SparkFS

David Pilling

Archimedes Software

SparkFS was written using Acorn's ANSI C compiler release 4, and the Objasm assembler.

This manual was produced using the Ovation DTP program.

My thanks to everyone who helped me with SparkFS.

(4th. printing for version 1.22, 24th February 1993)

If you have any comments, suggestions, bug reports or complaints or would like further copies of this program or details of the many other pieces of software available for the Archimedes, please write to;

David Pilling
P.O. Box 22,
Thornton Cleveleys,
Blackpool.
FY5 1LR.
United Kingdom.

From time to time enhanced versions of programs will appear. You can upgrade your copy to the latest one by sending your original disc with return postage to the above address.

Contents

1. Introduction

SparkFS, is a universal modular archive filing system for RISC OS. It allows access to all the popular archive file types, via a filing system interface.

Archive files

At the mention of 'archive', many peoples minds turn to backing up hard discs. Whilst the archive files SparkFS handles can be used to help do this, their main use is in communications, software distribution and backing up projects. An 'archive file' or 'archive', is a file which contains other files and directories. In communications it is much easier to transfer one file than many. So the archive file is widely used, and life would be difficult without it.

When files are put into archives, it is possible to use various algorithms to reduce their size. Such data compression works by locating redundancies in the data and coding them more efficiently. In communications, time is money, and it is more attractive to transfer a small file than a big one.

For certain types of data e.g. pictures and text, the space savings produced by data compression are so great, that it becomes worthwhile to only ever keep the files in compressed form even on disc. To cope with this situation, yet another type of program has evolved - the compressed filing system.

SparkFS, fulfils the roles of a compressed filing system, and a file archiver in one program.

Many different formats have appeared for archives. Often these are motivated by the special features of different computers, or improvements in compression techniques. SparkFS will cope with most of the file types in current use. Some of them are, Spark and ArcFS files from the Archimedes, Packit, Stuffit and Compactor from the Apple Macintosh, ARJ, Zip, Zoo, PKarc and SEAarc from the IBM PC, UNIX and many other systems and Compress and Tar from UNIX.

Because a program that can handle every file type is likely to be big, SparkFS has been made modular - you only need to install the modules for the archive types you want to use. Some users will want to handle files from other computers most of the time, they can use the full range of features of SparkFS. However, other users whose work is confined to the Archimedes, can turn SparkFS into a compact system. Extra modules can be produced as new archive files appear.

Often archive files will be transferred, using channels that are designed for text only communications. To get binary 8 bit data through such 7 bit paths, a layer of binary to text coding is applied to the files. SparkFS understands many of these files, including uucode, atob, FCET, boo, and HQX.

Getting started

SparkFS is supplied on a single floppy disc. You should make a backup copy of this for day to day use. A text file called ReadMe is supplied on the disc and notes new features and additions to the program. SparkFS can be installed on your hard disc or another floppy by just dragging the !SparkFS application to the desired destination. SparkFS is run by double clicking on the !SparkFS icon, it will then install on the icon bar. Archives can be recognised by the characteristic Spark sprite (a lightning flash), and can be opened by double clicking, or dragging to the icon bar icon. Opening an archive will make a desktop filer window appear showing the contents. Files in this can be used just like those in normal disc windows.

To use SparkFS, you need to have version 3.75 (or greater) of the shared C library in the modules subdirectory of your !System directory (usually on Applications disc 1). If this is not present, the program will fail to run and give an error message. A copy of CLib version 3.75 is included on this disc. To update your System resources (including the CLib module), you should insert the disc, open a window on it and double click the Sysmerge application and follow any instructions that are issued. This will ensure that both SparkFS and other new programs will run correctly.

Overview

SparkFS consists of several programs.

First the filer which sits on the icon bar. This is loaded when you double click on the !SparkFS icon. The filer, provides desktop control of the filing system, allowing new archives to be created, and old ones to be converted to different types. It also supports ASCII to binary conversion.

Secondly, the module SparkFS. This is the core filing system. It provides support such as memory management for the archive modules. SparkFS, knows nothing about individual archive formats. All such intelligence, is in the archive modules. This means that different copies of SparkFS may read and write completely different file types.

Finally, the archive modules. These live in the directory !SparkFS.Modules. The required ones will be automatically loaded when the filer is started up.

2. Using SparkFS

You start using SparkFS by loading it. This can be done by double clicking on the !SparkFS icon, by double clicking on an archive file or directory, or by running SparkFS from a !Boot file of some kind.

After one of these actions, SparkFS will appear on the left of the icon bar. To begin, you will need some archives to use SparkFS on. If you already have some, then you can double click them, and SparkFS will open a window showing their contents - the process is just like clicking on a directory icon.

Files in the archive can then be used in the normal way, double clicked to run, or dragged to applications.

If you don't have any archives, then you need to create one. To do this, click on the SparkFS icon on the icon bar with the Select button. A window like this one will pop-up.

The buttons down the left allow you to choose the type of archive.

Different types of archive are suitable for different uses. For example Spark dir (short for directory) archives are best for storing large files which you wish to update frequently - typically large DTP documents.

File archives are better for storing small files, or files that are not updated often. With experience you will work out which types of archive are best for your work.

Tar and PK arc files are only of interest if you want to swop files with UNIX or PC computers. Zip files can be exchanged with many other computers, but are perfectly useable on Acorn computers. They or Spark files should be your standard file archive.

The window is like a standard Save box. You can type in a file name, and drag the icon at the top to a normal desktop file viewer window. When you do that the archive will be created, and a window showing the contents of it opened.

So to create an archive, click on the icon bar icon with Select, click on the button for the type of archive you want to create, enter it's name, and then drag the Spark icon to where on disc you want your archive creating. An empty filer window will appear. This is a bit like creating a directory folder.

Once you have created an archive, files can be put into it in all the same ways that files are normally saved to disc. For example by dragging from elsewhere on disc, dragging from the save box of an application, or by key-press. Files can be loaded from archives, by double clicking, or dragging.

As files are loaded from archives, they will be automatically decompressed, and as they are saved, they will be compressed. The data compression, is therefore completely transparent.

Another use of SparkFS is reading archives from other machines. Suppose you have a disc with a PC Zip file on, or have downloaded one from another computer via modem. It will probably have file type data, so you can't double click it to open it. To look inside it, drag it onto the icon bar icon. If it can be handled, a filer window will open, it will also have its type set correctly. Files can now be dragged into other programs from inside the archive.

If the file from the other computer had been binary to text encoded, the procedure would be the same, except the Filer would automatically convert the text file to binary before trying to open it as an archive.

Theory

Although there is still intense competition for better practical methods of data compression, there is a complete mathematical theory of the subject. It makes certain predictions which provide useful rules of thumb. For example, there is a limit to how much data can be compressed. The practical effect of this, is that there is usually little point trying to compress something that is already compressed. Often a second attempted compression will expand data.

Secondly, compressed data becomes more random. An uncompressed file may be readable in Edit, whilst a compressed one is so much "alphabet soup". More seriously, completely random data won't compress. That means it will always be possible to find files which don't compress well or at all.

To cope with these situations often the option of a compression method of "No Compression" is offered.

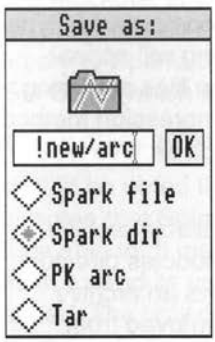
3. SparkFS Filer

Introduction

The filer provides easy control of SparkFS from the desktop. When you double click on !SparkFS, the filer program is run, and installs itself on the left of the icon bar. Various functions can be accessed by dragging files onto the icon bar, or popping up a menu. The menu is opened by pressing the mouse menu button over the icon bar icon.

Info

This leads to a dialogue box giving information about the version number of this copy of SparkFS. The version number should be quoted in any correspondence, and is useful to find out if you have the latest release of the program.

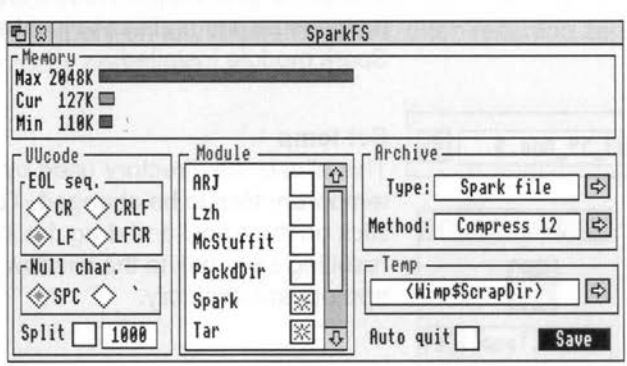


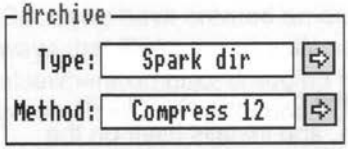
New Archive

This points to a standard save dialogue box which is used to create new archives. To create a new archive, simply drag the archive icon to where you want the new archive to appear. It will be created, and a window will be opened on the contents of the archive.
The buttons allow you to choose the type of archive to be created. The available types will depend on the currently installed archive modules.

Choices

Choosing this will display a large dialogue box which allows various aspects of SparkFS to be configured. Each section of it will now be described.



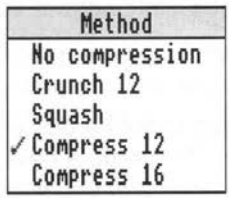


Archive type

This allows the current archive type and compression format to be chosen. Clicking on the two right pointing arrows will produce these two menus.

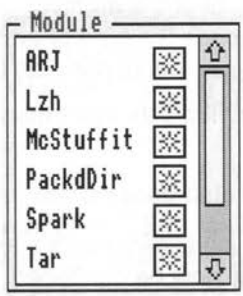
Type

Sets the current archive type. What appears on the sub menu, depends on which archive modules are loaded. Only archive types which can be written will appear. The current archive format will be selected when the new archive dialogue box is opened.



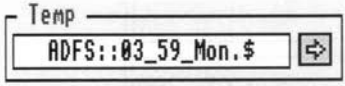
Method

Sets the current compression method for the current archive type. Only compression types which can be used for writing will appear. This option is relevant when files are being added to archives. The compression method is set independently for each archive type.



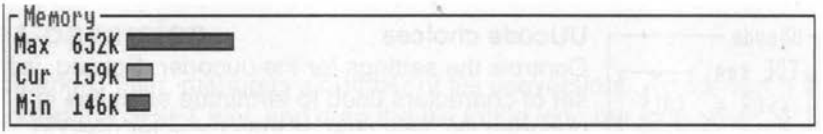
Module

This window shows the available archive modules in the !SparkFS.Modules directory. The button to the right allows an archive module to be installed or removed from memory. This can save lots of memory. For example if you are certain you will never want to open a Macintosh archive, then you can remove (or even delete) the McStuffit module. The same applies to the other modules. For many users it is realistic to just have the Spark module installed.



Set temp

This allows the directory used by SparkFS for temporary files to be changed. To do this click on the arrow and drag the icon from the resulting save box to the window of the appropriate directory.



Memory

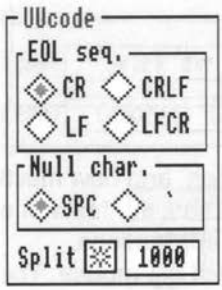
This allows you to control how much memory SparkFS has, and how much it can possibly claim. As usual the more memory SparkFS has, the faster it will work, however, there will then be less memory for other applications. Memory usage is changed by dragging the red sliders with the mouse. The top one 'Max' controls the maximum amount of memory SparkFS will use. The bottom one 'Min' sets the least memory SparkFS will use.

The maximum value is useful, because it stops SparkFS from using up all your memory, instead you may prefer to force temporary files out to disc. The minimum value prevents other programs from grabbing all the memory in the machine, and then trying to access a file from SparkFS. Unfortunately, such programs do exist, so it is necessary to permanently allocate some memory to SparkFS. This particularly applies to the *copy command on RISC OS 2 (which is used when files are dragged around the desktop).

Memory management is discussed in more technical terms in section 4.

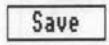
It should be noted that the maximum value, is just that. There is no guarantee that SparkFS will use the maximum amount of memory or that it will be able to. It may be that there is no free memory left when SparkFS tries to extend its usage in which case the maximum value will not be reached. The only certainties, are that SparkFS will claim the minimum setting amount of memory, that it will always have this much memory at its disposal and that it will never use more than the maximum setting.

To give an obvious example, if the maximum value is set to 4096K (4Mb.) on a computer with only 2048K (2Mb.) of RAM, it should not be surprising if SparkFS runs out of memory before it reaches 4096K. More realistically, the amount of free memory on any machine varies, often reaching zero during the use of the *copy command.



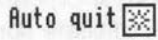
UUcode choices

Controls the settings for the uucoder. Eol seq, is the set of characters used to terminate each line in a uucoded file. Null char, is the character used to represent code 0. If the line split option is selected, uucoded files will not exceed the number of lines entered. Instead the uucoded file will be split into a number of smaller files.



Save Choices

Clicking on Save allows all the configurable options in SparkFS to be saved, so that the next time it is run, they will be reset.



Auto Quit

When selected this will make the filer terminate shortly after being run. The idea is that you can run SparkFS to load modules and so on and then it quits saving space. Obviously once Auto Quit has been selected and the choices have been saved it is impossible to ever get back to the choices window. To allow this, if the ALT key is held down whilst SparkFS is run, the Auto Quit feature will not take effect.

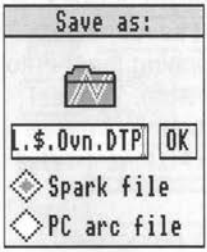
A similar effect can be obtained by putting -quit on the command line used to run !SparkFS. For example, you can make a copy of the !Run file, called !Install in which the final line has a -quit added. Then something like Run !SparkFS.!Install will install the modules, but not the filer.

Quit

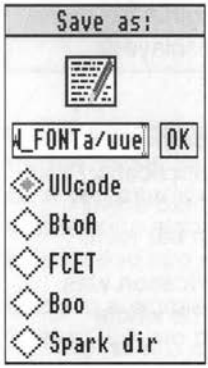
If Quit from the main menu is selected the Filer will terminate. However, SparkFS and all the archive modules will remain active. This can be a useful way of saving memory. The Filer is only vital for creating archives. By using 'FS too' on the sub menu, the entire system can be quit. This will remove SparkFS and all the archive modules from memory.

File Conversion

The SparkFS Filer, manages a number of file conversions. To convert a file, hold down the SHIFT key, and drag the file to the icon bar icon. The available conversions depend on the object type.

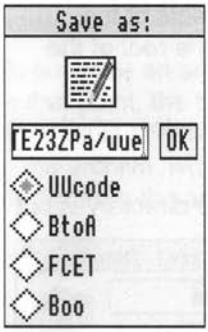


Converting Spark directory archives.
These can be converted into either Spark files or PC (SEA or PKarc type files). The Spark icon should be dragged to where you want the new file to go. The conversion type is chosen by clicking on the button of your choice.

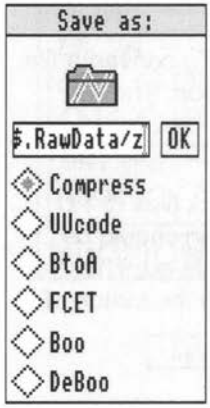


Converting Spark and ArcFS file archives
These can be converted into Spark directories, or turned into a text files in a number of ways. UUcode is the standard UNIX method for converting binary files to text. BtoA is a more efficient method also common in UNIX. In Viewdata terminology, BtoA is sometimes called CET+. FCET is another method used by Viewdata systems. Finally Boo is from the Kermit world.

In addition, ArcFS archives can be converted into Spark or PC arc files.



Converting any archive file.
Similar to Spark files, except they can't be converted into Spark directories.



Converting any file

These options can be applied to any file

Compress, is the standard UNIX method for squashing files. A compressed file, will have the same attributes, date stamp etc. as the uncompressed version but will usually have a 'z' tacked onto the end of its name. Compress files can be expanded by dropping them onto the icon bar icon (without holding down shift). The number of bits used for Compress, will be the same as set for Spark files.

Because Boo files can't be automatically recognised, when you want to convert a Boo file back to binary form, the DeBoo option must be used. The original file name, is contained in the Boo file and will be displayed.

Converting Directories and Applications

The contents of a directory or an entire application can be converted into an archive. As usual, hold down SHIFT and drop the directory on the icon bar icon.

In old versions of SparkFS when an application was converted, an archive was created and the whole application copied into the root directory. This is now only the behaviour if the archive name does not start with a !. If it does begin with a ! the contents of the application directory will be copied into the root of the archive.

Impression documents are a common form of application directory which it is useful to archive.

The radio buttons allow you to choose the sort of archive the directory is put into.

Passwords and global archive handling.

Dropping an archive file on the icon bar whilst holding down the CONTROL key, provides useful information about it, and may let a password be applied.

For archives that can be password protected, this dialogue box will appear. The password can be entered and the method of coding set. The password will become active after RETURN is pressed or the type of encryption is changed.

Passwords only apply to the files written to the archive whilst the password is set. Files can only be read if the correct password for them is currently set.

Password protecting files, is not a step that should be taken lightly. Once you have used a password, it is most unlikely that you will ever get your data back without the password. To SparkFS all files look the same, there is no flag that identifies data as password protected or not. If you attempt to load a file with the wrong password, usually the data compression algorithm will detect that the data is corrupt, or the program the file is loaded into will display rubbish.

Garble is a simple (but quick) method which originally appeared in the PC PK and SEA arc programs.

DES is a sophisticated algorithm devised by the US NSA. It is very difficult to break.

So suppose an archive is dropped on the icon bar, and a password entered. At that point, the files in the archive have not changed - as stated there is no flag that says a file is encrypted. To actually encrypt a file, it must be added to the archive, whilst the password is set. Later it will only be possible to read the file from the archive if the password has first been set for it.

For archives that do not support coding. A simplified version of the window appears. This shows the size and date stamp for the archive, and its type.

Finally clicking on the Test button will check that all the files in the archive can be unpacked.

Configuration

The directory SparkFS.Config, contains a number of files that can be used to configure SparkFS to your requirements.

Choices

A text file of the information stored by the save choices option.

Extensions

This file contains the data used to map file extensions to file types. When an archive from a system like the PC that does not support file types is opened, the file extensions (the group of letters after the final '.' in the file name) are used to generate an Archimedes file type. For example, all files ending in '.txt' will be given file type text. Because '.' cannot appear in Archimedes file names, they are displayed as '/'. This can be useful because the process works both ways, so you can create files with the correct names when your archives are accessed on other systems.

The extensions file consists of a list of entries like this;

```
0xfff txt
```

The first line in the file, is the default type that will be used for files whose extension is unknown, or not present. It is also possible to add to this file, the Macintosh type names and corresponding file types. For example;

```
0xfff TEXT
```

NoCo

Certain types of data won't compress. To save time, it is possible to tell SparkFS not to apply compression to some file types. These file types are put in a list in the NoCo file. For example;

```
0xFF8
0x695
0xDDC
```

FF8 is the filetype given to absolute code files, mostly these will have already been compressed using the 'squeeze' program. 695 is for GIF files which again are already compressed. Finally DDC is the archive file type.

Note that in the NoCo and Extensions files, numbers are entered in hexadecimal in C style notation.

AtExit

This is a file of * commands that will be executed when the SparkFS module is RMKilled. See the *SparkFSAtExit command for more details.

The AutoRun Directory

This can contain an obey file with the same name as an archive module. When the archive module is loaded the obey file will be executed. This mechanism can be used to issue specific *commands to archive modules. It provides a simple way of configuring them. For example you may have a file called 'Zip' with *ZipUseCentralDirectory 1 in it.

ASCII Encoded Binary files

There are many ways of encoding binary files as text - uucode, boo, atob etc. Sometimes when you retrieve the text file, it will not be in a form suitable for decoding. It is important to realise that these text files can be loaded into a normal editor (e.g. Edit) and cleaned up.

For example the text file may have been split into a number of pieces, or have extra text added by a mail system. Using your editor, you can assemble the pieces and strip out the extraneous text.

Often these files will have as the first line, the name of the file that they contain. It can be useful to edit this line if the contained file name is too long or unsuitable in some other way.

4. SparkFS Module

Introduction

This module is normally located in the !SparkFS.Resources.<Country> . directory. It is the heart of SparkFS. On one hand it provides support for the archive modules, and on the other the interface to the rest of the computer.

Before any archives can be accessed using SparkFS, the SparkFS module must be present in memory. You can find out if it is loaded by pressing the F12 key, and typing *modules. *Help SparkFS will produce a list of the *commands supported by SparkFS.

When an archive module is loaded, it will attempt to link itself to the SparkFS module. This means that the SparkFS module must be present before the archive modules are loaded.

All archive file types and compression methods are given unique numbers. SparkFS uses these archive and compression types as parameters to its *commands. A list of the current types is given in Appendix 1.

Memory Management

When the SparkFS module is first started, it reads the value of a variable SparkFS$Memory. This amount of memory is then claimed from the operating system. SparkFS will use the memory intelligently. There are two possible uses for memory, buffering file data, which is more or less optional, and providing work space for the compression code, which is not. This means that the amount of memory reserved, must be big enough for the compression code. At least 32K is needed and 64K should be a practical minimum. SparkFS$Memory is set in the !SparkFS.!Run file.

If the memory reserved is big enough, files will be held in memory whilst they are open. However, if there is not enough memory, they will be created temporarily on disc. The variable Spark$Scrap is used to set where the temporary files are put. In general, it should point at a large area of fast disc space. Spark$Scrap is set in the Filer configuration file, and can be set from the desktop, using the choices dialogue box.

If SparkFS$Memory, is given the value -1, then SparkFS will use memory from the system sprite area. Although this may seem bizarre, it offers efficient memory management. The only snag, is if a program uses the SNew command, which clears the system sprite area. Most RISC OS compliant programs will not do this. However, some older ones - Hearsay 1 for example will. If you want to use such programs, then the other method of memory management should be used.

Image Files

On RISC OS 2, and most conventional operating systems, there are only two types of object, files and directories. With RISC OS 3, Acorn invented a new kind of object, the "image file". An image file is a single file, that contains a directory structure. The motivation was to allow easy access to PC partitions on hard discs, and PC floppy discs. However it is obvious that archives are ideal candidates for image files.

If a filing system tells RISC OS 3 that a given file type is an image file for it, then files of that type will behave almost like directories.

The *command SparkFSImage (see below) is used to control whether SparkFS will register the file types of all the archive modules as images. Usually this command is issued in the SparkFS !Run file.

Image files are a new idea, and some software may become confused by them, so occasionally it may be useful to turn off the image FS feature.

Files are objects of type 1, directories of type 2, and image files of type 3.

ImageFSFix

This module allows applications to be stored as files (saving disc space), but still be used in the normal way by double clicking.

The ImageFSFix module tells the desktop that any image file whose name begins with a ! is a directory. The desktop then treats the image file as an application directory.

So suppose you create an archive file called !Fonts and copy into it, the contents of your usual !Fonts directory, !Boot etc. Normally you would end up with an archive on the desktop called !Fonts showing the Spark file flash icon. Double clicking it would open a viewer on the contents of the archive.

However with ImageFSFix loaded the archive will appear as an application, it will show the !Fonts icon on the desktop and will run when double clicked.

Applications can be converted to archives in the usual way by holding down SHIFT and dragging them to the icon bar. For use with ImageFSFix, the archive name should start with a !.

With ImageFSFix loaded, you can convert applications into archive files and although the applications will be compressed and turned into archives, from the desktop they will look and work as normal.

ImageFSFix is located inside !SparkFS.Resources and can be loaded from the !SparkFS.!Run file by uncommenting the appropriate line.

Paths

By default SparkFS, expects file paths to be in the following format

SparkFS#<full archive path/filename>:$.<path/filename inside archive>

In the archive filename, ':'s are replaced with '#'s

Example

cat SparkFS#SCSI##RISCiX121.$.SparkFS.FS.more_arc:$

If you are using RISC OS 3 and have enabled Image FS support, then paths for archive files, will be just like those for ordinary files - archives behave like directories.

The example would be;

cat RISCiX121.$.SparkFS.FS.more_arc

Filing System *commands

***SparkFS**
Selects SparkFS as the current filing system. Only of relevance to command line operation.

***Back**
Swaps the current and previous directories.

***URD [<directory>]**
Selects a directory as the current user root directory. Default is to restore the URD to $.

***NoURD**
Unsets the user root directory.

***NoDir**
Unsets the current directory.

***NoLib**
Unsets the current library directory.

SparkFS Specific *commands

***SparkFSConvert <source> <destination> <dest. type>**
Converts an archive from one type to another. Parameters are the paths to the original and destination archives and the destination archive type.

***SparkFSFiler_OpenDir <full archive name>**
Used to open a desktop filer on the root directory of an archive.

***SparkFSCreate <type> <name>**
Creates a new archive with the given archive type.

***SparkFSMethod <type> <method>**
Sets the compression method for a given archive type.

***SparkFSEncrypt <archive name> <method> <<password>>**
Sets the encryption method and password for an archive.

***SparkFSExtension <type> <<extension>>**
Sets the RISC OS file type to associate with a given file extension.

***SparkFSNoCo <type>**
Tells SparkFS to not compress a given file type.

***SparkFSTruncate <length>**
Sets the maximum length of displayed filenames. For RISC OS 2 file names must not be longer than 10 characters. The command (used in the !Run file by default) lets you specify this. <length> includes the 0 string terminator.

***SparkFSAtExit <filename>**
Loads a file of commands that will be executed when the SparkFS module is RMKilled. This is usually used to set the RunType variables for archives. For example, if an archive file is run whilst SparkFS is active open a window on it, otherwise Run SparkFS and then try to open the window. This can be seen in the default !Run and Config.AtExit files. By changing these files it is possible to choose which archive file types when double clicked will cause SparkFS to run.

***SparkFSImage <1|0>**
Usually put in the !Run file. SparkFSImage 1 makes SparkFS attempt to register as an image filing system and SparkFSImage 0 deregisters it.

***SparkFSMemory <min>[K] <max>[K]**
Sets the size of the min and max memory slots when using system sprite space for memory. One possible use is to allow SparkFS to be set up without running the filer program. Another possibility is to allow other programs to control the memory used automatically.

5. Spark

Although archive files have been going almost as long as computers. The first archive program and file type to gain great popularity, was SEA's arc for the IBM PC. This occurred because it was the first widely available implementation of the LZW algorithm. LZW coding had first been used in the UNIX compress program and offered much better compression and speed than previous techniques like Huffman coding. SEAs version of LZW was called 'Crunch' - it was a combination of LZW and a much older technique RLE (run length encoding).

LZW works by encoding repeated strings. A parameter of this technique is how many different strings can be used, this depends on the number of bits used for the codes. More bits give better compression, however more memory is required for the compression and expansion process. The lowest number of bits in common use is 12 and the greatest 16.

A variant of SEA arc was devised by Phil Katz. His PC program called PKarc, replaced SEA's 12 bit crunch with a straight 13 bit LZW technique called 'squashing' but retained the SEA file format.

Legal arguments between SEA and Phil Katz eventually led to the large scale abandonment of the SEA standard and the appearance of Phil Katz's Zip program and file. However, in 1988 the SEA format reigned supreme, and seemed a good basis for an archive file format for the Archimedes.

The Spark file format consists of using SEA/PK arc files and adding an extra 12 bytes to each file to contain the RISC OS file attributes. The original format did not support directories in archives. Spark files overcome this problem by storing directories as archives within archives.

Spark programs have always allowed PC format files to be written. The only catch is that you must be careful to use the correct compression methods when creating files to be used with the PC programs.

Later versions of Spark, allowed not just crunch and squash, but also LZW with a larger number of bits "compress". This is like squash (just the same as the UNIX compress program) but can use between 12 and 16 bits.

Spark files are a simple and effective way of archiving things on RISC OS. They offer reasonable data compression and are compact. They are far from ideal for fast file updating and access. In a Spark file the catalogue information (length, type etc.) for each file is stored with it. This extended catalogue provides the kind of robustness which is ideal for communications, but means there can be a long delay when an archive is first opened.

An alternative type of file was devised for the first compressed filing system for the Archimedes - ArcFS. ArcFS files have a central directory near the start. This means that ArcFS archives can be opened quickly.

Storing an entire directory structure in one file, offers many advantages. In almost all conventional filing systems some space is wasted each time a file is created. Although a file may only have 500 bytes in, often 1000 bytes will be allocated on disc for it. Putting everything in one file removes this waste. For small files the saving from this source is bigger than that offered by data compression. However, archive files are slow to update - they have to be compacted. They are also not robust. A single error can destroy many files.

To be able to update files quickly and in a robust way, it is better to store each file on disc separately. The Spark module supports a final type of archive. Spark directory archives, consist of an application directory in which many files are stored in squashed form.

These archive directories are not as efficient in disc use as Spark files, but they are fast and robust. They can be converted to archive files. So you might use directory archives for your work files and periodically convert them to archive files for backup purposes.

The Spark module will read, ArcFS, PKarc, SEA arc, Spark files and directories. It will let you write PKarc, SEAarc and Spark files and directories. Supported compression methods are No compression (files are just stored), 12 bit Crunch, Squash, 12 and 16 bit Compress. Squeeze (Huffman coding). 12 bit Compress will offer the fastest results. All these are compatible with Spark 2.XX. Spark 1.XX does not support Compress.

Spark$Template

This system variable points at a directory containing the files to be put in new Spark directory archives. It can be useful if you want to use a different type of sprite for directory archives or for utilities like 'chkspr' to be automatically included in new archives. The directory should have the same name as the sprites in the !Sprites file that you want renaming to have the same name as the new archive. So if the directory is called Temp, !Sprites may contain

!temp and sm!temp. All !Sprites* (e.g. 22) files will be scanned.

*Commands

***SparkCompress <source> <destination>**
Used to make UNIX compress files.

***SparkUncompress <source> <destination>**
Used to expand UNIX compress files.

6. Zip

Following his legal argument with SEA, Phil Katz developed a new file format and archive program - Zip. Zip is in many ways the current world standard archive format. Programs are available to read and write Zip files on all the common systems.

The Zip module can write Zip files as well as read them. Zip files can be extended to hold extra information, the Zips created by the Zip module contain the RISC OS file attributes, yet can still be read by PC and UNIX Zip programs.

Currently two compression methods are offered. Shrinking is an enhanced form of LZW. Deflation is a more powerful (but slower) technique (see the section on LZH).

Deflation is a development of 'Implosion'. In December 1992, Phil Katz released a version of Zip which supported Deflation, and because it gives superior results to Implosion, Deflation then became the preferred technique to use for Zip files.

Notice this implies that many of the installed copies of Zip or compatible programs will not be able to handle files that use Deflation.

*Commands

*ZipUseCentralDirectory <1|0>

When used with a parameter of 1, this will make the Zip module use the Zip central directory on loading Zip files. The main advantage is that this is much quicker than scanning the distributed directory (or file headers). The disadvantage is that the Central directory must be both present and in agreement with the rest of the file. Using 0 for the parameter will revert to the extended catalogue.

In other words if you have problems with some Zip files it may be worth turning off the use of central directories.

7. Tar

Tar is probably the oldest archive format still in common use. The name shows this deriving from Tape ARchive. Files stored in Tar archives, are not compressed at all. Usually the files are a number of 512 byte blocks long. Each file has a 512 byte header followed by as many 512 byte blocks as are necessary to store it. This gives Tar files their characteristic appearance when loaded into an editor - large areas of null characters.

Because there is no compression of the data in Tar files. It is common to squash a complete Tar with the UNIX Compress program. Since data compression is often more effective on larger files this will give better results than programs which store the data for each file in compressed form. However, it loses the advantage of random access to files.

The often encountered uucoded compressed Tar file (filename.tar.z.uue) will be automatically decoded, uncompressed and opened when dropped on the icon bar. Usually compressed Tar files have names of the form 'filename.tar.z', but sometimes this will be shortened to 'filename.tzr'

Ordinary UNIX Tar files do not retain RISC OS file attributes. It is possible to use some of the usual header for this purpose. Various methods have been proposed for doing this. They are shown on the Compression method menu when Tar is the current archive type.

Finally the method 'comma' involves tacking a comma followed by the RISC OS filetype onto the end of the file name. This has the considerable advantage, of allowing the Tar file to be read by standard UNIX Tar programs whilst also preserving file type information on the Archimedes. The comma and file type string will not appear under RISC OS.

Tar files have a RISC OS file type (0xC46) and corresponding icon.

8. Zoo

Whilst SEA and PKarc were shareware programs. Zoo was devised as a completely public domain archive standard. Implementations exist for all the usual hardware platforms, however the most common places to find Zoo files are the Amiga and UNIX.

The original Zoo program used a slight variation on the standard UNIX LZW

technique, later versions have borrowed algorithms from LZH.

9. ARJ

ARJ (the name derives from the initials of the author Robert Jung) is the latest standard to appear in the PC world. The main advantage of ARJ is that it offers amongst the best data compression currently available.

10. LZH

The original inventors of the LZW algorithm were Lempel and Ziv. They actually proposed two algorithms around the same time. One was the first to be widely exploited in the LZW form. Their other algorithm took longer to catch on, but its superior data compression, has made it the basis of all the latest generation of data compressors. The first practical implementation was by a group of Japanese enthusiasts. They made their data compressor into an archive standard.

11. PackdDir

PackdDir is the name of an archive program written for the Archimedes by John Kortinck. It's motivation, is to allow a directory structure to be rapidly converted into an archive file. The compression algorithms used are a slight variation on UNIX Compress - LZW with an adjustable number of bits.

12. McStuffit

Introduction

The filing system on the Macintosh is novel, in that files consist of two components or forks. The resource and data forks. There is a standard for handling such files on non-Macintosh systems (or for sending them via modem). A "Macbinary" file, consists of a 128 byte header, followed by the data and resource forks.

The Macintosh also has its own type of binary to ASCII conversion - HQX or BinHex. Files can be loaded in either HQX, Macbinary or Data fork form. Only the latter two should have type Archive, the first is a text file.

The original Macintosh archive format was Packit. This is fairly primitive and unsuitable for random file access. It uses Huffman data compression. By far the most popular Macintosh format is Stuffit. This is broadly comparable to SEA and PK arc, later versions add one of the LZH algorithms. The newest format is Compactor. This is analogous to Zip.

*Commands

*McStuffitMode <mode>

The McStuffit module supports a * command, to decide if files should consist of the resource, data or both forks. If the file consists of both forks, it will be created in Macbinary format.

*McStuffitMode controls which file fork will be accessed. Possible values are 1 Resource fork 2 Data fork 3 Both 4 Longest.

Mode 4, makes whichever of the forks is biggest represent the file. This can be useful when files have all the data of interest in varying forks, and nothing in the other fork.

This command should be used before any Macintosh archives are opened. Although changing the value at any other time will not of itself do any damage, applications may find files larger or smaller than they think with unfortunate consequences.

It is possible to set the mode value in an Obey file in the !SparkFS.AutoRun directory.

Appendix 1:Archive types

SPARK1	1	Spark files
SPARK2	2	Spark directories
PKARC	3	PK arc
ZIP	4	Zip
ZOO	5	Zoo
LZH	6	LZH
TAR	7	Tar
ARCFS	8	ArcFS
ARJ	9	ARJ
PIT	10	Packit
SIT	11	Stuffit
SITD	12	Stuffit Deluxe
SITSX	13	Stuffit self extracting
COMPAC	14	Compactor
COMPACSX	15	Compactor self extracting
PACKD	16	PackdDir

Appendix 2:Error Messages

SparkFS has filing system number &42

It gives the following errors

&014200	Not enough free memory *Try using the Memory section of the choices dialogue box to give SparkFS more memory.*
&014201	Bad parameters passed internally
&014202	SparkFS does not support this command
&014203	Archive not found
&014204	Too many archives open *There are open files in too many different archives.*
&014205	Can't handle this type of file
&014206	Bad archive
&014207	Archive already exists *You have tried to create an archive with the same name as a file that already exists.*
&014208	Name must start with! *Spark directory archive names must begin with a !.*
&014209	Catalogue not found
&01420A	Archive is read only
&01420B	Corruption in compressed data
&01420C	Bad compression method parameter
&01420D	Bad number of bits parameter
&01420E	Unsupported compression method
&01420F	Unsupported number of compression bits
&014210	Bad encryption method parameter
&014211	Password too long
&014212	Password missing
&014213	Unsupported encryption method
&014214	Cannot create scrap file *Check the setting of Spark$Scrap. Ensure !Scrap is installed.*

&014215	File is packed with a bad number of bits
&014216	No currently selected directory
&014217	No currently selected library
&014218	Error writing data
&014219	Error reading data
&014220	Not a compressed file
&014221	Bad Mac parameter
&014222	SparkFS memory corrupted
&014223	Unidentified header type found
&014224	Central directory offset inconsistent
&014225	Central directory has an inconsistent number of files
&014226	Central directory has an inconsistent file offset
&014227	Central directory not found
&014228	End directory not found
&014229	Archive does not support this command
&01422A	Too many files open
&01422B	Bad image file handle

Standard Filing system error messages

&0142B4	Directory not empty
&0142BD	Access violation
&0142C2	File open
&0142C3	Locked
&0142C4	Already exists
&0142C5	Types don't match
&0142CC	Bad file name
&0142D6	Not found

Appendix 3:SWI Calls

SparkFS supports a number of SWI calls.

```
#define FS_SWI  0x445C0
#define USERSWI (FS_SWI+5)
```

0 - Identify archive

Entry	Exit
r0=0	r0=type
r1=archive path	

1 - Read archive entries

This is passed the path to an archive, and the path to a directory inside the archive. In addition a wild-carded file name can be used.

Entry
r0=1
r1-> archive path i.e. path to archive file on external FS
r2->buffer to write information to
r3= number of objects to write
r4= position in catalogue to start reading from
r5= size of buffer
r6->wild carded file name to match. Can be NULL to match all.
r7->path to directory inside archive

Exit
r4=position in catalogue to read from next
r3=number of entries written to buffer

Format of entry

offset		
0	load address	
4	exec address	
8	length	i.e. real uncompressed size
12	size	i.e. compressed size
16	header version	loosely related to the compression method
20	attributes	
24	object type	1=file, 2=directory
28	object name	0 terminated, padded with 0's to word boundary.

Operation is similar in concept to OS_GBPB 10

```
#define LINKSWI    (FS_SWI+0)
```

0 - Add new link
Entry
r0=0
r1->flink
r2->private workspace
r3=compatibility level

1 - Remove link
Entry
r0=1
r1->flink

#define MEMSWI (FS_SWI+1)

0 - Allocate

Entry	Exit
r0=0	r0=0 failed
r1->anchor	
r2=size	

1 - Extend

Entry	Exit
r0=1	r0=0 failed
r1->anchor	
r2=size	

2 - Free

Entry	Exit
r0=2	
r1->anchor	

3 - Set

Entry	Exit
r0=3	r1=min
r1=min	r2=max
r2=max	r3=cur

#define INFOSWI (FS_SWI+2)

0 - Archive Info

Entry	Exit
r0=0	r1->archive structure
r1-> archive pathname	

1 - Module Info

Entry	Exit
r0=1	r0=0 no module, -1 no more modules
r1=n module number	r1->archive info
	r2->compression info
	r3->code info
	r4->convert info
	r5->module base

#define CODESWI (FS_SWI+3)

0 - CRC 16 block

Entry	Exit
r0=0	r1=crc out
r1=crc in	
r2->block	
r3=length	

1 - CRC 32 block

Entry	Exit
r0=1	r1=crc out
r1=crc in	
r2->block	
r3=length	

2 - Open Encrypt

Entry
r0=2
r1=type 1==Garble 2==DES
r2->password

3 - Encrypt block
r0=3
r1=type 1==Garble 2==DES
r2->block
r3=length

4 - Decrypt block
r0=4
r1=type 1==Garble 2==DES
r2->block
r3=length

5 - Close Encrypt
r0=5
r1=type 1==Garble 2==DES

#define UTILSWI (FS_SWI+4)
0 - Parent of directory
Entry
r0=0
r1->archive
r2=child

Exit
r0=parent

1 - Set no of files in directory
Entry
r1->archive
r2=no of files

2 - Insert entry
Entry
r0=2
r1->archive
r2=file number
r3=directory
r4=size

3 - Remove entry
Entry
r0=3
r1->archive
r2=file number
r4=size

4 - Map extension to type
Entry
r0=4
r1->extension

Exit
r0=file type

5 - Calc dir lens
Entry
r0=5
r1->archive

6 - Open scrap file
Entry
r0=6
r1=mode

Exit
r0=handle

7 - Close scrap file
Entry
r0=7
r1=handle

Bibliography

The Data Compression Book,
Mark Nelson, M&T Books 1991, ISBN 0-13-202854-9
A readable summary of the current state of play, with C source code for all the standard methods and explanations of how they work.

Data Compression (methods and theory),
James A Storer, Computer Science Press 1988, ISBN 0-88175-161-8
Theoretical survey from one of the contributors to the field. Contains Pascal source code for the methods discussed.

Text Compression,
Timothy C. Bell, John G. Cleary and Ian H. Witten,
Prentice Hall 1990, ISBN 0-13-911991-4
Excellent and readable survey of data compression, from some of the pioneers in arithmetic coding. Covers the theoretical basis of all the current techniques and provides comparisons of efficiency.