ECMA

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

STANDARD ECMA-85

VIRTUAL FILE PROTOCOL

September 1982

Free copies of this document are available from ECMA,
European Computer Manufacturers Association
114 Rue du Rhône – 1204 Geneva (Switzerland)

ECMA

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

STANDARD ECMA-85

VIRTUAL FILE PROTOCOL

September 1982

BRIEF HISTORY

This Standard ECMA-85 is one of a set of standards for Open Systems Interconnection (OSI). Open Systems Interconnection standards are intended to facilitate homogeneous interconnection between heterogeneous information processing systems. The standard is within the framework for the co-ordination of standards for Open Systems Interconnection which is defined by ISO/7498.

This ECMA Standard is based on the practical experience of ECMA member companies world-wide, and on the results of their active participation in the current work of ISO and national standard bodies in Europe and the USA. It represents a pragmatic and widely based consensus.

A particular emphasis of this Standard is to specify the homogeneous externally visible and verifiable characteristics needed for interconnection compatibility, while avoiding unnecessary constraints upon, and changes to, the heterogeneous internal design and implementation of the information processing systems to be interconnected.

In the interest of a rapid and effective standardization, the standard is oriented towards urgent and well understood needs. It is intended to be capable of modular extension to cover future developments in technology and needs.

Adopted as Standard ECMA-85 at the General Assembly of June 7-8, 1982.

TABLE OF CONTENTS

Table of Contents (cont'd)

Table of Contents (cont'd)

Table of Contents (cont'd)

1 General

1.1 INTRODUCTION

This Standard ECMA-85 is one of a set of standards for Open Systems Interconnection. Open Systems Interconnection standards are intended to facilitate homogeneous interconnection between heterogeneous information processing systems. The standard is within the framework for the coordination of standards for Open Systems Interconnection which is defined by ISO 7498.

This ECMA standard is based on the practical experience of ECMA member companies world-wide, and on the results of their active participation in the current work of ISO and national standard bodies in Europe and the USA. It represents a pragmatic and widely based consensus.

A particular emphasis of this standard is to specify the homogeneous externally visible and verifiable characteristics needed for interconnection compatibility, while avoiding unnecessary constraints upon and changes to the heterogeneous internal design and implementation of the information processing systems to be interconnected.

In the interest of a rapid and effective standardisation, the standard is oriented towards urgent and well understood needs. It is intended to be capable of modular extension to cover future developments in technology and needs.

1.2 SCOPE

This ECMA Virtual File Protocol Standard:

- defines a virtual file model (see section 2),
- defines operations on the virtual file model as abstract interactions between two virtual file service users, via the virtual file service (see section 3),
- defines the protocol to support the above service and its mapping into the underlying presentation service (see section 4),
- specifies the requirements for conformance with this protocol (see section 5).

The standard defines only what is needed for a basic file transfer. It provides the consistent technical basis for further virtual file protocol standards with extended scope.

The standard defines what is needed for compatible interconnection between information processing systems. It does not define local interactions between a virtual file service user and the virtual file service. It in no way defines interlayer interfaces.

This Standard is for the application layer of Open Systems Interconnection (see ISO 7498).

1.3 REFERENCES

ISO 7498	Data Processing - Open Systems Interconnection - Basic Reference Model.
ECMA-6	7-Bit Input/Output Coded Character Set.
ECMA-84	Data Presentation Protocol.
ISO 2014	Writing of calendar dates in all-numeric form.
ISO 3307	Representations of time of the day.

1.4 GENERAL OVERVIEW

The role of the virtual file protocol is to allow a standardized handling of files in the context of Open Systems interconnection, in a way independent from the location of files and from the specific host systems.

This is made possible by the definition of a generalized file model called the Virtual File: all the services provided by the virtual file protocol are applied to virtual files. Each implementation provides a local mapping of virtual files into its own real files.

The services provided by the virtual file protocol can be divided into two categories:

- virtual file access, which allows the inspection and manipulation of all or parts of the contents of a virtual file,
- virtual file management, which allows the inspection and manipulation of the attributes associated with a virtual file.

File transfer applications use a combination of virtual file access and virtual file management to move the contents and associated properties of a virtual file from one system to another. Definition of a file transfer application service is outside the scope of this standard; an example is given in appendix E to help understand the relationship between a file transfer application and this standard.

2 Virtual File

2.1 VIRTUAL FILE MODEL

This section describes a general open ended model, but it presents only those aspects which concern the file transfer.

2.1.1 General principles

Standardized file descriptions are achieved by using the concept of Virtual Filestore, which allows open working between dissimilar real files. The virtual filestore is built on the premise that standardized representations can be set up for a file or a collection of files (filestore). It allows the absorption of the differences in style and specifications by using mapping functions to relate the standard descriptions to local resources and vice-versa. Any particular system can then interwork with other different systems in terms which can be mutually understood. Although the objective is to deal with real files, the model and protocol will deal only with virtual files. The mapping between real and virtual files is considered a local concern. This is illustrated in figure 2.1/1.

The description of the virtual filestore is resolved into a set of distinct properties called attributes; the values of these attributes identify or describe the files to be handled. The real files which are mapped onto the virtual filestore description and the mechanism by which the mapping is implemented will vary considerably from case to case, but the exchange of information will still be in terms of the virtual filestore attributes.

Appendix H describes, as an example, the possible mapping of the real file attributes defined in the ECMA labelling and file structure standards for files on portable data interchange media.

2.1.2 Virtual file addressing

Each virtual file is identified in an unambiguous way by means of its name, defined in BNF as:

<virtual-file-name>::=<virtual-filestore-name><file-name>

<virtual-filestore-name>::= global title necessary to establish an application connection with the VFS entity that supports this virtual filestore.

<file-name>::= name that must be passed to the VFS entity in order to enable it to properly identify the file. The file name is unique within this virtual filestore. It is transferred as a character string. It may explicitly or implicitly define the media on which the file resides: there is no separate parameter for this.

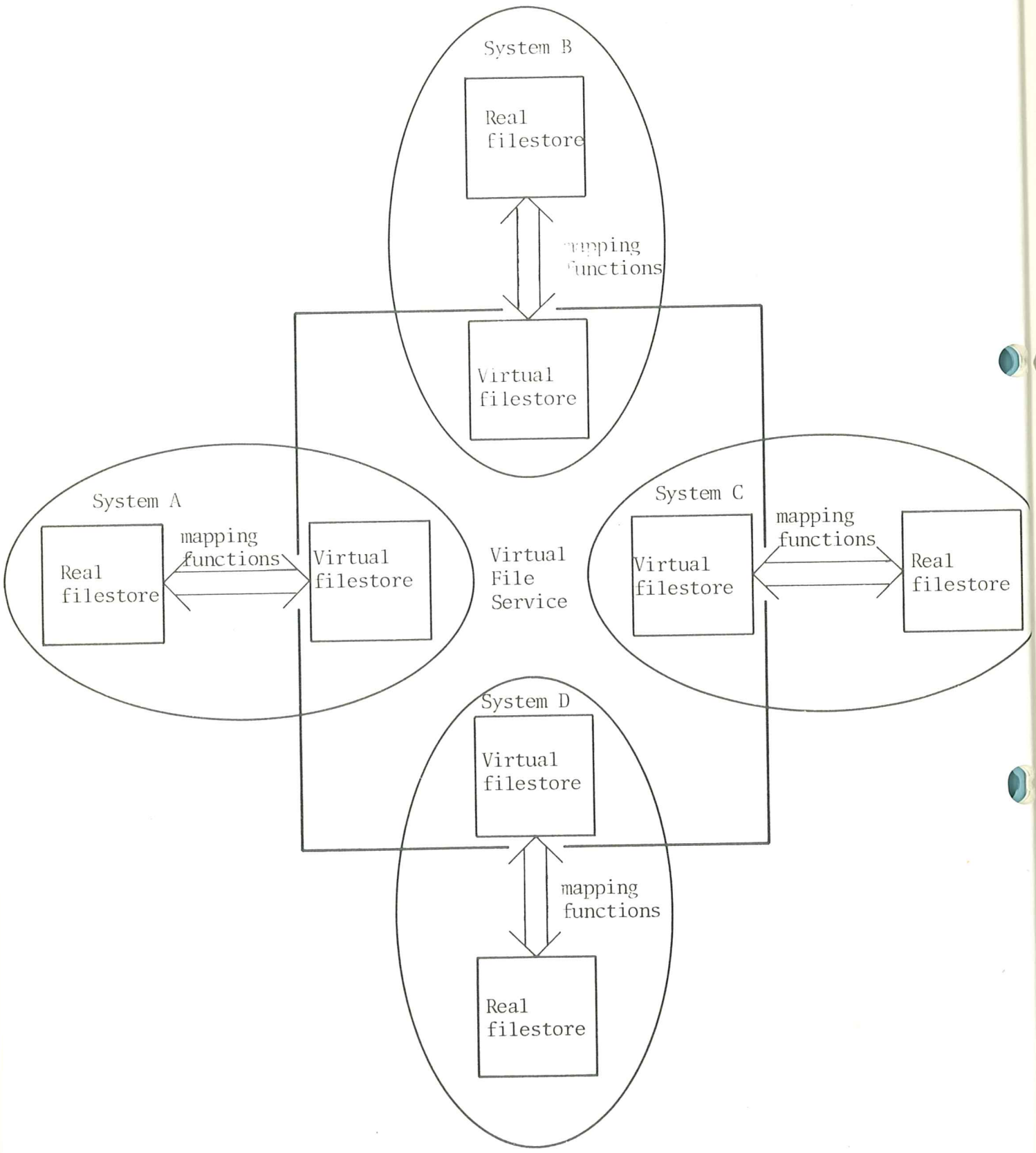

Fig. 2.1/1 - Virtual Filestore in an OSI Environment

2.1.3 Virtual file attributes

2.1.3.1 Attributes overview

The description of a virtual file is made by means of defining a set of distinct properties called attributes. The values of the attributes define, identify and describe the virtual file completely, allowing correct mapping between real and virtual files.

The attributes of a virtual file can be classified into 3 categories:

Container attributes: for addressing and protection. They include:

- file name
- file passwords
- access-control-list

History attributes: automatically maintained by the filestore to reflect the accumulated file activity.

Contents attributes: for description of the structure of the file contents. They are the most numerous and are subdivided for better modularity into subcategories as follows:

Global attributes:	file-structure file-data-type file-current-size file-maximum-size
Record attributes:	record-sequence direct-access record-size-type record-size
Key attributes:	key-position key length key-field-rank
Field attributes:	field-rank field-data-type field-size field-complement

2.1.3.2 File name

Each file within a given Virtual Filestore is identified in an unambiguous way by means of its name. Because the naming policy depends on the application environment, the Virtual File Service has to allow any types of names.

Value: character string

2.1.3.3 File passwords

Within the Virtual Filestore, security mechanisms are provided to protect a given file against unauthorized access. One of the commonly used mechanisms are passwords. As for the file name, the passwords attribute depends on local naming policies. So the Virtual Filestore has to allow any types of passwords.

Value: character string

2.1.3.4 Access control list

Another security mechanism is the access control list, identifying authorized users of the file and their specific permissions. This attribute defines whether the file is protected by an access control list or not. If yes, the file is created with only one authorized user having all permissions: the creator. Further updates of the access control list as well as types of permissions, are not standardized in this version and are considered to be a local matter.

Value: symbolic (yes/no)

2.1.3.5 History

The history attributes are for statistical purpose only. They are locally created at file creation time and locally updated for every subsequent operation on the file contents (either local or remote). They only appear in the Read Attributes service.

The following attributes may be supported:

- Date and time of creation
- User identification for creation
- Date and time of last access
- User identification for last access
- Date and time of last modification
- Total number of accesses
- Total number of modifications

Value: character string, for identifications
date-time
numeric, for the last two

2.1.3.6 File structure

Defines the internal structuring of the file. Two models are supported: unstructured and flat. Other models of file structure will be considered in future versions (see appendix F).

An unstructured file has no visible internal structure: it is composed only of a sequence of octets or characters (see 2.1.3.7).

A flat file is composed of records, without any relationship between records other than sequencing. This category includes most conventional files and the relational model.

Value: symbolic (flat/unstructured)

2.1.3.7 File data type

This attribute describes the type of data stored in the file. The value "heterogeneous" means that field descriptions are provided in the file attributes. Otherwise the file contents are considered as homogeneous and field descriptions are not supplied. The value "character" means that the file data is entirely composed of characters. The value "transparent" means that the file data is entirely composed of octets, the content of which is undefined.

When the file structure is "unstructured", this attribute cannot have the value "heterogeneous".

Value: symbolic (character/transparent/heterogeneous)

2.1.3.8 File current size

This attribute specifies the approximate amount of user data currently in the file. The value provided at creation time indicates the minimum amount of space to be allocated to the file. After creation, the value of this attribute is updated locally every time the file grows. The unit of measure is kilo-octets if file-data-type is "transparent", kilo-characters if file-data-type is "character", and records if file data type is "heterogeneous".

Value: numeric

2.1.3.9 File maximum size

This attribute defines the maximum size to which the file can grow. Crossing this boundary may cause an error. The unit of measure is the same as for the file current size.

Value: numeric

2.1.3.10 Record sequence

This attribute describes the order of the records when the file is sequentially accessed. Not applicable to unstructured files.

Value: symbolic (by position/by key)

NOTE 1

Since different systems may adopt different data syntaxes, it is not possible to guarantee that the key sequence is preserved in a file transfer.

2.1.3.11 Direct access

This attribute describes by which means the records are directly accessible. Not applicable to unstructured files.

Value: symbolic (by no means/by position/by key)

NOTE 2

"Record sequence" and "direct access" together comprise the so called "file organization". The table below shows the equivalence between some well known organizations and the values of these attributes.

Table 2.1/1 - Equivalence to file organizations

	Record sequence	
Direct access	by position	by key
by no means	sequential	-
by position	relative	-
by key	random	index sequential

2.1.3.12 Record size type

This attribute defines whether all records have the same size or not. Not applicable to unstructured files.

Value: symbolic (fixed/variable)

2.1.3.13 Record size

This attribute defines the maximum or fixed record size. Applicable only if file structure is "flat" and file data type is not "heterogeneous". If file structure is "flat" and file data type is "heterogeneous", record size is deduced from the field descriptions.

The unit of measure is octets if file data type is "transparent", characters if file data type is "character".

Value: numeric

2.1.3.14 Key position

This attribute is applicable only if at least one of the attributes record sequence and direct access has the value "by key" and file data type is not "heterogeneous". It defines the beginning of the key within a record, as the number of octets or characters (see 2.1.3.7) from the beginning of the record. A value of N means there are N octets or characters before the key.

Value: numeric

2.1.3.15 Key length

This attribute is applicable only when key position is. It defines the fixed length of the key, assumed contiguous, in octets or characters (see 2.1.3.7).

Value: numeric

2.1.3.16 Key field rank

This attribute is applicable only if at least one of the attributes record sequence and direct access has the value "by key" and file data type is "heterogeneous". It defines the key by the field rank (2.1.3.17) of the key field (a single field) in the field descriptions.

Value: numeric

2.1.3.17 Field rank

This attribute is applicable only if file data type is "heterogeneous". Like all the following attributes, it is specified once per record field. It indicates the rank of the field in the record, "1" being the leftmost field of the record.

Value: numeric

2.1.3.18 Field data type

This attribute is applicable only when field rank is. It specifies the data type of an elementary field.

There are two major data types: strings and numbers.

Strings are further subdivided according to their elements: character strings, octet strings (for transparent data) and bit strings.

Numbers are further characterized by a scale (fixed or floating point), a base (binary or decimal) and a sign option (signed or unsigned).

Value: character string
octet string
bit string
unsigned fixed binary
signed fixed binary
unsigned fixed decimal
signed fixed decimal
signed floating point

NOTE 3

It is assumed that the location of the fixed point is known only by the users.

NOTE 4

The base attribute for numbers is introduced at the application level because it influences the performance of file processing by user programs: users select the base according to the most frequent type of processing of the field (editing or computation).

NOTE 5

Floating point decimal is excluded.

2.1.3.19 Field size

This attribute is applicable only when field rank is. It specifies the size of an elementary field. For floating point data type, it designates the size of the fraction. For the other data types, it designates the total field size.

The unit of measure depends on the field data type, in the following way:

character string: characters
octet string: octets
bit string: bits
fixed binary: bits (sign not included)
fixed decimal: decimal digits (sign not included)
floating point: digits (currently restricted to hexadecimal base).

2.1.3.20 Field complement

This attribute is field data type dependent. In this standard, it applies only to "floating point" and specifies the size of the exponent, in bits.

Value: numeric

2.1.4 Summary of attributes applicability

The following table specifies the conditions for applicability of the different file attributes. Absence of condition means the attribute is always applicable.

The conditions are encoded as follows:

A: file structure value is not "unstructured"
B: file data type value is "heterogeneous"
C: record sequence value is "by key"
D: direct access value is "by key"
E: field data type value is "floating point"

attribute	conditions
file-name	
file-passwords	
access-control-list	
history-attributes	
file-structure	
file-data-type	
file-current-size	
file-maximum-size	
record-sequence	A
direct-access	A
record-size-type	A
record-size	A and not B
key-position	A and (C or D) and not B
key-length	A and (C or D) and not B
key-field-rank	A and (C or D) and B
field-rank	A and B
field-data-type	A and B
field-size	A and B
field-complement	A and B and E

2.2 VIRTUAL FILE MODEL SUBSETS

2.2.1 General

Subsets of the virtual file model are defined to achieve simplification and variety control. The virtual file model is subsetted in the following way:

- a kernel model, composed of all the attributes and attribute values that all implementations must support.
- optional extensions, each composed of an additional set of attributes:

  Unstructured files extension
  Field descriptions extension

Each of these options can be supported independently of whether the other is supported.

The class-of-filestore parameter of the F-CONNECT service (see 3.2.1.1) specifies the optional virtual file model extensions that may be used on the connection. The Primary proposes its maximum requirement and the Secondary responds with its capabilities within the proposed set of extensions. The Primary may decide to terminate the con-

nection if the selected set is too restricted.

Two sorts of file attributes are provided within the Virtual Filestore. Mandatory attributes are required to be supported by each Virtual Filestore implementation, while the support of optional attributes is not required. Even if it does not support an optional attribute within its own local filestore, a VFS entity may have to support this attribute when referring to a remote filestore which supports it.

In the following subclauses, mandatory attributes or attribute values are denoted with a "*" on the left of the attribute name. The others are all optional.

2.2.2 Kernel

The attributes and attribute values included in this subset are:

*file-name
file-passwords
access-control-list
history attributes

*file-structure (flat)
*file-data-type (character/transparent)
file-current-size
*file-maximum-size

*record-sequence (by position)
record-sequence (by key)
*direct-access (by no means)
direct-access (by position/by key)
*record-size-type (fixed)
record-size-type (variable)
*record-size

key-position
key-length

2.2.3 Unstructured files extension

The attributes and attribute values included in this extension are:

*file-structure (unstructured)

2.2.4 Field descriptions extension

The attributes and attribute values included in this extension are:

*file-data-type (heterogeneous)

key-field-rank

```
*field-rank
*field-data-type (character and octet string,
                  unsigned fixed binary number)
 field-data-type (all other types)
*field-size
 field-complement
```

3 Service

3.1 SERVICE OVERVIEW

3.1.1 Roles of partners

In a VFS connection between two VFS users, the dialogue is always asymmetrical, i.e. the two VFS users play different and complementary roles. The initiator of the VFS connection (called the Primary) is the one who defines the work to be performed on the Virtual Filestore through the connection: it is in a more or less direct relation with the end user on behalf of whom it acts. The other VFS user (called the Secondary) is there to execute the work proposed by the Primary and report to it; it resides on the same system as the Virtual Filestore and has no relation with an end user, except through the Primary.

There are however particular phases, in the life of a VFS connection, where the file data is being transferred: the direction of the transfer can vary from one such phase to another. During these phases, a temporary leadership is assumed by the Sender of data, while the Receiver acts as a slave, that is it can only accept data or report abnormal conditions. As soon as a file data transfer phase is completed, the roles switch back to Primary and Secondary.

3.1.2 Dynamic structuring of a VFS connection

The VFS allows operations on only one file at a time on a given VFS connection. Multiple files can be handled concurrently through several parallel VFS connections. Furthermore, within one VFS connection, operations on the current file are executed one after the other in the order of submission. This is necessary to keep total control on the sequence of events.

The work performed on a VFS connection can be dynamically structured as a set of nested enclosures, which must be opened in the hierarchical order and closed in the reverse order. If the VFS connection breaks or is abnormally terminated by one VFS user, all the currently opened enclosures are considered as being implicitly closed.

The enclosures are the following, in the hierarchical order, starting from the outmost one:

- Connection enclosure: the VFS connection exists (from establishment to termination of the VFS connection).
- File enclosure: a current file exists (from successful file selection to file release). A connection enclosure contains any number of file enclosures (including none).

- Open enclosure: the current file is ready for data access (from successful file opening to file closing). A file enclosure contains any number of open enclosures (including none).
- Transfer enclosure: file data is being transferred (from transfer begin to normal or abnormal transfer end). For file transfer purpose, only one transfer enclosure per open enclosure is necessary. Once a transfer enclosure is open, the roles switch to Sender and Receiver until the transfer enclosure is closed.

3.1.3 Connection facility

The connection facility service provides for establishment and release of the VFS connection.

At connection establishment, there is a negotiation of the particular class of service to be used and of any user special conventions that may be agreed (see Appendix G). Renegotiation is not provided in this standard.

Connection termination is normally requested by the Primary, when all work is completed. However, in emergency cases (e.g. system shutdown), the connection can be abnormally terminated by either VFS user at any point in time. The connection can also be accidentally lost: this is reported to both users by the VFS.

3.1.4 File management

The file management service provides to the Primary all file services with the exception of file data transfer. This includes:

- Selection of a current file, by designating an existing file.
- Release of the current file when all work on it has been completed.
- Creation of a new file, with specified attributes. This file then becomes the current file. An option specifies what to do if a file with the same name already exists.
- Deletion (and release) of the current file.
- Retrieval of selected attributes of the current file.
- Opening of the current file for data access, with a specified lock. Only sequential data access is provided in this standard: read the file or write to the file (either after its current content or overwriting it).

- Closing of the current file, with release of the lock.
- Starting of the file data transfer: this enters the next part of the service.

3.1.5 File data transfer

The file data transfer service provides all transfer of file data. The data flow is one way from Sender to Receiver. Facilities are provided for:

- Orderly termination of the data transfer by the Sender (with acknowledgement by the Receiver).
- Abnormal termination of the data transfer by either user.
- Checkpointing and checkpoint acknowledgement.
- Immediate restart of the data transfer to a negotiated previous position. This form of restart can be requested by either user.

3.1.6 Parameter value setting

A parameter is specified in the service in order to set its value.

The value of a parameter can be set in one of two ways: either by selection or by negotiation.

Selection

A number of parameters have their value set by selection, i.e. one of the two VFS users unilaterally sets the value. When this value is not acceptable to the other VFS user, the work in progress cannot be completed. Example: file name.

Negotiation

The remaining parameters have their value set by negotiation between the two VFS users. Negotiation is performed in one handshake: the initiator of the negotiation proposes its acceptable values and the other user responds with the selected value (or with a rejection of the service when no acceptable value may be selected). Negotiation rules are defined for each negotiable parameter.

Unset parameters

It is not necessary to supply all potential parameters in every case: when a parameter is omitted, its value remains unset. When the value is needed in the course of the processing, the result depends on whether the parameter has a default value or not.

- If it has a default value, this default value is automatically assigned to it. This is a matter of protocol encoding (see 4.3.2.3).
- If it has no default value, the parameter has no significant value and this may lead to rejection of the service if a value was mandatory in the particular circumstances.

Initial values

At the beginning of an enclosure, the initial values of parameters are as follows:

- all parameters set in a higher level enclosure retain their current values;
- all parameters that can be set by services of the enclosure being open are unset.

3.1.7 Recovery

The purpose of the restart facility is to avoid complete repetition of a transfer which was interrupted before completion. The prime objective is to minimize the amount of data retransmission, while eliminating any loss or duplication of data in the receiving file.

Restart may be either immediate or deferred.

An immediate restart occurs within a transfer enclosure, on request of either VFS user. In such case, negotiation of the restart position and retransmission occur immediately: there is no exit from the transfer enclosure.

A deferred restart occurs after the transfer enclosure has been terminated and the file closed and potentially released. The termination can be involuntary (failure of network or either node) or it can be voluntary (shutdown or desire to execute higher priority work). In any case all enclosures that have been closed will have to be reopened before the transfer can be resumed. All file attributes and processing attributes should be set to the same values as initially. Restart position will be negotiated while entering the open enclosure. A deferred restart can occur within the same or a different VFS connection. A deferred restart can be initiated only by the entity which was the Primary of the interrupted activity.

Open identification

Since a restart can be deferred, there is a need to relate several successive transfer enclosures as belonging to the same file transfer activity. As a result, each new file

transfer request will be identified by a unique identification, which will be recalled every time a deferred restart is attempted. This identification is supplied by the Primary when entering the open enclosure. It is unique only within the Primary and has to be prefixed by the address of the Primary to be unique within the Secondary.

The open identification is forgotten (and can therefore be reused for another activity) once the associated file transfer request is either successfully completed or abandoned (non recoverable failure).

Restart position negotiation

The restart position will be designated in terms of record position in the data flow, starting from the beginning of file data transfer. This type of designation is not applicable to unstructured files: for unstructured files, the position will be a character or octet position in the data flow (according to file-data-type value), starting from the beginning of file data transfer.

The negotiation rules are the following:

- one partner proposes a restart position,
- the other can agree or specify an earlier position.

The sender can abstain from specifying a restart position, since restart is normally driven by the receiver. However, the sender may specify a restart position to force retransmission of data.

Checkpoints

As seen above, checkpoints (i.e. the marking of particular points in the data flow) are not required for resynchronization of the data flow. However, they are useful whenever context information (i.e. information necessary to properly restart data transfer after a recoverable failure) is saved: checkpoint acknowledgement by the receiver will allow the sender to purge its context information (thus avoiding uncontrolled growth). Restart at a position before an acknowledged checkpoint is allowed, but might involve full retransmission of the data, dependent on implementations.

3.1.8 Grouping of service structures

The file management services are composed only of type 2 service structures (see Appendix C). Several of these will be needed to initialize or terminate file data processing: the Primary should normally not initiate a new request primitive before having received the confirmation

to the previous request. However, a better response time may be achieved, if needed, by issuing a series of consecutive request primitives and then waiting for the corresponding response primitives: this feature is called grouping of service structures and is applicable only to file management services. The use of this feature is an option, selected by the Primary at connection establishment.

Though the main concern and the initiative rest with the Primary, grouping is also visible to the Secondary, because of error situations. The rules for service grouping are the following:

- the end of each service group must be explicitly indicated (End of group service).
- the Secondary normally generates an indication primitive for each request, as if there were no grouping. However, in case of a negative response primitive, no more indication primitives will be generated, except for the end of group indication; the end of group response primitive will therefore directly follow the negative response primitive.
- the Primary processes each confirmation primitive individually and only after issuing the end of group request primitive. It must remember the sequence of request contained in the group, in order to progress its state machine after each positive confirmation primitive as if the next request primitive had just been sent. When the end of group confirmation is received, there is no more pending request.
- a new group can be initiated only after the end of group confirmation has been received.

3.1.9 List of services

The table below lists all the services of the VFS. For each service, it specifies its type (see Appendix C), the user who can initiate it (PR: Primary, SC: Secondary, SN: Sender, RC: Receiver), and its purpose.

Table 3.1/1 - Virtual file services

Service	Type	Init.	Description
			CONNECTION FACILITY
F-CONNECT	2	PR	Establish VFS connection
F-RELEASE	2	PR	Clean release of VFS connection
F-DISCONNECT	1	PR,SC	Unclean release of VFS connection
F-ABORT	3	-	Loss of presentation connection
			FILE MANAGEMENT
F-END-GROUP	2	PR	Delimit a service group
F-SELECT-FILE	2	PR	Establish current file
F-RELEASE-FILE	2	PR	Release current file
F-CREATE-FILE	2	PR	Create new file
F-DELETE-FILE	2	PR	Delete current file
F-READ-ATTRIBUTES	2	PR	Read attributes of current file
F-OPEN-FILE	2	PR	Open current file for data access
F-CLOSE-FILE	2	PR	Close current file
F-BEGIN-DATA	2	PR	Start transfer of file data
			FILE DATA TRANSFER
F-DATA	1	SN	Transfer file data
F-END-DATA	2	SN	End transfer of file data
F-ABORT-DATA	2	SN,RC	Abort transfer of file data
F-RESTART	2	SN,RC	Resynchronize transfer in progress
F-CHECKPOINT	2	SN	Request acknowledgement

3.2 SERVICE DESCRIPTION

3.2.1 Primitives

The service is described by using the service description notation and terminology defined in Appendix C.

3.2.1.1 The F-CONNECT service

Purpose: to establish a VFS connection. This includes establishment of a presentation connection, negotiation of the class of VFS service to be used and of any particular work option that will remain valid for the entire duration of the connection.

Structure: type 2

Initiated by: initiator becomes Primary

Parameters:

	REQ	IND	RESP	CONF
Diagnostic	x	x	B	U
Filestore-name	D	x	x	x
Authentication	D	U	x	x
Protocol-identifier	D	U	x	x
Protocol-version	D	U	D	U
Class-of-filestore	D	U	D	U
Class-of-service	D	U	D	U
Grouping-option	D	U	x	x
Special-conventions	D	U	D	U

Parameter descriptions:

Diagnostic: see 3.2.2

Filestore-name: global title necessary to establish a connection with the application entity supporting the virtual filestore (to be supplied to presentation service). No default.

Authentication: management information necessary for security and accounting of the connection (user identification and password, account identification).

Protocol-identifier: applicable value is "VFP".

Protocol-version: designates the VFP version. Negotiable. Applicable value for this version is "1".

Class-of-filestore: specifies the class of virtual file model which will be used on this connection. Negotiated. See 2.2 for values.

Class-of-service: specifies the class of the Virtual File Service which will be used on this connection. Negotiated. See 3.3 for values.

Grouping-option: specifies whether the grouping of service structures will be used (see 3.1.8).

Special-conventions: specifies which special conventions can be used on this connection. Negotiated. Successful negotiation allows the use of the special-information parameter in any service where it applies. Else, use of this facility is illegal. See Appendix G for purpose of defining special conventions.

3.2.1.2 The F-RELEASE service

Purpose: to obtain a clean release of a VFS connection. The effects are non-disruptive. The Secondary cannot refuse the release of the connection.

Structure: type 2

Initiated by: Primary

Parameters: this service contains no parameters.

3.2.1.3 The F-DISCONNECT service

Purpose: to obtain an unclean release of a VFS connection (emergency situation) and specify the reason for this. The effects are disruptive.

Structure: type 1

Initiated by: Primary or Secondary

Parameters:

	REQ	IND
Diagnostic	D	U

Parameter descriptions:

Diagnostic: see 3.2.2. Only one diagnostic may be provided, with a single diagnostic supplement (type 2).

3.2.1.4 The F-ABORT service

Purpose: to signal a spontaneous disconnection. The effects are disruptive.

Structure: type 3

Initiated by: either VFS entity

Parameters:

	IND	IND
Diagnostic	U	U

Parameter descriptions:

Diagnostic: see 3.2.2. Only one diagnostic may be provided, with a single diagnostic supplement (type 2).

3.2.1.5 The F-END-GROUP service

Purpose: to delimit the end of a service group. The Primary cannot initiate any other service (except F-DISCONNECT) between the F-END-GROUP request and confirmation primitives. The same constraint applies to the Secondary, between the F-END-GROUP indication and response primitives. This service applies only if the grouping option has been selected in F-CONNECT.

Structure: type 2

Initiated by: Primary

Parameters: this service contains no parameters.

3.2.1.6 The F-SELECT-FILE service

Purpose:to establish an existing file as the current file for the connection. In case of success, any further service will implicitly refer to this file, until either a F-RELEASE-FILE or a F-DELETE-FILE. This service is subject to grouping.

Structure: type 2

Initiated by: Primary

Parameters:

	REQ	IND	RESP	CONF
Diagnostic	x	x	B	U
File-name	D	U	x	x
File-passwords	D	U	x	x
Special-information	D	U	D	U

Parameter descriptions:

Diagnostic: see 3.2.2.

File-name: specifies the identification of the file. See 2.1.3.2.

File-passwords: specifies any password(s) needed to obtain access to the file. See 2.1.3.3.

Special-information: transparent data obeying special conventions between VFS users (see special-conventions parameter in F-CONNECT).

3.2.1.7 The F-RELEASE-FILE service

Purpose: to release the current file of the connection. The file enclosure is ended even if the diagnostic severity is "failure". This service is subject to grouping.

Structure: type 2

Initiated by: Primary

Parameters:

	REQ	IND	RESP	CONF
Diagnostic	x	x	B	U
Special-information	D	U	D	U

Parameter descriptions:

Diagnostic: see 3.2.2.

Special-information: transparent data obeying special conventions between VFS users (see special-conventions parameter in F-CONNECT).

3.2.1.8 The F-CREATE-FILE service

Purpose: to create a new file and establish it as the current file for the connection. In case of success, any further service will implicitly refer to this file, until either a F-RELEASE-FILE or a F-DELETE-FILE. This service is subject to grouping.

Structure: type 2

Initiated by: Primary

Parameters:

	REQ	IND	RESP	CONF
Diagnostic	x	x	B	U
File-name	D	U	x	x
File-passwords	D	U	x	x
File-attributes	D	U	x	x
Clash-option	D	U	x	x
Reversible-mapping	D	U	x	x
Special-information	D	U	D	U

Parameter descriptions:

Diagnostic: see 3.2.2.

File name: specifies the identification of the file. See 2.1.3.2.

File-passwords: specifies any password(s) to be used subsequently for protection of the file. See 2.1.3.3.

File-attributes: specifies the values to be assigned to file attributes other than name and passwords. See 2.1.3 for the complete list of attributes and applicability rules. History attributes are not specified in this service.

Clash-option: specifies what to do if the supplied file-name corresponds to an already existing file.

Legal values:

Reject: the existing file is kept; diagnostic severity is failure.

Keep: the existing file is kept and selected; diagnostic severity is success.

Replace: the existing file is replaced by the newly defined file; diagnostic severity is success.

Reversible-mapping: specifies that the mapping between virtual and real file must be such that all the attributes supplied with this F-CREATE-FILE are returned unchanged on any subsequent F-READ-ATTRIBUTES (unless changed by a user). If reversible mapping is requested, the F-CREATE-FILE must be rejected if the Secondary cannot guarantee it.

Legal values: yes/no.

Special-information: transparent data obeying special conventions between VFS users (see special-conventions parameter in F-CONNECT).

3.2.1.9 The F-DELETE-FILE service

Purpose: to delete and release the current file of the connection. The file enclosure is ended (with the file possibly not deleted) even if the diagnostic severity is "failure". This service is subject to grouping.

Structure: type 2

Initiated by: Primary

Parameters:

	REQ	IND	RESP	CONF
Diagnostic	x	x	B	U
Special-information	D	U	D	U

Parameter descriptions:

Diagnostic: see 3.2.2.

Special-information: transparent data obeying special conventions between VFS users (see special-conventions parameter in F-CONNECT).

3.2.1.10 The F-READ-ATTRIBUTES service

Purpose: to return specified attributes of the current file. This does not include file-name and file-passwords. This service is subject to grouping.

Structure: type 2

Initiated by: Primary

Parameters:

	REQ	IND	RESP	CONF
Diagnostic	x	x	B	U
Requested-attributes	D	U	x	x
File-attributes	x	x	D	U
Special-information	D	U	D	U

Parameter descriptions

Diagnostic: see 3.2.2.

Requested-attributes: specifies for which categories of file attributes (see 2.1.3.1) the attribute values are to be returned.

3.2.1.12 The F-CLOSE-FILE service

Purpose: to terminate the processing of the contents of the current file. The open enclosure is ended even if the diagnostic severity is "failure". This service is subject to grouping.

Structure: type 2

Initiated by: Primary

Parameters:

	REQ	IND	RESP	CONF
Diagnostic	x	x	B	U
Special-information	D	U	D	U

Parameter descriptions:

Diagnostic: see 3.2.2.

Special-information: transparent data obeying special conventions between VFS users (see special-conventions parameter in F-CONNECT).

3.2.1.13 The F-BEGIN-DATA service

Purpose: to start transfer of file data from or to the currently open file. The direction of data transfer is determined by the processing-mode parameter of the previous F-OPEN-FILE service. This service is subject to grouping.

Structure: type 2

Initiated by: Primary

Parameters:

	REQ	IND	RESP	CONF
Diagnostic	x	x	B	U
Special-information	D	U	D	U

Parameter descriptions:

Diagnostic: see 3.2.2.

Special-information: transparent data obeying special conventions between VFS users (see special-conventions parameter in F-CONNECT).

3.2.1.14 The F-DATA service

Purpose: to transfer file data

Structure: type 1

Initiated by: Sender

Parameters:

	REQ	IND
File-data	D	U

Parameter descriptions:

File-data: for a flat file, contains one complete record. Records are ordered as described by the "Record sequence" attribute (see 2.1.3.10). Empty records (null size) are allowed and are also taken into account for the determination of the record ranks (in the restart-position parameter). For unstructured files, contains an arbitrary number of consecutive characters or octets (according to file-data-type value), in their logical sequence.

The syntax in which the data is transferred is a concern of Data Presentation: see Standard ECMA-84.

3.2.1.15 The F-END-DATA service

Purpose: to specify normal completion of file data transfer. No restart is possible after a successful F-END-DATA.

The acceptor may reject a proposed F-END-DATA only by issuing a F-ABORT-DATA request or a F-RESTART request.

Structure: type 2

Initiated by: Sender

Parameters: this service contains no parameters.

3.2.1.16 The F-ABORT-DATA service

Purpose: to specify abnormal termination of file data transfer. If the diagnostic severity value is "failure" the transfer is abandoned. If the diagnostic severity value is "recoverable failure", the transfer is recoverable as soon as the cause of error has disappeared. The effects are disruptive.

In case of collision between two F-ABORT-DATA initiated at both ends, only the one issued by the Primary is retained.

Structure: type 2

Initiated by: Sender or Receiver

Parameters:

	REQ	IND	RESP	CONF
Diagnostic	D	U	D	U

Parameter descriptions:

Diagnostic: see 3.2.2. Only one diagnostic may be supplied, with no diagnostic supplement.

3.2.1.17 The F-RESTART service

Purpose: to resynchronize the current file data transfer at a previous point. The effects are disruptive.

The acceptor can reject a proposed F-RESTART only by issuing a F-ABORT-DATA request (purposeful service collision, in which only the F-ABORT-DATA is retained). In case of collision between two F-RESTART initiated at both ends, only the one issued by the Primary is retained.

Structure: type 2

Initiated by: Sender or Receiver

Parameters:

	REQ	IND	RESP	CONF
Restart-position	D	U	D	U

Parameter descriptions:

Restart-position: negotiates the position at which to resume data transfer.

Legal values:

For flat files, rank in the total transfer flow of the first record to be retransmitted. For unstructured files, rank in the total transfer flow of the first character or octet to be retransmitted. 1 designates the beginning of transfer. 0 is invalid.

3.2.1.18 The F-CHECKPOINT service

Purpose: to mark points in the data transfer at which acknowledgement of processing is desired. Other services can be initiated by the Sender between the request primitive and the confirmation primitive.

Structure: type 2

Initiated by: Sender

Parameters:

	REQ	IND	RESP	CONF
Checkpoint-ident.	U	U	D	U

Parameter descriptions:

Checkpoint-identification: serial number uniquely identifying each checkpoint within a given file data transfer. No rule is specified for the automatic incrementation or reset of this serial number.

3.2.2 Error reporting

Error reporting is provided by means of a diagnostic parameter appearing in response and confirmation primitives. It also appears in a few request and indication primitives (for disruptive services). The diagnostic parameter conveys up to three elements of information, corresponding to three levels of error analysis:

- severity
- reason
- diagnostic supplement

Each element can be supplied only if the preceding (more synthetic) elements have been supplied. A separate diagnostic parameter is used for each detected error. Limitations specific to some services are indicated within the description of these services in 3.2.1.

3.2.2.1 Severity

Specifies the degree of success or failure. The legal values are, by increasing order of severity:

- Success.
- Success with warning.
- Recoverable failure: applicable only to a non atomic operation, i.e. file data transfer. Used with F-ABORT-DATA primitives to indicate that the data transfer can be restarted after failure correction. If the negotiated service subset does not include the restart extension (see 3.3), there is no difference between "recoverable failure" and "failure".
- Failure: indicates non recoverable failure.

Severity is the minimum amount of information to be provided.

3.2.2.2 Reason

Specifies a summary diagnostic. Legal values are supplied in 4.3.2.4.

3.2.2.3 Diagnostic supplement

Specifies a detailed diagnostic. Several diagnostic supplements may be specified within the same diagnostic parameter. The legal types of diagnostic supplements are defined below.

DS-0: may accompany any reason value and is specifically expected when the reason value is "non standard reason". The legal value is a character string of not more than 31 characters.

DS-1: accompanies standard reason values like "unset parameter value", "illegally duplicated parameter", "illegal parameter value". Repeated once for each erroneous parameter. Contains the parameter type of the defective parameter.

DS-2: accompanies standard reason values like "time-out expiration" or "protocol violation". Contains the type of the expected or erroneous message.

3.3 SERVICE SUBJECTS

3.3.1 General

Subsets of the virtual file service are defined to achieve simplification and variety control. The Virtual File service is subsetted in the following way:

- a kernel, composed of all the services that all implementations must support,
- two optional extensions, each composed of an additional set of services:

  Basic file management extension,
  Restart extension.

Each of these options can be supported independently of whether the other is supported.

The class-of-service parameter of the F-CONNECT service specifies the optional extensions that may be used on the connection. The Primary proposes its maximum requirement and the Secondary responds with its capabilities within the proposed set of extensions. The Primary may decide to terminate the connection if the selected set is too restricted.

There is no relationship between the service subsets and the virtual file model subset in this version: any service subset can be used on any virtual file model subset.

3.3.2 Kernel

The services included in this subset are:

F-CONNECT
F-RELEASE
F-DISCONNECT
F-ABORT

F-END-GROUP (*)
F-SELECT-FILE
F-RELEASE-FILE
F-OPEN-FILE
F-CLOSE-FILE
F-BEGIN-DATA

F-DATA
F-END-DATA
F-ABORT-DATA

The open-identification and restart-position parameters of F-OPEN-FILE are not included in this subset.

(*) Support of the request and indication primitives is optional and indicated in the F-CONNECT request primitive.

3.3.3 Basic file management extension

The services included in this extension are:

F-READ-ATTRIBUTES
F-CREATE-FILE
F-DELETE-FILE

3.3.4 Restart extension

The services included in this extension are:

F-RESTART
F-CHECKPOINT

The open-identification and restart-position parameters of F-OPEN-FILE are included in this extension.

4 Protocol

4.1 PROTOCOL OVERVIEW

4.1.1 Roles of VFS entities

The asymmetry of the VFS is reflected in the protocol: the two VFS entities play different and complementary roles, corresponding to the roles played by their respective users: Primary and Secondary outside of a transfer enclosure, Sender and Receiver within a transfer enclosure (see 3.1.1).

4.1.2 Descriptive model

The Virtual File Protocol is modelled as an abstract machine, with protocol structures between the two VFS entities. A protocol structure is an elementary dialogue for the purpose of an indivisible operation. As such, it is totally successful or totally unsuccessful, never partly successful. It is composed of a request, issued by one VFS entity, and for most (but not all) types of structure, of a response, issued by the other VFS entity. Each response or request is a single protocol message.

There are two types of protocol structures:

- Type 1 structure: request without response.
- Type 2 structure: request with response.

A protocol message contains protocol control information (i.e. one or more parameters) and may in some cases also contain file data.

Dynamic execution of the VF protocol results in an ordered sequence of protocol structures. To describe the protocol, it is sufficient to separately describe each of its structures (or messages), plus any precedence relationship between structures (state transitions).

4.1.3 Grouping of protocol structures

The grouping of protocol structures corresponds to the grouping of service structures described in 3.1.8. It is applicable only to file management protocol structures (see 4.1.4 for the list of them). The effect on the protocol of the grouping mechanism is briefly described here. In order not to overload the description of the protocol in 4.2, the effects of grouping do not appear in the message descriptions. They are fully taken into account in the formal description of Appendix D.

Effects at the Primary

First, it is necessary to verify if a request message can be legally part of the current group before issuing this

message. Second, the state machine cannot be immediately progressed to the XX-pending state: the new state that the sending of the message should cause is enqueued in a first in first out queue. The enqueued states are successively dequeued as successful response messages are processed. There is no state dequeuing on a failed response message. The queue is purged when the end of group response is received.

Effects at the Secondary

The only effect at the Secondary is when a failed response message is issued: then all incoming request messages are ignored by the VFS entity until the next end of group message.

4.1.4 List of protocol structures

The table below lists all the structures of the VFP. For each structure, it specifies its type (see 4.1.2), the VFS entity which can initiate it (PR: Primary, SC: Secondary, SN: Sender, RC: Receiver), and its purpose.

Table 4.1/1 - Protocol structures

Structure	Type	Init.	Description
Select Protocol	2	PR	Initiate VFS connection
Release Protocol	2	PR	Release VFS connection
Disconnect Protocol	1	PR,SC	Abnormally terminate VFS conn.
End Group	2	PR	Delimit a structure group
Select File	2	PR	Establish current file
Release File	2	PR	Release current file
Create File	2	PR	Create new file
Delete File	2	PR	Delete current file
Read Attributes	2	PR	Read attributes of current file
Open File	2	PR	Open current file for data access
Close File	2	PR	Close current file
Begin Transfer	2	PR	Begin transfer of file data
Data	1	SN	Transfer file data
End Transfer	2	SN	End transfer of file data
Abort Transfer	2	SN,RC	Abort transfer of file data
Restart Transfer	2	SN,RC	Resynchronize transfer in progress
Checkpoint	2	SN	Request acknowledgement

4.2 PROTOCOL DESCRIPTION

4.2.1 Notation

This clause provides a narrative description of the protocol. Appendix D provides the formal description.

Each message is defined by the following items:

- Sender of message
- Function
- List of parameters
- Resulting state transition(s)
- Relationship with service primitives (sending/receiving)

A detailed description of parameters is supplied only when they differ from the service parameters; otherwise reference is made to the description of the equivalent parameter in the service.

4.2.2 Select Protocol request (SP)

Sent by: Primary

Function: Initiate a VFS connection
(requests opening of a VFS connection enclosure)

Content:
```
filestore-name
authentication
protocol-definition = {protocol-identifier
                      {protocol-version
                      {class-of-filestore
                      {class-of-service
grouping-option
special-conventions
```

Parameter descriptions: see F-CONNECT, 3.2.1.1.

Transition: Dormant --> SP pending

Sending: on F-CONNECT request primitive.

Receiving: generates a F-CONNECT indication primitive. The expected outcome is a F-CONNECT response primitive.

4.2.3 Select Protocol response (SPR)

Sent by: Secondary

Function: response to SP

Content:
```
diagnostic
protocol-definition = {protocol-version
                      {class-of-filestore
                      {class-of-service
special-conventions
```

Parameter descriptions: see F-CONNECT, 3.2.1.1.

Transition: SP pending --> No-file (successful)
SP pending --> Dormant (rejected)

Sending: on F-CONNECT response primitive.

Receiving: generates a F-CONNECT confirmation primitive.

4.2.4 Release Protocol request (RP)

Sent by: Primary.

Function: request normal termination of the VFS connection enclosure.

Content: none.

Transition: No-file --> RP-pending.

Sending: on F-RELEASE request primitive.

Receiving: generates a F-RELEASE indication primitive. The expected outcome is a F-RELEASE response primitive.

4.2.5 Release Protocol response (RPR)

Sent by: Secondary.

Function: response to RP.

Content: none.

Transition: RP pending --> Dormant.

Sending: on F-RELEASE response primitive.

Receiving: generates a F-RELEASE confirmation primitive.

4.2.6 Disconnect Protocol request (DP)

Sent by: Primary/Secondary.

Function: Request abnormal termination of the VFS connection enclosure.

Content: diagnostic.

Parameter descriptions: see F-DISCONNECT, 3.2.1.3.

Transition: Any state --> Dormant.

Sending: on F-DISCONNECT request primitive.

Receiving: generates a F-DISCONNECT indication primitive.

4.2.7 End Group request (EG)

Sent by: Primary.

Function: Delimit the end of a structure group.

Content: none.

Transition: none.

Sending: on F-END-GROUP request primitive.

Receiving: generates a F-END-GROUP indication primitive.
The expected outcome is a F-END-GROUP response primitive.

4.2.8 End Group response (EGR)

Sent by: Secondary.

Function: response to EG.

Content: none.

Transition: none.

Sending: on F-END-GROUP response primitive.

Receiving: generates a F-END-GROUP confirmation primitive.

4.2.9 Select File request (SL)

Sent by: Primary.

Function: Establish as current an existing file.
(requests opening of a file enclosure).

Content: file-name
file-passwords
special-information

Parameter descriptions: see F-SELECT-FILE, 3.2.1.6.

Transition: No-file --> SL-pending

Sending: on F-SELECT-FILE request primitive.

Receiving: generates a F-SELECT-FILE indication primitive.
The expected outcome is a F-SELECT-FILE response primitive.

4.2.10 Select file response (SLR)

Sent by: Secondary.

Function: response to SL.

Content: diagnostic
special - information

Parameter descriptions: see F-SELECT-FILE, 3.2.1.6.

Transition: SL-pending --> File-selected (successful)
SL-pending --> No-file (rejected)

Sending: on F-SELECT-FILE response primitive.

Receiving: generates a F-SELECT-FILE confirmation primitive.

4.2.11 Release File request (RL)

Sent by: Primary.

Function: Release the current file.
(requests closing of the file enclosure).

Content: special-information.

Parameter description: see F-RELEASE-FILE, 3.2.1.7.

Transition: File-selected --> RL-pending.

Sending: on F-RELEASE-FILE request primitive.

Receiving: generates a F-RELEASE-FILE indication primitive. The expected outcome is a F-RELEASE-FILE response primitive.

4.2.12 Release File response (RLR)

Sent by: Secondary.

Function: response to RL.

Content: diagnostic
special-information.

Parameter descriptions: see F-RELEASE-FILE, 3.2.1.7.

Transition: RL-pending --> No-file.

Sending: on F-RELEASE-FILE response primitive.

Receiving: generates a F-RELEASE-FILE confirmation primitive.

4.2.13 Create File request (CR)

Sent by: Primary.

Function: Create and establish as current a new file. (requests opening of a file enclosure)

Content: file-name
file-passwords
file-attributes = {global-attributes
{record-attributes
{key-attributes
{field-attributes
clash-option
reversible-mapping
special-information

Parameter descriptions: see F-CREATE-FILE, 3.2.1.8.

Transition: No-file --> CR-pending

Sending: on F-CREATE-FILE request primitive.

Receiving: generates a F-CREATE-FILE indication primitive. The expected outcome is a F-CREATE-FILE response primitive.

4.2.14 Create File response (CRR)

Sent by: Secondary.

Function: response to CR.

Content: diagnostic
special-information.

Parameter descriptions: see F-CREATE-FILE, 3.2.1.8.

Transition: CR-pending --> File-selected (successful)
CR-pending --> No-file (rejected)

Sending: on F-CREATE-FILE response primitive.

Receiving: generates a F-CREATE-FILE confirmation primitive.

4.2.15 Delete File request (DL)

Sent by: Primary.

Function: delete and release the current file. (requests closing of the file enclosure)

Content: special-information

Parameter descriptions: see F-DELETE-FILE, 3.2.1.9.

Transition: File-selected --> DL-pending

Sending: on F-DELETE-FILE request primitive.

Receiving: generates a F-DELETE-FILE indication primitive. The expected outcome is a F-DELETE-FILE response primitive.

4.2.16 Delete File response (DLR)

Sent by: Secondary.

Function: response to DL.

Content: diagnostic
special-information

Parameter descriptions: see F-DELETE-FILE, 3.2.1.9.

Transition: DL-pending --> No-file.

Sending: on F-DELETE-FILE response primitive.

Receiving: generates a F-DELETE-FILE confirmation primitive.

4.2.17 Read Attributes request (RA)

Sent by: Primary.

Function: retrieve specified attributes of the current file.

Content: requested-attributes
special-information

Parameter descriptions: see F-READ-ATTRIBUTES, 3.2.1.10.

Transition: File-selected --> RA-pending

Sending: on F-READ-ATTRIBUTES request primitive.

Receiving: generates a F-READ-ATTRIBUTES indication primitive. The expected outcome is a F-READ-ATTRIBUTES response primitive.

4.2.18 Read Attributes response (RAR)

Sent by: Secondary.

Function: response to RA.

Content: diagnostic
file-attributes = {history-attributes
{global-attributes
{record-attributes
{key-attributes
{field-attributes
special-information

Parameter descriptions: see F-READ-ATTRIBUTES, 3.2.1.10.

Transition: RA-pending --> File-selected.

Sending: on F-READ-ATTRIBUTES response primitive.

Receiving: generates a F-READ-ATTRIBUTES confirmation primitive.

4.2.19 Open File request (OP)

Sent by: Primary.

Function: initiate processing of contents of the current file. (requests opening of an open enclosure)

Content: access-mode
processing-mode
lock
failure-option
open-identification
restart-position
special-information

Parameter descriptions: see F-OPEN-FILE, 3.2.1.11.

Transition: File-selected --> OP-pending

Sending: on F-OPEN-FILE request primitive.

Receiving: generates a F-OPEN-FILE indication primitive. The expected outcome is a F-OPEN-FILE response primitive.

4.2.20 Open File response (OPR)

Sent by: Secondary.

Function: response to OP.

Content: diagnostic
restart-position
special-information

Parameter descriptions: see F-OPEN-FILE, 3.2.1.11.

Transition: OP-pending --> File-open (successful)
OP-pending --> File-selected (rejected)

Sending: on F-OPEN-FILE response primitive.

Receiving: generates a F-OPEN-FILE confirmation primitive.

4.2.21 Close File request (CL)

Sent by: Primary.

Function: terminate processing of contents of the current file. (requests closing of the open enclosure)

Content: special-information.

Parameter descriptions: see F-CLOSE-FILE, 3.2.1.12.

Transition: File-open --> CL-pending
File-aborted --> CL-pending

Sending: on F-CLOSE-FILE request primitive.

Receiving: generates a F-CLOSE-FILE indication primitive. The expected outcome is a F-CLOSE-FILE response primitive.

4.2.22 Close File response (CLR)

Sent by: Secondary

Function: response to CL.

Content: diagnostic
special-information

Parameter descriptions: see F-CLOSE-FILE, 3.2.1.12.

Transition: CL-pending --> File-selected.

Sending: on F-CLOSE-FILE response primitive.

Receiving: generates a F-CLOSE-FILE confirmation primitive.

4.2.23 Begin Transfer request (BT)

Sent by: Primary.

Function: cause transition to file data transfer level. (requests opening of a transfer enclosure)

Content: special-information.

Parameter descriptions: see F-BEGIN-DATA, 3.2.1.13.

Transition: File-open --> BT-pending.

Sending: on F-BEGIN-DATA request primitive.

Receiving: generates a F-BEGIN-DATA indication primitive. The expected outcome is a F-BEGIN-DATA response primitive.

4.2.24 Begin Transfer response (BTR)

Sent by: Secondary.

Function: response to BT.

Content: diagnostic
special-information

Parameter descriptions: see F-BEGIN-DATA, 3.2.1.13.

Transition: BT-pending --> Data
BT-pending --> File open (rejected)

Sending: on F-BEGIN-DATA response primitive.

Receiving: generates a F-BEGIN-DATA confirmation primitive.

4.2.25 Data request (DATA)

Sent by: Sender.

Function: transfer file data and/or a delimiter.

Content: file-data.

Parameter descriptions: see F-DATA, 3.2.1.14.

Transition: Data --> Data.

Sending: on F-DATA request primitive.

Receiving: generates a F-DATA indication primitive.

4.2.26 End Transfer request (ET)

Sent by: Sender.

Function: specify termination of the transfer without loss of data (requests closing of the transfer enclosure).

Content: none

Transition: Data --> ET-pending

Sending: on F-END-DATA request primitive.

Receiving: generates a F-END-DATA indication primitive. The expected outcome is a F-END-DATA response primitive or a F-RESTART or F-ABORT-DATA request primitive.

4.2.27 End Transfer response (ETR)

Sent by: Receiver.

Function: response to ET.

Content: none.

Transition: ET-pending --> File-open.

Sending: on F-END-DATA response primitive.

Receiving: generates a F-END-DATA indication primitive.

4.2.28 Checkpoint request (CK)

Sent by: Sender.

Function: specify a checkpoint position, for acknowledgement by the Receiver.

Content: checkpoint-identification.

Parameter descriptions: see F-CHECKPOINT, 3.2.1.18.

Transition: Data --> Data.

Sending: on F-CHECKPOINT request primitive.

Receiving: generates a F-CHECKPOINT indication primitive. The expected outcome is a F-CHECKPOINT response primitive.

4.2.29 Checkpoint response (CKR)

Sent by: Receiver.

Function: asynchronous response to CK. Acknowledges safe processing of all file data until specified checkpoint.

Content: checkpoint-identification.

Parameter descriptions: see F-CHECKPOINT, 3.2.1.18.

Transition: Any data transfer state --> same state.

Sending: on F-CHECKPOINT response primitive.

Receiving: generates a F-CHECKPOINT confirmation primitive.

4.2.30 Abort Transfer request (AT)

Sent by: Sender/Receiver.

Function: specify abnormal termination of the transfer, with destruction of data in transit. (requests closing of the transfer enclosure).

Content: diagnostic.

Parameter descriptions: see F-ABORT-DATA, 3.2.1.16.

Transition: Any other data transfer state --> AT-pending.

Sending: on F-ABORT-DATA request primitive.

Receiving: generates a F-ABORT-DATA indication primitive. The expected outcome is a F-ABORT-DATA response primitive.

4.2.31 Abort Transfer response (ATR)

Sent by: Sender/Receiver.

Function: response to AT.

Content: diagnostic.

Parameter descriptions: see F-ABORT-DATA, 3.2.1.16.

Transition: AT-pending --> File-aborted.

Sending: on F-ABORT-DATA response primitive.

Receiving: generates a F-ABORT-DATA confirmation primitive.

4.2.32 Restart Transfer request (RT)

Sent by: Sender/Receiver.

Function: request immediate restart of the transfer at a previous point. Destroys any data in transit.

Content: restart-position.

Parameter descriptions: see F-RESTART, 3.2.1.17.

Transition: Data --> RT-pending
ET-pending --> RT-pending

Sending: on F-RESTART request primitive.

Receiving: generates a F-RESTART indication primitive. The expected outcome is a F-RESTART response primitive or a F-ABORT-DATA request primitive.

4.2.33 Restart Transfer response (RTR)

Sent by: Sender/Receiver.

Function: response to RT.

Content: restart-position.

Parameter descriptions: see F-RESTART, 3.2.1.17.

Transition: RT-pending --> Data

Sending: on F-RESTART response primitive.

Receiving: generates a F-RESTART confirmation primitive.

4.3 PROTOCOL ENCODING

General convention: the bits of an octet are identified b1 b2 b3 b4 b5 b6 b7 b8, b1 being the leftmost and most significant bit.The same convention applies to strings of more than one octet, the rightmost bit becoming b16, b24 or b32.

4.3.1 Message structure

Each message of the VFP is composed of 2 parts:

- a one octet message header
- a variable length message content

The message header contains a 8-bit message type. The legal type values are listed in table 4.3/1.

The message content is composed of the message parameters, in any order. The representation of parameters is described in 4.3.2.

Table 4.3/1 - Message type values

1 SP	2 SPR
3 SL	4 SLR
5 OP	6 OPR
7 CL	8 CLR
9 RL	10 RLR
11 BT	12 BTR
13 ET	14 ETR
15 DATA	16-31 unassigned
32 RA	33 RAR
34 CR	35 CRR
36 DL	37 DLR
38-255 unassigned	

NOTE:

The VFP messages that do not appear in the above list are directly mapped onto lower layer services (see 4.4). They are the following: RP, RPR, DP, EG, EGR, AT, ATR, RT, RTR, CK, CKR.

4.3.2 Parameter encoding

4.3.2.1 Type/Length/Value technique (TLV technique)

The TLV technique is a method for coding an Information Unit.

Every information unit is encoded as a triplet, made of:

a type (1st field)
a length (2nd field)
a value (3rd field)

Type field (1 octet)

b1 = 0
b2-b8 = type value (1 to 127,0 reserved)

Length field (1 or 2 octets)

b1 = 0: the length is specified on 7 bits (b2-b8)
b1 = 1: the length is specified on 15 bits (b2-b16)

b2-b8 or b16: binary number of octets of the value field

Value field (0 to N octets)

all octets: data

4.3.2.2 Parameter value representation

The following value types are needed for representation of the various parameters of the VFP:

C: character string. Any size (unless limits are defined), character encoding is according to the ECMA 7-bit coded character set (see ECMA-6). Each character is mapped on bits b2-b8 of an octet with bit b1 equal to 0. The allowed characters are those in columns 2 to 7 of the International Reference Version, with the following exclusions: 4/0, 5/11 to 5/14, 6/0, 7/11 to 7/15.

N: numeric. Unsigned binary integer, with 4 possible sizes: 8, 16, 24 or 32 bits.

S: symbolic. Unsigned binary integer, the values of which encode specific meanings. Size is 8 bits.

M: bit map. Bit string in which each bit encodes a specific meaning. Size is 8 or 16 bits. Any bit the role of which is not defined must be encoded as 0. For bits representing specified options, the bit is set to 1 if the option is requested, 0 otherwise.

D: date (and optionally time). Encoded as specified in the Standards ISO 2014 and ISO 3307: YYMMDD(hhmmss), where YY= year, MM= month, DD= day, hh= hour, mm= minute, ss= second. It is encoded as a character string (see above), with a size of either 6 octets (date only) or 12 octets (date and time).

T: transparent. Encoded by VFS user.

A: aggregate. Composed of fields (sometimes smaller than 8 bits) of the above types.

Value truncation

For economy reasons, it is recommended to use the minimum variable length strings for expressing values.

- character strings should not contain unnecessary spaces.
- for numeric or symbolic, all unnecessary octets from left containing only zero bits should be removed.
- for bit maps, all unnecessary octets from right containing only zero bits should be removed (must be from right to allow future extension to the right).
- for transparent, no truncation is applied.

4.3.2.3 Parameter encoding

For each parameter of the protocol, table 4.3/2 gives the following description:

- parameter type (unique identifier)
- maximum length of value field ("-" means unlimited)
- type of value representation (C, N, S, M, D, T: see 4.3.2.2)
- encoding of all possible values (for S or M only).

The parameters which are directly mapped onto presentation service parameters do not appear in this table. These are: filestore-name (see 4.4.3) and checkpoint-identification (see 4.4.7.3).

Default values

The existence of a default value, to be used in case the parameter has not been explicitly specified, depends only on the parameter value type (see 4.3.2.2):

- for value types S and M, there is always a default value, equal to zero (0).
- for value types C, N, D and T, there is no default value.

Aggregates

A few parameters contain, instead of an elementary value, an aggregate of values. This aggregate is encoded as a fixed data structure when most elements are always present and there is no need for extendability. Otherwise, it is encoded recursively as a set of TLV items. In this latter case, the range of T's internal to an aggregate can overlap the range of T's defined in table 4.3/2, since this represents another level of encoding.

The internal encoding of aggregate parameters is specified in 4.3.2.4 to 4.3.2.10.

Table 4.3/2 - Parameter encoding

Type	Parameter	Value length	Value type	Value encoding
1	Diagnostic	-	-	Aggregate, see 4.3.2.4
2	Protocol-definition	-	-	Aggregate, see 4.3.2.5
3	File-name	64	C	
4	File-passwords	32	C	
5	Access-control-list	1	S	0: no, 1: yes
6	Access-mode	1	S	0: in sequence
7	Processing-mode	1	S	0: read, 1: write, 2: append
8	Lock	1	S	0: exclusive, 1: shared read 2: shared update
9	Failure-option	1	M	b1: rollback
10	Grouping-option	1	M	b1: Grouping
11	Authentication	-	-	Aggregate, see 4.3.2.11
16	Open-identification	4	N	
17	Restart-position	4	N	
24	Clash-option	1	S	0: reject, 1: keep, 2: replace
25	Reversible-mapping	1	S	0: no, 1: yes
26	Requested-attributes	1	M	b1: history, b2: global b3: record, b4: key b5: field
28	History-attributes	-	-	Aggregate, see 4.3.2.6
29	Global-attributes	-	-	Aggregate, see 4.3.2.7
30	Record-attributes	-	-	Aggregate, see 4.3.2.8
31	Key-attributes	-	-	Aggregate, see 4.3.2.9
32	Field-attributes	-	-	Aggregate, see 4.3.2.10
62	Special-conventions	2	T	
63	Special-information	-	T	
-	File-data	-	T	see 4.3.2.12

4.3.2.4 Value of diagnostic parameter

The value field of the diagnostic parameter is a structure defined as follows in BNF:

<diagnostic value> ::= <severity> <reason> {<DS>}

<DS> ::= <DS type> <DS value length> <DS value>

The terminal elements are encoded as follows:

<severity>

4-bit unsigned binary integer (b1-b4 of first octet of <diagnostic value>). See table 4.3/3 for values. When the diagnostic parameter is omitted, the default value of <severity> is "success" (0).

<reason>

12 bit unsigned binary integer (b5-b8 of first octet of <diagnostic value>, followed by b1-b8 of second octet). See table 4.3/4 for values. The table also indicates the corresponding severity (or severities) and the messages where each reason value can appear (except for values applicable to most messages). When the diagnostic parameter is omitted, the default value of <reason> is "no reason provided" (0).

<DS type>

3-bit unsigned binary integer (b1-b3 of first octet of <DS>). Legal values: 0-2.

<DS value length>

5-bit unsigned binary integer (b4-b8 of first octet of <DS>).

<DS value>

Dependent on <DS type> value, as follows:

0: character string, up to 31 characters.

1: one octet containing a parameter type (see 4.3.2.1) or 2 octets containing an aggregate type and a parameter type, in the case of parameters recursively encoded within aggregates.

2: one octet containing a message type (see 4.3.1)

Exception:

A special encoding is required for the diagnostic parameter when it is supplied as user data in a P-DISCONNECT, since the P-DISCONNECT can offer only 3 octets of user data (see 4.4.4).

In this special case, the following fields are omitted: parameter type, parameter length, DS type, DS value length. The 3 octets are encoded as follows:

first 2 octets: severity and reason (standard encoding)
third octet: message type (standard value field of DS type 2)

Table 4.3/3 - Severity values

0: success
1: success with warning
2: recoverable failure
3: failure

Table 4.3/4 - Reason values

Value	Sev.	Reason	Messages
0	0	No reason provided	
1	123	Non standard reason	
2	3	Unset parameter value	
3	3	Illegal parameter value	
4	3	Unsupported parameter value	
5	3	Illegally duplicated parameter	
6	3	Illegal parameter type	
7	3	Unsupported parameter type	
8	23	I/O error	AT, ATR
9	23	File space exhausted	AT, ATR
10	23	Transmission error	AT, ATR
11	23	Record size error	AT, ATR
12	23	Presentation error (formatting)	AT, ATR
13	23	Presentation error (compression)	AT, ATR
14	23	Presentation error (encryption)	AT, ATR
15	3	VF protocol violation	VF conn. abort
16	3	Time-out expiration	DP
17	3	Shut-down	DP
18	3	File does not exist	SLR, OPR
19	3	Insufficient permission	
20	3	Insufficient resources	
21	3	File not mountable	SLR, CRR, OPR
22	3	File busy	SLR, CRR, DLR, OPR
23	1	More restrictive lock	OPR
24	3	Rollback not supported	OPR
25	3	File already exists	CRR
26	1	Existing file kept	CRR
27	1	Existing file replaced	CRR
28	3	File mapping not reversible	CRR
29	3	Local filestore error	
30	3	Local filestore restriction	

Table 4.3/4 - Reason values (cont'd)

Value	Sev.	Reason	Messages
31	3	Password collision	CRR
32	1	Input file empty	OPR
33	3	Specification error in local filestore	
34	3	Illegal parameter value duplication	
35	3	Conflicting parameter value	
36	3	File not prepared for restart	OPR
37	3	File waiting restart	OPR
38	3	Open-identification not unique	OPR
39	3	Open-identification not found	OPR

4.3.2.5 Contents of protocol-definition aggregate

Parameters of this aggregate are encoded as a structure containing a fixed number of elements in the specified order. This structure is described by table 4.3/5.

Table 4.3/5 - Protocol-definition encoding

Type	Parameter	Value length	Value type	Value encoding
-	Protocol-identifier	1	S	0: VFP
-	Protocol-version	1	M	b1: version 1
-	Class-of-filestore	1	M	b1: unstructured files b2: field descriptions
-	Class-of-service	1	M	b1: basic file mgt b2: restart

4.3.2.6 Contents of history-attributes aggregate

Parameters of this aggregate are recursively encoded as a set of TLV's which are defined in table 4.3/6

Table 4.3/6 - History-attributes encoding

Type	Parameter	Value length	Value type	Value encoding
56	Creation-date	12	D	
57	Creation-user-id	8	C	
58	Last-access-date	12	D	
59	Last-access-user-id	8	C	
60	Last-modif-date	12	D	
61	Last-modif-user-id	8	C	
62	Total-number-of-accesses	4	N	
63	Total-number-of-modif	4	N	

4.3.2.7 Contents of global-attributes aggregate

Parameters of this aggregate are recursively encoded as a set of TLV's which are defined in table 4.3/7.

Table 4.3/7 - Global-attributes encoding

Type	Parameter	Value length	Value type	Value encoding
1	File-structure	1	S	0: flat, 1: unstructured
2	File-data-type	1	S	0: character, 1: transparent, 2: heterogeneous
3	File-current-size	4	N	
4	File-maximum-size	4	N	

4.3.2.8 Contents of record-attributes aggregate

Parameters of this aggregate are recursively encoded as a set of TLV's which are defined in table 4.3/8.

Table 4.3/8 - Record-attributes encoding

Type	Parameter	Value length	Value type	Value encoding
5	Record-size-type	1	S	0: fixed, 1: variable
6	Record-size	4	N	
7	Record-sequence	1	S	0: by position, 1: by key
8	Direct-access	1	S	0: by no means, 1: by sequence, 2: by key

4.3.2.9 Contents of key-attributes aggregate

Parameters of this aggregate are recursively encoded as a set of TLV's which are defined in table 4.3/9.

Table 4.3/9 - Key-attributes encoding

Type	Parameter	Value length	Value type	Value encoding
9	Key-position	2	N	
10	Key-length	2	N	
11	Key-field-rank	2	N	

4.3.2.10 Contents of field-attributes aggregate

Parameters of this aggregate are recursively encoded as a set of TLV's which are defined in table 4.3/10.

Table 4.3/10 - Field-attributes encoding

Type	Parameter	Value length	Value type	Value encoding
12	Field-rank	2	N	
13	Field-data-type	1	A	b1-b2: type (symbolic: 0=string, 1=number) b3-b8: attributes -for type = 0, b3: size type (0:fixed) b4-b6: reserved (0) b7-b8: element type (0: character, 1: octet, 2: bit) -for type = 1, bit map b3-b5: reserved (0) b6: base:0=binary,1=decimal (0 for floating point) b7: scale: 0=fixed, 1=float b8: sign: 0=no, 1=yes
14	Field-size	1	N	
15	Field-complement	1	N	

4.3.2.11 Contents of authentication aggregate

Parameters of this aggregate are recursively encoded as a set of TLV's which are defined in table 4.3/11.

Note 6

This description is given here provisionally, until some Security Protocol standard is produced.

Table 4.3/11 - Authentication encoding

Type	Parameter	Value length	Value type	Value encoding
1	User-identification	8	C	
2	User-password	8	C	
3	Account-identification	8	C	

4.3.2.12 File data encoding

Because the encoding of file data for the transfer is provided by the Presentation layer, the length of this data may vary from one VFS entity to the other, due to differences in local syntaxes. Therefore, this length cannot be specified in the file-data parameter

encoding, which is to be considered as a special kind of parameter.

Since the file-data parameter is the only parameter of the DATA message, it is encoded as transparent data of the VFP, without any parameter header. File data occupies the totality of the DATA message, with exception of the message header.

4.4 PRESENTATION SERVICES MAPPING

4.4.1 General

The virtual file protocol relies on the services offered by the data presentation protocol of the presentation layer (see Standard ECMA-84).

4.4.2 Connection mapping

Each VFS connection uses one and only one presentation connection. The presentation connection is used by only one VFS connection.

NOTE 7

Multiplexing of multiple concurrent VFS connection onto a presentation connection is excluded in ISO 7498.

4.4.3 Presentation connection establishment

The establishment of a presentation connection between the Primary and the Secondary is necessary before any Virtual File protocol activity. The P-CONNECT service maps the SP and SPR messages of the VFP in the following way:

- the filestore-name parameter of the SP is mapped onto the user-called parameter of P-CONNECT.
- the remainder of the SP message is mapped into the transp-data parameter of P-CONNECT request and indication.
- the SPR message is mapped into the transp-data parameter of P-CONNECT response and confirmation.
- the VFP sets the session parameters of P-CONNECT as follows:
  - Session-subset: "C" (basic synchronized subset)
  - Token assignment:
    data token: not defined
    all other tokens: Primary
  - Blocking: according to VFP implementations.

4.4.4 Presentation connection termination

The presentation connection is terminated in one of three ways:

- normal termination request only by the Primary (RP, RPR messages)
- abnormal termination request by either VFS entity, resulting either from a F-DISCONNECT request (DP message) or from a spontaneous action of the VFS entity itself.
- spontaneous disconnection by the presentation service.

The first case is mapped onto the P-RELEASE service. Though this service offers negotiated termination, the Secondary must accept the proposed termination.

The second case is mapped onto the P-DISCONNECT service. The diagnostic parameter of the VFP is mapped into the transp-data parameter of P-DISCONNECT (see note 8).

The third case is treated as a presentation service failure. It is signalled by the P-ABORT service.

NOTE 8

The transp-data parameter of the P-DISCONNECT is limited to 3 octets. This limit is expected to be removed in a future version.

4.4.5 Presentation negotiation

The VFP uses the P-PERFORM-NEGOTIATION service to negotiate the presentation characteristics for a given data transfer (DPP transfer environment). The P-PERFORM-NEGOTIATION is always invoked by the Primary. It may be used any number of times on a VFS connection (including zero). It must not be invoked within a file data transfer enclosure (as defined in 3.1.2); if grouping is used, it can be invoked only at the beginning of a structure group.

NOTE 9

This last rule is to eliminate any ambiguity on the current DPP transfer environment in case of a failed structure group.

4.4.6 Data exchange

A number of VFP messages (see 4.4.9) are mapped as presentation service data units of the P-DATA service. All VFP control information (message header and TLV parameters) are mapped into the transp-data parameter, while file data is mapped into the formatted-data parameter.

4.4.7 Dialogue facilities

The following presentation service dialogue facilities (implied by the choice of session subset C) are used for the mapping of VFP messages not sent through P-DATA.

4.4.7.1 Dialogue unit concept

The VFP uses the concept of dialogue unit (DU). A dialogue unit maps:

- a VFP structure group outside of file data transfer, if grouping is used,
- all consecutive VFP dialogue outside of file data transfer, if grouping is not used,
- part of a file data transfer enclosure. The transfer enclosure is only one dialogue unit if there is no restart during the transfer.

Therefore the P-END-DU service maps the EG and EGR VFP messages and must be used immediately after the ET and ETR VFP messages and, when grouping is not used, the BT and BTR messages.

When two consecutive dialogue units are of different nature, it may be necessary to exchange the End-DU and the Synchronize tokens: this is achieved through the P-TOKENS-GIVE service. Both tokens are always assigned to the same entity:

- Primary for file management dialogue units,
- Sender for file data transfer dialogue units.

4.4.7.2 Abnormal termination of DU

Outside of file data transfer, dialogue units are never abnormally terminated (except by disconnection). File data transfer dialogue units may be abnormally terminated. The P-RESYNC service maps the following VFP messages:

- AT and ATR: the resync-type value is then "abandon" and the user-data parameter of the P-RESYNC is used to map the diagnostic parameter of the VFP. Token exchange can be specified in this use of P-RESYNC: the rules are the same as in 4.4.7.1.
- RT and RTR: the resync-type value is then "restart" and the user-data parameter of the P-RESYNC is used to map the restart-position parameter of the VFP. Tokens are not exchanged in this use of P-RESYNC. The serial-number parameter must be set to its value at the beginning of the DU (in order that the Primary be always the contention winner).

4.4.7.3 Minor synchronization points

The P-SYNC service maps the CK and CKR messages of the VFP. The type is set to "normal".

4.4.8 Summary of presentation service usage

The presentation services used by the VFP are:

P-CONNECT
P-RELEASE
P-DISCONNECT
P-ABORT

P-PERFORM-NEGOTIATION
P-DATA

P-SYNC
P-END-DU
P-RESYNC
P-TOKENS-GIVE

4.4.9 Summary of VFP messages mapping

SP, SPR: P-CONNECT
RP, RPR: P-RELEASE
DP : P-DISCONNECT
SL, SLR: P-DATA
OP, OPR: P-DATA
CL, CLR: P-DATA
RL, RLR: P-DATA
BT, BTR: P-DATA [+ P-END-DU [+ P-TOKENS-GIVE]]
RA, RAR: P-DATA
CR, CRR: P-DATA
DL, DLR: P-DATA
DATA : P-DATA
EG, EGR: P-END-DU [+ P-TOKENS-GIVE]
ET, ETR: P-DATA + P-END-DU [+ P-TOKENS-GIVE]
AT, ATR: P-RESYNC
RT, RTR: P-RESYNC
CK, CKR: P-SYNC

NOTE 10

For optional mapping on P-END-DU and P-TOKENS-GIVE, see 4.4.7.1.

4.5 PROTOCOL SUBSETS

4.5.1 General

For each of the service subsets defined in 3.3, a corresponding protocol subset is defined in this clause.

4.5.2 Kernel

The protocol messages included in this subset are:

EG, EGR
SP, SPR
RP, RPR
DP
SL, SLR
OP, OPR
CL, CLR
RL, RLR

BT, BTR
DATA
ET, ETR
AT, ATR

The open-identification parameter in OP and the restart-position parameter in OP and OPR are not included in this subset.

4.5.3 Basic file management extension

The protocol messages included in this extension are:

RA, RAR
CR, CRR
DL, DLR

4.5.4 Restart extension

The protocol messages included in this extension are:

RT, RTR
CK, CKR

The open-identification parameter of OP and the restart-position parameter of OP and OPR are included in this extension.

5 Conformance

5.1 CONFORMANCE REQUIREMENTS

5.1.1 General

This clause defines the conformance requirements for the virtual file protocol defined in section 4 of the standard.

There is no conformance requirement for the abstract virtual file service defined in section 3 of the standard.

Only the externally visible and externally testable criteria are defined.

5.1.2 Equipment

The conformance requirement is for equipment which consists of hardware and/or software and has the purpose of conforming with this standard. The equipment may also have other purposes.

5.1.3 Peer equipment

Any execution of the protocol necessarily involves a peer equipment with which the subject equipment communicates. For purposes of verifying conformance, it is assumed that this other peer equipment:

- is operating in conformance with the standard;
- may be capable of controlled deviation, in that it may be the source of deliberate protocol errors for the purpose of testing.

This conformance requirement does not distinguish any differences of conformance status between the two equipments (i.e. the notion of a "reference equipment" with a "definitive implementation" is not used).

5.1.4 Protocol subsets

The equipment shall implement one or more of the protocol subsets defined in 4.5, in the role of Primary or Secondary or both. For each role, the supplier of the equipment shall nominate which of these subsets the equipment is intended to conform with. Any number of the following can be nominated:

(1) Kernel
(2) Kernel + restart
(3) Kernel + basic file management
(4) Kernel + restart + basic file management

NOTE 11

Conformance with subset (N) does not necessarily imply conformance with any subset (N-i).

NOTE 12

Primary and secondary need not conform with the same subsets.

The supplier shall also nominate which of the subsets of the virtual file model, as defined in 2.2, the equipment is intended to support.

5.1.5 Additional virtual file protocol

In addition to the subsets nominated as in 5.1.4, the equipment may also implement other virtual file protocol, including different subsets of the protocol defined in this standard.

Such additional provisions are themselves not in conformance with the standard, but do not prejudice conformance with the standard provided that they are separate and do not prevent use of the subsets nominated as in 5.1.4.

5.1.6 Requirements

For each role and subset nominated as in 5.1.4 above, the equipment shall support establishment, use and termination of a VFS connection by execution of the protocol according to the following criteria. The subject equipment:

- shall accept correct sequences of VFP messages received from peer equipment, and respond with correct VFP message sequences, for the defined states of a VFS connection;
- shall respond correctly to all incorrect sequences of VFP messages received for a defined state of a VFS connection;
- shall accept all correct sequences of the presentation mapping;
- shall only issue correct sequences of the presentation mapping;
- should demonstrate a series of file operations which give evidence that the virtual filestore implementation is behaving as implied by the virtual file model (for example, create a file, retrieve its attributes, write data to it, reread the data from it, delete the file, verify its absence).

NOTE 13

In particular implementations, however, reversible mapping may not be needed or may be difficult to achieve. For example, a real filestore which supports only variable size records can obviously map a virtual file with fixed size records, but may be unable to "remember" that the records were fixed size.

The terms "correct sequences" and "incorrect sequences" refer to the protocol defined in section 4 and Appendix D.

Appendices

APPENDIX A

BRIEF DESCRIPTION OF THE REFERENCE MODEL OF OPEN SYSTEMS INTERCONNECTION

A.1 SCOPE

This appendix is not an integral part of the standard. It is a copy of ISO/TC97/SC16 N 575.

This appendix provides a brief description of the Reference Model of Open Systems Interconnection.

A.2 GENERAL DESCRIPTION

A.2.1 Introduction

The Reference Model of Open Systems Interconnection provides a common basis for the co-ordination of the development of new standards for the interconnection of systems and also allows existing standards to be placed within a common framework. The model is concerned with systems comprising terminals, computers and associated devices and the means for transferring information between these systems.

A.2.2 Overall perspective

The model does not imply any particular systems implementation, technology or means of interconnection, but rather refers to the mutual recognition and support of the standardized information exchange procedures.

A.2.3 The Open Systems Interconnection environment

Open Systems Interconnection is not only concerned with the transfer of information between systems (i.e. with communication), but also with the capability of these systems to interwork to achieve a common (distributed) task. The objective of Open Systems Interconnection is to define a set of standards which allow interconnected systems to co-operate.

The Reference Model of Open Systems Interconnection recognizes three basic constituents (see fig. 1):

a) application processes within an OSI environment,

b) connections which permit information exchange,

c) the systems themselves.

NOTE A.1

The application processes may be manual, computer or physical processes.

A.2.4 Management Aspects

Within the Open Systems Interconnection architecture there is a need to recognize the special problems of initiating, terminating, monitoring on-going activities and assisting in their harmonious operations as well as handling abnormal conditions. These have been collectively considered as the management aspects of the Open Systems Interconnection architecture. These concepts are essential to the operation of the interconnected open systems and therefore are included in the comprehensive description of the Reference Model.

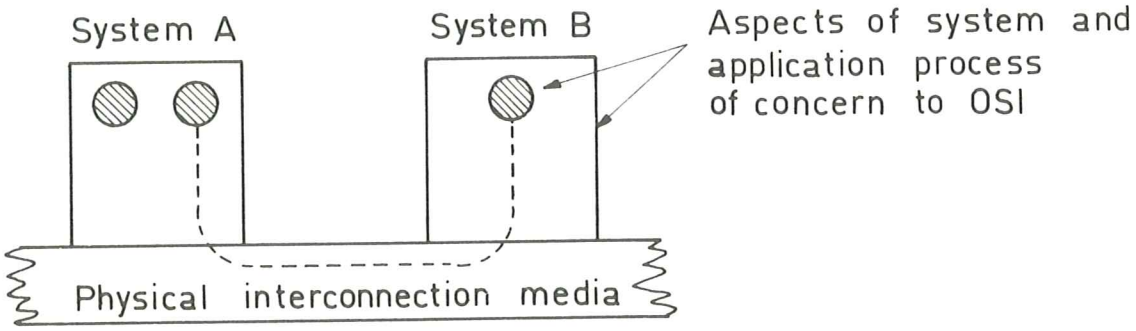

Fig. 1 - General schematic diagram illustrating the basic elements of Open Systems Interconnection

A.2.5 Concepts of a Layered Architecture

The open systems architecture is structured in Layers. Each system is composed of an ordered set of subsystems represented for convenience by Layers in a vertical sequence. Adjacent subsystems communicate through their common interface.

A Layer consists of all subsystems with the same rank. The operation of a layer is the sum of the co-operation between entities in that Layer. It is governed by a set of protocols specific to that Layer.

The services of a Layer are provided to the next higher Layer, using the functions performed within the Layer and the services available from the next lower Layer.

An entity in a Layer may provide services to one or more entities in the next higher Layer and use the services of one or more entities in the next lower Layer.

A.3 THE LAYERED MODEL

The seven-Layer Reference Model is illustrated in fig.2.

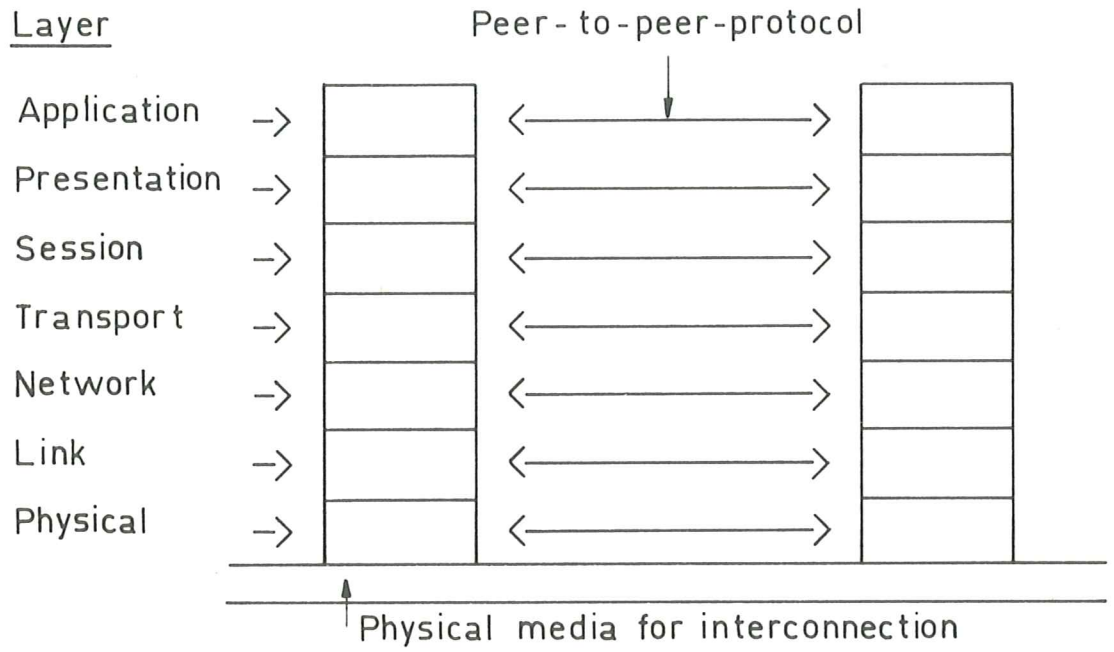

Fig. 2 - The seven-layer Reference Model and peer-to-peer protocol

A.3.1 The Application Layer

As the highest layer in the Reference Model of Open Systems Interconnection, the Application Layer provides services to the users of the OSI environment, not to a next higher layer.

The purpose of the Application Layer is to serve as the window between communicating users of the OSI environment through which all exchange of meaningful (to the users) information occurs.

The user is represented by the application-entity to its peer.

All user specifiable parameters of each communications instance are made known to the OSI environment (and, thus, to the mechanisms implementing the OSI environment) via the Application Layer.

A.3.2 The Presentation Layer

The purpose of the Presentation Layer is to represent information to communicating application-entities in a way that preserves meaning while resolving syntax differences.

The nature of the boundary between the Application Layer and the Presentation Layer is different from the nature of other Layer boundaries in the architecture.

The following principles are adopted to define a boundary between the Application Layer and the Presentation Layer:

a) internal attributes of the virtual resource and its manipulation functions exist in the Presentation Layer;

b) external attributes of the virtual resource and its manipulation functions exist in the Application Layer;

c) the functions to use the services of the Session Layer effectively exist in the Presentation Layer;

d) the functions to use services of the Presentation Layer effectively exist in the Application Layer.

A.3.3 The Session Layer

The purpose of the Session Layer is to provide the means necessary for cooperating presentation-entities to organize and synchronize their dialogue and manage their data exchange. To do this, the Session Layer provides services to establish a session-connection between two presentation entities, and to support their orderly data exchange interactions.

To implement the transfer of data between the presentation-entities, the session-connection is mapped onto and uses a transport-connection.

A.3.4 The Transport Layer

The Transport Layer exists to provide the transport-service in association with the underlying services provided by the supporting layers.

The transport-service provides transparent transfer of data between session entities. Transport Layer relieves the transport users from any concern with the detailed way in which reliable and cost effective transfer of data is achieved.

The Transport Layer is required to optimize the use of the available communication resources to provide the performance required by each communicating transport user at minimum cost. This optimization will be achieved within the constraints imposed by considering the global demands of all concurrent transport users and the overall limit of resources available to the Transport Layer. Since the network service provides network connections from any transport entity to any other, all protocols defined in the Transport Layer will have end-to-end significance, where the ends are defined as the correspondent transport-entities.

The transport functions invoked in the Transport Layer to provide requested service quality will depend on the quality of the network service. The quality of the network service will depend on the way the network service is achieved.

A.3.5 The Network Layer

The Network Layer provides the means to establish maintain and terminate network connections between systems containing communicating application-entities and the functional and procedural means to exchange network service data units between two transport entities over network connections.

A.3.6 The Link Layer

The purpose of the Link Layer is to provide the functional and procedural means to activate, maintain and deactivate one or more data link connections among network entities.

The objective of this layer is to detect and possibly correct errors which may occur in the Physical Layer. In addition, the Link Layer conveys to the Network Layer the capability to request assembly of data circuits within the Physical Layer (i.e. the capability of performing control of circuit switching).

A.3.7 The Physical Layer

The Physical Layer provides mechanical, electrical, functional and procedural characteristics to activate, maintain and deactivate physical connections for bit transmission between data link entities possibly through intermediate systems, each relaying bit transmission within the Physical Layer.

APPENDIX B

TERMINOLOGY

B.1 GENERAL

This Appendix is an integral part of the standard.

The terminology used in this standard consists of:

- Reference Model terminology, which is defined in ISO 7498.
- Terminology for virtual file model, services and protocol, which is defined in this Appendix (see B.2).
- Notation terminology, which is defined in Appendix C.

B.2 DEFINITIONS

File: see Virtual File.

Filestore: see Virtual Filestore.

File-Access: the inspection or manipulation of all or part of the contents of a file.

File-Attributes: the standardized properties of a file.

File-Management: the inspection or manipulation of the attributes associated with a file.

File Mapping: the process of relating a real file to a virtual file and vice-versa.

File-Transfer: a function which moves the contents and associated properties of a file from one filestore to another.

Message: used in this document to designate a Virtual File protocol-data-unit, consisting of Virtual File Protocol control information and possibly file data.

Primary: the Virtual File Service entity (or the VFS user) which initiates the file transfer.

Real File: a named collection of information with a common set of properties, as stored in a specific system.

Real Filestore: a collection of real files residing at a particular system, including description of their properties and names.

Record: the unit of data within a file, that can be individually accessed through the Virtual File Service. Applicable to most file structures; exceptions are named unstructured files.

Secondary: the Virtual File Service entity (or the VFS user) which accepts the file transfer initiated by the Primary.

Virtual File: a commonly agreed image of a file, created so as to allow open working between dissimilar real files. The Virtual File Service only operates on virtual files. In this document, "file" is always used as a short form for "virtual file".

Virtual Filestore: a collection of virtual files, residing at a particular system, including descriptions of their properties and names. In this document, "filestore" is always used as a short form for "virtual filestore".

Virtual File Service (VFS): a standardized service, which provides an independence of:

- the specific host system
- the filestore location
- the local data representation.

This service can operate on virtual files and virtual filestores. It is composed of file management, file access and file transfer.

Virtual File Service Entity (VFS entity): addressable entity to provide virtual file service.

APPENDIX C

NOTATION

C.1 INTRODUCTION AND SCOPE

This Appendix is an integral part of the standard.

It is a reproduction of notation defined in the ECMA register of common techniques for OSI standards, particularised by substitution of the acronym "VFS" into the terms defined.

C.2 DEFINITIONS

This terminology is for the notation defined in this Appendix.

(x)-service: a conceptual unit of the virtual file service, of which (x) is its particular name.

(Service) primitive: a discrete component of an (x)-service.

(x)-Request primitive: a type of primitive by means of which a VFS user causes an occurence of the (x)-service.

(x)-Indication primitive: a type of primitive by means of which a VFS user is informed of an occurence of the (x)-service.

(x)-Response primitive: a type of primitive by means of which a VFS user replies to an occurence of an (x)-indication primitive.

(x)-Confirmation primitive: a type of primitive by means of which a VFS user is informed of an occurence of an (x)-response primitive.

Service structure: the series of one or more primitives of which an (x)-service wholly consists.

Service structure type 1: a service structure with a request primitive and an indication primitive.

Service structure type 2: a service structure with a request primitive, an indication primitive, a response primitive and a confirmation primitive.

Service structure type 3: a service structure with two indication primitives.

Event: the occurence of a primitive.

Initiator: the VFS user who issues the request primitive of the (x)-service concerned.

Acceptor: the VFS user who receives the indication primitive of the (x)-service concerned.

C.3 SERVICE MODEL

The virtual file service is modelled as an abstract service to which VFS users gain access at VFS-access-points. All interactions are between two VFS users located at separate VFS-access-points. A single VFS connection is modelled.

C.4 PRIMITIVES

The virtual file service is defined by means of service primitives.

Primitives are conceptual and are not intended to be directly related to virtual file protocol elements or to the units of interaction across a procedural interface in an implementation. The descriptive technique is independent of such variable details.

Primitives which relate only to local conventions between a VFS user and an implementation are not defined.

The subdivision of the virtual file service into the particular primitives chosen is arbitrary, in that the same virtual file service could be described by other logically equivalent primitives. There is no notion that a primitive is "elementary".

A primitive occurs at one service access point (not both). It usually has parameters, containing values related to its occurence.

The occurence of a primitive is a logically instantaneous and indivisible event. The primitive occurs at a logically separate instant, which cannot be interrupted by the occurence of another primitive. It occurs either completely or not at all.

There are four types of primitives in this standard (see C.2):

a) request primitive
b) indication primitive
c) response primitive
d) confirmation primitive.

The primitives are given names prefixed by "F" (file) to distinguish them from primitives of adjacent layers. The names of the primitives are written in upper case, e.g. F-DATA.

C.5 SERVICE STRUCTURE

Each service consists of one or more primitives and affects both service access points. There are three different combinations of primitives. These combinations are referred to as service structures. The three service structure types used in this standard are defined in C.2.

Unlike the occurence of its constituent primitives, the occurence of a service structure is not logically instantaneous and indivisible. The intervals between its constituent primitives may be non-disruptively interspersed with the primitives of other service structures, subject to restrictions particular to the service concerned. Service structures may also be disrupted by the occurence of certain other primitives (see C.6).

C.6 EFFECTS OF SERVICES

The effects of a service are referred to as being sequentially transmitted if its successive primitives at one service access point result in the same sequence of corresponding primitives at the other service access point (unless disrupted, see below).

The effects of a service are referred to as being expedited if its indication or confirmation primitives may arrive at the other service access point before those of a previous occurence of a service.

The effects of a service are referred to as being disruptive if it may destroy, and therefore prevent the occurence of, indication and confirmation primitives corresponding to previous request or response primitives. The effects of disrupted services are expedited, unless stated otherwise.

The effects of a primitive are referred to as being non-disruptive if they do not have the above disruptive effects. Non-disruptive effects may include effects relating to or delaying other events without destroying them.

Unless otherwise specified, the effects of a service are sequentially transmitted and non-disruptive.

C.7 PARAMETER NOTATION

For each service structure, the parameters are defined by a table. followed by a list of parameter descriptions. The column headings in the parameter tables indicate the primitive types: REQ for Request, IND for Indication, RESP for Response, CONF for Confirmation.

The values in the columns of the parameter tables obey the following conventions:

D (down): value supplied by the VFS user in the primitive

U (up): value supplied by the VFS in the primitive

B (both): value supplied either by the user or by the VFS in the primitive

x: parameter not used in the primitive.

The detailed description of a parameter includes its purpose, the rule for setting its value, the default value and the legal values. All parameters which are supplied by the user (D) both in request and response primitives are negotiated. The others are not negotiated.

Unless otherwise stated, the parameter value in the indication is the same as that in the request, and the parameter value in the confirmation is the same as that in the response.

Parameter values are only defined to a level which distinguishes meaning, but generally without defining their absolute value or encoding. These details are outside the scope of the standard, being local conventions between the VFS user and an implementation.

APPENDIX D

FORMAL DESCRIPTION

D.1 INTRODUCTION

This Appendix is an integral part of the standard.

In section 4 of the standard, the virtual file protocol interactions between two VFS entities are described. That description references states, events and actions which in this appendix are consolidated into a formal description of the protocol, as a Finite State Machine (FSM).

NOTE D.1

There is no formal description of the presentation mapping.

D.2 ELEMENTS USED IN THE FORMAL DESCRIPTION

Table D/1 lists the states which are used in the formal description. For each entry there is a state code and a brief description.

Table D/2 lists the events which are used in the formal description. For each entry there is an event code and a brief description.

Table D/3 lists the message acronyms which are used in the formal description to identify messages sent.

Table D/4 lists the conditions which are used in the formal description. For each entry there is a condition code and a brief description.

Table D/5 lists the valid groupings of request messages, by describing which pairs of consecutive requests (preceding, following) are allowed in the same group, i.e. without an intervening EG message. "Y" entries indicate valid combinations, blank entries invalid combinations. "NS" entries refer to valid but irrational combinations (not significant).

The way in which these various elements are used is defined in D.3.

Table D/1 - FSM States

State-code	State-description	State tables
CF..	(Connection Facility states)	
CF01	Dormant	D/7 D/8
CF02	SP-pending	D/7 D/8
CF03	RP-pending	D/7 D/8
FM..	(File Management states)	
FM01	No-file	D/7 D/8 D/9 D/10
FM02	SL-pending	D/9 D/10
FM03	CR-pending	D/9 D/10
FM04	File-selected	D/9 D/10 D/11 D/12
FM05	RL-pending	D/9 D/10
FM06	DL-pending	D/9 D/10
FM07	RA-pending	D/9 D/10
FM08	OP-pending	D/11 D/12
FM09	File-open	D/11 D/12
FM10	CL-pending	D/11 D/12
FM11	BT-pending	D/11 D/12
FM12	EGR-pending	D/11B
FM13	wait-for-EG	D/12B
DT..	(Data Transfer states)	
DT01	Data	D/13 D/14 D/15 D/16
DT02	ET-pending	D/13 D/15 D/16
DT03A	RT-pending (Sender initiated)	D/15 D/16
DT03B	RT-pending (Receiver initiated)	D/15 D/16
DT04A	AT-pending (Sender initiated)	D/15 D/16
DT04B	AT-pending (Receiver initiated)	D/15 D/16
DT05	File-aborted	D/11 D/12
DT06	CK-pending	D/14 D/16

NOTE D.2

The state tables referenced are those where the state-code appears as a column heading. The state-code may also appear inside other tables as a resulting state. Tables D/7 and D/8 should be implicitly added to the list for all state-codes, since they contain "Any other state" as a column heading .

Table D/2 - FSM Events

Event-code	Event description
XX	XX request message
XXR	XX response message
XX-RQ	Request primitive associated to XX
XX-IN	Indication primitive associated to XX
XX-RP	Response primitive associated to XX
XX-CF	Confirmation primitive associated to XX

NOTE D.3

Table D/3 gives the list of valid values for XX. An example is given below, where XX = SL.

SL : SL request message
SLR : SLR response message
SL-RQ : F-SELECT-FILE request primitive
SL-IN : F-SELECT-FILE indication primitive
SL-RP : F-SELECT-FILE response primitive
SL-CF : F-SELECT-FILE confirmation primitive

Table D/3 - List of protocol messages

Acronym	Message name	State tables
SP	Select Protocol	D/7 D/8
RP	Release Protocol	D/7 D/8
DP	Disconnect Protocol	D/7 D/8
AB(*)	Abort Connection	D/7 D/8
EG	End Group	D/9 D/10 D/11 D/12
SL	Select File	D/9 D/10
RL	Release File	D/9 D/10
CR	Create File	D/9 D/10
DL	Delete File	D/9 D/10
RA	Read Attributes	D/9 D/10
OP	Open File	D/11 D/12
CL	Close File	D/11 D/12
BT	Begin Transfer	D/11 D/12
DATA	Data	D/13 D/14
ET	End Transfer	D/15 D/16
RT	Restart Transfer	D/15 D/16
AT	Abort Transfer	D/15 D/16
CK	Checkpoint	D/13 D/14

(*) There are only two events related to the acronym AB:
AB: Presentation connection abort indication from layer 6
AB-IN: VFS connection abort indication.

NOTE D.4

The state tables referenced are those where one of the events associated with the acronym (see table D/2) appears as a row heading.

Table D/4 - Conditions

Condition	Meaning
+	Value of diagnostic severity is "success" or "success with warning"
-	Value of diagnostic severity is "failure"
pr	VFS entity is "Primary"
sc	VFS entity is "Secondary"
ag	Current request group aborted
eg	End of request group pending
cc	Checkpoint counter >0
CONDV:cond	This intersection is valid only if cond is true

Initial setting of conditions

+,-: not applicable
pr,sc: according to connection establishment
ag, eg: false
cc: false whenever entering DT01.

Table D/5 - Request grouping

prev. \ next	SL	CR	RL	DL	RA	OP	CL	BT	EG
SL			NS	Y	Y	Y			Y
CR			Y	NS	Y	Y			Y
RL	Y	Y							Y
DL	Y	Y							Y
RA			Y	Y	NS	Y			Y
OP								Y	Y
CL			Y	Y	Y	Y			Y
BT									Y

NOTE D.5

A valid grouping needs not be supported by all implementations. It is up to each implementation of a Primary to decide which amount of grouping, if any, it wants to support.

D.3 Formal Description conventions

The formal description is in tables D/7 to D/16.

The horizontal dimension of each table is the set of all the states. If for any given state there is no valid event, then the state does not appear in the table, i.e. no column.

The vertical dimension of each table is the set of all relevant FSM request events, incoming message events and FSM response events. For each event there is an entry, i.e. a row.

Each valid intersection contains:

- one or more conditions (where relevant),
- one or more actions (where relevant),
- the new state (always).

The conditions are those defined in table D/4.

The action consists of sending a message or issuing a local primitive event (indication or confirmation) or setting a condition.

The new state is the state which is entered after the specified actions are completed. When the grouping option is supported, states may be enqueued in a FIFO queue for later retrieval (as specified in 4.1.3): this is indicated by the Q(XXnn) action, where XXnn is the enqueued state. Their dequeuing is indicated by the DQ action and results in the dequeued state being entered. The PQ action indicates purging of the queue. DQ and PQ have a null effect if the queue is empty. If the queue is not empty, the dequeued stated overrides any other state indicated in the intersection. Example:

> Intersection indicates ":FM01,DQ"; the final state will be:
>
> - the dequeued state of the queue is not empty,
> - FM01 if the queue is empty.
>
> (Note that DQ may also be conditioned by + or -)

Absence of a new state in a valid table entry means the new state is the current state.

When a request message is dropped as a result of contention solving or because it belongs to an aborted group, no state change occurs and the indication primitive is not issued.

An intersection is invalid either if it contains a blank entry or if the condition indicated by CONDV is not met. All invalid intersections between states and incoming message events are treated as protocol violations: the action is disconnect the VFS connection (DP message with "protocol violation" reason-code), the new state is CF01 (Dormant).

In case of grouping, there might arrive valid messages while waiting for a response primitive (at the Secondary) or an end-group request primitive (at the Primary). These messages should be thought of being gated until the expected response or end-group primitive has arrived, rather than taken as invalid.

All invalid intersections between states and incoming primitive events are treated as local errors, in a way not specified in this Standard.

NOTE D.6

Where contentions are solved by lower layer services, they are not shown in the state tables, since they cannot validly appear in this layer (this occurs with the following states: DT03A, DT03B, DT04A, DT04B).

The following editorial conventions are used:

":" = action with regard to state change or queuing
"," = connects list of actions
"(a&b)"= logical "and" of conditions a and b
"^" = logical negation
"set xx" = set condition xx on
"set ^ xx" = set condition xx off
"inc cc" = increment cc by one
"dec cc" = decrement cc by one.

D.4 Formal Description Tables

The complete state table for the virtual file protocol FSM is too large to be edited on a single page. In addition, it contains a majority of empty cells, representing error cases. Therefore this state table has been segmented into smaller tables and besides sub-tables with only empty cells have not been represented.

In addition, it is necessary to distinguish two FSM's according to the role played by the VFS entities: Primary/Secondary in some cases, Sender/Receiver in other cases. Table D/6 is an index of the formal description tables.

Table D/6 - Index of formal description tables

Table	Subprotocol	Role
D/7	Connection Facility	Primary
D/8	Connection Facility	Secondary
D/9	File Management (file enclosure)	Primay
D/10	File Management (file enclosure)	Secondary
D/11	File Management (open enclosure)	Primary
D/12	File Management (open enclosure)	Secondary
D/13	Data Transfer (data flow)	Sender
D/14	Data Transfer (data flow)	Receiver
D/15	Data Transfer (termination)	Sender
D/16	Data Transfer (termination)	Receiver

File Management tables are subdivided into 2 subtables: one for the case where grouping is not used (D/9A, D/10A, D/11A, D/12A), the other where grouping is used (D/9B, D/10B, D/11B, D/12B).

Table D/7 - Connection Facility, Primary

	CF01 Dormant	CF02 SP-PEND	CF03 RP-PEND	FM01 NO-FILE	Any other state			
SP-RQ	: CF02 SP							
SPR		SP-CF +: FM01 -: CF01						
RP-RQ				:CF03 RP				
RPR			RP-CF :CF01					
DP-RQ		:CF01 DP	:CF01 DP	:CF01 DP	:CF01 DP			
DPR		DP-IN :CF01	DP-IN CF01	DP-IN :CF01	DP-IN :CF01			
AB		AB-IN :CF01	AB-IN :CF01	AB-IN :CF01	AB-IN :CF01			

Table D/8 - Connection Facility, Secondary

	CF01 Dormant	CF02 SP-PEND	CF03 RP-PEND	FM01 NO-FILE	Any other state			
SP	SP-IN :CF02							
SP-RP		+:FM01 -:CF01 SPR						
RP				RP-IN :CF03				
RP-RP			:CF01 RPR					
DP-RQ		:CF01 DP	:CF01 DP	:CF01 DP	:CF01 DP			
DP		DP-IN :CF01	DP-IN :CF01	DP-IN :CF01	DP-IN :CF01			
AB		AB-IN :CF01	AB-IN :CF01	AB-IN :CF01	AB-IN :CF01			

Table D/9A - File Management (file enclosure), Primary no grouping

	FM01 NO-FILE	FM02 SL-PEND	FM03 CR-PEND	FM04 FILE- SELECTED	FM05 RL-PEND	FM06 DL-PEND	FM07 RA-PEND	
SL-RQ	:FM02 SL							
SLR		SL-CF +:FM04 -:FM01						
CR-RQ	:FM03 CR							
CRR			CR-CF +:FM04 -:FM01					
RL-RQ				:FM05 RL				
RLR					RL-CF :FM01			
DL-RQ				:FM06 DL				
DLR						DL-CF :FM01		
RA-RQ				:FM07 RA				
RAR							RA-CF :FM04	

Table D/9B - File Management (File enclosure), Primary, grouping

	FM01 NO-FILE	FM02 SL-PEND	FM03 CR-PEND	FM04 FILE-SELECTED	FM05 RL-PEND	FM06 DL-PEND	FM07 RA-PEND	FM09 DT05 FILE-OPEN FILE-ABORT
SL-RQ	CONDV:^ eg Q(FM02) SL			CONDV:^ eg Q(FM02) SL				CONDV:^eg Q(FM02) SL
SLR		SL-CF +:FM04,DQ -:FM01						
CR-RQ	CONDV:^ eg Q(FM03) CR			CONDV:^eg Q(FM03) CR				CONDV:^eg Q(FM03) CR
CRR			CR-CF +:FM04,DQ -:FM01					
RL-RQ	CONDV:^ eg Q(FM05) RL			CONDV:^eg Q(FM05) RL				CONDV:^eg Q(FM05) RL
RLR					RL-CF :FM01 + DQ			
DL-RQ	CONDV:^eg Q(FM06) DL			CONDV:^eg Q(FM06) DL				CONDV:^eg Q(FM06) DL
DLR						DL-CF :FM01 +DQ		
RA-RQ	CONDV:^eg Q(FM07) RA			CONDV:^eg Q(FM07) RA				CONDV:^eg Q(FM07) RA
RAR							RA-CF :FM04 +DQ	
EG-RQ	CONDV:^eg DQ, set eg EG			CONDV:^eg DQ, set eg EG				CONDV:^eg DQ, set eg EG
EGR	CONDV: eg EG-CF PQ, set ^eg			CONDV: eg EG-CF PQ, set ^eg				CONDV: eg EG-CF PQ, set ^eg

Table D/10A - File Management (File enclosure), Secondary, no grouping

	FM01 NO-FILE	FM02 SL-PEND	FM03 CR-PEND	FM04 FILE- SELECTED	FM05 RL-PEND	FM06 DL-PEND	FM07 RA-PEND	
SL	SL-IN :FM02							
SL-RP		+:FM04 -:FM01 SLR						
CR	CR-IN :FM03							
CR-RP			+:FM04 -:FM01 CRR					
RL				RL-IN :FM05				
RL-RP					:FM01 RLR			
DL				DL-IN :FM06				
DL-RP						:FM01 DLR		
RA				RA-IN :FM07				
RA-RP							:FM04 RAR	

Table D/10B - File Management (File enclosure), Secondary, grouping

	FM01 NO-FILE	FM02 SL-PEND	FM03 CR-PEND	FM04 FILE-SELECTED	FM05 RL-PEND	FM06 DL-PEND	FM07 RA-PEND	FM09 FILE-OPEN
SL	CONDV:^eg ^ag SL-IN ^ag:FM02 ag:FM01			CONDV: (^eg&ag) :FM04				CONDV: (^eg&ag) :FM09
SL-RP		+:FM04 -:FM01 - set ag SLR						
CR	CONDV:^eg ^ag CR-IN ^ag:FMC3 ag:FM01			CONDV: (^eg&ag) :FM04				CONDV: (^eg&ag) :FM09
CR-RP			+:FM04 -:FM01 - set ag CRR					
RL	CONDV: (^eg&ag) :FM01			CONDV:^eg ^ag RL-IN ^ag:FM05 ag:FM04				CONDV: (^eg&ag) :FM09
RL-RP					:FM01 -set ag RLR			
DL	CONDV: (^eg&ag) FM01			CONDV:^eg ^ag DL-IN ^ag:FM06 ag:FM04				CONDV: (^eg&ag) :FM09
DL-RP						:FM01 - set ag DLR		
RA	CONDV: (^eg&ag) : FM01			CONDV:^eg ^ag RA-IN ^ag:FM07 ag:FM04				CONDV: (^eg&ag) :FM09
RA-RP							:FM04 - set ag RAR	
EG	CONDV:^eg EG-IN set eg			CONDV:^eg EG-IN set eg				CONDV:^eg EG-IN set eg
EG-RP	CONDV:eg set (^ag,^eg) EGR			CONDV:eg set (^ag,^eg) EGR				CONDV:eg set (^ag,^eg) EGR

Table D/11A - File Management (open enclosure), Primary, no grouping

	FM04 FILE- SELECTED	FM08 OP-PEND	FM09 FILE-OPEN	FM10 CL-PEND	FM11 BT-PEND	DT05 FILE- ABORTED		
OP-RQ	:FM08 OP							
OPR		OP-CF +:FM09 -:FM04						
CL-RQ			:FM10 CL			:FM10 CL		
CLR				CL-CF :FM04				
BT-RQ			:FM11 BT					
BTR					BT-CF +:DT01 -:FM09			

Table D/11B - File Management (open enclosure), Primary, grouping

	FM01 NO-FILE	FM04 FILE-SELECTED	FM08 OP-PEND	FM09 FILE-OPEN	FM10 CL-PEND	FM11 BT-PEND	FM12 EGR-PEND	DT05 FILE-ABORTED
OP-RQ	CONDV:^eg Q(FM08) OP	CONDV:^eg Q(FM08) OP		CONDV:^eg Q(FM08) OP				CONDV:^eg Q(FM08) OP
OPR			OP-CF +:FM09,DQ -:FM04					
CL-RQ				CONDV:^eg Q(FM10) CL				CONDV:^eg Q(FM10) CL
CLR					CL-CF :FM04 +DQ			
BT-RQ	CONDV:^eg Q(FM11) BT	CONDV:^eg Q(FM11) BT		CONDV:^eg Q(FM11) BT				CONDV:^eg Q(FM11) BT
BTR						BT-CF +:FM12 -:FM09		
EG-RQ	see D/9B	see D/9B		see D/9B				see D/9B
EGR	see D/9B	see D/9B		see D/9B			EG-CF set ^eg :DT01	see D/9B

Table D/12A - File Management (open enclosure), Secondary, no grouping

	FM04 FILE - SELECTED	FM08 OP-PEND	FM09 FILE-OPEN	FM10 CL-PEND	FM11 BT-PEND	DT05 FILE - ABORTED		
OP	OP-IN :FM08							
OP-RP		+:FM09 -:FM04 OPR						
CL			CL-IN :FM10			CL-IN :FM10		
CL-RP				:FM04 CLR				
BT			BT-IN :FM11					
BT-RP					+:DT01 -:FM09 BTR			

Table D/12B - File Management (open enclosure), Secondary, grouping

	FM01 NO-FILE	FM04 FILE - SELECTED	FM08 OP-PEND	FM09 FILE-OPEN	FM10 CL-PEND	FM11 BT-PEND	FM13 WAIT FOR EG	DT05 FILE - ABORTED
OP	CONDV: (^eg&ag) :FM01	CONDV:^eg ^ag OP-IN ag:FM08 ag:FM04						
OP-RP			+:FM09 -:FM04 - set ag OPR					
CL				CONDV:^ eg CL-IN :FM10				CONDV:^ eg CL-IN :FM10
CL-RP					:FM04 - set ag CLR			
BT	CONDV: (^eg&ag) :FM01	CONDV: (^eg&ag) :FM04		CONDV:^ eg ^ag BT-IN ^ag:FM11 ag:FM09				
BT-RP						+:FM13 -:FM09 - set ag BTR		
EG	see D/10B	see D/10B		see D/10B			CONDV:^ eg EG-IN set eg	
EG-RP	see D/10B	see D/10B		see D/10B			CONDV:eg set (^ag,^eg) :DT01 EGR	

Table D/13 - Data Transfer (data flow), sender

	DT01 DATA	DT02 ET-PEND						
DATA-RQ	:DT01 DATA							
CK-RQ	:DT01 inc cc CK							
CKR	CONDV: cc CK-CF dec cc	CONDV: cc CK-CF dec cc						

Table D/14 - Data Transfer (data flow), Receiver

	DT01 DATA	DT06 CK-PEND						
DATA	DATA-IN :DT01							
CK	CK-IN :DT06							
CK-RP		:DT01 CKR						

Table D/15 - Data Transfer (termination), Sender

	DT01 DATA	DT02 ET-PEND	DT03A RT-PEND (INIT=SN)	DT03B RT-PEND (INIT=RC)	DT04A AT-PEND (INIT=SN)	DT04B AT-PEND (INIT=RC)		
ET-RQ	:DT02 ET							
ETR		ET-CF :FM09						
RT-RQ	:DT03A RT							
RTR			RT-CF :DT01					
RT	RT-IN :DT03B	RT-IN :DT03B	CONDV:sc RT-IN :DT03B		:DT04A			
RT-RP				:DT01 RTR				
AT-RQ	:DT04A AT			:DT04A AT				
ATR					AT-CF :DT05			
AT	AT-IN :DT04B	AT-IN :DT04B	AT-IN :DT04B		CONDV:sc AT-IN :DT04B			
AT-RP						:DT05 ATR		

Table D/16 - Data Transfer (termination) Receiver

	DT01 DATA	DT02 ET-PEND	DT03A RT-PEND (INIT=SN)	DT03B RT-PEND (INIT=RC)	DT04A AT-PEND (INIT=SN)	DT04B AT-PEND (INIT=RC)	DT06 CK-PEND	
ET	ET-IN :DT02							
ET-RP		:FM09 ETR						
RT	RT-IN :DT03A			CONDV:sc RT-IN :DT03A		:DT04B	RT-IN :DT03A	
RT-RP			:DT01 RTR					
RT-RQ	:DT03B RT	:DT03B RT					:DT03B RT	
RTR				RT-CF :DT01				
AT	AT-IN :DT04A			AT-IN :DT04A		CONDV:sc AT-IN :DT04A	AT-IN :DT04A	
AT-RP					:DT05 ATR			
AT-RQ	:DT04B AT	:DT04B AT	:DT04B AT				:DT04B AT	
ATR						AT-CF :DT05		

APPENDIX E

FILE TRANSFER APPLICATION SERVICE

E.1 Introduction

This Appendix is not an integral part of the Standard, since standardizing a file transfer application service is not a purpose of this Standard.

This Appendix describes an example of a File Transfer Application service, which can be provided to an end user by using the VFS. The distinction between the File Transfer Application service and the VFS is shown in figure E-1.

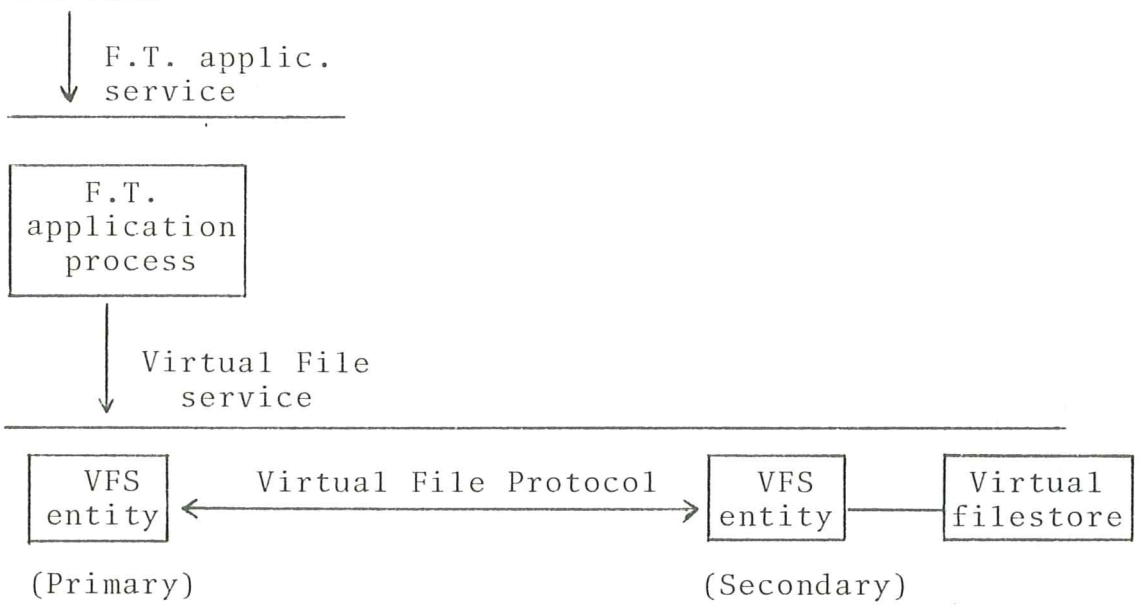

Fig. E-1: File Transfer Application and VFS

The example identifies 4 possible file transfer application service elements: COPY, SUSPEND, RESTART, CANCEL.

E.2 COPY Service

The COPY service always involves two files: a source file and a sink file. The effect of the service is to copy the first one into the second one. In particular conditions and on user request, the sink file can be created or the source file can be deleted. The operation succeeds only if the whole source file is transferred and copied. Partial file transfer service is not provided.

Referring to the source and sink files, the following cases must be distinguished:

a) both files are local

This case is normally resolved locally without involving OSI, but the implementor may provide the same service interface to the end user.

b) both files are remote

This case requires the definition of a management protocol, not part of this Standard, in order to transfer the request to one of the concerned systems. This management protocol has no impact on the VFS. It will require standardization of the File Transfer Application service. Using this protocol, the first File Transfer Application (local to the user) asks a corresponding File Transfer Application on one of the two remote systems to perform the transfer and report the result, as shown in figure E-2.

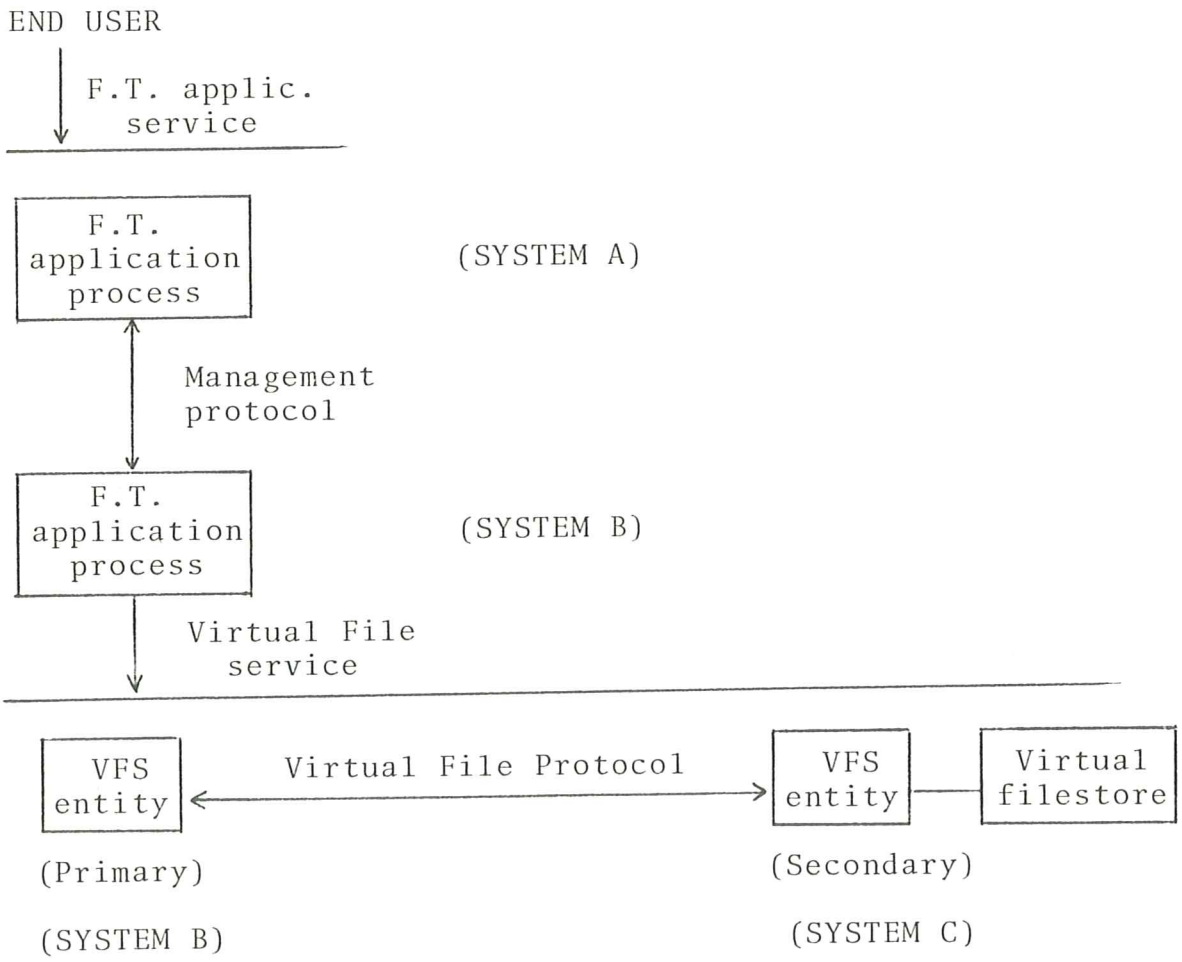

Fig. E-2: File transfer between 2 remote files

c) one of the files is local

This case is directly supported by the Virtual File Service.

In each case, the COPY service element can be defined as follows:

COPY SERVICE {(SOURCE FILE) (SINK FILE) (TRANSFER OPTIONS)}

The parameters indicate three sets of information that the user must supply to access the service.

The one concerning the SOURCE FILE includes:

File name: to identify the file and its location.
File password(s): to gain access to it.
Lock options: to allow the user to specify if the source file must be locked during the transfer or can be concurrently accessed by other file system users.
Delete option: to allow the user to require the deletion of the source file after a successful transfer.

The parameters concerning the SINK FILE include:

File name: as above
File password(s): as above
Operation mode: to specify the effect of the copy on the sink file.

APPEND ONLY or REPLACE ONLY: successful only if the sink file exists.
MAKE ONLY: successful only if the sink file does not exist.
MAKE OR APPEND or MAKE OR REPLACE: independent of the existence of the sink file.

The TRANSFER OPTIONS parameters include:

Transfer priority.
Transfer reliability options: recovery of interrupted transfer; rollback of sink file to its previous content in case the transfer cannot be successfully completed. Recovery and rollback should be user-selectable options, because they may affect the performance of the transfer.

E.3 SUSPEND service

The SUSPEND allows the user to immediately suspend a transfer in progress, for any time duration (minutes, hours or days). It is significant only if the recovery option has been selected.

E.4 RESTART service

The RESTART service allows the user to restart an interrupted transfer; interruption may be due either to accidental failure or to use of the SUSPEND service. In both cases, the transfer will be resumed at a point automatically determined (as near as possible to the point of interruption), unless the user otherwise specifies.

The RESTART service is significant only if the recovery option has been selected. It should however be used with some caution. The source and sink file should not be modified between interruption and restart of the transfer: this extended lock facility may not always be provided by the local file systems.

E.5 CANCEL service

This service allows the user to cancel a transfer in progress or pending restart. If the rollback option has been specified in the COPY service, the content of the sink file is restored to its previous state. If the sink file has been dynamically created for the transfer (MAKE option of the COPY service), it is deleted.

APPENDIX F

FUTURE EXTENSIONS

F.1 General

This Appendix is not an integral part of the Standard.

It is provided to aid understanding and use of the Standard by explaining some future extensions and related matters which have been considered by ECMA during its development.

This is a basic Virtual File protocol, designed to allow such extensions in a modular way which preserves compatibility.

F.2 Liaison with other Standardization Bodies

During the development of this Standard by ECMA, there has been close liaison with several other standard bodies, especially the ISO Open System Interconnection Sub-committee (ISO/TC97/SC16).

ECMA is an active contributor to the development of ISO virtual file standards by SC16/WG-5. The technical content of this ECMA Standard has a high degree of commonality with the current (1982) ISO work, which is still at a formative stage prior to agreement of ISO standards. The intention is that future developments will continue to benefit from this close liaison.

F.3 Flexibility for Future Extension

The flexibility for future modular extension while preserving compatibility is obtained in three main ways.

Firstly, the file model, services and protocol included in this initial Virtual File Standard have been selected to support this purpose. They are sufficiently open-ended to allow future development.

Secondly, the protocol definition has been separated from its mapping on the lower layer services. To a certain degree, each of these can be changed independently, without affecting the other.

Thirdly, the encoding is designed to be highly flexible:

- all variables within an element of protocol are individually fully encoded wherever practical (instead of collective encoding of groups of variables, restricting the possible combinations).
- self-defining formats are used to enable future insertion of new variables and the extension of existing ones.

- the encoding scheme selected for file attributes has been designed in order to easily accommodate any foreseen extension of the Virtual File model (see F.4).

F.4 Virtual File Model Enhancements

F.4.1 Principles

In the next version, priority will be given to enhancing the Virtual File model.

ECMA believes that Virtual File standardization cannot be successful if it imposes major changes to the current file system, except in the long term. Therefore the Virtual File model must not propose innovations: it must rather faithfully reflect the major existing file and data base models.

Another design guideline is to eliminate from the standardization existing file constructs which are not widely used. It is recognized that such specific file structures may require support in homogeneous networks, but it is believed that this support is better provided by a specific extension mechanism (see Appendix G) than by increasing the complexity of the Virtual File model.

For all file model enhancements, a liaison will be established with ECMA TC22 and other bodies active in data base standardization.

F.4.2 New file structures

There are currently four models which are general enough to be candidate for inclusion in the Virtual File model. By order of increasing complexity and power, these are:

- the unstructured model: the file has no elementary structure, except the character (or the octet);
- the flat model: the file is structured into records, without any relationship between these records other than record sequence. The simplest and most frequent form has a single type of records, but there may be multiple record types. The relational model is a particular case of single record type flat file;
- the hierarchical model: the file has multiple record types, with hierarchical relationships between the record types. The relationships are such that a given record type may be subordinate in only one relationship. No loop can be defined in the series of relationships;
- the network model: the file has multiple record types with relationships between the record types. Many types of relationships are allowed.

In this version, the Virtual File model supports unstructured files and single record type flat files. The next version will probably support multiple record type flat files and hierarchical files.

Support of multiple record types implies the addition of a record-type attribute, the capability to iterate the record-attributes (once for each record type), the capability to link field-attributes to the appropriate record-type, and the capability to specify the record-type in the F-DATA. All these will default to the current situation, where this information is implicit.

Support of hierarchical files implies the addition of relationship-attributes and the definition of an order of sequential access to the hierarchy.

The network model is not well adapted to file transfer, because it cannot be easily "flattened". Its support will therefore be considered only for later versions, in conjunction with the introduction of services for direct access.

F.4.3 Enhanced field description

The currently specified field descriptions have been voluntarily limited to the simplest cases. Consideration will be given to inclusion in the next version of some useful features:

- variable size character fields (the size of which is specified by the content of a previous field in the record);
- field aggregates: repeating items or repeating groups of items (the number of repetitions may be fixed or may depend on the content of a previous field in the record).

F.4.4 Key enhancements

Some concerns for a future version are:

- multicomponent keys, i.e. keys composed of several (not necessarily contiguous) record fields. This is frequently a property of secondary keys;
- secondary keys: it is not certain, however, that these should have any visibility in a file transfer.

F.5 Service Enhancements

F.5.1 Principles

The currently defined service is considered as requiring few urgent extensions, as far as the support of file transfer applications is concerned. Services for direct access or for transfer of record updates are not considered as a short term objective.

F.5.2 Partial file transfer

The most appealing service enhancement seems to be the capability to transfer parts of files instead of complete files. This capability is needed only for reading of a file.

While it is easy to agree to such a general functionality, its scope is more difficult to decide. In the simplest form, it may be limited to "from here to there", "here" and "there" being record positions or record key values. More flexible forms can involve a selection based on record types or on the values of one or more fields up to a full query capability.

The first attempt will offer a reasonably limited capability for the most common needs.

Partial file transfer will imply the addition of selection attributes to the F-BEGIN-DATA service and the capability for multiple data transfer enclosures within an open enclosure: this capability already exists in the design, but is not legal in this version since it has little significance for transfer of complete files.

F.5.3 Collections of files

Considerations will be given to the transfer of collections of files, designated by a generic name. It is perhaps sufficient to add a file management service returning the list of file names matching a generic name.

F.5.4 JTMP support

If and when requirements on the virtual file service are expressed for the support of a Job Transfer and Manipulation Protocol, these will be taken in consideration with the appropriate priority.

F.6 Conformance Testing

This Standard does not specify detailed testing methods to verify conformance. The technical work to define such testing methods is as yet only at a formative stage. It is intended to specify conformance testing and associated diagnostic aids in a future version of the Standard.

APPENDIX G

USE OF THE SPECIAL EXTENSION MECHANISM

G.1 Scope

This Appendix is not an integral part of the Standard. It is provided as an aid to implementations of the Standard.

The services and protocols defined in this Standard apply to fully heterogeneous networks. As such, they cover only general needs, since they cannot reasonably include features which are not widely supported in current file systems. However, in many cases, they will be used within less heterogeneous networks (or parts of networks) where the systems or the end users can agree to particular common conventions. For such cases, a mechanism is provided whereby special extensions can be unilaterally defined and implemented, according to particular conventions, and without degrading the openness to fully heterogeneous environments.

Particular conventions may serve two main purposes:

- addition of specific services, file models or data models;
- simplification, by assuming implicit prenegotiation of a number of parameters (useful for small systems offering a single choice for each capability).

If particular conventions have been used for general needs not satisfied in an early version of the VFP, it is strongly advised to abandon them as soon as the corresponding needs are satisfied by a new version of the VFP.

G.2 Special Extension Mechanism

A pre-requisite is to recognize if the other entity supports the same conventions. This is achieved through the special-conventions parameter of the F-CONNECT service, allowing to negotiate any number of special conventions (by repetition of the parameter). The value of this parameter is an arbitrary string of 1 or 2 octets identifying a set of conventions.

When at least one set of special conventions has been retained, the following is allowed in the other services:

- omission of mandatory parameters: their value is either pre-negotiated or supplied by special extensions;
- usage of the special-information parameter, where applicable, for conveying special extensions. If several special parameters are required for a given service, two possibilities are offered:

. repeat the special-information parameter,

. use any encoding scheme allowing to group several distinct special parameters into a single occurence of the special-information parameter (e.g. the encoding scheme defined in 4.3.2.1).

APPENDIX H

EXAMPLE OF FILE ATTRIBUTES MAPPING

H.1 Introduction

This Appendix is not an integral part of the Standard. It describes the possible mapping of the real file attributes defined in the ECMA Labelling Standards for Magnetic Tape (ECMA-13), Magnetic Tape Cassette (ECMA-41) and Flexible Disk Cartridges (ECMA-58 and ECMA-67).

Of the file attributes appearing in the specified labels, those that are medium-independent may be transferred within the VFP by means of the corresponding virtual file attributes as identified in the following table.

H.2 Mapping Table

In the table below, the "*" has the following meaning:

- under MT (Magnetic Tape): the field appears in ECMA-13 and ECMA-41;
- under FD (Flexible Disk): the field appears in ECMA-58 and ECMA-67;
- under MI (Medium Independent): the field is medium independent.

Field Name	FD	MT	MI	Equivalent VFP attribute
Label Identifier/Number	*	*		
File Identifier	*	*	*	File-name (part of)
Block Length	*	*		
Begin of Extent	*			
End of Extent	*			
Record Format	*	*	*	Record-size-type
Bypass Indicator	*			
File Accessibility	*	*	*	Access-control-list
Write Protect	*		*	(no equivalent)
Interchange Type	*			
Multivolume Indicator	*			
File Section Number	*	*		
Creation Date	*	*	*	Creation-date
Record Length	*	*	*	Record-size
Offset to Next Record Space	*			
Record Attribute	*			
File Organization	*		*	Direct-access plus record-sequence

Field Name	FD	MT	MI	Equivalent VFP attribute
Expiration Date	*	*	*	(no equivalent)
Verify Copy Indicator	*		*	(see note H.1)
End of Data	*			
File Set Identifier		*	*	File-name (part of)
File Sequence Number		*	*	File-name (part of)
Generation Number		*	*	File-name (part of)
Generation Version Number		*	*	File-name (part of)
Block Count		*		
System Code		*	*	(see note H.1)
Buffer Offset Length		*		

NOTE H.1

These fields relate to the processing history of the file, and have no equivalent in VFP.