INTERNATIONAL STANDARD

**ISO 8571-2**

First edition
1988-10-01

**AMENDMENT 1**
1992-12-15

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

## Part 2 : Virtual Filestore Definition

## AMENDMENT 1 : Filestore Management

*Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Transfert, accès et gestion de fichiers –*

*Partie 2: Définition du système de fichiers virtuel*

*AMENDEMENT 1 : Gestion du système de fichiers*

Reference number
ISO 8571-2:1988/Amd.1:1992 (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 1 to International Standard ISO 8571-2:1988 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

ISO 8571-2 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*

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

## Part 2 : Virtual Filestore Definition

## AMENDMENT 1 : Filestore Management

NOTE – This amendment has additional subclauses and tables to ISO 8571 which are indicated by the use of lower case Roman letters beginning with "a" and imply ordering alphabetically, following the clause with the same numerical value in ISO 8571. These and all subsequent subclauses, tables, and cross references will be renumbered in subsequent editions.

## Introduction

*(amend 3rd paragraph, page 1)*

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.

*(amend 4th paragraph, page 1)*

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.

## 1 Scope and field of application

*(amend 1st paragraph)*

This part of ISO 8571

a) defines an abstract model of the virtual filestore for describing files and filestores (see section one);

b) defines the set of actions available to manipulate the elements of the model (see section two);

c) defines the properties of individual objects and associations in terms of attributes (see section three).

d) defines the form of representations of files with hierarchical structures (see clause 7 in section one).

# Section one: The filestore model

## 5 Basic concepts

*(amend 3rd paragraph (after note), page 2)*

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

*(amend 4th paragraph, page 2)*

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.

*(add following paragraph 5, page 2)*

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.

*(amend 7th paragraph, page 3)*

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.

*(amend 9th paragraph, pages 3 and 4)*

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.

*(add after clause 5, page 4)*

## 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 who's 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.)

#### 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:

a) 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);

b) 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);

c) 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:

d) if the operation is specific to reference objects, then the operation is performed on the reference object located;

e) 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.

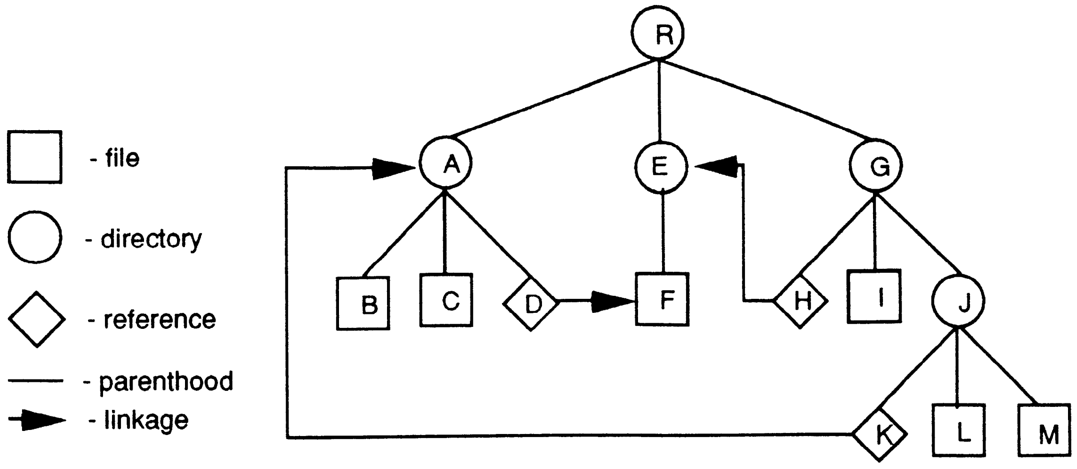

**Figure 1a – An example tree structure of a VFS**

### 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.

NOTES

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.

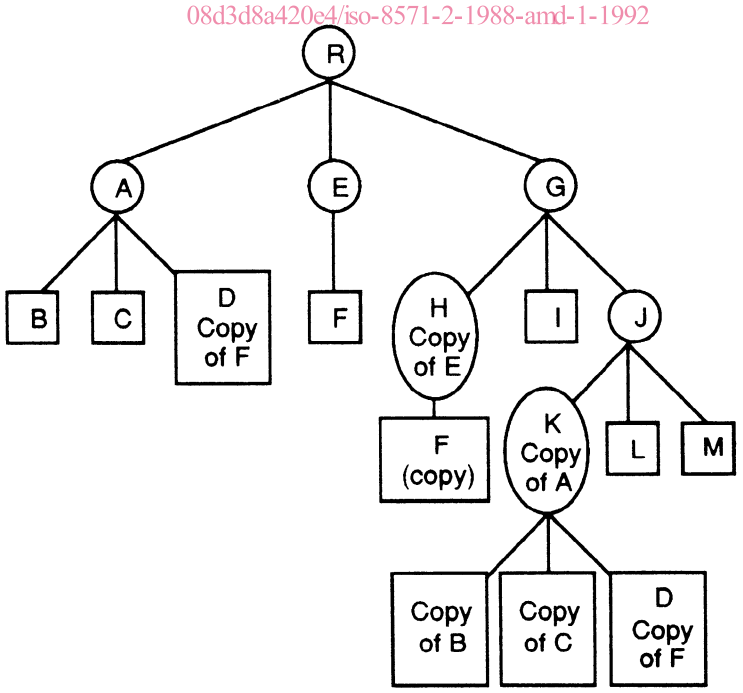

**Figure 1b – An example of the apparent structure of a VFS**

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.

*(amend title of clause 6, page 4)*

## 6 Object selection

*(amend 1st paragraph, page 4)*

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.

*(insert after 1st paragraph, page 4)*

### 6.1 Methods of object selection

Two methods of object selection are provided.

#### 6.1.1 Simple object selection

*(amend 2nd paragraph of clause 6, page 4)*

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.

*(replace 3rd paragraph of clause 6, page 4)*

#### 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 it's 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.

*(add after clause 7, page 7)*

## 7a Actions on objects

The virtual filestore defines actions which manipulate the objects within the filestore. The definition of the individual actions (see section two) 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).

### 9.1 Attribute scope

*(amend 1st paragraph, page 8)*

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.

*(add after clause 9.4, page 9)*

### 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 it's 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);

2) there are no characters in any string pattern of type 'a', or positions within string pattern type 'b' which do not have a corresponding character from the GraphicString.

Table 1a – Bitstring and bitstring pattern relationships

| Significance | Value of assertion | Meaning |
|---|---|---|
| Not significant | any | bit is not significant for matching |
| Significant | 0 | bit is significant and matches 0 |
| Significant | 1 | bit is significant and matches 1 |

### 9a.1.2 Relations for Pathnames

Assertions regarding Pathnames are made in terms of the logical relation "equality" in comparison to pathname patterns. Pathname patterns can be either complete pathname patterns, specifying pathname searches are to be made from the filestore root; or incomplete pathnames, specifying pathname searches are to be made from the file-directory identified by the current name prefix.

Either pathname pattern consists of a sequence of component patterns. A component pattern can take any of two forms:

a) a string pattern;

b) a specification for zero or more object names.

A Pathname is equal to a pathname pattern if

1) every object name in the Pathname can be sequentially matched with a string pattern in a component pattern of type 'a', or a component pattern of type 'b' (pattern types correspond to the numbering of the list, above);

2) there are no component patterns of type 'a' which do not have a corresponding object name from the Pathname.

NOTES

1) An object name is said to be "explicit" if the component pattern is of type "a", and that string pattern consists of a single substring pattern of type "a" (see 9a.1.1). Otherwise, the object name is said to be "implicit".

2) In each attribute value assertion sublist, there is always a pathname attribute assertion in effect, even if one is not supplied by the initiator. In that case, a pathname pattern resolving to all objects in the file-directory specified by the current name prefix is implied.

Access control passwords are only used with explicit pathnames.

### 9a.1.3 Relations for dates and times

Assertions regarding dates and times can be made in terms of

a) "less than" (i.e. before, or older than);

b) "greater than" (i.e. after, or younger than); or

c) "equality" (i.e. concurrent, or same age).

Assertions are evaluated according to both the precision given and the precision available on the local system.

### 9a.1.4 Relations for integers

Assertions regarding integers can be made in terms of the logical relations

a) "less than",

b) "greater than", or

c) "equality",

Where all are taken with their standard mathematical meanings.

### 9a.1.5 Relations for bitstrings

Assertions regarding bitstrings can be made in terms of "equality" with a pattern. A bitstring pattern consists of a significance mask and a value assertion. Matches against bitstring patterns are done on significant bits as shown in Table 1a.

A bitstring is equal to a pattern if each significant bit in the bitstring matches the corresponding assertion value in the pattern. Any bits appearing in the bitstring type attribute beyond the length of the significance mask are assumed to be not significant. Any bits set to significant appearing in the significance mask beyond the length of the bitstring type attribute are assumed to not match, making the attribute value not equal to the pattern.

### 9a.1.6 Relations for object identifiers

Table 1b – Attribute relationship combinations

| | less than | equality | greater than |
|---|---|---|---|
| less than | – | less than or equal | not equal |
| equality | less than or equal | – | greater than or equal |
| greater than | not equal | greater than or equal | – |

Assertions regarding object identifiers can be made in terms of "equality". An object identifier pattern consists of an object identifier. An object identifier attribute is equal to an object identifier pattern if they are identical object identifiers.

### 9a.1.7 Relations for externally defined attributes

Relations for externally defined attributes must be defined within the specification of the external attribute.

### 9a.1.8 Relations for boolean

Assertions regarding boolean attributes are made in terms of "equality", taken with its standard mathematical meanings.

### 9a.1.9 Relations for enumerated values

Assertions regarding enumerated attributes are made in terms of "equality", taken with its standard mathematical meanings.

### 9a.1.10 Relations for octetstring values

Assertions regarding octetstring attributes are made in terms of "equality". An octetstring is equal to an octetstring provided in an attribute value assertion if the two octetstrings are the same length, and each octet within one octetstring is equal in numerical value to the corresponding octet of the other octetstring.

## 9a.2 Attribute value assertion structure

An attribute value assertion consists of an identification of an attribute, a target attribute value, and a relationship.

### 9a.2.1 Attribute identification

The way an attribute value assertion identifies the attribute against which it is to be compared depends on the specific attribute. Attributes can be either internal to the files protocol, or else outside the files protocol, using attribute extensions.

An attribute value assertion identifies itself as pertaining to an attribute specified within the files protocol implicitly by position within a list.

An attribute value assertion identifies itself as pertaining to an attribute within an attribute extension set by identifying the attribute by its object identifier. Mapping to specific attributes within an extension set will be defined by the extension set definition, and are outside the scope of this part of ISO 8571.

### 9a.2.2 Attribute value

The value of an attribute value assertion is either a pattern describing one or more possible values of the attribute (see 9a.1), or the indication "no value available".

### 9a.2.3 Attribute relationship

The attribute value assertion relationships defined in 9a.1 are defined in some subset of the terms "equality", "greater than", and "less than".

Where only the "equality" relationship is provided, the files protocol provides means for the negation of the attribute value assertion, resulting in the ability to identify an object pathname based on it's "inequality" to a specified attribute pattern.

Where the "greater than" and "less than" relationships are also provided, combinations of the relationships may be expressed to form new relationships by taking the logical "or" result of the truth value of each of the relationships individually. Table 1b shows the allowed combinations of the relationships, and the resulting relationships.