call:               DBBottom()

-- 13 --------------------------------------------------------------
Direct Call:                    FileFlex(13, seekExpr)
FileFlex DB Call:               DBSeek(seekExpr)

-- 14 --------------------------------------------------------------
Direct Call:                    FileFlex(14, num)
FileFlex DB Call:               DBSkip(num)

-- 15 --------------------------------------------------------------
Direct Call:                    FileFlex(15, startRec, endRec)
FileFlex DB Call:               DBDeleteRecs(startRec, endRec)

-- 16 --------------------------------------------------------------
Direct Call:                    FileFlex(16, startRec, endRec)
FileFlex DB Call:               DBRecallRecs(startRec, endRec)

-- 17 --------------------------------------------------------------
Direct Call:                    FileFlex(17, startRec, endRec)
FileFlex DB Call:               DBZapRecs(startRec, endRec)

-- 18 --------------------------------------------------------------
Direct Call:                    FileFlex(18)
FileFlex DB Call:               DBPack()

-- 19 --------------------------------------------------------------
Direct Call:                    FileFlex(19)
FileFlex DB Call:               DBCount()

-- 20 --------------------------------------------------------------
Direct Call:                    FileFlex(20, recNum)
```

```
FileFlex DB Call:          DBRecordDeleted(recNum)

-- 21 ------------------------------------------------------------
Direct Call:               FileFlex(21)
FileFlex DB Call:          DBCurrRecNum()

-- 22 ------------------------------------------------------------
Direct Call:               FileFlex(22, numFieldName)
FileFlex DB Call:          DBSum(numFieldName)

-- 23 ------------------------------------------------------------
Direct Call:               FileFlex(22, numFieldName) /
                             FileFlex(19)
FileFlex DB Call:          DBAverage(numFieldName)
```

Note: In FileFlex v1.5/v1.6 there is no function number 23. Instead, to compute the average, divide the sum of a given field by the number of records.

```
-- 24 ------------------------------------------------------------
Direct Call:               FileFlex(24, memoField)
FileFlex DB Call:          DBGetMemo(memoField)
```

Note: This function supports decryption. See Office-Quality Encryption for syntax.

```
-- 25 ------------------------------------------------------------
Direct Call:               FileFlex(25, memoField, memoVal)
FileFlex DB Call:          DBWriteMemo(memoField, memoVal)
```

Note: This function supports encryption. See Office-Quality Encryption for syntax.

```
-- 26 ------------------------------------------------------------
Direct Call:               FileFlex(26, fieldName)
FileFlex DB Call:          DBGetFieldByName(fieldName)
```

Note: This function supports decryption. See Office-Quality Encryption for syntax.

```
-- 27 ------------------------------------------------------------
Direct Call:               FileFlex(27, fieldNum)
FileFlex DB Call:          DBGetFieldByNum(fieldNum)
```

Note: This function supports decryption. See Office-Quality Encryption for syntax.

```
-- 28 ------------------------------------------------------------
Direct Call:               FileFlex(28, containerType)
FileFlex DB Call:          DBGetCurrRecVal(containerType)
```

Note: If containerType is "B" or "C", FileFlex attempts to place its values into HyperCard or SuperCard background or card fields, respectively. This is one of the few places where FileFlex makes direct HyperCard callbacks. If you're not in HyperCard or SuperCard, don't pass "B" or "C" (choose some other character, like "L", instead). If containerType is "G", FileFlex attempts

to place its field values into matching global variables, creating them if they don't exist. This function supports decryption. See Office-Quality Encryption for syntax.

```
-- 29 ----------------------------------------------------------
Direct Call:                FileFlex(29)
FileFlex DB Call:           DBListFields()

-- 30 ----------------------------------------------------------
Direct Call:                FileFlex(30, containerType, recNum,
                               value)
FileFlex DB Call:           DBWriteRec(containerType, recNum,
                               value)
```

Note: This function supports encryption. See Office-Quality Encryption for syntax.

```
-- 31 ----------------------------------------------------------
Direct Call:                FileFlex(31, indexName, indexExpr,
                               unique, safety)
FileFlex DB Call:           DBCreateIndex(indexName, indexExpr,
                               unique, safety)

-- 32 ----------------------------------------------------------
Direct Call:                FileFlex(32, indexID)
FileFlex DB Call:           DBReindex(indexID)

-- 33 ----------------------------------------------------------
Direct Call:                FileFlex(33, dbName, numFields,
                               fields, safety)
FileFlex DB Call:           DBCreate(dbName, numFields, fields,
                               safety)

-- 34 ----------------------------------------------------------
Direct Call:                FileFlex(34, indexID)
FileFlex DB Call:           DBCheckIndex(indexID)

-- 35 ----------------------------------------------------------
Direct Call:                FileFlex(35, expr)
FileFlex DB Call:           DBQuery(expr)
```

Note: This function number was assigned to DBLocate, which has been replaced by DBQuery.

```
-- 36 ----------------------------------------------------------
Direct Call:                FileFlex(36)
FileFlex DB Call:           DBCurrDBNum()
```

Note: Returns the current db number

```
-- 38 ----------------------------------------------------------
Direct Call:                FileFlex(38, string, key)
FileFlex DB Call:           DBEncrypt(string, key)

-- 39 ----------------------------------------------------------
```

Direct Call: FileFlex(39, string, key)
FileFlex DB Call: DBDecrypt(string, key)

-- 40 ----------
Direct Call: FileFlex(40, memofield, string)
FileFlex DB Call: DBFindMemo(memofield, string)

-- 41 ----------
Direct Call: FileFlex(41,data,option)
FileFlex DB Call: DBConvertCRLF(data,option)

-- 42 ----------
Direct Call: FileFlex(42)
FileFlex DB Call: DBPlatform()

-- 43 ----------
Direct Call: FileFlex(43, datafile[, path])
FileFlex DB Call: DBDatabaseExists(datafile[, path])

-- 43 ----------
Direct Call: FileFlex(44,indexID)
FileFlex DB Call: DBIndexExpr(indexID)

-- 45 ----------
Direct Call: FileFlex(45, indexID, val, ...)
FileFlex DB Call: DBBuildSeekExpr(indexID, val, ...)

-- 46 ----------
Direct Call: FileFlex(46, indexID, delim)
FileFlex DB Call: DBListIndexFields(indexID, delim)

-- 47 ----------
Direct Call: FileFlex(47)
FileFlex DB Call: DBMaxRecs()

-- 201 ----------
Direct Call: FileFlex(201, globalName)
FileFlex DB Call: DBGetGlobal(globalName)

-- 202 ----------
Direct Call: FileFlex(202,globalName, globalVal)
FileFlex DB Call: DBSetGlobal(globalName, globalVal)

-- 999 ----------
Direct Call: FileFlex(999)
FileFlex DB Call: DBCopyright()

22. Distributing FileFlex

- FileFlex Runtime Edition
- Special Note for Macromedia Director 5 Users
- "R" and "Licensed Runtime Version" from DBVersion
- DBCreate and DBCreateIndex Disabled
- DBUse and DBUseIndex File Encryption Keys
- Opening Distribution Doors Wide

You can license FileFlex for distribution, whether to others within your organization or to customers and users outside.

FileFlex Runtime Edition

Most developers don't develop software solely for their own use. Most develop software for use by others. FileFlex exists so that you can embed relational database functionality within your own projects and applications, and can then provide those finished applications to your users.

You can purchase a license to distribute FileFlex to your users for a nominal fee: $100 per product title you're distributing. This fee is the same whether you're distributing it internally, for profit in a consumer product, as a shareware product, or even giving it away.

Along with your distribution license (more described below), you will receive a special Runtime Version of FileFlex. This version is identical to the Developer Version with a few exceptions.

Special Note for Macromedia Director 5 Users

If you are a registered Director 5 user, Macromedia has purchased for you the right to distribute your productions with FileFlex Lite at no additional fee. You are still required to agree to the licensing terms and send in your license form, but you will not be asked to pay the $100 per-product fee.

Further, FileFlex Lite users do not need to use the special Runtime Edition of FileFlex in the products you distribute. You may take the FileFlex Lite engine and distribute that with your application.

Please note that this special privilege is limited to distribution of the Lite edition--if you choose to upgrade to the professional edition with access to an unlimited number of records, you will need to pay a runtime fee for distribution of runtime software.

"R" and "Licensed Runtime Version" from DBVersion

In the professional edition, you can issue a DBVersion function call [FileFlex(0) in the direct XFCN interface] and get back a version number. In the Runtime Version, when you issue a DBVersion call, you'll get the version number with an "R" appended, as in "1.7R". Also, the full version string will include the phrase "Licensed Runtime Version", telling you and your users that they have a legitimate copy of the FileFlex runtime.

DBCreate and DBCreateIndex Disabled

Because we expect you to distribute pre-constructed databases to your customers, we've eliminated the DBCreate and DBCreateIndex functions from the FileFlex Runtime Version. If you want to distribute empty databases, use DBCreate in the Developer Version then use the command:

```
get DBZapRecs(1,DBCount())
```

to clear the database of information. This will give you an empty shell database you can distribute.

Eliminating these two functions will make it harder for users of your applications to pull out the FileFlex XFCNs and use them directly--thereby protecting our extensive product development investment while still allowing you widespread distribution.

FileFlex Lite: If you plan to distribute FileFlex Lite only, you're not subject to this limitation. However, if you ever expect to expand beyond 1,000 records, you should design your software with this limitation in mind.

Note: If you run into some unexpected snag with this limitation, please contact us and we'll try to figure out a good way to handle it.

DBUse and DBUseIndex File Encryption Keys

Even with DBCreate and DBCreateIndex disabled, it was still possible for anyone who had an existing DBF file to make full use of the FileFlex runtimes. And since you can create DBF files in many other products, including FoxPro, FileMaker, Excel, and other tools, there was very little protection for the product.

We hesitated to grant licenses for widespread distribution, especially shareware authors, because anyone who had a copy of the distributed product would, in actuality, have a fully working copy of FileFlex. This was a doomsday scenario for us, because with a small market, each unlicensed copy is less we can invest in creating new versions.

But we're nothing if not creative developers. And a simple, elegant solution presented itself. The key, if you'll pardon the expression, was a very limited form of encryption-but not of the data or the program code. Encryption of the file names.

Here's how it works. The only way FileFlex knows what files to operate on are with the DBUse and DBUseIndex functions. In each case, a filename (or full path) string is passed to the function. The function then looks on disk for a file of the corresponding name and opens it for further processing.

But if the DBUse and DBUseIndex functions expect an encrypted string, which they then translate to your normal file name internally, anyone who gets the FileFlex Runtime as part of your product can't open any files other than those you specify.

Here's an example. Let's say you want to open the file VIDEO.DBF with a DBUse and VIDEOBYNAME.NDX with a DBUseIndex. Here's what the program would look like using the Developer Version:

```
put DBUse("VIDEO") into vidID
if vidID < 0 then
     -- process some kind of error handler
     -- and leave the routine
end if
put DBUseIndex("VIDEOBYNAME") into vidDexID
if vidDexID < 0 then
     -- process some kind of error handler
     -- and leave the routine
end if
```

In the Runtime Version, your code would be identical, except the filenames would be changed (encrypted) to protect the innocent:

```
put DBUse("%#Jd0") into vidID
if vidID < 0 then
     -- process some kind of error handler
     -- and leave the routine
end if
put DBUseIndex("Kdfue&g-mne") into vidDexID
if vidDexID < 0 then
     -- process some kind of error handler
     -- and leave the routine
end if
```

Your data files would still be named VIDEO.DBF and VIDEOBYNAME.NDX respectively, but the runtime version wouldn't know those names until it unscrambled the encrypted file names provided to it.

When you get your copy of the FileFlex Runtime Version with your approved license, you'll get a copy of a program that will generate the encrypted file names. Just type in the normal file names and it'll tell you the encrypted character sequence.

FileFlex Lite: Again, for FileFlex Lite, this restriction does not exist. However, if you ever expect to expand beyond 1,000 records, you should design your software with this limitation in mind.

Opening Distribution Doors Wide

The benefit of this is that we at Component no longer need to live in fear of FileFlex being distributed to thousands of users who can then pull the FileFlex engine and use it on their own.

Because of the protection offered by file encryption keys, we're able to license FileFlex runtimes for distribution by you to anyone you please, whether commercial, shareware, freeware, or however you'd like.

And since the only changes you need make to your code are a few text strings for file names, and since you can generate new strings any time you wish, we think (and hope) this protection won't be too painful for you to tolerate.

Before we go on to the Runtime License itself, we want to thank Brian Molyneaux for taking the time to discuss licensing issues and for his permission to copy some of the material from his WindowScript license agreement for use in ours.

23. FileFlex Distribution Terms and Conditions

Fees Paid--Lite Edition

The following sections apply to users of FileFlex Lite who received it with Director 5 and who wish to distribute FileFlex Lite (limited to 1,000 records) in their software.

You are licensed to run FileFlex on one personal computer per copy of Director 5 purchased.

If you intend to distribute (in any way, including but not limited to internally within your organization, for profit in a consumer product, as a shareware product, or giving it away) any product or program containing FileFlex Lite, you must return the FileFlex Lite runtime license. Your license fee has been paid by Macromedia. No further fee will be required.

If you are a consultant and you are distributing a finished product to clients that is to contain FileFlex Lite, your client must license FileFlex Lite in their name.

There is a license form at the end of this document. Fill this form out and mail it back to Component. You must complete and return the form to Component Software before you may begin distribution of FileFlex Lite.

Fees--Professional Edition

The following sections apply to customers of the full, professional edition of FileFlex who wish to distribute an unlimited record number version of the software.

You are licensed to run FileFlex on one personal computer per copy of FileFlex professional edition purchased. If you are developing software using FileFlex, you must own a licensed professional edition, latest release. If you have more than one developer building software using FileFlex, each must have a purchased copy of the FileFlex professional, latest release.

You may not purchase a runtime license (and thereby be granted permission to distribute the unlimited FileFlex engine in your application) unless you are a Registered User of the latest release of FileFlex, having returned your registration card to Component Software.

If you intend to distribute (in any way, including but not limited to internally within your organization, for profit in a consumer product, as a shareware product, or giving it away) any product or program containing FileFlex, you must license the FileFlex Runtime Version. Your license fee is One Hundred US Dollars (US$100) payable before you begin distribution of your application. No further fee will be required. Net terms purchase orders will not be accepted unless accompanied by full payment.

If you are a consultant and you are distributing a finished product to clients that is to contain the full FileFlex runtime, you must purchase a runtime license or your clients must own a copy of FileFlex for each machine they intend to run the software on. If you are developing software that uses FileFlex that your client intends to distribute, your Client must purchase (if they don't already own) a professional edition of FileFlex, return the registration card and thereby become a registered user, and then license the FileFlex Runtime for the program in their name.

There is a license form at the end of this document. Fill this form out and mail it back to Component Software with a check. You must complete and return the form to Component Software, and receive a signed copy back, before you may begin distribution.

If you distribute your product containing FileFlex before recieving back a signed license agreement from Component, you are liable for the full list price of FileFlex (US$195) for EACH COPY DISTRIBUTED, plus a penalty fee of One Hundred US Dollars (US$100) for EACH COPY DISTRIBUTED until such time as Component has approved and executed a signed license. No penalty fees will be returned. Component reserves the right to withhold licensing and, in that case, you will be required to stop distributing FileFlex. In other words, don't distribute FileFlex unless you have a license.

Terms and Conditions

The following terms apply to users of the full, professional edition FileFlex and FileFlex Lite. For the purpose of this document, the term "program" shall mean the thing you've developed for distribution, whether it might also be called a production, product, program, application, creation, or something else.

This license is non-exclusive and non-transferable and grants the Licensee the right to distribute the FileFlex resources in the product named in the Distribution License, provide the following terms are complied with and for the fee detailed above.

Display of Copyright and Trademark: The following must be displayed to users of your program:

> "This program uses certain copyrighted resources from FileFlex which are included under license from Component Software of Rocky Hill, NJ. The FileFlex resources may not be removed from this program for any reason and may not be modified. Component Software makes no warranty of any kind to users of this program regarding FileFlex.
>
> FileFlex ©1992-96 by David Gewirtz. All rights reserved. FileFlex is a trademark of David Gewirtz under license to Component Software."

The above notice must be displayed in a prominent location in Licensee's product on a title screen, an about box, or other similar location. It also must be displayed prominently in the documentation. Further, any place your copyright notice is displayed (like, for example, on a product package), the copyright notice above (second paragraph only) must be displayed with it.

The first occurance of any mention of FileFlex in each of Licensee's program, literature, or advertising must include a trademark symbol following the word FileFlex (e.g., FileFlex) and the following line must appear in an appropriate location on the same material:

> "FileFlex is a trademark of David Gewirtz under license to Component Software Industries Corp."

You may not use the phrase "The Relational Database Inside" (a trademark of David Gewirtz under license to Component Software), the Component "plug-in module" logo (a trademark of Component Software), or any other Component trade or service marks without special written permission of Licensor.

Licensed Resources--FileFlex Lite Edition: The following FileFlex resources from FileFlex Lite only are covered by this license when licensing the Macintosh version:

```
XTRA      "FileFlex Engine Lite"
```

The following FileFlex Lite files from FileFlex Lite only are covered by this license when licensing the Windows version:

```
XTRA      "FFDIR5LT.X16"
XTRA      "FFDIR5LT.X32"
```

These are the only resources which may be distributed under the terms of this license. You may not distribute any other resources from the Lite edition of the FileFlex product.

Licensed Resources--Professional Edition: The following FileFlex resources from the FileFlex Runtime Version only are covered by this license when licensing the Macintosh version:

```
XFCN      "FileFlex"
XFCN      "SmartFields"
```

```
XTRA        "FileFlex Engine"
```

The following FileFlex files from the FileFlex Runtime Version only are covered by this license when licensing the Windows version:

```
XTRA        "FFRDIR5.X16"
XTRA        "FFRDIR5.X32"
```

These are the only resources which may be distributed under the terms of this license. You may not distribute any other resources from the professional edition of the FileFlex product.

Documentation: No documentation on the use of the licensed FileFlex resources may be provided with Licensee's program.

Title: Licensee acknowledges the FileFlex resources are the sole property of Licensor. Licensee is not granted any title whatsoever in the FileFlex resources.

Indemnification: Licensee agrees to indemnify and hold harmless Licensor from any and all claims made against it regarding the FileFlex resources.

Warranty: No warranty of any kind is extended to Licensee's product. Licensee shall include the following in the program's warranty information or elsewhere in the program in an appropriate place if Licensee does not explicitly state a warranty policy:

This program includes certain resources from FileFlex. THE FILEFLEX RESOURCES ARE PROVIDED "AS IS" AND NO WARRANTY OF ANY KIND IS PROVIDED. IN NO CASE SHALL COMPONENT SOFTWARE OR DAVID GEWIRTZ BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMANGES (OR DAMAGES OF ANY SORT) IN CONNECTION WITH OR ARISING OUT OF THE FURNISHING, PERFORMANCE, OR USE OF THE FILEFLEX RESOURCES.

Acceptance: Licensor reserves the right to refuse to license the FileFlex resources to any party at its sole discretion. If Licensor rejects a license, any fee received will be promptly returned to Licensee. Upon acceptance, Licensor will send Licensee a signed copy of the license agreement.

Effective Date: The effective date of a license will be the date that Licensor signs the agreement.

Termination: If Licensee breaches any of the terms of this license, Licensor may terminate this license by notifying Licensee in writing. Licensee will have 30 days from the date notice is received to correct the breach or otherwise renegotiate this license, otherwise, this license will terminate. In the event of termination, Licensor will NOT return any license fees which may have been paid by Licensee.

Right to Monitor: Licensor may at any time and at its sole discretion, request a copy of Licensee's program covered by this license and/or any related manuals, literature, or advertising copy which make mention of any edition of FileFlex in order to monitor Licensee's compliance with the terms of this license. If Licensor exercises this right, any materials received will be kept

in confidence and will be used solely to monitor compliance with the terms of this agreement. In no event will Licensor redistribute any materials received under this clause nor will Licensor return said materials to Licensee.

Future Versions of Licensee's Program: This license shall apply to only one Licensee program at a time. If more than one version of a program is distributed at the same time (e.g., a "Lite" version and a "Full" version) then each version will require a separate license. However, if a new version is released and the previous version discontinued, this license will continue to be valid and will apply to the new version (e.g., v1.0, v1.1, v1.2.5, etc).

Future Versions of FileFlex: This license shall apply to the version of FileFlex licensed at the Effective Date only. You may continue to distribute the licensed version of the FileFlex runtime after new versions of FileFlex are released. Should you desire to distribute a newer version of FileFlex, you may be required to purchase an upgraded version of the FileFlex Runtime and/or execute a new or revised license agreement. THE FILEFLEX RESOURCES ARE PROVIDED TO LICENSEE "AS IS" AND NO WARRANTY OF ANY KIND IS PROVIDED. FURTHER, LICENSOR MAKES NO REPRESENTATION OR GUARANTEE OF FUTURE UPGRADES OR VERSIONS OF FILEFLEX, WHETHER TO ADD FEATURES, EXPAND COMPATIBILITY, OR FOR ANY OTHER REASON. LICENSEE RECOGNIZES AND ACCEPTS THAT IT IS LICENSING THE CURRENT VERSION OF FILEFLEX ONLY.

Validity: In the event that any portion of this license shall be rendered in a court of law as invalid, unenforceable, or illegal, all remaining portions of this agreement shall remain in effect.

Governing Law: This license shall be governed in accordance with the laws of the State of New Jersey.

Amendments: Licensor may amend this license from time to time as new versions of FileFlex are released. Such amendments will only apply to Licensee if Licensee chooses to incorporate the upgraded version of FileFlex into Licensee's program.

Nothing in this license shall be construed as the formation of a partnership or joint venture between the parties.

This license supercedes any and all oral or written communications between the parties and is the entire agreement between the parties.

A. FileFlex v2.0 Distribution License

If you will be distributing a program or product that uses the FileFlex professional product, you must fill in this license form and agree to the terms stated in the FileFlex Distribution Terms and Conditions chapter of this manual and return this form to Component Software along with a check for the license fee plus $8 shipping/handling if in the US and $18 from outside the US.

Photocopy or reprint this form as needed for additional licenses. No faxes will be accepted. All license requests must be mailed to Component Software Industries Corp., PO Box 201, Rocky Hill, NJ 08553. If your product requires a custom version of FileFlex or otherwise has special licensing requirements, call us at 609-497-4501. We'll do our best to meet your needs. Do not assume that you can modify the terms of the license agreement or this form, send it to us without our approval, and be granted a license. Signed licenses, if approved, will be returned within four weeks of receipt at Component Software. You may call at any time to inquire about status.

This license is between Component Software Corp., (Licensor) acting for and on behalf of David Gewirtz, and:

Licensee:

Name_____ Company_____

Address_____

City_____ State____ Zip _____ Phone_____

Development Environment _____

Mac or Windows (you must purchase a separate license for each) _____

Estimated number of users (for our total user estimates) _____

Licensee wishes to purchase a license to use the FileFlex resources, as set forth in the "FileFlex Runtime Licensing" chapter of the FileFlex Online Guide, with the following program or product:

Program or Product Name_____

Brief Description of Program or Product (attach a product brochure if you prefer):

I agree with and understand the terms of the license stated in the FileFlex Terms and Conditions.

Name (print or type)_____ Title_____

Signature_____ Date_____

Upon acceptance of this license by Component Software and receipt of valid payment of the license fee, a signed copy of this agreement will be returned for your records.

A license is hereby granted to Licensee by Component Software:

Name (print or type)_____ Title_____

Signature_____ Date_____

B. FileFlex Lite Distribution License

If you will be distributing a program or product that uses the FileFlex Lite edition, you must fill in this license form and agree to the terms stated in the FileFlex Distribution Terms and Conditions chapter of this manual and return this form to Component Software.

Photocopy or reprint this form as needed for additional licenses. No faxes will be accepted. All license requests must be mailed to Component Software Industries Corp., PO Box 201, Rocky Hill, NJ 08553. Do not assume that you can modify the terms of the license agreement or this form, send it to us without our approval, and be granted a license. Signed licenses, if approved, will be returned within four weeks of receipt at Component Software. You may call at any time to inquire about status.

This license is between Component Software Corp., (Licensor) acting for and on behalf of David Gewirtz, and:

Licensee:

Name_____ Company_____

Address_____

City_____ State____ Zip _____ Phone_____

Development Environment _____

Mac or Windows (you must purchase a separate license for each) _____

Estimated number of users (for our total user estimates) _____

Licensee wishes to purchase a license to use the FileFlex resources, as set forth in the "FileFlex Runtime Licensing" chapter of the FileFlex Online Guide, with the following program or product:

Program or Product Name_____

Brief Description of Program or Product (attach a product brochure if you prefer):

I agree with and understand the terms of the license stated in the FileFlex Terms and Conditions.

Name (print or type)_____ Title_____

Signature_____ Date_____

Upon acceptance of this license by Component Software, a signed copy of this agreement will be returned for your records.

A license is hereby granted to Licensee by Component Software:

Name (print or type)_____ Title_____

Signature_____ Date_____

C. FileFlex Order Form

Yes, I want to purchase the full, unlimited FileFlex product! Please send me FileFlex today.

Where I got this Inside FileFlex document: _____

Name _____

Company _____

Address _____

City _____ State _____ Zip_____

Country _____

Phone _____

Internet E-mail address _____

The regular purchase price for FileFlex is US$195. The special price for Internet and Macromedia Director users is $119. US shipping and handling is $8 per unit, international is $18. New Jersey recipients, please add applicable sales tax to your order.

OS	Location	Quantity	Price (each)	Total
Mac	USA	_____	X $119 + $7.00	= $ _____
Win	USA	_____	X $119 + $7.00	= $ _____
Mac	International	_____	X $119 + $18.00	= $ _____
Win	International	_____	X $119 + $18.00	= $ _____

You can also order a printed FileFlex manual (limit one per registered copy of FileFlex). Versions are identical for Macintosh and Windows. The special price for FileFlex users is $27. US shipping and handling is $8 per unit, international is $18. New Jersey recipients, please add applicable sales tax to your order.

Location	Quantity	Price (each)	Total
USA	_____	X $27 + $8.00	= $ _____
International	_____	X $27 + $18.00	= $ _____

Sorry, but we cannot send FileFlex without payment nor can we accept purchase orders. We accept VISA, MasterCard, and American Express. International Orders: Please pay with US Currency from a US Bank, Credit Card, or US Money Order. International orders not sent with payment in US currency from a US bank will be returned.

Send order form and payment to:

Component Software Corporation
PO Box 201
Rocky Hill, NJ 08553
609-497-4501
609-497-4008 FAX
info@component-net.com
www.component-net.com