

*This is a draft specification which will be
released as a controlled document once
approved*

Resource Manager

A Network Filing System

DISTRIBUTION LIST

John Juer	
Dave Storey	
Ian Hughes	

By
M Fox

DOCUMENT TYPE
Software Design Specification

File: roc:

Contents

1	Scope	4
2	Related Documents	4
3	Introduction	4
4	Protocol	4
4.1	FileOpen	5
4.2	FileRead	6
4.3	FileWrite	7
4.4	FileAppend	7
4.5	FileClose	8
4.6	FileDelete	8
5	The File Manager	8
6	File System Programming Interfaces	9
6.1	Common Features	9
6.2	Local	9
6.2.1	LfsOpen	9
6.2.2	LfsRead	10
6.2.3	LfsWrite	10
6.2.4	LfsAppend	10
6.2.5	LfsClose	10
6.2.6	LfsDelete	10
6.3	Remote	11
6.3.1	RfsOpen	11
6.3.2	RfsRead	11
6.3.3	RfsWrite	11
6.3.4	RfsAppend	11
6.3.5	RfsClose	12
6.3.6	RfsDelete	12
6.3.7	RfsGetStatus	12
6.3.8	RfsFree	12

For information only

REVISION HISTORY

Revision	Date	Changes
1	September 9, 1992	First formal issue

1 Scope

This document describes the protocol and programming interface of a file system that may supported on many different targets. This document is intended for the implementors of file systems and remote file system interfaces. The document includes the definition of local file system interface to encourage the writing of portable applications using local files.

2 Related Documents

- [1] HP081079 EILIN: Application Layer
- [2] HP024105C301 Communications Messaging Services

3 Introduction

The file protocol and associated programming interfaces described here are to used on a variety of products and target executives and operating systems to provide a unified view of "file systems". This then allows bulk transfer of data between different instruments and computer systems without embedding any knowledge about the characteristics of the file system. The result is a simple distributed file system. This then allows a utilities such as download of userware or upload of error logs without the need for special code.

4 Protocol

The following services will be supported, derived from EILIN ([1]).

- Open a file
- Close a file
- Read from a file
- Write to a file
- Delete a file
- Append to a file

Each service is described below, including an ordered listing of the fields of the service in network order. All multi-byte quantities are to be in network byte order, that is most significant byte first.

All services will return an error code which will be one of:

Symbol	Value	Description
OK	1	Operation was successful
NOSUCHFILE	2	No such file and/or directory
NOPERM	3	No permission for required operation
BADREF	4	No such FileReference
EOF	5	Unexpected EOF encountered during operation
EXISTS	6	File already exists
TOOMANY	7	Too many files already open
BADNAME	8	Invalid file name
SYSTEM	9	General file system error

These will constitute the protocol CMS [2] “FILE”.

This protocol does not impose any constraints about how a file name and/or directory is structured or composed. A file and directory specification is simply a character string. It will be a local issue for each target as to what constitutes a valid file name, however a set of conventions for file and directory naming will be used (§7).

4.1 FileOpen

Opens a file for read and write operations.

This service will open a file.

Its request will consist of:

Field	Type	Value(s)
Service	uint8	2 - RequestFileOpen
NameLength	uint16	Length of FileName
FileName	byte[NameLength]	Name of file to be opened
OpenMode	uint8	1 - CREATE 2 - READ 3 - WRITE

The open mode implies:

CREATE Open a new file for reading and writing provided a file of the same name does not already exist.

READ Open a file for read access only. The file will not be modified.

WRITE Open an existing file for read and write. It will be an issue for the local file system as to whether it is possible to extend an existing file by writing beyond the end of the file. If a file is extended and an unwritten “hole” is left then the values in the “hole” are undefined.

Its response will consist of:

Field	Type	Value(s)
Service	uint8	3 - ResponseFileOpen
FileError	uint8	OK NOSUCHFILE (not CREATE) NOPERM EXISTS (CREATE only) TOOMANY BADNAME
FileReference	uint16	0 if != OK else != 0
FileSize	uint32	0 if FileError = NOSUCHFILE Size of file in bytes <i>Is this still needed now we have append</i>

These services do not impose any restrictions about shared access to files. This is an issue for the local file system (§6.2).

4.2 FileRead

Used to read the contents of a file starting from a given position in the file.

Its request will consist of:

Field	Type	Value(s)
Service	uint8	4 - RequestFileRead
FileReference	uint16	Value returned by ResponseFileOpen
BytePosition	uint32	Position in file from which to start read
Length	uint16	Number of bytes to read

Its response will consist of:

Field	Type	Value(s)
Service	uint8	5 - ResponseFileRead
FileError	uint8	OK BADREF EOF
Length	uint16	0 if != OK and != EOF Number of bytes read
DataOctets	byte[Length]	Data read

It is not an error to return less bytes than requested, nor does this imply that the end of file has been reached. Only a read starting at or beyond the end of file should ever return an EOF.

4.3 FileWrite

Used to write data to a file starting from a given position in the file.

Its request will consist of:

Field	Type	Value(s)
Service	uint8	6 - RequestFileWrite
FileReference	uint16	Value returned by ResponseFileOpen
BytePosition	uint32	Position in file from which to start write
Length	uint16	Number of bytes to write
DataOctets	byte[Length]	Data to write

Its response will consist of:

Field	Type	Value(s)
Service	uint8	7 - ResponseFileWrite
Length	uint16	Number of bytes written
FileError	uint8	OK NOPERM (File opened for READ only) BADREF EOF

4.4 FileAppend

Used to write data to a file starting from the end of file.

Its request will consist of:

Field	Type	Value(s)
Service	uint8	8 - RequestFileAppend
FileReference	uint16	Value returned by ResponseFileOpen
Length	uint16	Number of bytes to write
DataOctets	byte[Length]	Data to write

Its response will consist of:

Field	Type	Value(s)
Service	uint8	9 - ResponseFileAppend
Length	uint16	Number of bytes written
FileError	uint8	OK NOPERM (File opened for READ only) BADREF EOF

4.5 FileClose

Used to close a file opened with the FileOpen service

Its request will consist of:

Field	Type	Value(s)
Service	uint8	10 - RequestFileClose
FileReference	uint16	Value returned by ResponseFileOpen

Its response will consists of:

Field	Type	Value(s)
Service	uint8	11 - ResponseFileClose
FileError	uint8	OK BADREF

4.6 FileDelete

Used to delete a file which exists in a file system.

Its request will consist of:

Field	Type	Value(s)
Service	uint8	12 - RequestFileDelete
NameLength	uint16	Length of FileName
FileName	byte[NameLength]	Name of file to be deleted

Its response will consists of:

Field	Type	Value(s)
Service	uint8	13 - ResponseFileDelete
FileError	uint8	OK NOSUCHFILE NOPERM

5 The File Manager

A file manager is a task which provides an external view of a file system, ie allows remote access to the file system. Each node that wishes to permit remote access to its file system must instantiate a file manager by convention this will have the application entity name "FILEMAN" [2]. The file manager is a generic program that uses the local file system programming interface (§6.2). This then provides a direct mapping between the local and remote views of the file system. The file manager may have a limit on the number of files open concurrently. The file manager may also choose to close a file if no operation has been performed upon a open file after a timeout period.

6 File System Programming Interfaces

This section describes the programmers view of the file systems.

6.1 Common Features

The two interfaces (local and remote) consist of very similar interfaces which both map onto the file services. There are a number of common types used by both interfaces, these are:

```
enum FsOpenMode = { FsModeCreate,  
                    FsModeRead,  
                    FsModeWrite } ;  
  
typedef uint32 FsBytePosition ;  
  
typedef uint16 FsBufferLength ;
```

6.2 Local

For each target file system a C++ interface consisting of a set of functions that map directly onto the file services must be provided. The details of how the functionality is provided is a target dependent issue, and so only the interface is described. The implementation of the local file system may limit the number of currently open files. It is also a target dependent issue as to whether files may be "shared". In the case of targets which already have a mature file system (eg unix, DOS) then these services will easily map onto the existing file system.

Many operations return an error (**LfsError**) which will be one of :

LfsErrorOk Operation successful.

LfsErrorNoSuchFile No such file and/or directory.

LfsErrorNoPerm No permission for operation.

LfsErrorBadFile Bad LocalFile*

LfsErrorEOF Unexpected EOF encountered

LfsErrorExists File already exists

6.2.1 LfsOpen

Used to open a file on the local file system.

```
LfsFile* LfsOpen ( char* Name, FsOpenMode Mode ) ;
```

6.2.2 LfsRead

Used to read the contents of a local file starting from a given position in the file.

```
LfsError LfsRead ( LfsFile* File,  
                  FsBytePosition Position,  
                  FsBufferLength Length, char* Buffer,  
                  FsBufferLength* NumRead ) ;
```

6.2.3 LfsWrite

Used to write data to a local file starting from a given position in the file.

```
LfsError LfsWrite ( LfsFile* File,  
                   FsBytePosition Position,  
                   FsBufferLength Length, char* Buffer ) ;
```

6.2.4 LfsAppend

Used to write data to a local file starting from the end of file.

```
LfsError LfsAppend ( LfsFile* File,  
                    FsBufferLength Length, char* Buffer ) ;
```

6.2.5 LfsClose

Used to close a file opened with LfsOpen.

```
LfsError LfsClose ( LfsFile* File ) ;
```

6.2.6 LfsDelete

Used to delete a file from the local file system.

```
LfsError LfsDelete ( char* Name ) ;
```

6.3 Remote

This provides a users programming interface for remote files. It may also be used to access local files provided that a local file manager exists.

All operations consist of a request to perform the operation which must be checked later to determine if the operation was successful.

Many operations return an error (`RfsError`) which will be one of :

`RfsErrorAccepted` Request for operation accepted.

`RfsErrorFileBad` Bad `RfsFile*`

`RfsErrorInProgress` Operation already in progress on file.

6.3.1 RfsOpen

Open a remote file. A NULL pointer indicates that there insufficient local resources (`RfsFiles`) to do the open. A non-NULL `RfsFile*` pointer indicates that the operation has been accepted locally.

```
RfsFile* RfsOpen ( CmsNodeType* Node, char* FileName, FileOpenMode Mode ) ;
```

6.3.2 RfsRead

Used to read the contents of a remote file starting from a given position in the file.

```
RfsError RfsRead ( RfsFile* File,  
                  FsBytePosition Position,  
                  FsBufferLength Length, char* Buffer ) ;
```

Until the operation has completed or resulted in failure the Buffer must continue to exist.

6.3.3 RfsWrite

Used to write data to a remote file starting from a given position in the file.

```
RfsError RfsWrite ( RfsFile* File,  
                   FsBytePosition Position,  
                   FsBufferLength Length, char* Buffer ) ;
```

6.3.4 RfsAppend

Used to write data to a remote file starting from the end of file.

```
RfsError RfsAppend ( RfsFile* File,  
                    FsBufferLength Length, char* Buffer ) ;
```

6.3.5 RfsClose

Used to close a remote file previously opened with **RfsOpen**.

```
RfsError RfsClose ( RfsFile* File ) ;
```

6.3.6 RfsDelete

Used to delete a file which exists in a file system.

```
RfsFile* RfsDelete ( CmsNodeType* Node, char* FileName ) ;
```

A NULL pointer indicates that there insufficient local resources (**RfsFiles**) to do the open. A non-NULL **RfsFile*** pointer indicates that the operation has been accepted locally.

6.3.7 RfsGetStatus

```
RfsStatus RfsGetStatus ( RfsFile* File, FileBufferLength* NumBytes ) ;
```

The status of an operation may be derived from local conditions, network conditions or from the remote file manager. The function also returns the number of bytes read or written so far (if a read or write is in progress or has completed). The possible values of the status are :

RfsStatusOk Last operation completed successfully.

RfsStatusNotExist No such **RfsFile***

RfsStatusInProgress Operation in progress.

RfsStatusTimeout The last operation timed out.

RfsStatusNoSuchFile No such file reported in the remote file system.

RfsStatusNoPerm No permission for the operation.

RfsStatusEOF Unexpected EOF encountered

RfsStatusExists File already exists.

6.3.8 RfsFree

Used to release a **RfsFile*** allocated by either **RfsOpen** or **RfsDelete**.

```
void RfsFree ( RfsFile* File ) ;
```

After completion of the **RfsFree** the **RfsFile*** is no longer valid.

7 Conventions

This section defines conventions that are recommended to be used for the file systems. These are not constraints imposed by the protocol.

```
base_name : [A-Z][A-Z0-9]* ; /* Maximum 8 characters */
extension : [A-Z0-9]* ; /* Maximum 3 characters */
file_name : base_name [ "." [ extension ] ] ;
directory_name : base_name ;
directory_path : "" | "/" directory_path | directory_name "/" directory_path ;
device_name : "" | [A-Z] ":" ;

full_file_name : [ device_name ] [ directory_path ] file_name ;
```

→ ○ ←