

Open Systems Interconnection: file transfer, access and management —

Part 2: Virtual filestore definition

[ISO title: Information processing systems — Open Systems Interconnection — File Transfer, Access and Management — Part 2: Virtual Filestore Definition]

UDC 681.324.02.071.8

Committees responsible for this British Standard

The preparation of this British Standard was entrusted by the Information Systems Technology Standards Policy Committee (IST/-) to Technical Committee IST/21, upon which the following bodies were represented:

Association for Payment Clearing Services
 British Computer Society Ltd.
 British Gas plc
 British Telecommunications plc
 Business Equipment and Information Technology Association
 Computer Services Association
 Department of Trade and Industry (Alvey: Advanced Network Systems Architecture)
 Department of Trade and Industry (Information Technology Division)
 Department of Trade and Industry (Information Technology Standards Unit)
 Department of Trade and Industry (National Physical Laboratory)
 Electronic Engineering Association
 HM Treasury (Central Computer and Telecommunications Agency)
 Institute of Chartered Secretaries and Administrators
 Institute of Data Processing Management
 Institution of Electrical Engineers
 Inter-universities Computing Committee
 Joint Academic Network
 Joint Network Team
 Ministry of Defence
 National Computing Centre Ltd.
 Post Office
 Science and Engineering Research Council
 Telecommunications Engineering and Manufacturing Association
 United Kingdom Atomic Energy Authority

This British Standard, having been prepared under the direction of the Information Systems Technology Standards Policy Committee, was published under the authority of the Board of BSI and comes into effect on 31 July 1989

© BSI 02-2000

The following BSI references relate to the work on this standard:
 Committee reference IST/21
 Draft for comment DD 113-2

ISBN 0 580 17281 3

Amendments issued since publication

Amd. No.	Date of issue	Comments
6827 Corr.	September 1991	
7937	November 1993	
8063	January 1994	Indicated by a sideline in the margin

Contents

	Page
Committees responsible	Inside front cover
National foreword	iii
Foreword	ii
0 Introduction	1
1 Scope and field of application	1
2 References	1
3 Definitions	2
4 Abbreviations	2
Section 1. The filestore model	
5 Basic concepts	3
6 File selection	6
7 File structures	8
8 Actions on files	13
9 Attributes	14
Section 2. Actions on the Filestore	
10 Actions on complete files	19
11 Actions for file access	20
Section 3. Attribute definitions	
12 File Attributes	26
13 Activity attributes	28
14 Attribute groups	31
15 Minimum attribute ranges	32
Annex A File access structure constraint sets	36
Annex B Document types	45
Annex C Reading of structured files	58
Annex D Insertion in a structured file	59
Annex E ASN.1 cross reference	64
Figure 1 — Relationship of files, attributes and associations	4
Figure 1a — An example tree structure of a VFS	5
Figure 1b — An example of the apparent structure of a VFS	6
Figure 2 — The Access Structure as a Tree Structure	9
Figure 3 — ASN.1 definition of file contents	10
Figure 4 — ASN.1 definition of file structure	11
Figure 5 — Unstructured file transfer	58
Figure 6 — Flat file transfer	58
Figure 7 — Hierarchical file transfer	59
Figure 8 — Source of transferred data	60
Figure 9 — Initial state of destination file	60
Figure 10 — Final state of insert as sisters to A	60
Figure 11 — Final state of insert as children of C (normal)	61
Figure 12 — Final state of insert as children of C (variant)	62
Figure 13 — Final state of insert subtree as sister	62
Figure 14 — Final state of insert subtree as child	63
Figure 15 — Initial state of ordered flat file	63
Figure 16 — Final state of merge of ordered flat files	64
Table 1 — Result of reading in an access context	12
Table 1a — Bitstring and bitstring pattern relationships	16
Table 1b — Attribute relationship combinations	16

	Page
Table 2 — Effect of actions on location	21
Table 2a — access control and permitted actions by object type	25
Table 3 — Concurrency control options	29
Table 3a — Object attribute groups	32
Table 4 — Activity attributes	33
Table 5 — Object attributes	34
Table 6 — Basic constraints in the unstructured constraint set	37
Table 7 — Basic constraints in the sequential flat constraint set	38
Table 8 — Identity constraints in the sequential flat constraint set	38
Table 9 — Basic constraints in the ordered flat constraint set	39
Table 10 — Identity constraints in the ordered flat constraint set	39
Table 11 — Basic constraints in the ordered flat constraint set with unique names	40
Table 12 — Identity constraints in the ordered flat constraint set with unique names	41
Table 13 — Basic constraints in the ordered hierarchical constraint set	41
Table 14 — Identity constraints in the ordered hierarchical constraint set	42
Table 15 — Basic constraints in the general hierarchical constraint set	43
Table 16 — Identity constraints in the general hierarchical constraint set	43
Table 17 — Basic constraints in the general hierarchical constraint set with unique names	44
Table 18 — Identity constraints in the general hierarchical constraint set with unique names	44
Table 19 — Information objects in the unstructured text document type	45
Table 20 — Information objects in the sequential text document type	48
Table 21 — Information objects in the unstructured binary document type	52
Table 22 — Information objects in the sequential binary document type	54
Table 23 — Information objects in the hierarchical document type	57
Publications referred to	Inside back cover

National foreword

This Part of BS 7090 has been prepared under the direction of the Information Systems Technology Standards Policy Committee. It is identical with ISO 8571-2:1988 *“Information processing systems — Open Systems Interconnection — File Transfer, Access and Management — Part 2: Virtual Filestore Definition”*, including Technical Corrigendum 1, Amendment 1 and Amendment 2. The International Standard was prepared and published by the International Organization for Standardization (ISO) with the active participation and approval of the UK. It is a revision of DD 113-2:1987 which is withdrawn.

The other Parts of BS 7090 are as follows:

- *Part 1: General introduction;*
- *Part 3: File service definition;*
- *Part 4: File protocol specification;*
- *Part 5: PICS proforma¹⁾.*

Up-to-date information on existing and projected British Standards publications for Open Systems Interconnection is included in the “Basic List”, published as part of the BSI Information Technology Services (BITS) initiative. For details contact: Marketing Department, BSI, Linford Wood, Milton Keynes, MK14 6LE.

Terminology and conventions. The text of the International Standard has been approved as suitable for publication as a British Standard without deviation. Some terminology and certain conventions are not identical with those used in British Standards; attention is drawn especially to the following.

Wherever the words “International Standard” or “ISO 8571” appear, referring to this standard, they should be read as “British Standard” or “BS 7090” respectively.

Cross-references

International Standard	Corresponding British Standard
ISO 7498:1984	BS 6568 <i>Reference model of open systems interconnection</i> Part 1:1988 <i>Basic reference model (incorporating connectionless-mode transmission)</i> (Identical) BS 7090 <i>Open Systems Interconnection: file transfer, access and management</i>
ISO 8571-1:1988	Part 1:1989 <i>General introduction</i> (Identical)
ISO 8571-3:1988	Part 3:1989 <i>File service definition</i> (Identical)
ISO 8571-4:1988	Part 4:1989 <i>File protocol specification</i> (Identical)
ISO 8650:1988	BS 7092:1989 <i>Open Systems Interconnection: protocol specification for the Association Control Service Element</i> (Identical)
ISO 8822:1988	BS 7093:1989 <i>Open Systems Interconnection: connection-oriented presentation service definition</i> (Identical)
ISO 8824:1987	BS 6962:1988 <i>Open Systems Interconnection: specification for Abstract Syntax Notation One (ASN.1)</i> (Identical)

¹⁾ In preparation.

International Standard	Corresponding British Standard
ISO 8825:1987	BS 6963:1988 <i>Open Systems Interconnection. Specification for basic encoding rules for Abstract Syntax Notation One (ASN.1)</i> (Identical)

The Technical Committee has reviewed the provisions of ISO 6429, ISO 8601, ISO 8832¹⁾, ISO 9804¹⁾ and ISO 9834-2¹⁾, to which reference is made in the text, and has decided that they are acceptable for use in conjunction with this standard. A related British Standard to ISO 8601 is BS 5249 “*Representation of elements of data in interchanges using data processing systems*” Part 1 “*Representation of dates and times*”.

A British Standard does not purport to include all the necessary provisions of a contract. Users of British Standards are responsible for their correct application.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Summary of pages

This document comprises a front cover, an inside front cover, pages i to iv, the ISO title page, page ii, pages 1 to 66, an inside back cover and a back cover. This standard has been updated (see copyright date) and may have had amendments incorporated. This will be indicated in the amendment table on the inside front cover.

INTERNATIONAL STANDARD

ISO
8571-2

First edition
1988-10-01

AMENDMENT 2
1993-08-15

Information processing systems – Open Systems Interconnection – File Transfer, Access and Management –

Part 2 : Virtual Filestore Definition

AMENDMENT 2 : Overlapped access

*Systèmes de traitement de l'information – Interconnexion de systèmes
ouverts – Transfert, accès et gestion de fichiers –*

Partie 2: Définition du système de fichiers virtuel

AMENDEMENT 2 : Chevauchement d'accès



Reference number
ISO 8571-2:1988/Amd. 2:1993 (E)

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Amendment 2 to International Standard ISO 8571-2:1988 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

ISO 8571 consists of the following parts, under the general title *Information processing systems — Open Systems Interconnection — File Transfer, Access and Management*:

- *Part 1: General introduction;*
- *Part 2: Virtual Filestore Definition;*
- *Part 3: File Service Definition;*
- *Part 4: File Protocol Specification;*
- *Part 5: Protocol Implementation Conformance Statement Proforma.*

0 Introduction

ISO 8571 is one of a set of International Standards produced to facilitate the interconnection of computer systems. It is related to other International Standards in the set as defined by the Reference Model for Open Systems Interconnection (ISO 7498). The Reference Model subdivides the area of standardization for interconnection into a series of layers of specification, each of manageable size.

The aim of Open Systems Interconnection is to allow, with a minimum of technical agreement outside the interconnection standards, the interconnection of computer systems

- a) from different manufacturers,
- b) under different managements,
- c) of different levels of complexity,
- d) of different ages.

ISO 8571 defines services for file transfer, access and management. It also specifies a protocol available within the application layer of the Reference Model. The service defined is of the category Application Service Element (ASE). It is concerned with identifiable bodies of information which can be treated as files, stored and managed within open systems, or passed between application processes.

ISO 8571 defines a basic file service. It provides sufficient facilities to support file transfer, file access, and management of files stored on open systems. ISO 8571 does not specify the interfaces to a file transfer, access or management facility within the local system.

ISO 8571 consists of the following four parts:

- *Part 1: General introduction;*
- *Part 2: Virtual Filestore definition;*
- *Part 3: File Service definition;*
- *Part 4: File Protocol specification.*

The definition in this part of ISO 8571 is used in the subsequent parts of ISO 8571 which specify services and protocols. They reference the filestore definition in order to assign meaning to the various descriptive data items which they manipulate. This definition will also be used by protocol implementors when choosing a mapping from the protocol items onto their real storage mechanism.

This part of ISO 8571 contains the following annexes which form part of the standard:

- Annex A — File access structure constraint sets;
- Annex B — Document types;

and the following annexes which do not form part of the standard:

- Annex C — Reading of structured files;
- Annex D — Insertion in a structured file;
- Annex E — ASN.1 cross reference.

1 Scope and field of application

This part of ISO 8571

- a) defines an abstract model of the virtual filestore for describing files and filestores (see section 1);
- b) defines the set of actions available to manipulate the elements of the model (see section 2);
- c) defines the properties of individual objects and associations in terms of attributes (see section 3).
- d) defines the form of representations of files with hierarchical structures (see clause 7 in section 1).

This part of ISO 8571 does not specify

- e) requirements relating to the mapping from real to virtual filestores;
- f) requirements for implementations of the real filestore.

The Virtual Filestore definition is provided for reference by the other parts of ISO 8571 defining the file service (ISO 8571-3) and specifying the file protocol (ISO 8571-4).

2 References

ISO 6429, *Information processing — ISO 7-bit and 8-bit coded character sets — Additional control functions for character imaging devices.*

ISO 7498, *Information Processing Systems — Open Systems Interconnection — Basic Reference Model.*

ISO 8571, *Information Processing Systems — Open systems interconnection — File transfer, access and management — Part 1: General introduction — Part 3: File Service definition — Part 4: File Protocol specification.*

ISO 8601, *Data elements and interchange formats — Information interchange — Representation of dates and times.*

ISO 8650, *Information Processing Systems — Open Systems Interconnection — Protocol specification for the Association Control Service Element.*

ISO 8822, *Information Processing Systems — Open Systems Interconnection — Connection-oriented presentation service definition.*

ISO 8824, *Information Processing Systems — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1)*.

ISO 8825, *Information Processing Systems — Open Systems Interconnection — Specification of Basic encoding rules for Abstract Syntax Notation One (ASN.1)*.

ISO 8832, *Information Processing Systems — Open Systems Interconnection — Specification of the basic class protocol for Job Transfer and Manipulation*²⁾.

ISO 9804, *Information Processing Systems — Open Systems Interconnection — Definition of Application Service Elements — Commitment, Concurrency and Recovery*²⁾.

ISO 9834-2, *Information Processing Systems — Procedures for specific OSI registration authorities — Part 2: Registration of Document Types*²⁾.

3 Definitions

Terms are defined in ISO 8571-1.

4 Abbreviations

Abbreviations are defined in ISO 8571-1.

²⁾ At present at the stage of draft; publication anticipated in due course.

Section 1. The filestore model

5 Basic concepts

A virtual filestore is handled by one or more application entities; an initiator may communicate with these by initializing an FTAM regime, in which the file service is provided. Each application entity that is capable of operating as a responder in an FTAM regime supports a single virtual filestore. Thus an application entity title serves to identify a virtual filestore.

NOTE Mechanisms for allocation of unambiguous application entity titles and for the corresponding directory services to provide addresses for communication with the entity are outside the scope of ISO 8571.

A filestore may contain an arbitrary number (greater than or equal to one) of objects (see Figure 1).

The properties of each object are defined by the values of a set of object attributes. These attributes are global; at any one time, a single attribute value is available to all initiators. Different object types may have distinct types of attributes, as well as types of attributes in common.

Each file is either empty or has some contents and a structure. Some of the file attributes identify structural aspects of the contents.

Each file-directory maintains a parenthood relationship with zero or more subordinate objects. Some of the file-directory attributes may identify access control information to subordinate objects.

Each reference maintains a link to exactly one other object. The referent is either a file or a file-directory. The identity of the referent is available as an attribute of the reference, in the form of a (possibly incomplete) primary pathname. This attribute can not be changed. Other reference attributes may identify the object type and access control information to the linked object. If the identity of the referent changes, the corresponding reference ceases to exist.

There is a set of activity attributes associated with each FTAM regime. The activity attributes are of two types.

The first are in one to one correspondence with the object attributes, and indicate the active value of those attributes as perceived by the initiator.

The second type of activity attributes are current values of state information concerning the exchanges taking place within an FTAM regime, and the facts established about the initiator. These are usually derived from corresponding parameters in protocol exchanges. An initiator has available the description of only the FTAM regimes in which it is participating.

An arbitrary number (greater than or equal to zero) of initiators may have initialized FTAM regimes at any one time. Exchanges between the initiator and the responder lead to the selection of at most one object in the responder's virtual filestore to be bound to a particular FTAM regime at any one time. Note that multiple file objects may be identified for later selection via the generalized selection service. However only one object may be selected at a time. Further, no guarantees are placed on the availability of any file object in this group if it is eventually selected.

5a The virtual filestore model

5a.1 Filestore Objects

A virtual filestore is comprised of one or more of three kinds of objects:

- a) files;
- b) file-directories;
- c) references.

5a.1.1 Files

File objects contain data, and provide structuring information to access the data within them (see clause 7).

5a.1.2 File-directories

File-directory objects maintain a set of relationships to zero or more other objects within the filestore, whether those objects are files, references, or other file-directories. This relationship is parenthood. A file-directory is said to be the parent of an object if it maintains the relationship of parenthood for that object. Similarly, an object is said to be the child of a specified directory if that directory is the object's parent. In this way, file-directories provide a means of grouping objects within the virtual filestore. These groups can then be used to provide a structural order (the filestore tree) to the data files within the filestore.

An object is "in" a file-directory, if either

- a) that file-directory is the parent of the object
- b) there is a reference whose parent is the file-directory, linking to the object.

An object is "under" a file-directory if either

- a) the object is in the file-directory
- b) the object is in another file-directory that is under the file-directory. (Note this is a recursive definition.)

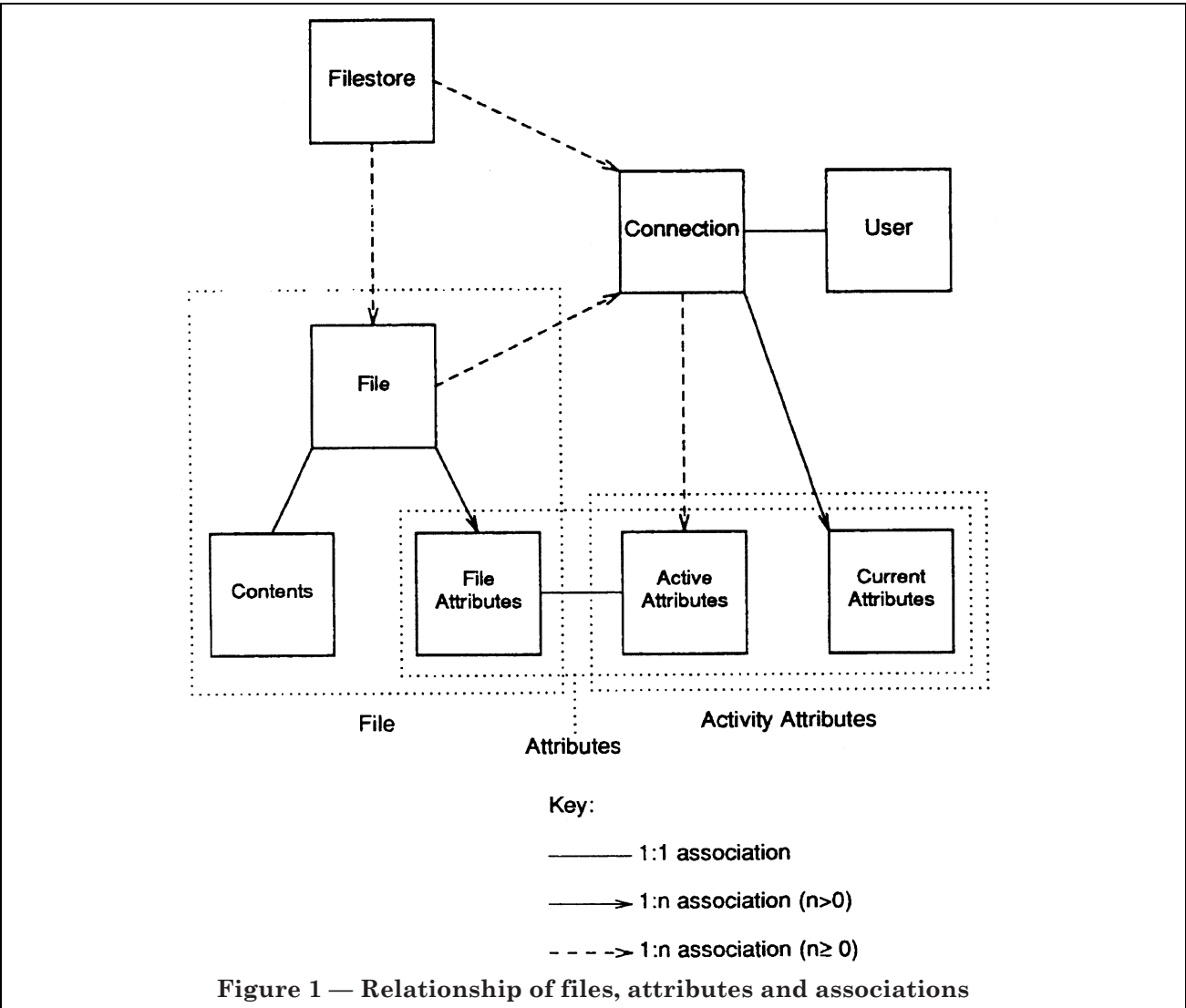


Figure 1 — Relationship of files, attributes and associations

5a.1.3 References

Reference objects maintain exactly one relationship to exactly one other object within the filestore. That relationship is linkage. The object linked by the reference must be either a file or a file-directory. The structure defined by the parent and linkage relations is called the filestore structure.

5a.2 Filestore structure

Every virtual filestore has a root object. The root is the only object in the filestore that has no parent. This root is either a file or a file-directory. It cannot be a reference. In the case where it is a file, that file will be the only object within that filestore.

The relationship of parenthood results in a hierarchical model of the filestore, where the root node is represented by the filestore root object, intermediate nodes are represented by file-directories maintaining at least one parenthood relationship, and leaf nodes are represented by files, references, and file-directories maintaining no parenthood relationships.

References may be used for convenience of access in special situations, or for special security needs. References provide a simple means of allowing an object to appear in more than one place in the filestore hierarchy without having to duplicate the object, or worry about maintaining consistency between duplicate objects. In normal use a user will not observe any difference in behavior whether an object is accessed via parenthood or reference.

5a.3 Name resolution

An object is identified within the virtual filestore by a pathname. A pathname is comprised of a series of object names. Each object name in the series identifies the next child object in the virtual filestore. The last object name in the series identifies the target object. The root object in a filestore is identified by a pathname comprised of zero object names. The exact algorithm is described in 5a.3.2.

5a.3.1 The current name prefix

When the pathname of an object begins its series of object names at the root of the filestore, it is called a complete pathname. Otherwise, to uniquely identify an object within the virtual filestore, the incomplete pathname must be resolved to a complete pathname. This is done with the current name prefix activity attribute. The current name prefix is assigned to the association by the responder. The current name prefix is a complete pathname of a file-directory object. The actual mechanisms for this assignment are outside the scope of FTAM, but possible uses could be for providing default file-directories to users, protecting filestore users from potential filestore organizational changes, or for enhanced security control.

An incomplete pathname is resolved to a complete pathname by prepending the series of object names within the current name prefix to the incomplete pathname.

Objects within a virtual filestore may be referenced by complete pathname, or by an incomplete pathname. In the latter case, the responder resolves the incomplete pathname to a complete pathname using the current name prefix. The file protocol is designed such that the responder need not reveal the current name prefix to the initiator, should it be desirable to conceal the filestore structure above this file-directory for security or other reasons.

5a.3.2 Resolving a pathname

A complete pathname is resolved to an object by a series of steps using the object names of the pathname to locate the intermediate objects along the path in turn.

Initially, the root node is located.

For each step, while object names of the pathname remain to be resolved:

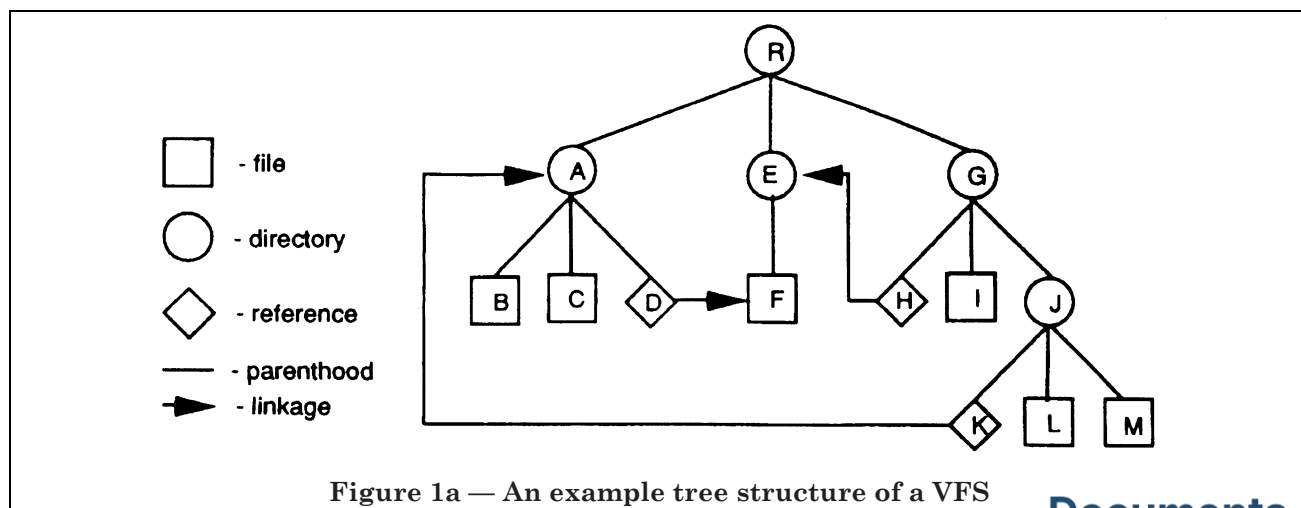
- if the object located is a reference, and the filestore user has passthrough access to this reference then the object which it references is located (if the user does not have passthrough access to this reference, or if the referenced object is not found, an error is reported);
- if the object located is a file-directory, and the filestore user has passthrough access to this file-directory, then the child object named by the next object name of the pathname is located (if the user does not have passthrough access to this directory, or the next object name does not correspond to any child of this directory, an error is reported);
- if the object located is a file, then an error is reported.

If the object located when all object names of the pathname have been exhausted is a reference, then the final action taken depends on the operation being performed:

- if the operation is specific to reference objects, then the operation is performed on the reference object located;
- if the operation is not specific to reference objects, then the object to which the reference refers is located, and the operation is performed on the referenced object.

5a.4 Object type checking

If the object located when a pathname is resolved is not of the type required for the operation to be performed then an error is reported.



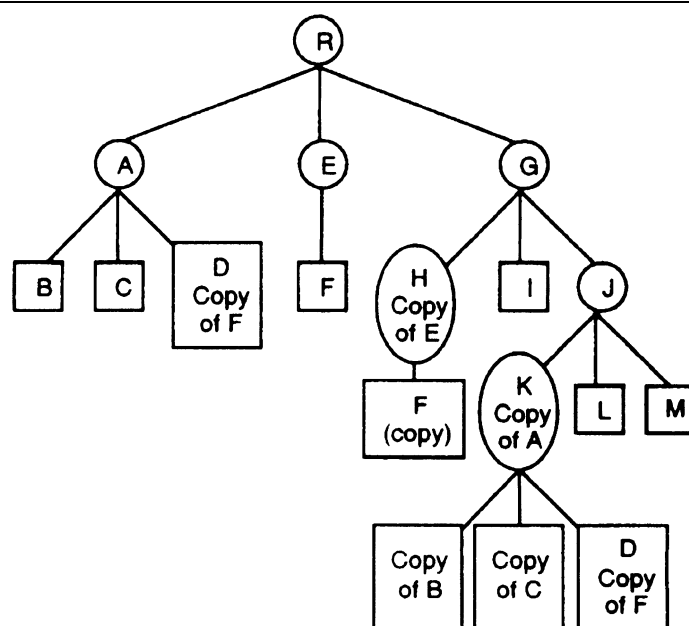


Figure 1b — An example of the apparent structure of a VFS

5a.5 Example

Figure 1a shows an example of a filestore containing references.

The file F has primary pathname E, F. However, it may also be accessed by the following names involving references:

- a) A, D
- b) G, H, F
- c) G, J, K, D

Thus for file selection, the filestore in this example appears as if duplications of data took place as in Figure 1b.

NOTE 1 In normal use, except when explicit manipulation of the reference object is carried out, a user will not observe any difference in behaviour whether an object is accessed via parenthood or reference.

NOTE 2 References may be used for convenience of access in special situations. References may have, in conjunction with the path access control attribute, applications to security and secure views of the filestore structure.

6 Object selection

From outside the filestore, selection of an object is always made using the pathname of the object. Even in the case of generalized selection services, the actual selection of a single object from within the group of file objects maintained in the generalized selection group activity attribute is made by implicit (i.e., internal to the responder controlling the filestore) reference to the pathname of the object. The reference to an object is within the context of a particular filestore identified by the application entity title. The application entity title refers to the location of file storage, and is known to the file service users, but lies outside the scope of FTAM. The pathname of an object is defined in clause 13.19.

6.1 Methods of object selection

Two methods of object selection are provided.

6.1.1 Simple object selection

Simple object selection takes place in two stages. First, an FTAM regime is initialized with the application entity handling the virtual filestore, and then information is given to this entity to identify the object unambiguously from among all the objects in the filestore. This information is the pathname of the object. The current pathname activity attribute is set to the pathname used to identify the selected object.

6.1.2 Generalized object selection

The second form of object selection works only with file objects and references to file objects. It is the generalized selection mechanism. First, an FTAM regime is initialized with the application entity handling the virtual filestore. Assertions regarding the file objects' attributes are provided by the initiator to the responder. This group of complete pathnames is created and maintained by the responder in the generalized selection group activity attribute. Inclusion in this group is based on the attribute assertions provided by the initiator. Access permission by the initiator to the files based on requested actions and access authorization provided by the initiator is implicitly an assertion in identifying the group of pathnames.

NOTE The generalized file selection mechanism does not imply formal selection of the objects identified by pathname within the generalized selection group. It merely collects the pathnames for later use in other operations.

Within the FTAM regime, multiple sequential select regimes can be established. These sequential select regimes are created either by selecting another pathname in the generalized selection group, or by the simple object selection mechanism, described above.

Selecting another object in the generalized selection group operates by requesting the responder to choose a previously unselected pathname from the pathname group, and attempting to select it. If it cannot be selected (for example, it has been renamed or deleted since it became a member of the group, or some access control attributes controlling access to the object have been changed to exclude the initiator), then that pathname is removed from the pathname group, a new previously unselected pathname is chosen, and the responder tries again. The current pathname activity attribute is the pathname chosen. No status codes are provided to the initiator to identify this condition. A pathname is considered previously unselected until it is chosen by the responder during a select another action. Selecting an object explicitly by pathname does not affect its status as previously selected or not within the pathname group.

The initiator deselects an object, which was selected in this manner, by deselecting or deleting the file or its reference. The initiator is notified that no more unselected pathnames exist in the group through a last member indicator. If additional pathnames exist within the group, but upon attempting to select the next one, none are found that can be selected by this initiator, (for example, a concurrency control lock is now in place), then an error is returned.

If, after either of these notifications, another request to select another object is received from the initiator, the responder then considers all remaining pathnames in the group as previously unselected, and begins again. If no pathnames remain in the list, a permanent error is returned. No access guarantees are made regarding the objects listed in the generalized selection group.

Pathnames may be removed from the group by the responder if the responder determines that the object cannot be selected by the initiator. The initiator will not be notified of any deletions from the group. It is possible that, because of references, the same file object will appear in the group multiple times, under different pathnames.

All generalized actions are considered to be specific to a file object. Any references to file objects that may be included in the generalized selection group are treated transparently, so that users are not necessarily aware that they are dealing with a reference.

6.2 Selection of references

After selecting or creating a reference object, actions appropriate to a reference object and actions appropriate to the type of object referenced are allowed (see 5a.3.2). Actions specific to a reference object operate on the selected reference object and its attributes. Actions not specific to reference objects operate on the referenced object and its attributes, with the exception of the object name attribute; in this case, the referenced object "inherits" the object name attribute of the reference object for this specific association for the duration of the select regime. The current pathname activity attribute is set from the pathname used to identify the reference object.

A change attribute action not specific to a reference object results in changes to the referenced object's attributes, with the exception of the object name attribute; in this case, the object name (and possibly the primary pathname) of the reference object is changed.

Actions specific to reference objects are:

- a) F-LINK
- b) F-UNLINK
- c) F-READ-LINK-ATTRIB
- d) F-CHANGE-LINK-ATTRIB

Actions involving attribute value assertion lists may operate directly on references, depending on the settings of the object type attribute value assertions, if any.

When invoking an attribute value assertion list, the reference object's attributes and the referenced object's attributes (with the inherited object name attribute) are considered independently. The reference object successfully matches the attribute value assertion list if either of the reference object's or referenced object's attributes match the assertions.

7 File structures

7.1 File access structure

This clause defines the properties of a hierarchical structure. The abstract structure of hierarchical files, including the simple cases of flat and unstructured files, is defined in this part using ASN.1 in the ASN.1 module ISO 8571-FADU (see 7.2). A hierarchical structure has the following properties:

- a) The file access structure is an ordered tree.
- b) Zero or one Data Units are assigned to a node.
- c) Each node within the structure gives access to its subtree. The access unit (that is, the subtree) is known as a File Access Data Unit (FADU) and it is comprised of the structural nodes of the subtree and the Data Units contained within the subtree. The root node of the tree gives access to the entire file.
- d) Optionally a node has a name assigned to it.
- e) The number of levels, the arc lengths and the number of arcs originating from each node are not restricted.

NOTE 1 Applicable constraint sets may limit the maximum level, and the arc lengths allowed.

NOTE 2 These terms are illustrated in the Figure 2, where, for tutorial purposes, each node has been uniquely identified by the letters A to F.

A preorder traversal sequence is a specific sequence of nodes in the tree; it is established by traversing the subtree corresponding to the whole file. A subtree is traversed by appending the root node of the subtree to the end of the traversal sequence established so far, and then, for each of the children of the subtree in their order of appearance, traversing that subtree.

NOTE The algorithm describes the abstract construction of a sequence, which is used for establishing FADU Identity, defining the location operations and determining the order of transmission. This part of ISO 8571 does not define how the algorithm is to be implemented.

The File Access Data Units (FADUs), being in one-to-one correspondence with the subtrees, can be identified in the same manner as subtrees (that is, by their root nodes). Likewise, data units are always associated with a node and may be identified by the identification of the node. The general hierarchical structure can represent a wide range of different practical file structures. However, real systems can only support a limited range of structures, and there are restrictions on the way files can be modified. To express this, the concept of a constraint set is introduced. A constraint set defines limitations on the range of structures allowed, and expresses the way in which the basic access actions can modify the structure. Constraint sets reflecting certain common file types are defined in this part of ISO 8571 (see Annex A), but other constraint sets may subsequently be defined and registered.

NOTE It is expected that a registration authority will be established to maintain a register of constraint sets.

7.2 Abstract structure definition

The access structure of the hierarchical file model is specified in Figure 4, using ASN.1. The data unit contents may be expressed in ASN.1, as in Figure 3, or in some other abstract syntax notation.

Files are accessed in terms of File Access Data Units (FADU), which are equivalent to the subtrees in the general hierarchical model. The smallest amount of data to be specified for access is one Data Unit (DU). For the purpose of transfer with checkpointing, the Data Units may be divided into smaller parts called data elements (DE). It is not possible to access the individual data elements of a Data Unit by means of the filestore actions defined in ISO 8571; the data unit is accessed as a whole.

7.3 Abstract syntax definition

For the purpose of providing access to the structure of files in the FTAM hierarchical file model this International Standard assigns the ASN.1 object identifier value

```
{ iso standard 8571 abstract-syntax (2)
  ftam-fadu (2) }
```

as an abstract syntax name for the set of presentation data values, each of which is a value of the ASN.1 type ISO 8571-FADU. Structuring-Data-Element.

The corresponding object descriptor value shall be "FTAM FADU"

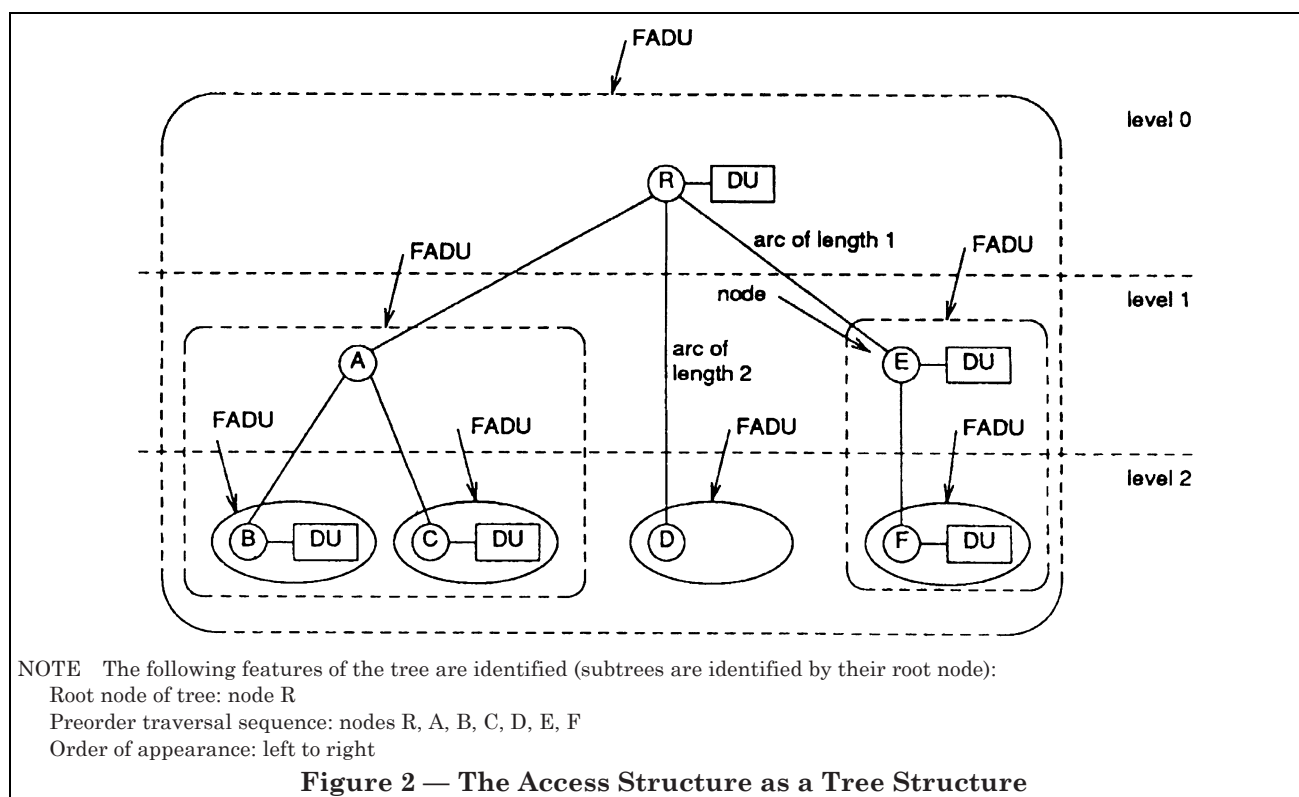
The ASN.1 object identifier and object descriptor values

```
{ joint-iso-ccitt asn1 (1) basic-encoding (1) }
```

and

"Basic Encoding of a single ASN.1 type"

(assigned to an information object in ISO 8825) can be used as a transfer syntax name with this abstract syntax.



7.4 File transfer structure

The ASN.1 module ISO 8571-FADU defines the access structure of the general hierarchical model. The syntax for transfer of files' contents is derived from this and consists of values of type Data-Element. Only values of type Data-Element are transferred from the FTAM sender to the FTAM receiver by means of the Presentation Service, ISO 8822. However these values shall appear in their syntactical order.

NOTE The syntax is constructed so that the syntactical order is equivalent to the preorder traversal sequence.

Structuring information (i.e. values of type Structuring-Data-Element) is communicated in FTAM FADU context (the one corresponding to the FTAM FADU abstract syntax, see 7.3). Node names are either in the same context or in a different, but embedded context. Values of type File-Contents-Data-Element are communicated in the file contents presentation context.

The abstract syntaxes used to transfer part or all of the contents of a file are indicated by the contents type file attribute. Each abstract syntax used shall be associated with a different presentation context. There are two possibilities:

- The contents type file attribute value specifies an abstract syntax and a constraint set; a presentation context is required corresponding to the specified abstract syntax. This presentation context is used to transfer both Node-Names and the actual contents of the file. A distinct presentation context, which corresponds to the file structuring abstract syntax (see 7.3), is required if the constraint set in use supports file structuring information.
- The contents type file attribute value specifies a document type; a presentation context is required corresponding to each abstract syntax specified in the document type register entry. The document type also defines which presentation context is to be used for transfer of Node-Names, if present. They can be
 - in a presentation context corresponding to one of the abstract syntaxes defined by the document type (user-coded);
 - in the same presentation context as the file structuring information (ftam-coded).

1	ISO8571-CONTENTS DEFINITIONS ::=
2	
3	BEGIN
4	
5	File-Contents-Data-Element ::= ANY
6	— Values of File-Contents-Data-Element are always transferred in a
7	— presentation context which is different from the presentation context
8	— used to transfer FTAM PCI. The actual presentation data values allowed
9	— are found in the abstract syntax for the file contents, as specified
10	— in the contents type file attribute for the file.
11	
12	END

Figure 3 — ASN.1 definition of file contents

7.5 Access context

Use of the abstract structure defined in the ISO 8571-FADU module to derive the corresponding sequence of information for transfer will yield the full hierarchical structure of the files, i.e. all the structuring information and all the data in the specified FADU will be transferred. However it is possible to access files for reading with a restricted view of their structure by use of different access contexts. In all cases, those data elements which are transferred are in the order defined in ISO 8571-FADU, and nodes are transferred in the order in which they appear in the preorder traversal sequence.

7.5.1 HA — Hierarchical All Data Units Access Context

In the HA access context, all four types of data elements (Node-Descriptor-Data-Element, Enter-Subtree-Data-Element, Exit-Subtree-Data-Element and File-Contents-Data-Element) in the addressed FADU are transferred.

7.5.2 HN — Hierarchical No Data Units Access Context

In the HN access context, all data elements of the types Node-Descriptor-Data-Element, Enter-Subtree-Data-Element and Exit-Subtree-Data-Element from the addressed FADU are transferred.

7.5.3 FA — Flat All Data Units Access Context

In the FA access context, data elements of the types Node-Descriptor-Data-Element and File-Contents-Data-Element from the addressed FADU are transferred. Only those Node-Descriptor-Data-Elements in which data-exists has the value TRUE are transferred.

7.5.4 FL — Flat One Level Data Units Access Context

In the FL access context, data elements of the types Node-Descriptor-Data-Element and File-Contents-Data-Element from those nodes in the addressed FADU which belongs to the specified level are transferred. Only those Node-Descriptor-Data-Elements in which data-exists has the value TRUE are transferred.

7.5.5 FS — Flat Single Data Unit Access Context

In the FS access context, the single Node-Descriptor-Data-Element and all the File-Contents-Data-Elements of the single DU belonging to the root node of the addressed FADU are transferred.

7.5.6 UA — Unstructured All Data Units Access Context

In the UA access context, only the data elements of type File-Contents-Data-Element from the addressed FADU are transferred.

7.5.7 US — Unstructured Single Data Unit Access Context

In the US access context, all the data elements of type File-Contents-Data-Element of the single DU belonging to the root node of the addressed FADU are transferred.

1	ISO8571-FADU DEFINITIONS ::=
2	
3	BEGIN
4	
5	Subtree ::= SEQUENCE {
6	node Node-Descriptor-Data-Element,
7	data [0] IMPLICIT DU OPTIONAL,
8	— present if and only if a DU is connected to the node.
9	children [1] IMPLICIT Children OPTIONAL }
10	— a leaf node is characterised by having no children
11	
12	Children ::= SEQUENCE {
13	enter-subtree Enter-Subtree-Data-Element,
14	SEQUENCE OF Subtree,
15	— subtrees must appear in their proper order according to
16	— their proper ordering as children of their parent node.
17	exit-subtree Exit-Subtree-Data-Element }
18	
19	DU ::= SEQUENCE OF ISO8571-CONTENTS.File-Contents-Data-Element
20	
21	Node-Descriptor-Data-Element ::= [APPLICATION 0] IMPLICIT SEQUENCE {
22	name Node-Name OPTIONAL,
23	— present only if the root node of the subtree is a named node.
24	arc-length [1] IMPLICIT INTEGER DEFAULT 1,
25	— used to specify the length of the arc to the root node of the subtree
26	— from its parent node.
27	data-exists [2] IMPLICIT BOOLEAN DEFAULT TRUE }
28	— data-exists = TRUE indicates that a DU is connected to the root node
29	— of the subtree.
30	
31	Node-Name ::= CHOICE {
32	ftam-coded [0] IMPLICIT GraphicString,
33	— when ftam-coded is used, the Node-Name belongs to the same abstract syntax
34	— as the structuring information. Node-Names are then transferred in
35	— the presentation context established to support the FTAM FADU abstract
36	— syntax. This form of Node-Name is only allowed when the contents type
37	— file attribute contains a document type name. To support this
38	— alternative, at least the GO character set registered in character
39	— set register entry 2 shall be supported.
40	user-coded EXTERNAL }
41	— the actual types allowed are found in the abstract syntax for the files
42	— contents, as specified in the contents type file attribute for the file.
43	
44	Enter-Subtree-Data-Element ::= [APPLICATION 1] IMPLICIT NULL
45	
46	Exit-Subtree-Data-Element ::= [APPLICATION 2] IMPLICIT NULL
47	— the enter-subtree and exit-subtree data elements are used to bracket
48	— the list of subtrees, which are children of the preceding node.
49	
50	FADU ::= Subtree
51	
52	Structuring-Data-Element ::= CHOICE {
53	Node-Descriptor-Data-Element,
54	Enter-Subtree-Data-Element,
55	Exit-Subtree-Data-Element }
56	
57	— Data-Element is defined to be a general data type whose values are
58	—
59	— a) a value of the ASN.1 type Structuring-Data-Element in the abstract
60	— syntax “FTAM FADU”; or
61	—
62	— b) a value of the ASN.1 type ISO8571-CONTENTS.File-Contents-Data-Element
63	— in the abstract syntax derived from the contents type file attribute.
64	END

Figure 4 — ASN.1 definition of file structure

Table 1 — Result of reading in an access context

Access Context	Result
HA	Single Subtree undefined type — because Node-Descriptor-Data-Elements with data exists = TRUE are transferred without the data elements constituting the DU.
HN	
FA	
FL	
FS	
UA	series of Subtree each with one node
US	Single DU

7.5.8 Summary of access contexts

Reading a non empty hierarchical subtree in these access contexts will have the results shown in Table 1, in terms of the data types defined in ISO 8571-FADU.

7.6 Identification structure

A FADU is identified by referencing the root node of the corresponding subtree. A FADU can be identified by any of the following mechanisms:

- a) “first”: the first FADU in the preorder traversal sequence for the file structure for which data exists is identified;
- b) “last”: the last FADU in the preorder traversal sequence for the file structure is identified;
- c) “previous”: the FADU preceding the currently identified FADU in the preorder traversal sequence of the file access structure is identified;
- d) “current”: the current location (see clause 8) in the file remains unchanged;
- e) “next”: the FADU following the currently identified FADU in the preorder traversal sequence of the file access structure is identified;
- f) “begin”: the exact meaning of begin depends on the constraint set in use, but is such that “locate next” will identify the “first” FADU.
- g) “end”: establishes a state of the file where there is no current location, but use of “previous” will identify the “last” FADU in the preorder traversal sequence for the file access structure;
- h) Node-Name: specifies the identifier of the FADU which is to be identified. The search for the specific Node-Name is restricted to the children of the currently located node;

i) Sequence of Node-Names: specifies a path of FADU identifiers from the root node of the file to the node to be located. The first Node-Name is a child of the root node of the file, so that the root node itself is identified by an empty sequence of Node-Names;

j) Node number: specifies the node to be selected by its number in the preorder traversal sequence for the file access structure. The root node of the file has node number zero;

If concurrent access is in use then the current locations for read and write actions are independent. The semantics of the FADU identifiers previous, current and next, are dependent on the action to be performed.

7.7 Constraint sets

The general hierarchical structure is constrained and its dynamics further qualified by the definition of a number of constraint sets, each identifying a particular structure in common use. These are specified in Annex A. Constraint sets are referenced in protocol exchanges by names, which are of ASN.1 type “object identifier”.

The set of actions that are ultimately allowable in the open regime is actually further constrained beyond the actions specified in the constraint set. This further restriction is imposed by the permitted actions file attribute, the processing mode activity attribute, the access control file attribute, the concurrency control activity attribute and the functional units selected for the current FTAM regime.

7a Actions on objects

The virtual filestore defines actions which manipulate the objects within the filestore. The definition of the individual actions (see section 2) states the objects to which actions apply, and the effects on those objects. Some actions also establish filestore state, such as the current name prefix, or generalized selection group.

The actions are invoked by service primitives. Their semantics are defined in conjunction with the filestore management primitives defined in ISO 8571-3.

Use of each action is subject to access control by the responder (see 12.16).

8 Actions on files

The virtual filestore defines actions which manipulate the data units in a file. The definition of the individual actions (see section 2) states the data units to which the actions apply, and the effects on those data units. Some actions also establish filestore state, such as the state “file selected” or FADU location.

The actions are invoked by service primitives. Their semantics are defined in conjunction with those of the file service primitives defined in ISO 8571-3 (see note 1).

Use of each action is subject to access control by the responder (see 12.16). The applicability of an action is subject to concurrency control governing the parallel activities during and after that action (see note 2 and 3).

NOTE 1 Some of the primitives of the service definition are thus closely related to particular actions, but the two are always logically distinct.

NOTE 2 When the use of commitment, concurrency and recovery has been agreed (see ISO 9804), concurrency controls should be applied as specified in that standard.

NOTE 3 The concurrency control permits an accessor to request either action not required, shared access, exclusive access or no access for various actions on a resource. If no access or exclusive access is requested and granted, actions by other associations do not occur. If shared access is requested and granted, exclusive access by other associations does not occur.

NOTE 4 The responder performs actions (as defined in section 2) so that they are serializable. Serializable execution is defined to be execution of the operations of concurrently executing activities in a way that produces the same effect as a serial execution of the same activities. A serial execution is one in which each action is completed before the next action begins. If concurrent access is in use then the effects of a set of read actions, and of a set of write actions, are independently serializable. However, the data transferred as a result of a read action may not be consistent with the current state of the file in some situations.

8.1 Relation to bulk data transfer

The hierarchical model of the contents of a file is defined in ASN.1 module ISO 8571-FADU. This model defines three data types that are called file structural items. These are

- a) Subtree: the main entry point data type of the module, representing a complete FADU;
- b) Children: a data type which consists of a sequence of an enter subtree, a sequence of one or more subtrees, and an exit subtree. Use of this data type at the outer level does not represent a single FADU but delimits a series of FADUs requiring special action (see 8.3);
- c) DU: a data type which consists only of a single Data Unit.

The receiver of transferred bulk data is said to recognise a file structural item when it receives the corresponding series of data types Data-Element. That is, the series of data types Data-Element in a value of that file structural item data type.

Certain constraint sets distinguish two variants of some actions, defining different semantics for them. These are called either

- a) a normal action, resulting from transfer of a complete subtree at the outer level;
- b) a qualified action resulting from the transfer of a Children file structural item. The qualified action is applied to each subtree in the sequence within the Children data value.

NOTE This mechanism is used to distinguish between the “insert as sister” and “insert as child” actions in the general hierarchical constraint set.

8.2 Read bulk data transfer

On performing an FADU read action (see 11.2), the responder shall generate one or more Subtree file structural items, or a single DU file structural item, depending on the access context used.

8.3 Write bulk data transfer

Before performing a write action, the responder shall perform a locate action using the FADU-Identity provided.

While performing an insert, replace or extend action (see 11.3 to 11.5), a responder shall parse the series of data values received, in order to recognise a file structural item. The file structural items acceptable will depend on the constraint set in use, but in general the responder will recognise

- a) the series of data values which can be recognised as a Subtree; this file structural item represents an FADU.
- b) the series of data values which can be recognised as Children; this represents a series of FADU with which qualified actions are to be performed. The meaning of the qualified action and its effect is specified in each constraint set for which the file structural item is valid.
- c) the series of data values which can be recognised as a DU, and which is not recognisable as part of any other file structural item which consists of a longer series of data values; this file structural item represents a Data Unit. In certain constraint sets, this file structural item is used to extend or replace the data unit of the currently located FADU.

NOTE 1 When the transfer is performed with FTAM, the data values are passed to the responder via the F-DATA indication primitive.

NOTE 2 The file structural item Children consists of an Enter Subtree, a series of Subtree, and Exit Subtree (see 7.2).

NOTE 3 The action qualifier mechanism is used, for example, to distinguish between the insert actions in a general hierarchical file; the normal action is insert as sister, and the qualified action is insert as child.

NOTE 4 A DU file structural item is not self delimiting. It is terminated by the end of the transferred data, or by a following Subtree or Children file structural item.

NOTE 5 The recognition of a file structural item will in general require a look-ahead of one data value.

For each file structural item received, the responder shall

- d) perform the action requested for the transfer as a whole, on the received Subtree or DU, or on each member of the qualified series in Children.
- e) after performing each action, update the current location as specified in the constraint set.

This process is terminated immediately if an error occurs, but if no failure has occurred when the actions specified for a recognised file structural item have been performed, the responder shall attempt to recognise a further file structural item and repeat the above process. This continues until the data transfer terminates or an error is detected. It is an error to receive a series of data values which cannot be recognised as a complete file structural item. However a single data transfer may contain many separate file structural items.

9 Attributes

9.1 Attribute scope

Two classes of attributes are defined:

- a) object attributes; each object is described by one set of object attribute values. The scope of the object attributes is the virtual filestore, and if an object attribute value is changed by the actions of one initiator, the new value is seen by any other initiators subsequently reading that attribute. Some attributes are specific to the type of object.
- b) activity attributes; each activity takes place within an FTAM regime and is described by one set of activity attribute values. The scope of the activity attributes is at most the FTAM regime, and a distinct and independent set of activity attribute values is bound to each FTAM regime. There are two distinct subdivisions of the activity attributes.

- 1) The active attributes are in one to one correspondence with the object attributes.

NOTE In most cases the mapping is trivial, since many file attributes are fixed at object creation time. However several of the active attributes such as active contents type and active legal qualifications have distinct values which are subsets of the object attribute values.

- 2) The current attributes concern the initiator and, are in general derived from the parameters on the protocol exchanges.

NOTE The current attributes are not exactly equivalent to static object attributes, but in some cases are closely related. For example the current access passwords must be members of the access passwords term in the access control attribute.

9.2 Scalar, vector and set attributes

Three types of attributes are defined:

- a) scalar attributes; each scalar attribute has one value at a given time;
- b) vector attributes; each vector attribute has a value which is a list of zero or more elements, each of which has an independent value. The elements are ordered;
- c) set attributes; each set attribute has a value which is itself an unordered set of zero or more elements, each of which has a distinct value.

The effect of the change attribute action on each of these different types of attribute is defined in 10.3.

9.3 Attribute values

For each attribute, the type of its value (or of the value of its elements for a vector or set attribute) is defined. The value type is one of the following:

- a) a sequence of characters (excluding space) from the character sets defined by GraphicString in ISO 8824 (ASN.1);
- b) a sequence of octets, as defined by OCTET STRING in ISO 8824 (ASN.1);
- c) an integer, as defined by INTEGER in ISO 8824 (ASN.1);
- d) a boolean, as defined by BOOLEAN in ISO 8824 (ASN.1);
- e) an application entity title, as defined by AE-title in ASN.1 module ACSE-1 in ISO 8650 (ACSE);
- f) a date and time; date and time values are limited to those expressible in GeneralizedTime; as defined in ISO 8601 (see note 1);
- g) an item from a named set of values defined by enumeration in this part of ISO 8571;
- h) an OBJECT IDENTIFIER, as defined in ISO 8824;
- i) an indication that the value cannot be determined (see note 2).

NOTE 1 The resolution and accuracy with which date and time values are maintained are determined by the responder, and are not defined in this part of ISO 8571. Optionally, the truncation of the representation from the right is used to indicate precision (see ISO 8601); truncation from the left is not used.

NOTE 2 This response would be made, for instance, if the real system environment had not been able to preserve the information.

9.4 Support of file attributes

Three levels of support for a file attribute within a filestore are defined:

- a) the attribute is not supported; an initiator not supporting an attribute will never reference it. An attempt to reference an attribute which a responder does not support results in an error.

b) the attribute is partially supported; an initiator shall not claim partial support for any attribute. Reference by an initiator to an attribute which is partially supported by a responder will yield the result “no value available”; attempts to change such an attribute will not succeed.

c) the attribute is supported; an initiator claiming support for an attribute shall be capable of requesting filestore actions specifying attribute values with at least the minimum range, as defined in clause 15.

NOTE This capability may be subject to conformance tests. Reference to an attribute supported by a responder will yield meaningful values with at least the minimum range, as defined in clause 15. Responders claiming support of an attribute shall support the associated semantics of the attribute as defined in clauses 12 and 13.

9.5 Extension attribute sets

The file protocol provides a mechanism for access to object attribute sets which are defined externally to this standard. This is done through extension attribute sets. An extension attribute set consists of an object identifier to identify the attribute set definition, and some number of attributes belonging to the identified attribute set. Each attribute is identified by its own Object Identifier, and maintains a value specific to that attribute.

If the initiator sends or requests the value of attribute sets not understood by the responder, the responder merely ignores those attribute sets it does not understand, completing the action as though they had not been present.

The responder must never send information about an attribute set not specifically requested by the initiator.

Requesting or recognizing an attribute extension set implies support for all attributes defined within the attribute extension set, and their mechanics.

9a Attribute value assertion lists

The file protocol provides a means for identifying a set of object pathnames based on the attributes of the objects they identify. This mechanism is by attribute value assertion list.

An attribute value assertion consists of an identification of an attribute, a target attribute value, and a relationship. An attribute value assertion is true for a specified object pathname if, for the object identified by the pathname,

- a) the identified attribute exists for this object type, and

- b) the identified attribute for that object has the specified relationship to the supplied target attribute value.

An attribute value assertion list consists of a set of attribute value assertion sublists, each sublist consisting of a set of attribute value assertions. An attribute value assertion list describes a subset of all pathnames of objects within the virtual filestore.

An object pathname is described by an attribute value assertion list if all of the following are true:

- a) the initiator has read-attribute access to the object via that pathname;
- b) the initiator has read access to each file-directory specified implicitly in the pathname pattern (see the note in 9a.1.2);
- c) the initiator has passthrough access to each file-directory specified implicitly in the pathname pattern;
- d) there exists at least one attribute value assertion sublist in the attribute value assertion list for which every attribute value assertion within it has the value “true” for the object identified by that pathname.

When performing actions on objects using attribute descriptions to identify the set of objects, the initiator provides an attribute value assertion list to describe the desired objects.

The responder then creates a list of pathnames based on the above criteria. The objects in this list of pathnames may then be operated upon singly, or as a group.

9a.1 Assertion types and components

9a.1.1 Relations for GraphicStrings

Assertions regarding GraphicStrings are made in terms of the logical relation “equality” in comparison to string patterns. A string pattern consists of a sequence of substring patterns. A substring pattern can be any of three types:

- a) a specific sequence of characters;
- b) a specification for an exact number of characters (those characters which are unimportant);
- c) a specification for zero or more characters.

A GraphicString is equal to a string pattern if

- 1) every character in the GraphicString can be sequentially matched with a character in a pattern of type “a”, a position in a substring pattern of type “b”, or a substring pattern of type “c” (pattern types correspond to the numbering of the list, above);