INTERNATIONAL STANDARD

ISO 8571-3

First edition
1988-10-01

AMENDMENT 2
1993-08-15

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

Part 3 :
File Service Definition

AMENDMENT 2 : Overlapped access

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

Partie 3: Définition du service de fichiers

AMENDEMENT 2 : Chevauchement d'accès



Reference number
ISO 8571-3:1988/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 8571-3:1988 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*

© ISO/IEC 1993

All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

ISO 8571 PT*3 88 ■ 4851903 0545605 015 ■

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

Part 3 :

File Service Definition

AMENDMENT 2 : Overlapped access

0 Introduction

Clause 0 provides an introduction to this amendment. The text in this clause is not intended for inclusion in ISO 8571 part 3.

0.1 General

ISO 8571 part 3 defines in an abstract way the externally visible file transfer, access and management service.

This amendment extends this service definition to incorporate the services offered by overlapped access.

0.2 Rationale

The objective in introducing overlapped access is to allow more efficient access to structured files when a single initiator has a need to perform many reading and updating operations; the serial nature of the current FTAM data transfer services introduces a significant control overhead if the FADUs are small. In this context, an FADU is small if its transmission time is comparable with the time to complete a confirmed service on the association (the association's round trip delay).

0.3 Summary

The current design envelope that there should be at most one file selection per association and one file open per file selection is maintained. If access to more than one file is to be overlapped, more than one association is necessary, The overlapped access takes place within a constant set of presentation contexts established as at present when the file is opened, or previously.

Two different degrees of overlap have been identified. Firstly, requests for future accesses may be issued whilst a previously requested BDT action is in progress, allowing the creation of a queue of read and write requests. In general, PCI relating to a given BDT action may be overlapped with other BDT actions, subject to restrictions; this is called consecutive access. Secondly, read and write actions can be performed in parallel, so that both directions of data transfer are exploited at any one time. Requests are then taken from the queue whenever either direction of transfer becomes free; this is called concurrent access.

The transfer of a single FADU, specified in a single F-READ request has the same interpretation as in ISO 8571. The resultant effect on the virtual filestore of a set of overlapped requests using consecutive access shall be the same as that of the equivalent set of requests issued in series; the service provided is serializable. If concurrent access is used then the resultant effect of a set of write actions on the virtual filestore, is also serializable. However, due to the non-determinism introduced by the use of concurrent access, it is also possible that, in some uses of the service, the data transferred as a result of a read action is not consistent with the current state of the file.

1 Scope and field of application

This amendment makes no additions to clause 1.

2 References

This amendment makes no additions to clause 2.

3 Definitions

This amendment makes no additions to clause 3.

4 Abbreviations

This amendment makes no additions to clause 4.

5 Conventions

This amendment makes no additions to clause 5.

Section one: General

6 Model of the file service

6.2 File service levels

Replace second sentence of a):

Transfer of file data is modelled in the external file service as error free operations.

6.3 Regimes of the file service

Replace figure 2.

Replace d) and g):

d) the data transfer regime during which particular bulk data transfer specifications are in force.

g) a sequence of data transfer regimes within a file open regime; the data transfer regimes may each be for either read transfer, write transfer or both, if overlapped access is in use. Write data transfer permits the operations insert, replace, or extend.

Add note after last paragraph:

NOTE - If overlapped access is in use, then a data transfer regime is deemed to have terminated when there are no requests for bulk data transfer outstanding or in progress.

7 Services of the file service

7.8 Bulk data transfer

Replace sub-clause 7.8:

Bulk data transfer refers to the transfer, optionally with checkpointing, of one or more file access data units (see 20.1). There are nine additional services associated with bulk data transfer.

a) the consecutive access service may be requested by an initiator to allow the overlap of the protocol control information for read and write bulk data transfer procedures (see ISO 8571-1);

b) the concurrent access service may be requested by an initiator to allow the concurrent progression of read and write bulk data transfer procedures ((see ISO 8571-1)

c) the read bulk data service (see 24.1) is used by the initiator to initiate a bulk data transfer from the responder (in

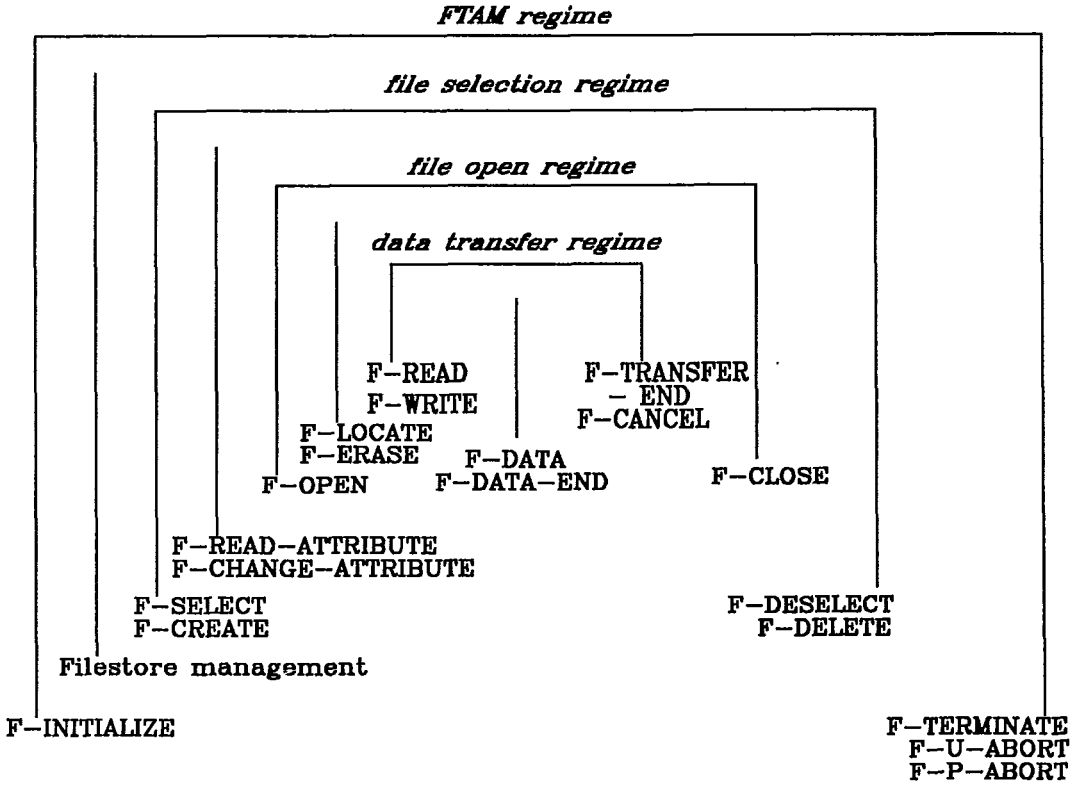


Figure 2 - File service regimes and related primitivees

Table 1 - Services and functional units of the External File Service

U15 Consecutive access	O				see note
U16 Concurrent access	O				see note

the role of sender), to the initiator (in the role of receiver);

d) the write bulk data service (see 24.2) is used by the initiator to initiate a bulk data transfer from the initiator (in the role of sender) to the responder (in the role of receiver);

e) the data unit transfer service (see 24.3) is used by the sender to transmit bulk data;

f) the end of data transfer service (see 24.4) is used by the sender to indicate completion of the data transfer;

g) the end of transfer service (see 24.5) is used by the initiator to confirm that the data transfer procedure is complete;

h) the cancel data transfer service (see 24.6) is used by either the sender or the receiver to cancel a data transfer procedure. When overlapped access is allowed either direction may be cancelled independently of the other.

7.10 Checkpointing and restarting

Add sentence to end of b):

If overlapped access is in use then either direction of transfer may be restarted.

8 Functional units and service classes

8.1 Functional units

Add the following clauses:

8.1.15 Consecutive access functional unit

The consecutive access functional unit allows the overlap of read and write data transfer procedures as defined in ISO 8571-1.

8.1.16 Concurrent access functional unit

The concurrent access functional unit allows the overlap of read and write data transfer procedures as defined in ISO 8571-1.

8.1.17 Service classes and functional units

Add entries to Table 1 and re-number entries accordingly:

Add note after Table 1:

Note - the consecutive access and concurrent access functional units allow the services provided by the read and write functional units to be overlapped. The description of the overlap allowed is contained within the definitions of individual services.

8.2 Service classes

8.2.2 File access class

Insert the following items and re-label accordingly:

d) optionally, the consecutive access functional unit;

e) optionally, the concurrent access functional unit;

9 Levels of file service

Replace Table 3 - Functional units in the file services

Insert note:

Note - Pending the specification of presentation symmetric synchronisation services, recovery mechanisms will not be available for use during overlapped access.

Table 3 - Functional units in the file service

External file service	Internal file service
Kernel	Kernel
Read	Read
Write	Write
File access	File access
Limited file management	Limited file management
Enhanced file management	Enhanced file management
Grouping	Grouping
FADU locking	FADU locking
	Recovery
	Restart data transfer
Consecutive access	Consecutive access
Concurrent access	Concurrent access

10 Negotiation of service class, FTAM QoS and functional units

Amend third paragraph:

The availability of the functional units, read, write, consecutive access, concurrent access, file access, limited file management, enhanced file management, grouping and FADU locking is controlled by the service class negotiated (see Tables 1 and 2).

Section two : Definition of file service primitives

11 File service primitives

Amend entries in Table 6.

12 Sequence of primitives

This amendment makes no additions to clause 12.

13 Common file service parameters

13.8 Concurrency Control

Replace fifth paragraph:

If FADU locking is negotiated the locks set at file open have the scope of the relevant bulk data transfer procedure. If overlapped access is not in use then this is equivalent to the bulk data transfer regime.

If overlapped access is in use then the scope of the FADU locking is that of the read and write action.

13.12 File Access Data Unit Identity

Add note.

Note: If concurrent access is in use then the FADU identifiers previous, current and next are dependent on whether the action is a read or a write.

Add the following clauses:

13.14 Bulk Transfer Number

The bulk transfer number parameter is only visible in the internal file service. The parameter is used to identify a particular bulk data transfer within an open regime. If the use of concurrent access is negotiated when the file open regime is established then the first read data transfer within an open regime is numbered 1, and subsequent read transfers are consecutively numbered. Similarly, the first write data transfer within the open regime is numbered 1, and subsequent write transfers are consecutively numbered. If overlapped access is not negotiated, or if consecutive access is negotiated, then read and write bulk transfers are numbered consecutively, starting from 1 for the first bulk data transfer procedure i.e. there is one

Table 6 - File service primitives

Primitive	Confirmed	Request by	Parameters
F-OPEN	Yes	Initiator	State result Action result Processing mode Contents type Concurrency control Shared ASE information Enable FADU locking Diagnostic Degree of overlap Transfer window [Activity identifier] [Recovery mode]
[F-RECOVER]	Yes	Initiator	[State result Action result Activity identifier Bulk transfer number Requested access Access passwords Contents type Recovery point Diagnostic Concurrent bulk transfer number Concurrent recovery point Last transfer end read request Last transfer end read response Last transfer end write request Last transfer end write response]

numbering sequence.

13.15 Transfer Number

The transfer number parameter is conditional on the use of overlapped access. The parameter is used by the initiator and responder to identify an uncompleted bulk data transfer. The value of the parameter is an integer. If overlapped access is not in use, or consecutive access is in use, then overlapping read and write transfers are consecutively numbered, starting from 1. The value is reset when all outstanding requests are completed.

Note - The transfer number of a bulk data transfer may exceed the value of the transfer window (17.1.2.10) as the latter parameter constrains the maximum number of transfers that are uncompleted at any one time.

13.16 Last transfer end read request

The last transfer end read request parameter is only visible in the internal file service, and is conditional on the use of overlapped access for the open regime. The value of the parameter is a transfer number (see 13.14). For the initiator it indicates the transfer number of the last read data transfer for which a F-TRANSFER-END request has been issued; for the responder it indicates the transfer number of the last read data transfer for which a F-TRANSFER-END indication has been received.

The default value 0 indicates that there are no outstanding F-TRANSFER-END requests or indications to be issued or received.

13.17 Last transfer end read response

The last transfer end read response parameter is only visible in the internal file service, and is conditional on the use of overlapped access for the open regime. The value of the parameter is a transfer number (see 13.13). For the initiator it indicates the transfer number of the last read data transfer for which a F-TRANSFER-END confirm has been received; for the responder it indicates the transfer number of the last read data transfer for which a F-TRANSFER-END response has been issued.

The default value 0 indicates that there are no outstanding F-TRANSFER-END responses or confirms to be issued or received.

13.18 Last transfer end write request

The last transfer end write request parameter is only visible in the internal file service, and is conditional on the use of overlapped access for the open regime. The value of the parameter is a transfer number (see 13.14). For the initiator it indicates the transfer number of the last write data transfer for which a F-TRANSFER-END request has been issued; for the responder it indicates the transfer number of the last write data transfer for which a F-TRANSFER-END indication has been received.

The default value 0 indicates that there are no outstanding F-TRANSFER-END requests or indications to be issued or received.

13.19 Last transfer end write response

The last transfer end write response parameter is only visible in the internal file service, and is conditional on the use of concurrent access for the open regime. The value of the parameter is a transfer number (see 13.14). For the initiator it indicates the transfer number of the last write data transfer for which a F-TRANSFER-END confirm has been received; for the responder it indicates the transfer number of the last write data transfer for which a F-TRANSFER-END response has been issued.

The default value zero indicates that there are no outstanding F-TRANSFER-END responses or confirms to be issued or received.

14 FTAM regime control

This amendment makes no additions to clause 14.

15 File selection regime control

This amendment makes no additions to clause 15.

16 File management

This amendment makes no additions to clause 16.

17 File open regime control

Amend second sentence:

This regime establishes the degree of overlapped access, processing mode, presentation contexts, and concurrency control for the data transfer activity which is to be performed.

17.1 File open service

17.1.1 Function

Amend last sentence of first paragraph:

It also establishes concurrency control, the degree of overlapped access and possible processing modes.

17.1.2 Types of primitives and parameters

Replace entries in table 21.

Add the following sub clauses and re-number:

17.1.2.9 Degree Of Overlap

The degree of overlap parameter indicates the degree of overlapped access required during the file open regime: either no overlapped access (normal), consecutive access or concurrent access.

The degree of overlap available during the open regime is the lowest of the values offered by the initiator and responder. For the purposes of negotiation, the degrees of overlapped access are rated as: concurrent access (highest), consecutive access, no overlapped access (lowest). This value is used to set the current degree of overlap attribute.

17.1.2.10 Transfer Window

The transfer window parameter indicates the number of bulk data transfers that may be uncompleted at any time. There is an independent transfer window for reads and writes in concurrent access, although the value of the limit is the same, otherwise the transfer window is for both reads and writes.

The value of the transfer window is the lowest of the values offered by the initiator or responder.

18 Grouping control

This amendment makes no additions to Clause 18 of ISO 8571 part 3.

19 Recovery (Internal service only)

19.1 Regime recovery

19.1.2 Types of primitives and parameters

Replace table 24.

19.1.2.4 Bulk Transfer Number

The bulk transfer number is defined in 13.14. The parameter indicates the bulk transfer procedure that is to be recovered.

If concurrent access is in use then this parameter takes the value of the read transfer that is to be recovered (the write transfer is identified by the concurrent bulk transfer number parameter). The default value 0 indicates that there is no read bulk transfer to be recovered.

Add the following clauses.

19.1.2.9 Recovery Point

Add last paragraph.

If concurrent overlapped access is in use then the parameter is with respect to the read data transfer that is identified by the bulk data transfer number. If the bulk transfer number is zero then the recovery point parameter shall take the value zero.

19.1.2.10 Concurrent Bulk Transfer Number

The concurrent bulk transfer number parameter is conditional on the selection of the concurrent access functional unit and the degree of overlap (see 17.1.2.9) being set to concurrent access. The parameter takes the value of a bulk transfer number (defined in 13.14) and identifies the write transfer that is to be recovered (see 19.1.2.4).

The default value 0 indicates that there is no write bulk transfer to be recovered.

19.1.2.11 Concurrent Recovery Point

The concurrent recovery point parameter is conditional on the concurrent access functional unit and the degree of overlap (see 17.1.2.9) being set to concurrent access. The parameter indicates that recovery is to a point before the start of the write bulk data transfer (value zero), or to a checkpoint within the bulk data transfer, or to a point following its completion (see note). The recovery point is within the write bulk data transfer identified by the concurrent bulk transfer number (see 19.1.2.10); if the write bulk transfer number parameter is zero then the write recovery point parameter shall take the value zero. The recovery point is determined by the entity which was receiving data at the time of failure. Consequently the parameter is required to be present on the response primitive.

Table 21 - F-OPEN parameters

Degree of overlap	Conditional	Conditional (=)	Conditional	Conditional (=)
Transfer window	Conditional	Conditional (=)	Conditional	Conditional (=)

If the concurrent bulk transfer number is zero, the concurrent recovery point parameter shall take the value zero.

NOTE - Recovery to a point following a write bulk data transfer completion is defined for the case where the receiver has received an F-DATA-END indication but the F-TRANSFER-END exchange has not completed. In such a situation it would be unnecessary to go back to the last confirmed checkpoint. The only requirement is for both sender and receiver to have an agreed understanding of the completion of the transfer. A value one larger than the highest checkpoint number issued by the sender is used to indicate recovery after bulk data transfer.

19.1.2.12 Last Transfer End Read Request

The last transfer end read request parameter is defined in 13.16. It is only present on the response primitive, and therefore indicates the bulk transfer number of the last read data transfer for which a transfer end read indication primitive had been received by the responder. It is only present if recovery is to an open regime that had negotiated the use of concurrent access.

19.1.2.13 Last Transfer End Read Response

The last transfer end read response parameter is defined in 13.17. It is only present on the request primitive, and therefore indicates the bulk transfer number of the last read data transfer for which a transfer end read confirm primitive had been received by the initiator. It is only present if recovery is to an open regime that had negotiated the use of concurrent access.

19.1.2.14 Last Transfer End Write Request

The last transfer end write request parameter is defined in 13.18. It is only present on the response primitive, and therefore indicates the bulk transfer number of the last write data transfer for which a transfer end read indication primitive had been received by the responder. It is only present if recovery is to an open regime that had negotiated the use of concurrent access.

19.1.2.15 Last Transfer End Write Response

The last transfer end write response parameter is defined in 13.19. It is only present on the request primitive, and therefore indicates the bulk transfer number of the last write data transfer for which a transfer end read confirm primitive had been received by the initiator. It is only present if recovery is to an open regime that had negotiated the use of concurrent access.

Replace Table 24.

20 Access to file contents

20.1 Bulk data transfer

Replace second paragraph:

When overlapped access is not in use, these procedures start and finish in a single data transfer idle state, and so can be considered as a self-contained procedural unit, which is primitive in the definition of the remainder of the file service.

If overlapped access is in use then these procedures may be overlapped. It is therefore not necessarily true that read and write procedures start and finish in a data transfer idle state.

21 Bulk data transfer service primitives

Replace Table 30.

Add sentence to end of fourth paragraph:

If overlapped access is in use then the initiator and responder may each be acting as both sender and receiver, with respect to the read and write procedures, as previously described.

Table 24 - F-RECOVER parameters

Parameter	F-RECOVER request	F-RECOVER indication	F-RECOVER response	F-RECOVER confirm
State Result			Mandatory	Mandatory
Action Result			Mandatory	Mandatory
Activity Identifier	Mandatory	Mandatory (=)		
Bulk Transfer number	Mandatory	Mandatory (=)		
Requested Access	Mandatory	Mandatory (=)		
Access Passwords	Optional	Optional (=)		
Contents Type			Mandatory	Mandatory (=)
Recovery Point	Conditional	Conditional (=)	Conditional	Conditional (=)
Diagnostic			Optional	Optional
Concurrent Bulk Transfer Number	Conditional	Conditional (=)	Conditional	Conditional (=)
Concurrent Recovery Point	Conditional	Conditional (=)	Conditional	Conditional (=)
Last Transfer End Read Request			Conditional	Conditional (=)
Last Transfer End Read Response	Conditional	Conditional (=)		
Last Transfer End Write Request			Conditional	Conditional (=)
Last Transfer End Write Response	Conditional	Conditional (=)		

Section three: Definition of bulk data transfer primitives

22 Sequence of bulk data transfer primitives

22.1 Normal sequences

The normal progress of individual read and write bulk data transfer procedures is illustrated by the state transition diagram shown in Figure 8. Full state transition diagrams are contained in Annex E.

The sequences of primitives allowed as a result of the use of the bulk data transfer service with overlapped access, is specified in Annex F.

Note - The use of a common Data Transfer Idle state for read and write bulk data transfer procedures in figure 8, serves to illustrate the progress of use of the bulk transfer service itself when overlapped access is not in use.

Re-label figure 8.

Figure 8 - Simplified State Transition Diagram for Read and Write Bulk Data Transfer Procedures

22.2 Constraints on issue of primitives

Table 30 - Bulk data transfer service primitives

Primitive Name	Confirmed Service	Request by	Parameters	Failure Notification
F-READ	No	Initiator	Bulk data transfer specification Transfer number	F-DATA-END action result
F-WRITE	No	Initiator	Bulk data transfer specification Transfer number	F-CANCEL
F-DATA	No	Sender	Data value	F-CANCEL
F-DATA-END	No	Sender	Action result Diagnostic	F-CANCEL
F-TRANSFER-END	Yes	Initiator	Action result Shared ASE information Diagnostic Request type Transfer number [Last transfer end read response] [Last transfer end write response]	action result
F-CANCEL	Yes	Either	Action result Shared ASE information Diagnostic Request type Transfer number [Last transfer end read request] [Last transfer end read response] [Last transfer end write request] [Last transfer end write response]	none
[F-CHECK]	Yes	Either	[Checkpoint identifier Transfer number]	F-CANCEL
[F-RESTART]	Yes	Either	[Checkpoint identifier Request type Transfer number Last transfer end read request Last transfer end read response Last transfer end write request Last transferend write response]	F-CANCEL

The primitives may be issued in any sequence consistent with the LOTOS specification in Annex F. In addition, for the non-overlapped case, the constraints are presented in tables 31 and 32. The sequences given with the individual primitive definitions apply.

22.3 Key to tables 31 and 32

This amendment makes no changes to sub-clause 22.3.

23 Common bulk data transfer parameters

23.2 Checkpoint identifier

Replace last sentence of first paragraph:

If the use of concurrent access has been negotiated for the open regime then there may be more than one checkpoint number sequence valid at any one time. The bulk transfer number is used to identify the correct sequence. For the F-RESTART primitive, the value is between 0 and 999998 for each direction of transfer.

23.43 Request type

The request type parameter identifies the type of transfer to which a primitive is related i.e. read or write.

24 Bulk data transfer

Replace the first paragraph:

This group of services performs the transfer of bulk data. Each procedure begins with the service initiator issuing either an F-READ request or an F-WRITE request as appropriate. This leads to the issue of a sequence of F-DATA requests followed by an F-DATA-END request by the sender of the data. A procedure is completed by the initiator issuing an F-TRANSFER-END request, as appropriate. If overlapped access has been negotiated, then the service primitives for read and write procedures may be interspersed, subject to the constraints defined for consecutive access and concurrent access. The primitives are defined in 24.1 to 24.6, and the valid sequences, with and without overlapped access, are defined in 24.7 and 24.8.

24.1 Read bulk data service

Replace sub-clause 24.1.1:

24.1.1 Function

The F-READ service specifies a data transfer from the service responder (that is the sender) to the service initiator (that is the receiver).

24.1.1.1 Service without overlapped access

Only one F-READ procedure may be in progress at any time on a single application association. The direction of data flow established continues until the exchange of F-TRANSFER-END primitives. Rejection of an F-READ indication is by issue of an F-DATA-END with an action result indicating unsuccessful.

These primitives signal a transfer of control from the initiator to the sender. They mark a reversal of the service asymmetry for the duration or the data transfer.

24.1.1.2 Service with consecutive access

F-READ procedures may be requested at any time during the open regime, up to the limit negotiated as the transfer window when the open regime was established. The requests for bulk transfer procedures are queued by the responder. If there are no outstanding bulk transfer requests then the F-DATA requests may be issued immediately, otherwise the F-DATA requests are issued subject to the following constraints:

a) if the preceding bulk transfer procedure was a read procedure then the responder may issue F-DATA requests after issuing the F-DATA-END request for the preceding read procedure;

b) if the preceding bulk transfer procedure was a write procedure then the responder may issue F-DATA requests after issuing the F-TRANSFER-END response for that write procedure.

A F-READ procedure concludes with the exchange of F-TRANSFER-END primitives.

Rejection of an F-READ indication is by issue of an F-DATA-END request with an action result indicating unsuccessful, although this may not be issued until the responder has issued a F-DATA-END indication or F-TRANSFER-END (write) response, as appropriate, for the preceding bulk transfer procedure.

24.1.1.3 Service with concurrent access

F-READ procedures may be requested at any time during the open regime, up to the limit negotiated as the transfer window when the open regime was established. The responder maintains two independent queues for the read and write bulk transfer requests. If there are no outstanding requests for read procedures then the responder may issue F-DATA requests immediately, otherwise the F-DATA requests are issued after the issue of an F-DATA-END

request for the preceding read procedure. A F-READ procedure concludes with the exchange of F-TRANSFER-END primitives.

Rejection of an F-READ indication is by issue of an F-DATA-END request with an action result indicating unsuccessful, although this may not be issued until the responder has issued a F-DATA-END indication for the preceding read procedure.

24.1.2 Types of primitives and parameters

Replace table 33.

Add sub-clause:

24.1.2.2 Transfer Number

The transfer number parameter is defined in 13.15.

24.2 Write bulk data service

Replace sub-clause 24.2.1:

24.2.1 Function

The F-WRITE service specifies a data transfer from the service initiator (that is the sender) to the service responder (that is the receiver).

24.2.1.1 Service without overlapped access

Only one F-WRITE procedure may be in progress at any time on a single application association. The direction of data flow established continues until the exchange of F-TRANSFER-END primitives. An F-WRITE indication can be rejected by issuing an F-CANCEL request (see 24.6). If the transfer is rejected, no further F-DATA indication primitives are received by the responder.

24.2.1.2 Service with consecutive overlapped access

F-WRITE procedures may be requested at any time during the open regime, up to the limit negotiated as the transfer window when the open regime was established. The requests for bulk transfer procedures are queued. If there are no outstanding bulk transfer requests then the F-DATA requests may be issued immediately, otherwise the F-DATA requests are issued subject to the following constraints:

Table 33 - F-READ parameters

Parameter	F-READ request	F-READ indication
Bulk data transfer specification	Mandator	Mandatory (=)
Transfer number	Conditional	Conditional (=)

a) if the preceding bulk transfer procedure was a read procedure then the initiator may issue F-DATA requests after issuing a F-TRANSFER-END request for the preceding read procedure;

b) if the preceding bulk transfer procedure was a write procedure then the initiator may issue F-DATA requests after issuing a F-TRANSFER-END request for the preceding write procedure.

An F-WRITE procedure concludes with the exchange of F-TRANSFER-END primitives.

A responder may reject an F-WRITE indication by issuing an F-CANCEL request, although this may not be issued until the responder has received an F-TRANSFER-END (read) indication or issued an F-TRANSFER-END (write) response, as appropriate, for the preceding bulk transfer procedure (see 24.7).

24.2.1.3 Service with concurrent access

F-WRITE procedures may be requested at any time during the open regime, up to the limit negotiated as the transfer window when the open regime was established. Two independent queues for the read and write bulk transfer requests. If there are no outstanding requests for write procedures then the initiator may issue F-DATA requests immediately, otherwise the F-DATA requests are issued after the issue of an F-TRANSFER-END request for the preceding write procedure. A F-WRITE procedure concludes with the exchange of F-TRANSFER-END primitives.

Rejection of an F-WRITE indication is by issue of an F-CANCEL request, although this may not be issued until the responder has issued a F-TRANSFER-END response, for the preceding write procedure (see 24.6).

24.2.2 Types of primitives and parameters

Replace table 34.

Add sub-clause:

24.2.2.2 Transfer Number

The transfer number parameter is defined in 13.15.

24.4 End of data transfer service

24.4.1 Function

Replace last sentence:

The sender may issue an F-DATA-END request with an unsuccessful action as a rejection of an F-READ indication, subject to the constraints defined in 24.1.

24.4.2 Types of primitives and parameters

Table 34 - F-WRITE parameters

Parameter	F-WRITE request	F-WRITE indication
Bulk Data Transfer Specification	Mandatory	Mandatory (=)
Transfer Number	Conditional	Conditional (=)

24.4.2.1 Action Result

Replace last sentence:

The initiator responds with an F-TRANSFER-END request as in the non error case.

24.5 End of transfer service

24.5.2 Types of primitives and parameters

Replace Table 37.

Add the following sub-clauses:

24.5.2.4 Request Type

The request type parameter is defined in 23.3.

24.5.2.5 Transfer Number

The transfer number parameter is defined in 13.15.

24.5.2.6 Last Transfer End Read Response

The last transfer end read response parameter is defined in 13.17.

24.5.2.7 Last Transfer End Write Response

The last transfer end write response parameter is defined in 13.18.

Replace sub-clause 24.6:

24.6 Cancel data transfer

24.6.1 Function

Either of the service users may cancel a data transfer activity by issuing an F-CANCEL request primitive. The conditions for use are described in the following clauses.

24.6.1.1 Service without overlapped access

The F-CANCEL primitive may be issued during data transfer after the issue or receipt of an F-READ or F-WRITE request or indication. At the end of data transfer, it may not be issued:

a) by the initiator acting as sender after issue of an F-DATA-END request;

b) by the responder acting as receiver after issue of an F-TRANSFER-END response;

c) by the responder acting as sender after the issue of an F-DATA-END request;

d) by the initiator acting as receiver after the issue of an F-TRANSFER-END request.

If either the cancel data transfer service or the end of data transfer service is used, the data transfer regime ceases.

Table 37 - F-TRANSFER-END parameters

Parameter	F-TRANSFER-END request	F-TRANSFER-END indication	F-TRANSFER-END response	F-TRANSFER-END confirm
Action result			Mandatory	Mandatory
Shared ASE Information	Optional	Optional (=)	Optional	Optional (=)
Diagnostic			Optional	Optional
Request Type	Conditional	Conditional (=)	Conditional	Conditional (=)
Transfer Number	Conditional	Conditional (=)	Conditional	Conditional (=)
[Last transfer end read response	Conditional	Conditional (=)		
Last transfer end write response	Conditional	Conditional (=)]		

Where use of these services collides, the cancel data transfer service takes precedence.

After an F-CANCEL procedure the two users may have different views of the state of the activity. The F-CANCEL primitives interrupt any activity in progress (including an F-RESTART sequence) and any undelivered indications or confirms may be discarded.

The file remains open after a sequence of F-CANCEL primitives, although the result of interrupted operations is not defined. Further F-READ or F-WRITE operations, not necessarily related to any previous read or write attempt, may be attempted after the completion of the sequence of F-CANCEL primitives has disposed of any previous activity.

24.6.1.2 Service with consecutive access

The cancel data transfer service may be used to cancel either a read transfer or a write transfer. If there are no uncompleted bulk transfer procedures then the constraints on the use of the service are as defined for the service without overlapped access. Otherwise, it may not be used:

a) by the initiator acting as sender before the issue of an F-TRANSFER-END request (as appropriate) for the preceding bulk transfer procedure, or after the issue of an F-DATA-END request for that write transfer;

b) by the responder acting as receiver before the receipt of an F-TRANSFER-END (read) indication or the issue of an F-TRANSFER-END (write) response (as appropriate) for the preceding bulk transfer procedure, or after the issue of an F-TRANSFER-END response for that write transfer;

c) by the responder acting as sender before the issue of an F-DATA-END request or F-TRANSFER-END (write) response (as appropriate) for the preceding bulk transfer, or after the issue of an F-DATA-END request for that read transfer;

d) by the initiator acting as receiver before the issue of an F-TRANSFER-END (read) request or receipt of an F-TRANSFER-END (write) confirm (as appropriate) for the preceding bulk transfer procedure, or after the issue of an F-TRANSFER-END request for that read transfer.

Outstanding F-TRANSFER-END primitives may not be exchanged until the F-CANCEL sequence has been completed. Also, the initiator may not issue further requests for bulk transfer procedures during the exchange of F-CANCEL primitives.

If the cancel data transfer service is used then all outstanding bulk transfer requests are cancelled and the data transfer regime ceases. Where uses of this service and the end of data transfer service collide, the cancel data transfer service takes precedence.

After an F-CANCEL procedure the two users may have different views of the state of the activity. The F-CANCEL primitives interrupt any activity in progress (including an F-RESTART sequence) and any undelivered indications or confirms may be discarded.

The file remains open after a sequence of F-CANCEL primitives, although the result of interrupted operations is not defined. Further F-READ or F-WRITE operations, not necessarily related to any previous read or write attempt, may be attempted after the completion of the sequence of F-CANCEL primitives has disposed of any previous activity.

24.6.1.3 Service with concurrent access

The cancel data transfer may be used to cancel a read transfer or a write transfer. The constraints that apply to cancelling read and write transfers are independent of each other and the F-CANCEL service may only be used to cancel one type of transfer. To cancel both the current read transfer and the current write transfer, the file service user must issue two F-CANCEL requests.

24.6.1.3.1 Cancel write bulk transfer constraints

If there are no uncompleted write bulk transfer procedures the constraints on the use of the service are as defined for the service without overlapped access. Otherwise, the service may not be used to cancel a write transfer:

a) by the initiator before the receipt of an F-TRANSFER-END confirm for the preceding write transfer, or after the issue of an F-DATA-END request for that write transfer;

b) by the responder before the issue of an F-TRANSFER-END response for the preceding write transfer, or after the issue of an F-TRANSFER-END response for that write transfer.

Outstanding F-TRANSFER-END (write) primitives may not be exchanged until the F-CANCEL sequence has been completed. Also, the initiator may not issue further requests for write bulk transfer procedures during the exchange of F-CANCEL primitives.

If the cancel data transfer service is used then all outstanding write bulk transfer requests are cancelled and the data transfer regime ceases. Where uses of this service and the end of data transfer service collide, the cancel data transfer service takes precedence.

After an F-CANCEL procedure the two users may have different views of the state of the activity. The F-CANCEL primitives interrupt any activity in progress (including an F-RESTART sequence) and any undelivered indications or confirms may be discarded.

The file remains open after a sequence of F-CANCEL primitives, although the result of interrupted operations is not defined. Further F-WRITE operations, not necessarily

related to any previous write attempt, may be attempted after the completion of the sequence of F-CANCEL primitives has disposed of any previous activity.

24.6.1.3.2 Cancel read bulk transfer constraints

If there are no uncompleted read bulk transfer procedures then the constraints on the service user are as defined for the service without overlapped access. Otherwise, the service may not be used to cancel a read transfer:

a) by the initiator before the issue of an F-TRANSFER-END request for the preceding read bulk transfer procedure, or after the issue of an F-TRANSFER-END request for that procedure;

b) by the responder before the receipt of an F-TRANSFER-END indication for the preceding read bulk transfer procedure, or after the issue of an F-DATA-END request for that read transfer.

Outstanding F-TRANSFER-END (read) primitives may not be exchanged until the F-CANCEL sequence has been completed. Also, the initiator may not issue further requests for read bulk transfer procedures during the exchange of F-CANCEL primitives.

If the cancel data transfer service is used then all outstanding read bulk transfer requests are cancelled and the data transfer regime ceases. Where uses of this service and the end of data transfer service collide, the cancel data transfer service takes precedence.

After an F-CANCEL procedure the two users may have different views of the state of the activity. The F-CANCEL primitives interrupt any activity in progress (including an F-RESTART sequence) and any undelivered indications or confirms may be discarded.

The file remains open after a sequence of F-CANCEL primitives, although the result of interrupted operations is not defined. Further F-READ operations, not necessarily related to any previous read attempt, may be attempted after the completion of the sequence of F-CANCEL primitives has disposed of any previous activity.

24.6.2 Types of primitives and parameters

Replace table 38.

Add the following sub-clauses:

24.6.2.4 Request Type

The request type parameter is defined in 23.3.

24.6.2.5 Transfer Number

The transfer number parameter is defined in 13.15.

24.6.2.7 Last Transfer End Read Request

The last transfer end read request parameter is defined in 13.16.

24.6.2.8 Last Transfer End Read Response

The last transfer end read response parameter is defined in 13.17.

24.6.2.9 Last Transfer End Write Request

The last transfer end write request parameter is defined in

Table 38 - F-CANCEL parameters

Parameter	F-CANCEL request	F-CANCEL indication	F-CANCEL response	F-CANCEL confirm
Action result	Mandatory	Mandatory	Mandatory	Mandatory
Shared ASE information	Optional	Optional (=)	Optional	Optional (=)
Diagnostic	Optional	Optional	Optional	Optional
Request type	Conditional	Conditional (=)	Conditional	Conditional (=)
Transfer number	Conditional	Conditional (=)	Conditional	Conditional (=)
[Last transfer end read request	Conditional (Responder only)	Conditional (=)	Conditional (Initiator only)	Conditional (=)
Last transfer end read response	Conditional (Initiator only)	Conditional (=)	Conditional (Responder only)	Conditional (=)
Last transfer end write request	Conditional (Responder only)	Conditional (=)	Conditional (Initiator only)	Conditional (=)
Last transfer end write response	Conditional (Initiator only)	Conditional (=)	Conditional (Responder only)	Conditional (=)]

13.18.

24.6.2.10 Last Transfer End Write Response

The last transfer end write response parameter is defined in 13.19.

24.7 Sequence of primitives on write

Add the following after first sentence:

If overlapped access is in use then the resultant sequence of primitives is the result of interleaving the two sequences for read and write operations. The rules for interleaving are defined for each individual primitive.

24.8 Sequence of primitives on Read

Add the following after first sentence:

If overlapped access is in use then the resultant sequence of primitives is the result of interleaving the two sequences for read and write operations. The rules for interleaving are defined for each individual primitive.

25 Checkpointing and restart (Internal BDT Service Only)

25.1 Checkpointing

Replace last sentence:

The checkpointing and restart primitives may be issued while a data transfer activity is in progress to control the progress of the data transfer.

25.1.1 Function

Add following paragraphs:

If overlapped access is not in use then checkpoint primitives may be issued after an F-READ or F-WRITE and before an F-DATA-END, for the sender and F-TRANSFER-END (read) or F-TRANSFER-END (write) for the recipient.

If the use of overlapped access has been negotiated for the open regime then checkpoints for read and write data transfers are independent of each other. A sender may only issue an F-CHECK request if all data has been sent for the preceding bulk data transfers; a responder may only issue an F-CHECK response if all data has been secured for the preceding bulk data transfer.

25.1.2 Types of primitives and parameters

Replace table 39.

Add the following sub-clause:

25.1.2.2 Transfer Number

The transfer number is defined in 13.15. The parameter identifies the data transfer to which the checkpoint is associated.

25.2 Restarting data transfer service

Replace sub-clause 25.2.1:

25.2.1 Function

The F-RESTART group of primitives interrupts any bulk data transfer activity in progress, with possible loss of any undelivered indications or confirms. It negotiates a point at which data transfer is to be restarted.

25.2.1.1 Service without overlapped access

At the end of data transfer, the service may not be used:

a) by the initiator acting as sender after issue of an F-DATA-END request;

b) by the responder acting as receiver after the issue of an F-TRANSFER-END response;

c) by the responder acting as sender after the issue of an F-DATA-END request;

d) by the initiator acting as receiver after the issue of an F-TRANSFER-END request.

If the receiver and the sender of data both issue an F-RESTART request, the service provider resolves the

Table 39 - F-CHECK parameters

Parameter	F-CHECK request	F-CHECK indication	F-CHECK response	F-CHECK confirm
Checkpoint identifier	Mandatory	Mandatory (=)	Mandatory	Mandatory (=)
Transfer number	Conditional	Conditional (=)	Conditional	Conditional (=)

collision and issues a confirmation to each user giving the point at which the transfer is to be restarted (see ISO 8571-4). An F-RESTART indication is rejected by using an F-CANCEL request primitive.

25.2.1.2 Service with consecutive access

The restarting data transfer service may be used to restart either a read transfer or a write transfer. If there are no uncompleted bulk transfer procedures then the constraints on the use of the service are as defined for the service without overlapped access. Otherwise, it may not be used:

a) by the initiator acting as sender before the issue of an F-TRANSFER-END request for the preceding bulk transfer procedure, or after the issue of an F-DATA-END request for that write transfer;

b) by the responder acting as receiver before the receipt of an F-TRANSFER-END (read) indication or the issue of an F-TRANSFER-END (write) response (as appropriate) for the preceding bulk transfer procedure, or after the issue of an F-TRANSFER-END response for that write transfer;

c) by the responder acting as sender before the issue of an F-DATA-END request (read) or F-TRANSFER-END (write) response, as appropriate, for the preceding bulk transfer, or after the issue of an F-DATA-END request for that read transfer;

d) by the initiator acting as receiver before the issue of an F-TRANSFER-END (read) request or receipt of an F-TRANSFER-END (write) confirm, as appropriate, for the preceding bulk transfer procedure, or after the issue of an F-TRANSFER-END request for that read transfer.

If the restarting data transfer service is used then all outstanding bulk transfer requests are cancelled and the initiator must reissue the bulk transfer requests. Where use of this service causes a collision, the service provider resolves the collision and issues a confirmation to each user giving the transfer, and the point at which the transfer is to be restarted (see ISO 8571-4). An F-RESTART indication is rejected by using an F-CANCEL request primitive.

25.2.1.3 Service with concurrent access

The restarting data transfer may be used to restart a read transfer or a write transfer. The constraints that apply to restarting read and write procedures are independent of each other. To issue an F-RESTART request for a read transfer and a write transfer, a service user must satisfy both sets of constraints.

25.2.1.3.1 Restart write bulk transfer constraints

If there are no uncompleted write bulk transfer procedures then the constraints on the use of the service are as defined for the service without overlapped access. Otherwise, the service may not be used to restart a write transfer:

a) by the initiator before the issue of an F-TRANSFER-END request for the preceding write transfer, or after the issue of an F-DATA-END request for that write transfer;

b) by the responder before the issue of an F-TRANSFER-END response for the preceding write transfer, or after the issue of an F-TRANSFER-END response for that write transfer.

25.2.1.3.2 Restart read bulk transfer constraints

If there are no uncompleted read bulk transfer procedures then the constraints on the service user are as defined for the service without overlapped access. Otherwise, the service may not be used to restart a read transfer:

a) by the initiator before the issue of an F-TRANSFER-END request for the preceding read bulk transfer procedure, or after the issue of an F-TRANSFER-END request for that read procedure;

b) by the responder before the issue of an F-DATA-END request for the preceding read bulk transfer procedure, or after the issue of an F-DATA-END request for that read transfer.

Subject to the following, a bulk transfer procedure that is not in a direction to be restarted, may continue as normal:

i) outstanding F-TRANSFER-END primitives may not be exchanged until the F-RESTART sequence has been completed;

ii) the initiator may not issue further requests for bulk transfer procedures during the exchange of F-RESTART primitives.

If the receiver and the sender of data both issue an F-RESTART request for the same direction, the service provider resolves the collision and issues a confirmation to each user giving the transfer, and the point at which the transfer is to be restarted (see ISO 8571-4).

25.2.2 Types of primitives and parameters

Replace table 40.

Add the following clauses:

25.2.2.2 Request Type

The request type parameter is defined in 23.4.

25.2.2.3 Transfer Number

The transfer number is defined in 13.15.

25.2.2.4 Last Transfer End Read Request

The last transfer end read request parameter is defined in 13.16.

25.2.2.5 Last Transfer End Read Response

The last transfer end read response parameter is defined in 13.17.

25.2.2.6 Last Transfer End Write Request

The last transfer end write request parameter is defined in 13.18.

25.2.2.7 Last Transfer End Write Response

The last transfer end write response parameter is defined in 13.19.

Table 40 - F-RESTART parameters

Parameter	F-RESTART request	F-RESTART indication	F-RESTART response	F-RESTART confirm
Checkpoint identifier	Mandatory	Mandatory (=)	Mandatory	Mandatory (=)
Request type	Conditional	Conditional (=)	Conditional	Conditional (=)
Transfer number	Mandatory	Mandatory (=)	Mandatory	Mandatory (=)
Last transfer end read request	Conditional (Responder only)	Conditional (=)	Conditional (Initiator only)	Conditional (=)
Last transfer end read response	Conditional (Initiator only)	Conditional (=)	Conditional (Responder only)	Conditional (=)
Last transfer end write request	Conditional (Responder only)	Conditional (=)	Conditional (Initiator only)	Conditional (=)
Last transfer end write response	Conditional (Initiator only)	Conditional (=)	Conditional (Responder only)	Conditional (=)

Annex A
Diagnostic parameter values

(This annex forms part of the standard.)

Amend table 48:

Table 48 - Access related diagnostics

Type	Identifier	Observer	Source	Reason
01	5nnn	5	5	Degree of overlap not available
0 2	5nnn	5	1	Transfer window - too large
0 2	5nnn	5	1	Transfer window - too small

Amend table 49:

Type	Identifier	Observer	Source	Reason
012	5nnn	5	5	Degree of overlap not available

The identifiers 5nnn are the next numbers in the sequence of identifiers for diagnostics.

Annex B
Relation of attributes to primitives

(This annex forms part of the standard.)

Amend table 51:

Table 51 - Activity attributes

Attribute Name	F-INITIALIZE	F-SELECT	F-CREATE	F-OPEN	F-READ & F-WRITE	F-LOCATE & F-ERASE	F-RECOVER
current degree of overlap				change			change

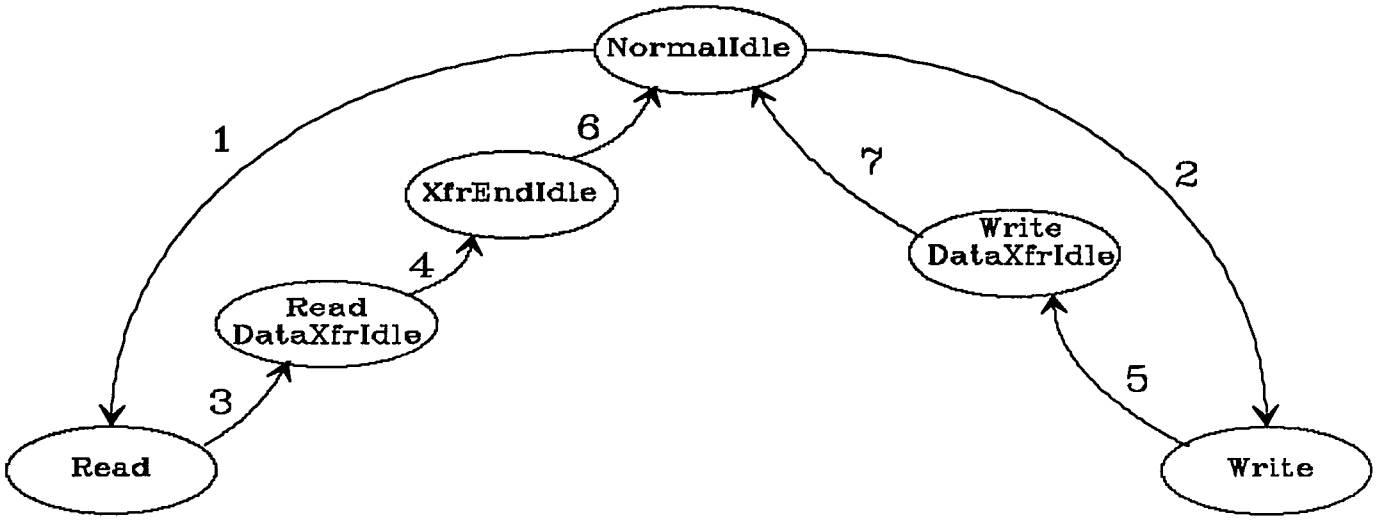
Annex E
State transition diagrams

(This annex forms part of the standard.)

Replace first sentence.

This annex contains descriptions of the sequences of service primitives in the external file service. State transition diagrams are used to define all sequences of primitives except those in overlapped access. The sequences of bulk data transfer primitives in overlapped access are defined by the LOTOS specifications in Annex F. This annex contains diagrams that illustrate the structure of the specifications and identify those events that (may) result in a change in the allowed behaviour i.e. the 'transition' between LOTOS processes (see note 1).

Note 1 - The notion of 'transition' between different processes is used in an informal manner for the purpose of this introduction. A transition relates to the situation in the LOTOS specifications where the use of a service primitive does not result in the same LOTOS process being instantiated.



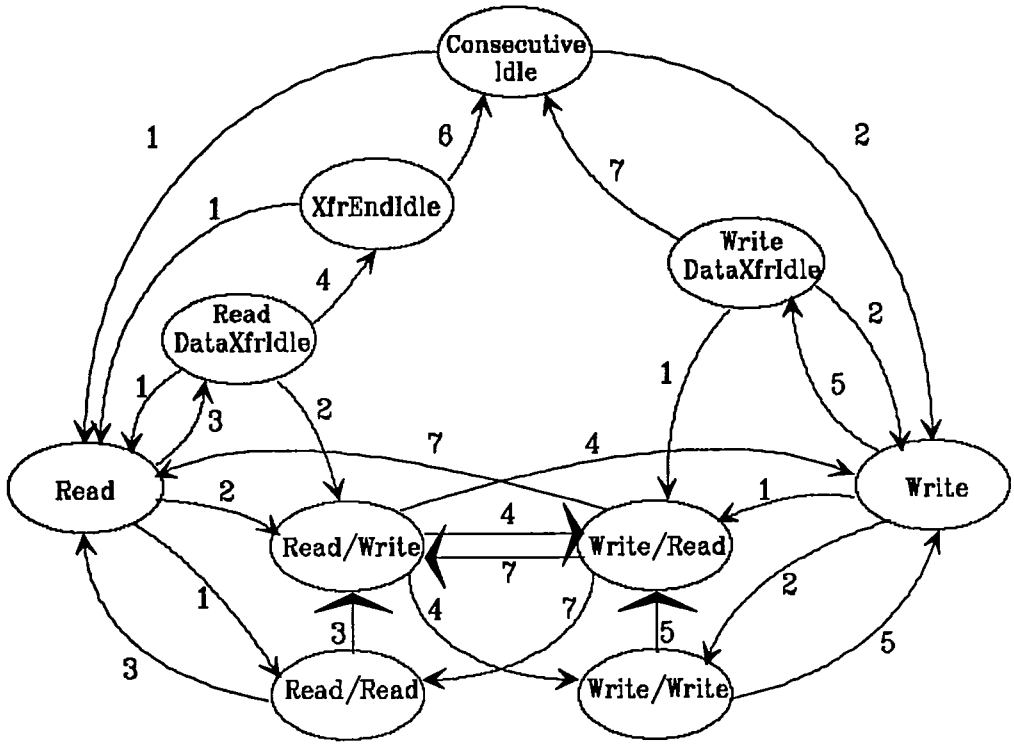
Initiator

1 - F-READ request
2 - F-WRITE request
3 - F-DATA-END indication
4 - F-TRANSFER-END request (Read)
5 - F-TRANSFER-END request (Write)
6 - F-TRANSFER-END confirm (Read)
7 - F-TRANSFER-END confirm (Write)

Responder

1 - F-READ indication
2 - F-WRITE indication
3 - F-DATA-END request
4 - F-TRANSFER-END indication (Read)
5 - F-TRANSFER-END indication (Write)
6 - F-TRANSFER-END response (Read)
7 - F-TRANSFER-END response (Write)

Figure 19 - A LOTOS Process transition diagram for normal (non-overlapped) access



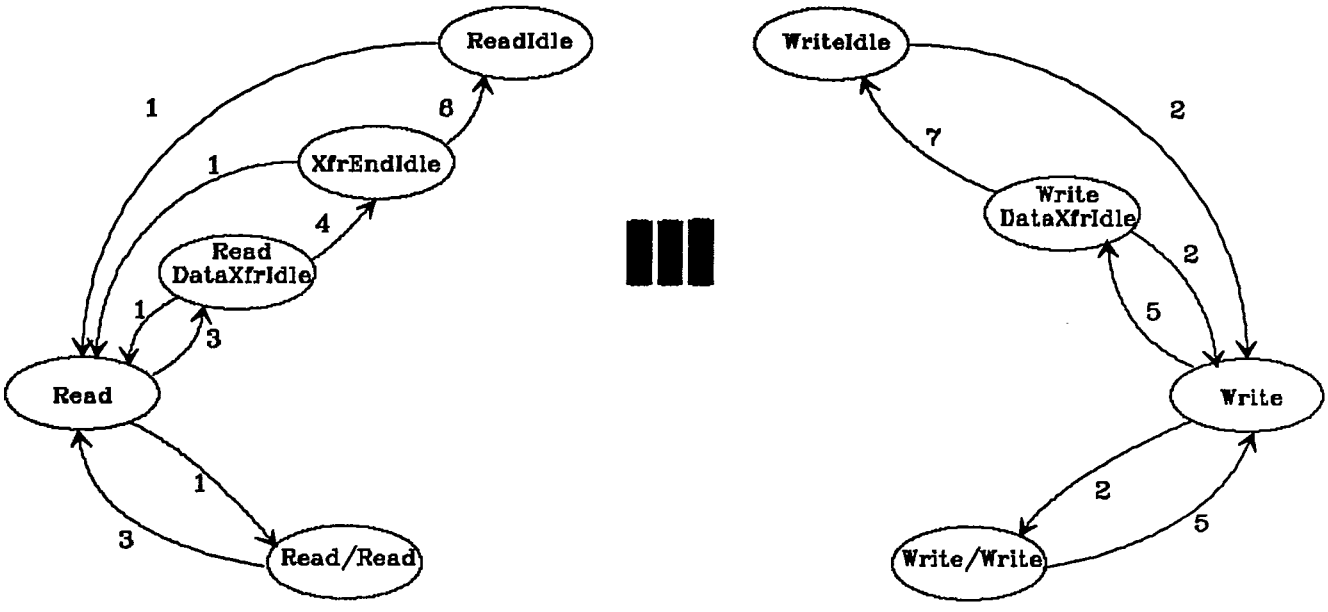
Initiator

1 - F-READ request
2 - F-WRITE request
3 - F-DATA-END indication
4 - F-TRANSFER-END request (Read)
5 - F-TRANSFER-END request (Write)
6 - F-TRANSFER-END confirm (Read)
7 - F-TRANSFER-END confirm (Write)

Responder

1 - F-READ indication
2 - F-WRITE indication
3 - F-DATA-END request
4 - F-TRANSFER-END indication (Read)
5 - F-TRANSFER-END indication (Write)
6 - F-TRANSFER-END response (Read)
7 - F-TRANSFER-END response (Write)

Figure 20 - A LOTOS Process transition diagram for consecutive access



Initiator

1 - F-READ request
2 - F-WRITE request
3 - F-DATA-END indication
4 - F-TRANSFER-END request (Read)
5 - F-TRANSFER-END request (Write)
6 - F-TRANSFER-END confirm (Read)
7 - F-TRANSFER-END confirm (Write)

Responder

1 - F-READ indication
2 - F-WRITE indication
3 - F-DATA-END request
4 - F-TRANSFER-END indication (Read)
5 - F-TRANSFER-END indication (Write)
6 - F-TRANSFER-END response (Read)
7 - F-TRANSFER-END response (Write)

Figure 21 - LOTOS Process transition diagram for concurrent access

Note - Figure 21 uses the LOTOS operator " |||" to denote the concurrent execution of read and write operations.

Annex F
Lotos specifications

(This annex forms part of the standard.)

The LOTOS specifications in this annex specify the sequences of bulk data transfer service primitives in the external file service. The specifications of the initiator and responder are provided separately.

Each specification (figures 22 and 23) specifies the allowed sequence of bulk data transfer service primitives for normal (non-overlapped), consecutive and concurrent overlapped access (see note 1). Figures 19, 20 and 21 illustrate the structure of the specifications and identify those events that (may) result in a change in the allowed behaviour i.e. the 'transition' between LOTOS processes (see note 2).

Note 1 - The behaviour defined in the non-overlapped case is equivalent to that for consecutive access, where the number of transfers that may be uncompleted at any time is 1. Also, the behaviour for concurrent access is equivalent to the interleaving of consecutively overlapped read procedures with consecutively overlapped write procedures.

Note 2 - The notion of 'transition' between different processes is used in an informal manner for the purpose of this introduction. A transition relates to the situation in the LOTOS specifications where the use of a service primitive does not result in the same LOTOS process being instantiated.

```
(*
Specification

The specification Initiator_External_Service specifies the sequence of service primitives in the external file service of the
initiator, for the duration of the bulk data transfer regime, for normal, consecutive and concurrent overlapped access.*)

Specification    Initiator_External_Service[Iin,Iout,Local]    :exit


(*
Data types

The specification imports the standard library definitions for the types Boolean and NaturalNumber, these are used to define
the following types:

ExtendedNaturalNumber - this extends the definition of NaturalNumber to include the operation pred, which decrements
a parameter of type Nat.

FileServicePrimitives - values are defined to represent the file service primitives. Boolean operations are defined that
identify a particular primitive e.g IsDataEndReq(transferEndReq)=false.

RequestQueue, QueueElement - a queue is defined with operations: empty (creates an empty queue), add (adds an
element to the queue), tail (removes the first element on the queue), head (returns the value of the first element on the queue),
last (returns the value of the last element on the queue), IsEmpty (determines whether the queue is empty), removeLast
(removes the last element on the queue), size (determines the number of elements on the queue), firstWriteReq (finds the
first write request on a queue), removeReqs (removes all elements from the end of a queue, up to and including the first write
request on the queue) and reverse (reverses the order of the elements on the queue). The elements of the queue are comprised
of a file service primitive and it's transfer number.

Note - the firstWriteInd and removeReqs operations are required to support the cancellation of a write data transfer after the
receipt of an F-TRANSFER-END indication by the responder. Equivalent operations are not required for read data transfer
operations.

DegreeOfOverlap - values are defined for normal, consecutive and concurrent; boolean operations are defined that
identify a given value.

FunctionalUnits - values are defined to represent the selection of the Read, Write and the combination of the Read &
Write functional units; boolean operations are defined that identify a given value.
*)


library

Boolean,NaturalNumber

endlib

type ExtendedNaturalNumber is NaturalNumber
     opns pred: Nat -> Nat
     eqns forall x:Nat
     ofsort Nat
          pred(0)          = 0;
          pred(Succ(x))    = x;
endtype
```

Figure 22 - Initiator service definition

```
type FileServicePrimitives is Boolean
     sorts FSPrim
     opns null,
          readReq,
          writeReq,
          dataReq,
          dataInd,
          dataEndReq,
          dataEndInd,
          xfrEndReadReq,
          xfrEndReadConf,
          xfrEndWriteReq,
          xfrEndWriteConf,
          cancelReadReq,
          cancelReadInd,
          cancelReadResp,
          cancelReadConf,
          cancelWriteReq,
          cancelWriteInd,
          cancelWriteResp,
          cancelWriteConf:          -> FSPrim

          Mp:          FSPrim -> Nat

          isNull,
          isReadReq,
          isWriteReq,
          isDataReq,
          isDataInd,
          isDataEndReq,
          isDataEndInd,
          isXfrEndReadReq,
          isXfrEndReadConf,
          isXfrEndWriteReq,
          isXfrEndWriteConf,
          isCancelReadReq,
          isCancelReadInd,
          isCancelReadResp,
          isCancelReadConf,
          isCancelWriteReq,
          isCancelWriteInd,
          isCancelWriteResp,
          isCancelWriteConf: FSPrim -> Bool


     eqns forall fsp:FSPrim

     ofsort Nat

          Mp(null)                  = 0;
          Mp(readReq)               = Succ(0);
          Mp(writeReq)        = Succ(Mp(readReq));
          Mp(dataReq)               = Succ(Mp(writeReq));
```

Figure 22 (continued) - Initiator service definition

```
            Mp(dataInd)              = Succ(Mp(dataReq));
            Mp(dataEndReq)           = Succ(Mp(dataInd));
            Mp(dataEndInd)           = Succ(Mp(dataEndReq));
            Mp(xfrEndReadReq)        = Succ(Mp(dataEndInd));
            Mp(xfrEndReadConf) = Succ(Mp(xfrEndReadReq));
            Mp(xfrEndWriteReq) = Succ(Mp(xfrEndReadConf));
            Mp(xfrEndWriteConf)      = Succ(Mp(xfrEndWriteReq));
            Mp(cancelReadReq)        = Succ(Mp(xfrEndWriteConf));
            Mp(cancelReadInd)        = Succ(Mp(xfrEndWriteConf));
            Mp(cancelReadResp) = Succ(Mp(cancelReadInd));
            Mp(cancelReadConf) = Succ(Mp(cancelReadResp));
            Mp(cancelWriteReq) = Succ(Mp(cancelReadConf));
            Mp(cancelWriteInd) = Succ(Mp(cancelWriteReq));
            Mp(cancelWriteResp)      = Succ(Mp(cancelWriteInd));
            Mp(cancelWriteConf)      = Succ(Mp(cancelWriteResp));

      ofsort  Bool
            isNull(fsp)        = Mp(fsp) eq Mp(null);
            isReadReq(fsp)           = Mp(fsp) eq Mp(readReq);
            isWriteReq(fsp)          = Mp(fsp) eq Mp(writeReq);
            isDataInd(fsp)           = Mp(fsp) eq Mp(dataInd);
            isDataReq(fsp)           = Mp(fsp) eq Mp(dataReq);
            isDataEndInd(fsp)        = Mp(fsp) eq Mp(dataEndInd);
            isDataEndReq(fsp)        = Mp(fsp) eq Mp(dataEndReq);
            isXfrEndReadReq(fsp)     = Mp(fsp) eq Mp(xfrEndReadReq);
            isXfrEndWriteReq(fsp)    = Mp(fsp) eq Mp(xfrEndWriteReq);
            isXfrEndReadConf(fsp)    = Mp(fsp) eq Mp(xfrEndReadConf);
            isXfrEndWriteConf(fsp)   = Mp(fsp) eq Mp(xfrEndWriteConf);
            isCancelReadInd(fsp)     = Mp(fsp) eq Mp(cancelReadInd);
            isCancelReadReq(fsp)     = Mp(fsp) eq Mp(cancelReadReq);
            isCancelReadConf(fsp)    = Mp(fsp) eq Mp(cancelReadConf);
            isCancelReadResp(fsp)    = Mp(fsp) eq Mp(cancelReadResp);
            isCancelWriteInd(fsp)    = Mp(fsp) eq Mp(cancelWriteInd);
            isCancelWriteReq(fsp)    = Mp(fsp) eq Mp(cancelWriteReq);
            isCancelWriteConf(fsp)   = Mp(fsp) eq Mp(cancelWriteConf);
            isCancelWriteResp(fsp)   = Mp(fsp) eq Mp(cancelWriteResp);
endtype


type QueueElement is FileServicePrimitives,Boolean,ExtendedNaturalNumber

      sorts  Element
      opns  nullEl:                    -> Element
            makeEl:      FSPrim,Nat    -> Element
            number:      Element       -> Nat
            prim: Element       -> FSPrim
            isNullEl:    Element       -> Bool

      eqns  forall  fsp:FSPrim,tn:Nat,q_el:Element
      ofsort  FSPrim
            prim(makeEl(fsp,tn))       = fsp;
            prim(nullEl)         = null;
      ofsort  Nat
            number(nullEl)             = 0;
            number(makeEl(fsp,tn))     = tn;
```

Figure 22 - (continued) Initiator service definition

```
      ofsort  Bool
            isNullEl(nullEl)          = true;
            isNullEl(q_el)            = isNull(prim(q_el));
      ofsort  Element
            makeEl(null,0)            = nullEl;
      endtype



type RequestQueue is QueueElement, Boolean, ExtendedNaturalNumber
      sorts  Queue
      opns  empty:                              -> Queue
            add:              Element,Queue     -> Queue
            tailOf:           Queue             -> Queue
            head:             Queue             -> Element
            last:             Queue             -> Element
            isEmpty:          Queue             -> Bool
            removeLast:       Queue             -> Queue
            size:             Queue             -> Nat
            firstWriteReq:    Queue             -> Element
            removeReqs:       Queue             -> Queue
            reverse:          Queue             -> Queue

      eqns  forall  x,y:Element,q:Queue
      ofsort  Element
            head(empty)               = nullEl;
            head(add(x,empty))        = x;
            head(add(x,add(y,q)))           = head(add(y,q));
            last(empty)               = nullEl;
            last(add(x,empty))        = x;
            last(add(x,q))                  = x;
            firstWriteReq(empty)            = nullEl;
            isWriteReq(prim(head(q)))
                  => firstWriteReq(q)       = head(q);
            not(isWriteReq(prim(head(q))))
                  => firstWriteReq(q)       = firstWriteReq(tailOf(q))

      ofsort  Queue
            tailOf(empty) = empty;
            tailOf(add(x,empty)) = empty;
            tailOf(add(x,q)) = add(x,tailOf(q));
            add(nullEl,q) = q;
            removeLast(empty) = empty;
            removeLast(add(x,q)) = q;
            removeReqs(empty) = empty;
            isWriteReq(prim(head(q)))
                  => removeReqs(q) = empty;
            not(isWriteReq(prim(head(q))))
                  => removeReqs(q) =reverse(add(head(q),removeReqs(tailOf(q))));

      ofsort  Bool
            isEmpty(empty) = true;
            isEmpty(add(x,q)) = false;

      ofsort  Nat
```

Figure 22 (continued) - Initiator service definition

```
            size(empty) = 0;
            size(add(x,q)) = Succ(size(q));

endtype

type DegreeOfOverlap is Boolean
      sorts Overlap
      opns normal: -> Overlap
            consecutive: -> Overlap
            concurrent: -> Overlap
            isNormal: Overlap -> Bool
            isConsecutive: Overlap -> Bool
            isConcurrent: Overlap -> Bool

      eqns forall degree:Overlap
      ofsort Bool
            isNormal(normal) = true;
            isNormal(consecutive) = false;
            isNormal(concurrent) = false;

            isConsecutive(normal) = false;
            isConsecutive(consecutive) = true;
            isConsecutive(concurrent) = false;

            isConcurrent(normal) = false;
            isConcurrent(consecutive) = false;
            isConcurrent(concurrent) = true;
endtype


type FunctionalUnits is Boolean
      sorts FUnits
      opns setRead: -> FUnits
            setWrite: -> FUnits
            setReadWrite: -> FUnits
            isAvailableRead: FUnits -> Bool
            isAvailableWrite: FUnits -> Bool

      eqns forall x:FUnits
            ofsort Bool
            isAvailableRead(setRead) = true;
            isAvailableRead(setWrite) = false;
            isAvailableRead(setReadWrite) = true;

            isAvailableWrite(setRead) = false;
            isAvailableWrite(setWrite) = true;
            isAvailableWrite(setReadWrite) = true;
endtype


(*
```

Behaviour

All events occur at the gates **Iin, Iout** or **Local**. Events at the gate **Iin** represent receipt of file service primitives by the file service provider; events at the gate **Iout** represent file service primitives that are issued to the file service user by the file service provider. The events at the gate **Local** establish the bounds on the behaviour of the file service i.e. the degree of overlap.

Figure 22 - (continued) Initiator service definition

```
the functional units available and, if overlapped access is allowed, the maximum number of request that may be outstanding.

The sequence of primitives allowed in the external file service is specified by the behaviour the process NormalIdle,
ConsecutiveIdle or ConcurrentIdle (chosen on the basis of the degreeOfOverlap, established by an event at the gate
Local).
*)

behaviour

 Local?degree:Overlap;
 ([isNormal(degree)]->
  Local?functionalUnits:FUnits;
    NormalIdle[Iin,Iout](functionalUnits)

[]

  [isConsecutive(degree)]->
  Local?functionalUnits:FUnits?maxRequests:Nat;
    ConsecutiveIdle[Iin,Iout](functionalUnits,maxRequests)

[]

  [isConcurrent(degree)]->
  Local?functionalUnits:FUnits?maxRequests:Nat;
    ConcurrentIdle[Iin,Iout](functionalUnits,maxRequests)
 )

where


(*
process NormalIdle

This process specifies the allowed behaviour when overlapped access is not in use. This point represents the completion of
the exchange of F-OPEN primitives.
*)

process  NormalIdle[Iin,Iout](functionalUnits:FUnits)  :exit:=

(* If the appropriate functional unit is available, the file service user may issue an F-READ request or an F-WRITE request. The request becomes the
current data transfer. *)

 ( [isAvailableRead(functionalUnits)]->
   Iin?fsp:FSPrim[isReadReq(fsp)];
    Read[Iin,Iout]
     (normal,functionalUnits,Succ(0),Succ(0),makeEl(fsp,Succ(0)),Succ(0),empty,empty,empty)

   []

   [isAvailableWrite(functionalUnits)]->
   Iin?fsp:FSPrim[isWriteReq(fsp)];
    Write[Iin,Iout]
     (normal,functionalUnits,Succ(0),Succ(0),makeEl(fsp,Succ(0)),Succ(0),empty,empty,empty)
 )

>>
```

Figure 22 (continued) - Initiator service definition

```
(* Upon completion of the previous bulk data transfer the file service user may issue an F-READ or F-WRITE request (as specified by the process
NormalIdle) or may exit the specification (i.e. initiate an exchange of F-CLOSE primitives). This point in the specification represents the data transfer
idle state. *)

        ( NormalIdle[Iin,Iout](functionalUnits)

      []

         exit
       )

endproc  (* NormalIdle*)


(*
process ConsecutiveIdle

This process specifies the allowed behaviour when consecutive overlapped access is in use. This point represents the
completion of the exchange of F-OPEN primitives.
*)

process   ConsecutiveIdle[Iin,Iout](functionalUnits:FUnits,maxRequests:Nat)   :exit:=

(* If the appropriate functional unit is available, the file service user may issue an F-READ request or an F-WRITE request. The request becomes the
current data transfer. *)

 ( [isAvailableRead(functionalUnits)]->
   Iin?fsp:FSPrim?tn:Nat[isReadReq(fsp) and (tn eq Succ(0))];
    Read[Iin,Iout]
     (consecutive,functionalUnits,maxRequests,tn,makeEl(fsp,tn),tn,empty,empty,empty)

  []

   [isAvailableWrite(functionalUnits)]->
   Iin?fsp:FSPrim?tn:Nat[isWriteReq(fsp) and (tn eq Succ(0))];
    Write[Iin,Iout]
     (consecutive,functionalUnits,maxRequests,tn,makeEl(fsp,tn),tn,empty,empty,empty)
 )

>>

(* Upon completion of all outstanding data transfers the file service user may exit the specification (i.e. initiate an exchange of F-CLOSE primitives)
or may issue an F-READ or F-WRITE request (as specified by the process ConsecutiveIdle) . This point in the specification represents the data
transfer idle state for consecutive access. *)

        ( ConsecutiveIdle[Iin,Iout](functionalUnits,maxRequests)

      []

         exit
       )

endproc  (* ConsecutiveIdle*)
```

Figure 22 - (continued) Initiator service definition

```
(*
process ConcurrentIdle

This process specifies the allowed behaviour when concurrent overlapped access is in use. This point represents the completion of the exchange of F-OPEN primitives.
*)

process  ConcurrentIdle[Iin,Iout](functionalUnits:FUnits,maxRequests:Nat)   :exit:=

(* The file service user may issue an F-READ or F-WRITE request. *)

 Iin?fsp:FSPrim?tn:Nat[((isAvailableRead(functionalUnits) and isReadReq(fsp)) or (isAvailableWrite(functionalUnits)
and isWriteReq(fsp))) and (tn eq Succ(0))];

(

(* The request for a data transfer becomes the current read or write data transfer as appropriate. Read data transfers will be performed concurrently with any write data transfers that are subsequently requested. *)

( ( [isAvailableRead(functionalUnits) and isReadReq(fsp)]->
    Read[Iin,Iout](concurrent,functionalUnits,maxRequests,tn,makeEl(fsp,tn),tn,empty,empty,empty)

    >>      ReadIdle[Iin,Iout](functionalUnits,maxRequests)
  )

   []

  ( [isAvailableWrite(functionalUnits) and isWriteReq(fsp)]->
    Write[Iin,Iout](concurrent,functionalUnits,maxRequests,tn,makeEl(fsp,tn),tn,empty,empty,empty)

    >>      WriteIdle[Iin,Iout](functionalUnits,maxRequests)
  )
)

 |||

(* The file service user may issue an F-READ or F-WRITE request depending on the type of the data transfer that was first requested. *)

  (  [isAvailableWrite(functionalUnits) and isReadReq(fsp)]->
     WriteIdle[Iin,Iout](functionalUnits,maxRequests)

   []

   [isAvailableRead(functionalUnits) and isWriteReq(fsp)]->
     ReadIdle[Iin,Iout](functionalUnits,maxRequests)
  )
)

>>

(* The file service user has the opportunity to exit the specification. Alternatively, further data transfers may be requested, as specified by process ConcurrentIdle. This represents the data transfer idle state. *)

        (  ConcurrentIdle[Iin,Iout](functionalUnits,maxRequests)

      []
```

Figure 22 (continued) - Initiator service definition

```
        exit
      )

where

(*
process ReadIdle

This process specifies the behaviour for the progress of read data transfers.
*)

 process ReadIdle[Iin,Iout](functionalUnits:FUnits,maxRequests:Nat)  :exit:=

(* The file service user may issue an F-READ request, this request will become the current read data transfer. *)

 ( Iin?fsp:FSPrim?tn:Nat[isReadReq(fsp) and (tn eq Succ(0))];
   Read[Iin,Iout]
   (concurrent,functionalUnits,maxRequests,tn,makeEl(fsp,tn),tn,empty,empty,empty)

  >>  ReadIdle[Iin,Iout](functionalUnits,maxRequests)
 )

  []

   exit

 endproc  (*ReadIdle*)

(*
process WriteIdle

The process specifies the behaviour for the progress of write data transfers.
*)

 process WriteIdle[Iin,Iout](functionalUnits:FUnits,maxRequests:Nat)  :exit:=

(* The file service user may issue an F-WRITE request, this request will become the current write data transfer. *)

 ( Iin?fsp:FSPrim?tn:Nat[isWriteReq(fsp) and (tn eq Succ(0))];
   Write[Iin,Iout](concurrent,functionalUnits,maxRequests,tn,makeEl(fsp,tn),tn,empty,empty,empty)

  >>  WriteIdle[Iin,Iout](functionalUnits,maxRequests)
 )

 []

  exit

 endproc  (*WriteIdle*)

endproc  (*ConcurrentIdle*)


(*
```

Figure 22 - (continued) Initiator service definition

process Read

This process specifies the behaviour for the current read data transfer in the case where there are no outstanding requests on the **requestQ**.
*)

```
process  Read[Iin,Iout]
            (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

(* An F-DATA indication may be issued to the file service user.*)

Iout?fsp:FSPrim[isDataInd(fsp)];
 Read[Iin,Iout]
  (degree,functionalUnits,maxRequests,tnCounter,current,reqOutstanding,reqQ,dataEndQ,xfrEndQ)

[]

(* An F-DATA-END indication may be issued to the file service user. The type and transfer number of the current read data transfer is added to the dataEndQ. *)

Iout?fsp:FSPrim[isDataEndInd(fsp)];
   ReadDataXfrIdle[Iin,Iout]
   (degree,functionalUnits,maxRequests,tnCounter,
   nullEl,reqOutstanding,reqQ,add(current,dataEndQ),xfrEndQ)

[]

(* If overlapped access is in use and the number of outstanding requests has not exceeded the maximum, then the file service user may issue an F-READ request. If consecutive access is in use and the Write functional unit has been negotiated then the file service user may issue an F-WRITE request. The type and transfer number of the request is added to the requestQ and the number of outstanding requests is incremented. The subsequent behaviour is dependant on the type of the request. *)

 [not(isNormal(degree)) and (reqOutstanding lt maxRequests)]->
 (   Iin?fsp:FSPrim?tn:Nat[isReadReq(fsp) and (tn eq (tnCounter+Succ(0)))];
     ReadRead[Iin,Iout]
     (degree,functionalUnits,maxRequests,tn,
     current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)

   []

   [isAvailableWrite(functionalUnits) and isConsecutive(degree)]->
   Iin?fsp:FSPrim?tn:Nat[isWriteReq(fsp) and (tn eq (tnCounter+Succ(0)))];
     ReadWrite[Iin,Iout]
     (degree,functionalUnits,maxRequests,tn,
     current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)
 )

[]

(* The file service user may issue an F-TRANSFER-END request for the data transfer at the head of the dataEndQ. The data transfer is removed from the dataEndQ and added to the xfrEndQ. *)

 [not(isEmpty(dataEndQ))]->
 Iin?fsp:FSPrim?tn:Nat[isXfrEndReadReq(fsp) and (tn eq number(head(dataEndQ)))];
  Read[Iin,Iout]
```

Figure 22 (continued) - Initiator service definition

```
  (degree,functionalUnits,maxRequests,tnCounter,
  current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

[]

(* An F-TRANSFER-END confirm may be issued to the file service user for the data transfer at the head of the xfrEndQ. The data transfer is removed
from the xfrEndQ. *)

 [not(isEmpty(xfrEndQ))]->
 Iout?fsp:FSPrim?tn:Nat[isXfrEndReadConf(fsp) and (tn eq number(head(xfrEndQ)))];
  Read[Iin,Iout]
   (degree,functionalUnits,maxRequests,tnCounter,
   current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))

[]

(* The initiating file service user may issue an F-CANCEL request for the data transfer at the head of the dataEndQ. If this is empty then the file
service user may issue an F-CANCEL request for the current data transfer. If the dataEndQ is empty then an F-CANCEL indication may be issued
to the file service user for the current data transfer. *)

Interrupt[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endproc (* Read *)


(*
process ReadRead

This process specifies the behaviour in the case where the current transfer is a read and the next requested data transfer
is also a read. The behaviour is only allowed if consecutive or concurrent access is in use. *)

process  ReadRead[Iin,Iout](degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

(* An F-DATA indication may be issued to the file service user for the current data transfer. *)

 Iout?fsp:FSPrim[isDataInd(fsp)];
   ReadRead[Iin,Iout]
   (degree,functionalUnits,maxRequests,tnCounter,current,reqOutstanding,reqQ,dataEndQ,xfrEndQ)

[]

(* An F-DATA-END indication may be issued to the file service user for the current data transfer. The current data transfer is added to the
dataEndQ; the request at the head of the dataEndQ becomes the current data transfer. The subsequent behaviour is determined on the basis
of the request that will now be at the head of the requestQ. *)

 Iout?fsp:FSPrim[isDataEndInd(fsp)];
 ( [isWriteReq(prim(head(tailOf(reqQ))))]->
    ReadWrite[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,
     head(reqQ),reqOutstanding,tailOf(reqQ),add(current,dataEndQ),xfrEndQ)

   []

   [isReadReq(prim(head(tailOf(reqQ))))]->
    ReadRead[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,
     head(reqQ),reqOutstanding,tailOf(reqQ),add(current,dataEndQ),xfrEndQ)
```

Figure 22 - (continued) Initiator service definition

```
  []

   [isEmpty(tailOf(reqQ))]->
   Read[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,
    head(reqQ),reqOutstanding,tailOf(reqQ),add(current,dataEndQ),xfrEndQ)
 )

[]

(* Further requests for read data transfers may be requested if the number of requests outstanding does not exceed the maximum. Write requests may
also be requested only if consecutive access is in use and the write functional unit is available. The type and transfer number of the request is added
to the requestQ*)

 [reqOutstanding lt maxRequests] ->
 ( Iin?fsp:FSPrim?tn:Nat[isReadReq(fsp) and (tn eq (tnCounter+Succ(0)))];
     ReadRead[Iin,Iout]
     (degree,functionalUnits,maxRequests,tn,
    current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)

   []

   [isAvailableWrite(functionalUnits) and isConsecutive(degree)]->
   Iin?fsp:FSPrim?tn:Nat[isWriteReq(fsp) and (tn eq (tnCounter+Succ(0)))];
     ReadRead[Iin,Iout]
     (degree,functionalUnits,maxRequests,tn,
    current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)
 )

[]

(* The file service user may issue an F-TRANSFER-END request for the data transfer at the head of the dataEndQ. The head of the dataEndQ
is removed and added to the xfrEndQ. *)

 [not(isEmpty(dataEndQ))]->
 Iin?fsp:FSPrim?tn:Nat[isXfrEndReadReq(fsp) and (tn eq number(head(dataEndQ)))];
  ReadRead[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,
   current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

[]

(* An F-TRANSFER-END confirm may be issued to the file service user for the data transfer at the head of the xfrEndQ. The head of the xfrEndQ
is removed. *)

 [not(isEmpty(xfrEndQ))]->
 Iout?fsp:FSPrim?tn:Nat[isXfrEndReadConf(fsp) and (tn eq number(head(xfrEndQ)))];
  ReadRead[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,
   current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))

[]

(* The initiating file service user may issue an F-CANCEL for the data transfer at the head of the dataEndQ. If this is empty then the file service user
may issue an F-CANCEL request for the current read data transfer. If the dataEndQ is empty then an F-CANCEL indication may be issued to the
file service user for the current read data transfer. *)
```

Figure 22 (continued) - Initiator service definition

```
Interrupt[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endproc (* ReadRead *)


(*
process ReadWrite

This process specifies the behaviour in the case where the current transfer is a read and the next requested data transfer
is a write. The behaviour is only allowed if consecutive access is in use. *)

process  ReadWrite[Iin,Iout](degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
           current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

[not(isNullEl(current))]->

(* An F-DATA indication may issued to the file service user for the current data transfer. *)


( Iout?fsp:FSPrim[isDataInd(fsp)];
    ReadWrite[Iin,Iout]
   (degree,functionalUnits,maxRequests,tnCounter,current,reqOutstanding,reqQ,dataEndQ,xfrEndQ)

  []

(* An F-DATA-END indication may be issued to the file service user for the current read data transfer. The current read data transfer is added to
the dataEndQ and the current data transfer becomes null. *)

  Iout?fsp:FSPrim[isDataEndInd(fsp)];
   ReadWrite[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,
    nullEl,reqOutstanding,reqQ,add(current,dataEndQ),xfrEndQ)
)

[]

(* The file service user may issue an F-TRANSFER-END request for the (read) data transfer that is at the head of the dataEndQ. The head of the
dataEndQ is removed and added to the xfrEndQ.

If the dataEndQ is empty and there is no current data transfer, then the head of the requestQ becomes the current data transfer. The
subsequent behaviour is determined on the basis of the type of the request that is now at the head of the requestQ. If however, there is a current
data transfer, or the dataEndQ is not empty, then the behaviour is specified by process ReadWrite. *)

[not(isEmpty(dataEndQ))]->
Iin?fsp:FSPrim?tn:Nat[isXfrEndReadReq(fsp) and (tn eq number(head(dataEndQ)))];
( [isEmpty(tailOf(dataEndQ)) and isNullEl(current)]->
  ( [isReadReq(prim(head(tailOf(reqQ))))]->
     WriteRead[Iin,Iout]
      (degree,functionalUnits,maxRequests,tnCounter,
     head(reqQ),reqOutstanding,tailOf(reqQ),tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

    []

    [isWriteReq(prim(head(tailOf(reqQ))))]->
      WriteWrite[Iin,Iout]
      (degree,functionalUnits,maxRequests,tnCounter,
     head(reqQ),reqOutstanding,tailOf(reqQ),tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))
```

Figure 22 - (continued) Initiator service definition

```
    []

     [isEmpty(tailOf(reqQ))]->
     Write[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,
      head(reqQ),reqOutstanding,tailOf(reqQ),tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))
   )

   []

    [not(isEmpty(tailOf(dataEndQ))) or not(isNullEl(current))]->
     ReadWrite[Iin,Iout]
      (degree,functionalUnits,maxRequests,tnCounter,
     current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))
 )

[]

(* The file service user may issue an F-WRITE or F-READ request. The type and transfer number of the request is added to the requestQ *)

 [reqOutstanding lt maxRequests]->
 Iin?fsp:FSPrim?tn:Nat[(isReadReq(fsp) or isWriteReq(fsp)) and (tn eq (tnCounter+Succ(0)))];
   ReadWrite[Iin,Iout]
   (degree,functionalUnits,maxRequests,tn,
  current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)

[]

(* An F-TRANSFER-END confirm may be issued to the file service user for the data transfer at the head of the xfrEndQ. The head of the xfrEndQ is
then removed. *)

 [not(isEmpty(xfrEndQ))]->
 Iout?fsp:FSPrim?tn:Nat[isXfrEndReadConf(fsp) and (tn eq number(head(xfrEndQ)))];
   ReadWrite[Iin,Iout]
   (degree,functionalUnits,maxRequests,tnCounter,
   current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))

[]

(* If the dataEndQ is empty then the file service user may issue an F-CANCEL request for the current read data transfer and an F-CANCEL
indication may be issued to the file service user for the current read data transfer. If the dataEndQ is not empty then the file service user may issue
an F-CANCEL request for the read data transfer at the head of the dataEndQ. *)

Interrupt[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endproc (* ReadWrite *)


(*
process ReadDataXfrIdle

The process specifies the behaviour where there are no data transfer requests on the requestQ and there is no current data
transfer in progress. *)

process     ReadDataXfrIdle[Iin,Iout](degree:Overlap,functionalUnits:FUnits,
              maxRequests,tnCounter:Nat,current:Element,reqOutstanding:Nat,reqQ,dataEndQ,
```

Figure 22 (continued) - Initiator service definition

```
            xfrEndQ:Queue)  :exit:=

(* The file service user may issue an F-TRANSFER-END request for the (read) data transfer at the head of the dataEndQ. The head of the
dataEndQ is removed and added to the xfrEndQ. The subsequent behaviour is dependant on whether the dataEndQ is empty. *)

Iin?fsp:FSPrim?tn:Nat[isXfrEndReadReq(fsp) and (tn eq number(head(dataEndQ)))];
( [isEmpty(tailOf(dataEndQ))]->
    ReadXfrEndIdle[Iin,Iout]
    (degree,functionalUnits,maxRequests,tnCounter,
   current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

  []

  [not(isEmpty(tailOf(dataEndQ)))]->
    ReadDataXfrIdle[Iin,Iout]
    (degree,functionalUnits,maxRequests,tnCounter,
      current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))
)

[]

(* If overlapped access is in use then the file service user may issue an F-READ request. If consecutive access is in use then the user may issue an
F-WRITE request. A request for a read data transfer will become the current data transfer with subsequent behaviour specified in process Read;
a request for a write data transfer is added to the requestQ and subsequent behaviour is specified by process ReadWrite. *)

[not(isNormal(degree)) and (reqOutstanding lt maxRequests)]->
( Iin?fsp:FSPrim?tn:Nat[isReadReq(fsp) and (tn eq (tnCounter+Succ(0)))];
   Read[Iin,Iout]
    (degree,functionalUnits,maxRequests,tn,
   makeEl(fsp,tn),Succ(reqOutstanding),reqQ,dataEndQ,xfrEndQ)

  []

  [isAvailableWrite(functionalUnits) and isConsecutive(degree)]->
  Iin?fsp:FSPrim?tn:Nat[isWriteReq(fsp) and (tn eq (tnCounter+Succ(0)))];
    ReadWrite[Iin,Iout]
    (degree,functionalUnits,maxRequests,tn,
   current,reqOutstanding,add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)
)

[]

(* An F-TRANSFER-END confirm may be issued to the file service user for the data transfer at the head of the xfrEndQ. The data transfer is removed
from the head of the xfrEndQ. *)

[not(isEmpty(xfrEndQ))]->
Iout?fsp:FSPrim?tn:Nat[isXfrEndReadConf(fsp) and (tn eq number(head(xfrEndQ)))];
  ReadDataXfrIdle[Iin,Iout]
  (degree,functionalUnits,maxRequests,tnCounter,
 current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))

[]

(* The file service user may issue an F-CANCEL request for the read data transfer at the head of the dataEndQ. *)
```

Figure 22 - (continued) Initiator service definition

```
Interrupt[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endproc (* ReadDataXfrIdle *)


(*
process ReadXfrEndIdle

The process specifies the behaviour when there is no current data transfer and the dataEndQ is empty. *)

process     ReadXfrEndIdle[Iin,Iout]
              (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
              current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

(* An F-TRANSFER-END confirm may be issued to the file service user for the data transfer at the head of the xfrEndQ. The data transfer is removed
from the xfrEndQ. If the xfrEndQ is empty then the process may exit. *)

 Iout?fsp:FSPrim?tn:Nat[isXfrEndReadConf(fsp) and (tn eq number(head(xfrEndQ)))];
 ([isEmpty(tailOf(xfrEndQ))]->
  exit

 []

  [not(isEmpty(tailOf(xfrEndQ)))]->
    ReadXfrEndIdle[Iin,Iout]
    (degree,functionalUnits,maxRequests,tnCounter,
   nullEl,pred(reqOutstanding),empty,empty,tailOf(xfrEndQ))
 )

[]

(* If overlapped access is in use then the file service user may issue an F-READ request. If consecutive access is in use then the user may issue an
F-WRITE request. A request for a read or write data transfer will become the current data transfer. The subsequent behaviour is dependant on the
type of the request. *)

 [not(isNormal(degree)) and (reqOutstanding lt maxRequests)]->
 ( Iin?fsp:FSPrim?tn:Nat[isReadReq(fsp) and (tn eq (tnCounter+Succ(0)))];
    Read[Iin,Iout]
     (degree,functionalUnits,maxRequests,tn,
    makeEl(fsp,tn),Succ(reqOutstanding),reqQ,dataEndQ,xfrEndQ)

  []

   [isAvailableWrite(functionalUnits) and isConsecutive(degree)]->
   Iin?fsp:FSPrim?tn:Nat[isWriteReq(fsp) and (tn eq (tnCounter+Succ(0)))];
    Write[Iin,Iout]
     (degree,functionalUnits,maxRequests,tn,
    makeEl(fsp,tn),Succ(reqOutstanding),reqQ,dataEndQ,xfrEndQ)
 )

endproc (* ReadXfrEndIdle *)


(*
process Write
```

Figure 22 (continued) - Initiator service definition

```
This process specifies the behaviour for the current write data transfer in the case where there are no outstanding requests
on the requestQ.
*)

process   Write[Iin,Iout]
             (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
             current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

[not(isNullEl(current))]->

(* The file service user may issue an F-DATA request. *)

( Iin?fsp:FSPrim[isDataReq(fsp)];
   Write[Iin,Iout]
   (degree,functionalUnits,maxRequests,tnCounter,current,reqOutstanding,reqQ,dataEndQ,xfrEndQ)

  []

(* The file service user may issue an F-DATA-END request. The current data transfer is added to the dataEndQ and the current data transfer
becomes null.*)

  Iin?fsp:FSPrim[isDataEndReq(fsp)];
    Write[Iin,Iout]
     (degree,functionalUnits,maxRequests,tnCounter,
     nullEl,reqOutstanding,reqQ,add(current,dataEndQ),xfrEndQ)
)

[]

(* The file service user may issue an F-TRANSFER-END request for the (write) data transfer at the head of the dataEndQ. The data transfer is
removed from the dataEndQ and added to the xfrEndQ. *)

[not(isEmpty(dataEndQ))]->
Iin?fsp:FSPrim?tn:Nat[isXfrEndWriteReq(fsp) and (tn eq number(head(dataEndQ)))];
   WriteDataXfrIdle[Iin,Iout]
  (degree,functionalUnits,maxRequests,tnCounter,
 current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

[]

(* If overlapped access is in use and the number of outstanding requests has not exceeded the maximum, then the file service user may issue an F-
Write request. If consecutive access is in use and the Write functional unit has been negotiated then the file service user may issue an F-READ
request. The type and transfer number of the request is added to the requestQ, and the number of outstanding requests is incremented. The
subsequent behaviour is dependant on the type of the request. *)

[not(isNormal(degree)) and (reqOutstanding lt maxRequests)]->
( [isAvailableRead(functionalUnits) and isConsecutive(degree)]->
  Iin?fsp:FSPrim?tn:Nat[isReadReq(fsp) and (tn eq (tnCounter+Succ(0)))];
    WriteRead[Iin,Iout]
    (degree,functionalUnits,maxRequests,tn,
    current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)

  []

  Iin?fsp:FSPrim?tn:Nat[isWriteReq(fsp) and (tn eq (tnCounter+Succ(0)))];
    WriteWrite[Iin,Iout]
```

Figure 22 - (continued) Initiator service definition

```
    (degree,functionalUnits,maxRequests,tn,
    current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)
 )

[]

(* An F-TRANSFER-END confirm may be issued to the file service user for the data transfer at the head of the xfrEndQ. The data transfer is removed
from the xfrEndQ. *)

 [not(isEmpty(xfrEndQ))]->
 Iout?fsp:FSPrim?tn:Nat[((isXfrEndReadConf(fsp) and isReadReq(prim(head(xfrEndQ))))
   or (isXfrEndWriteConf(fsp) and isWriteReq(prim(head(xfrEndQ)))))
   and (tn eq number(head(xfrEndQ)))];
  Write[Iin,Iout]
   (degree,functionalUnits,maxRequests,tnCounter,
   current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))

[]

(* The file service user may issue an F-CANCEL request for the current write data transfer. An F-CANCEL request may be issued to the file service
user for the first write data transfer on the xfrEndQ. If there is no write data transfer on the xfrEndQ then an F-CANCEL request may be issued to
the file service user for the data transfer at the head of the dataEndQ, or if this is empty, the current data transfer. *)

Interrupt[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endproc (* Write *)


(*
process WriteWrite

This process specifies the behaviour in the case where the current data transfer is a write and the next requested data transfer
is also a write. The behaviour is only allowed if consecutive or concurrent access is in use. *)

process    WriteWrite[Iin,Iout]
             (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
             current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

 [not(isNullEl(current))]->
 (

(* The file service user may issue an F-DATA request for the current data transfer. *)

   Iin?fsp:FSPrim[isDataReq(fsp)];
     WriteWrite[Iin,Iout]
    (degree,functionalUnits,maxRequests,tnCounter,current,reqOutstanding,reqQ,dataEndQ,xfrEndQ)

  []

(* The file service user may issue an F-DATA-END request for the current data transfer. The current data transfer is added to the dataEndQ and
the current data transfer becomes null. *)

   Iin?fsp:FSPrim[isDataEndReq(fsp)];
    WriteWrite[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,
    nullEl,reqOutstanding,reqQ,add(current,dataEndQ),xfrEndQ)
 )
```

Figure 22 (continued) - Initiator service definition

```
[]

(* The file service user may issue an F-TRANSFER-END request for the data transfer at the head of the dataEndQ. The head of the dataEndQ
is removed and added to the xfrEndQ. The subsequent behaviour is dependant on the request at the head of the requestQ; the current data
transfer remains null. *)

 [not(isEmpty(dataEndQ))]->
 Iin?fsp:FSPrim?tn:Nat[isXfrEndWriteReq(fsp) and (tn eq number(head(dataEndQ)))];
 ( [isEmpty(tailOf(reqQ))]->
    Write[Iin,Iout]
     (degree,functionalUnits,maxRequests,tnCounter,
    head(reqQ),reqOutstanding,tailOf(reqQ),tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

  []

   [isReadReq(prim(head(tailOf(reqQ))))]->
     WriteRead[Iin,Iout]
     (degree,functionalUnits,maxRequests,tnCounter,
    head(reqQ),reqOutstanding,tailOf(reqQ),tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

  []

   [isWriteReq(prim(head(tailOf(reqQ))))]->
     WriteWrite[Iin,Iout]
     (degree,functionalUnits,maxRequests,tnCounter,
    head(reqQ),reqOutstanding,tailOf(reqQ),tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))
 )

[]

(* Further requests for write data transfers may be requested if the number of requests outstanding does not exceed the maximum. Read requests may
be requested only if consecutive access is in use and the read functional unit is available. The type and transfer number of the request is added to
the requestQ *)

 [reqOutstanding lt maxRequests]->
 ( [isAvailableRead(functionalUnits) and isConsecutive(degree)]->
   Iin?fsp:FSPrim?tn:Nat[isReadReq(fsp) and (tn eq (tnCounter+Succ(0)))];
     WriteWrite[Iin,Iout]
     (degree,functionalUnits,maxRequests,tn,
    current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)



  []

   Iin?fsp:FSPrim?tn:Nat[isWriteReq(fsp) and (tn eq (tnCounter+Succ(0)))];
     WriteWrite[Iin,Iout]
     (degree,functionalUnits,maxRequests,tn,
    current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)
 )

[]

(* An F-TRANSFER-END confirm may be issued to the file service user for the data transfer at the head of the xfrEndQ. The data transfer is removed
```

Figure 22 - (continued) Initiator service definition

```
from the xfrEndQ. *)

 [not(isEmpty(xfrEndQ))]->
 Iout?fsp:FSPrim?tn:Nat[((isXfrEndReadConf(fsp) and isReadReq(prim(head(xfrEndQ))))
   or (isXfrEndWriteConf(fsp) and isWriteReq(prim(head(xfrEndQ)))))
   and (tn eq number(head(xfrEndQ)))];
   WriteWrite[Iin,Iout]
   (degree,functionalUnits,maxRequests,tnCounter,
   current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))

 []

(* The file service user may issue an F-CANCEL request for the current write data transfer. An F-CANCEL request may be issued to the file service
user for the first write data transfer on the xfrEndQ. If there is no write data transfer on the xfrEndQ then an F-CANCEL request may be issued to
the file service user for the data transfer at the head of the dataEndQ, or if this is empty, the current data transfer. *)

Interrupt[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endproc (* WriteWrite *)


(*
process WriteRead

This process specifies the behaviour in the case where the current data transfer is a write and the next requested data transfer
on the requestQ is a read. *)

process   WriteRead[Iin,Iout]
            (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

 [not(isNullEl(current))]->
 (

(* The file service user may issue an F-DATA request. *)

Iin?fsp:FSPrim[isDataReq(fsp)];
    WriteRead[Iin,Iout]
   (degree,functionalUnits,maxRequests,tnCounter,current,reqOutstanding,reqQ,dataEndQ,xfrEndQ)

  []

(* The file service user may issue an F-DATA-END request for the current data transfer. The data transfer is added to the dataEndQ and the
current data transfer becomes null. *)

  Iin?fsp:FSPrim[isDataEndReq(fsp)];
    WriteRead[Iin,Iout]
    (degree,functionalUnits,maxRequests,tnCounter,
    nullEl,reqOutstanding,reqQ,add(current,dataEndQ),xfrEndQ)
 )

[]

(* The file service user may issue an F-TRANSFER-END request for the data transfer at the head of the dataEndQ. The data transfer is removed
from the dataEndQ and added to the xfrEndQ. *)
```

Figure 22 (continued) - Initiator service definition

```
[not(isEmpty(dataEndQ))]->
Iin?fsp:FSPrim?tn:Nat[isXfrEndWriteReq(fsp) and (tn eq number(head(dataEndQ)))];
  WriteRead[Iin,Iout]
   (degree,functionalUnits,maxRequests,tnCounter,
  current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

[]

(* An F-TRANSFER-END confirm may be issued to the file service user for the data transfer at the head of the xfrEndQ. The data transfer is removed
from the xfrEndQ. If the xfrEndQ and the dataEndQ are empty, and the current data transfer is null, then the data transfer at the head of the
requestQ becomes the current data transfer and is removed from the requestQ; the subsequent behaviour is dependant on the type of the data
transfer that is now at the head of the requestQ. *)

[not(isEmpty(xfrEndQ))]->
Iout?fsp:FSPrim?tn:Nat[((isXfrEndReadConf(fsp) and isReadReq(prim(head(xfrEndQ))))
  or (isXfrEndWriteConf(fsp) and isWriteReq(prim(head(xfrEndQ)))))
  and (tn eq number(head(xfrEndQ)))];
( [isEmpty(tailOf(xfrEndQ)) and isEmpty(dataEndQ) and isNullEl(current)]->
  ( [isEmpty(tailOf(reqQ))]->
    Read[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,
     head(reqQ),pred(reqOutstanding),tailOf(reqQ),dataEndQ,tailOf(xfrEndQ))

    []

    [isReadReq(prim(head(tailOf(reqQ))))]->
     ReadRead[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,
      head(reqQ),pred(reqOutstanding),tailOf(reqQ),dataEndQ,tailOf(xfrEndQ))

    []

    [isWriteReq(prim(head(tailOf(reqQ))))]->
      ReadWrite[Iin,Iout]
       (degree,functionalUnits,maxRequests,tnCounter,
      head(reqQ),pred(reqOutstanding),tailOf(reqQ),dataEndQ,tailOf(xfrEndQ))

  )

  []

  [not(isEmpty(tailOf(xfrEndQ))) or not(isEmpty(dataEndQ)) or not(isNullEl(current))]->
    WriteRead[Iin,Iout]
    (degree,functionalUnits,maxRequests,tnCounter,
    current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))
)

[]

(* The file service user may issue an F-WRITE or F-READ request. The type and transfer number of the request is added to the requestQ. *)

[reqOutstanding lt maxRequests]->
Iin?fsp:FSPrim?tn:Nat[(isReadReq(fsp) or isWriteReq(fsp)) and (tn eq (tnCounter+Succ(0)))];
  WriteRead[Iin,Iout]
   (degree,functionalUnits,maxRequests,tn,
  current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)
```

Figure 22 - (continued) Initiator service definition

```
[]

(* The file service user may issue an F-CANCEL request for the current write data transfer. An F-CANCEL request may be issued to the file service
user for the first write data transfer on the xfrEndQ. If there is no write data transfer on the xfrEndQ then an F-CANCEL request may be issued to
the file service user for the data transfer at the head of the dataEndQ, or if this is empty, the current data transfer. *)

Interrupt[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endproc (* WriteRead *)


(*
process WriteDataXfrIdle

This process specifies the behaviour when the current data transfer is null, the dataEndQ is empty and the xfrEndQ is not empty.
*)

process     WriteDataXfrIdle[Iin,Iout]
             (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
             current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

(* An F-TRANSFER-END confirm may be issued to the file service user for the data transfer at the head of the xfrEndQ. The data transfer is removed
from the xfrEndQ. The subsequent behaviour is dependant on whether or not the xfrEndQ is now empty. *)

 (( [isReadReq(prim(head(xfrEndQ)))]->
    Iout?fsp:FSPrim?tn:Nat[isXfrEndReadConf(fsp) and (tn eq number(head(xfrEndQ)))];
     exit

    []

    [isWriteReq(prim(head(xfrEndQ)))]->
    Iout?fsp:FSPrim?tn:Nat[isXfrEndWriteConf(fsp) and (tn eq number(head(xfrEndQ)))];
     exit
  )

  >>

  ( [isEmpty(tailOf(xfrEndQ))]->
     exit

    []

    [not(isEmpty(tailOf(xfrEndQ)))]->
      WriteDataXfrIdle[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,
     current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))
  )
 )

[]

(* Further requests for write data transfers may be requested if consecutive or concurrent access is in use and the number of requests outstanding
does not exceed the maximum. Read requests may be requested only if consecutive access is in use and the read functional unit is available. The
subsequent behaviour is dependant on the type of the request. *)

 [not(isNormal(degree)) and (reqOutstanding lt maxRequests)]->
 ( [isAvailableRead(functionalUnits) and isConsecutive(degree)]->
```

Figure 22 (continued) - Initiator service definition

```
  Iin?fsp:FSPrim?tn:Nat[isReadReq(fsp) and (tn eq (tnCounter+Succ(0)))];
    WriteRead[Iin,Iout]
    (degree,functionalUnits,maxRequests,tn,
   current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)

 []

  Iin?fsp:FSPrim?tn:Nat[isWriteReq(fsp) and (tn eq (tnCounter+Succ(0)))];
    Write[Iin,Iout]
    (degree,functionalUnits,maxRequests,tn,
   makeEl(fsp,tn),Succ(reqOutstanding),reqQ,dataEndQ,xfrEndQ)
)

[]

(* An F-CANCEL indication may be issued to the file service user for the first write request on the xfrEndQ. *)

Interrupt[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endproc (* WriteDataXfrIdle *)


(*
process Interrupt

This process specifies the behaviour when a data transfer request is cancelled. *)

process   Interrupt[Iin,Iout]
            (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

  InitiatorCancel[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

[]

  ResponderCancel[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

where


(*
process InitiatorCancel

This process specifies the predicates and behaviour for the cancellation of a data transfer by the initiating file service user.
*)

  process   InitiatorCancel[Iin,Iout]
            (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

 [isAvailableRead(functionalUnits)]->
  ReadCancel[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

 []

 [isAvailableWrite(functionalUnits)]->
```

Figure 22 - (continued) Initiator service definition

```
  WriteCancel[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

 where

(*
process ReadCancel

This process specifies the predicates and the behaviour for the cancellation of a read data transfer by the file service user.
*)

   process ReadCancel[Iin,Iout](degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
           current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

(* The file service user may issue an F-CANCEL request for the read data transfer at the head of the dataEndQ. *)

   [isReadReq(prim(head(dataEndQ)))]->
   Iin?fsp:FSPrim?tn:Nat[isCancelReadReq(fsp) and (tn eq number(head(dataEndQ)))];
    CancelPending[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),xfrEndQ)

  []

(* The file service user may issue an F-CANCEL request for the current data transfer is the dataEndQ is empty. *)

   [isEmpty(dataEndQ) and isReadReq(prim(current))]->
   Iin?fsp:FSPrim?tn:Nat[isCancelReadReq(fsp) and (tn eq number(current))];
    CancelPending[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),xfrEndQ)

  endproc (* ReadCancel *)


(*
process WriteCancel

This process specifies the predicates and the behaviour for the cancellation of a write data transfer by the file service user.
*)

   process WriteCancel[Iin,Iout]
           (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
           current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

(* The file service user may issue an F-CANCEL request for the current write data transfer if the xfrEndQ is empty. *)

   [isEmpty(xfrEndQ) and isWriteReq(prim(current))]->
   Iin?fsp:FSPrim?tn:Nat[isCancelWriteReq(fsp) and (tn eq number(current))];
    CancelPending[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),xfrEndQ)

  endproc (* WriteCancel *)


(*
process CancelPending

This process specifies the completion of a data transfer cancellation.
*)

   process CancelPending[Iin,Iout]
```

Figure 22 (continued) - Initiator service definition

```
            (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            cancelData:Element,xfrEndQ:Queue) :exit:=

(* An F-CANCEL confirm may be issued for the data transfer that is being cancelled. *)

  ( [isCancelReadReq(prim(cancelData))]->
    Iout?fsp:FSPrim?tn:Nat[isCancelReadConf(fsp) and (tn eq number(cancelData))];
     exit

   []

    [isCancelWriteReq(prim(cancelData))]->
    Iout?fsp:FSPrim?tn:Nat[isCancelWriteConf(fsp) and (tn eq number(cancelData))];
     exit
  )

  >>

(* If there are any data transfers on the xfrEndQ then these will be read data transfers, subsequent behaviour is therefore as specified in process
ReadXfrEndIdle, otherwise the file service user may exit the specification. *)

   ( [isEmpty(xfrEndQ)]->
      exit

    []

     [not(isEmpty(xfrEndQ))]->
       ReadXfrEndIdle[Iin,Iout]
      (degree,functionalUnits,maxRequests,tnCounter,nullEl,0,empty,empty,xfrEndQ)
   )

 endproc (* CancelPending *)

endproc (* InitiatorCancel *)


(*
process ResponderCancel

This process specifies the predicates and behaviour for the cancellation of a data transfer by the responding file service user.
*)

  process  ResponderCancel[Iin,Iout]
           (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
           current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

 [isAvailableRead(functionalUnits)]->
  ReadCancel[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

 []

 [isAvailableWrite(functionalUnits)]->
  WriteCancel[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

 where
```

Figure 22 - (continued) Initiator service definition

```
(*
process ReadCancel

This process specifies the predicates and the behaviour for the cancellation of a read data transfer by the responding file service user. *)

   process ReadCancel[Iin,Iout]
           (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
           current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

(* An F-CANCEL indication may be issued to the file service user for the current data transfer if the dataEndQ is empty. *)

 [isEmpty(dataEndQ) and isReadReq(prim(current))]->
 Iout?fsp:FSPrim?tn:Nat[isCancelReadInd(fsp) and (tn eq number(current))];
  CancelPending[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),xfrEndQ)

  endproc (* ReadCancel *)


(*
process WriteCancel

This process specifies the predicates and the behaviour for the cancellation of a write data transfer by the responding file service user. *)

   process WriteCancel[Iin,Iout](degree:Overlap,functionalUnits:FUnits,maxRequests:Nat,
           tnCounter:Nat,current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

(* If there are no write data transfers on the xfrEndQ then an F-CANCEL indication may be issued to the file service user for the current write data transfer, if the dataEndQ is empty, or the write data transfer at the head of the dataEndQ. *)

 [isNull(prim(firstWriteReq(xfrEndQ)))]->
 ( [isEmpty(dataEndQ) and isWriteReq(prim(current))]->
   Iout?fsp:FSPrim?tn:Nat[isCancelWriteInd(fsp) and (tn eq number(current))];
    CancelPending[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),xfrEndQ)

   []

   [isNullEl(current) and isWriteReq(prim(head(dataEndQ)))]->
   Iout?fsp:FSPrim?tn:Nat[isCancelWriteInd(fsp) and (tn eq number(head(dataEndQ)))];
    CancelPending[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),xfrEndQ)
 )

 []

(* An F-CANCEL indication may be issued to the file service user for the first write data transfer on the xfrEndQ. *)

  [not(isNullEl(firstWriteReq(xfrEndQ)))]->
  Iout?fsp:FSPrim?tn:Nat[isCancelWriteInd(fsp) and (tn eq number(firstWriteReq(xfrEndQ)))];
   CancelPending[Iin,Iout](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),removeReqs(xfrEndQ))

  endproc (* WriteCancel *)


(*
```

Figure 22 (continued) - Initiator service definition

```
process CancelPending

This process specifies the completion of a data transfer cancellation.
*)

 process CancelPending[Iin,Iout]
           (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
           cancelData:Element,xfrEndQ:Queue) :exit:=

(* An F-CANCEL confirm may be issued for the data transfer that is being cancelled. *)

 ( [isCancelReadInd(prim(cancelData))]->
   Iin?fsp:FSPrim?tn:Nat[isCancelReadResp(fsp) and (tn eq number(cancelData))];
    exit

  []

   [isCancelWriteInd(prim(cancelData))]->
   Iin?fsp:FSPrim?tn:Nat[isCancelWriteResp(fsp) and (tn eq number(cancelData))];
    exit
  )

  >>

(* If there are any data transfers on the xfrEndQ then these will be read data transfers, subsequent behaviour is therefore as specified in process
ReadXfrEndIdle, otherwise the file service user may exit the specification. *)

  ( [isEmpty(xfrEndQ)]->
     exit

   []

    [not(isEmpty(xfrEndQ))]->
      ReadXfrEndIdle[Iin,Iout]
      (degree,functionalUnits,maxRequests,tnCounter,
      nullEl,size(xfrEndQ),empty,empty,removeLast(xfrEndQ))
  )

  endproc (* CancelPending *)

  endproc (* ResponderCancel *)

 endproc (* Interrupt *)

endspec (* Initiator_External_Service *)
```

Figure 22 - (continued) Initiator service definition

```
(*
Specification

The specification Responder_External_Service specifies the sequence of service primitives in the external file service of the
responder, for the duration of the bulk data transfer regime, for normal, consecutive and concurrent overlapped access.*)

Specification    Responder_External_Service[Rout,Rin,Local]    :exit


(*
Data types

The specification imports the standard library definitions for the types Boolean and NaturalNumber, these are used to define
the following types:

ExtendedNaturalNumber - this extends the definition of NaturalNumber to include the operation pred, which decrements
a parameter of type Nat.

FileServicePrimitives - values are defined to represent the file service primitives. Boolean operations are defined that
identify a particular primitive e.g. IsDataEndInd(transferEndInd)=false.

RequestQueue, QueueElement - a queue is defined with operations: empty (creates an empty queue), add (adds an
element to the queue), tail (removes the first element on the queue), head (returns the value of the first element on the queue),
last (returns the value of the last element on the queue), IsEmpty (determines whether the queue is empty), removeLast
(removes the last element on the queue), size (determines the number of elements on the queue), firstWriteInd (finds the
first write request on a queue), removeReqs (removes all elements from the end of a queue, up to and including the first write
request on the queue) and reverse (reverses the order of the elements on the queue). The elements of the queue are comprised
of a file service primitive and it's transfer number.

Note - the firstWriteInd and removeReqs operations are required to support the cancellation of a write data transfer after the
receipt of an F-TRANSFER-END indication by the responder. Equivalent operations are not required for read data transfer
operations.

DegreeOfOverlap - values are defined for normal, consecutive and concurrent; boolean operations are defined that
identify a given value.

FunctionalUnits - values are defined to represent the selection of the Read, Write and the combination of the Read &
Write functional units; boolean operations are defined that identify a given value.
*)


library

Boolean,NaturalNumber

endlib

type ExtendedNaturalNumber is NaturalNumber
     opns pred: Nat -> Nat
     eqns forall x:Nat
     ofsort Nat
          pred(0)          = 0;
          pred(Succ(x))    = x;
endtype

type FileServicePrimitives is Boolean
     sorts FSPrim
     opns null,
```

Figure 23 - Responder service definition

```
        readInd,
        writeInd,
        dataInd,
        dataReq,
        dataEndInd,
        dataEndReq,
        xfrEndReadInd,
        xfrEndReadResp,
        xfrEndWriteInd,
        xfrEndWriteResp,
        cancelReadInd,
        cancelReadReq,
        cancelReadConf,
        cancelReadResp,
        cancelWriteInd,
        cancelWriteReq,
        cancelWriteConf,
        cancelWriteResp: -> FSPrim

        Mp:          FSPrim -> Nat

        isNull,
        isReadInd,
        isWriteInd,
        isDataInd,
        isDataReq,
        isDataEndInd,
        isDataEndReq,
        isXfrEndReadInd,
        isXfrEndReadResp,
        isXfrEndWriteInd,
        isXfrEndWriteResp,
        isCancelReadInd,
        isCancelReadReq,
        isCancelReadConf,
        isCancelReadResp,
        isCancelWriteInd,
        isCancelWriteReq,
        isCancelWriteConf,
        isCancelWriteResp: FSPrim -> Bool

    eqns forall x,y,fsp:FSPrim

    ofsort Nat

        Mp(readInd)              = 0;
        Mp(writeInd)       = Succ(Mp(readInd));
        Mp(dataInd)              = Succ((writeInd));
        Mp(dataReq)              = Succ(Mp(dataInd));
        Mp(dataEndInd)           = Succ(Mp(dataReq));
        Mp(dataEndReq)           = Succ(Mp(dataEndInd));
        Mp(xfrEndReadInd)        = Succ(Mp(dataEndReq));
        Mp(xfrEndReadResp) = Succ(Mp(xfrEndReadInd));
        Mp(xfrEndWriteInd) = Succ(Mp(xfrEndReadResp));
        Mp(xfrEndWriteResp)      = Succ(Mp(xfrEndWriteInd));
```

Figure 23 (continued) - Responder service definition

```
            Mp(cancelReadInd)        = Succ(Mp(xfrEndWriteResp));
            Mp(cancelReadReq)        = Succ(Mp(cancelReadInd));
            Mp(cancelReadConf) = Succ(Mp(cancelReadReq));
            Mp(cancelReadResp) = Succ(Mp(cancelReadConf));
            Mp(cancelWriteInd) = Succ(Mp(cancelReadResp));
            Mp(cancelWriteReq) = Succ(Mp(cancelWriteInd));
            Mp(cancelWriteConf)      = Succ(Mp(cancelWriteReq));
            Mp(cancelWriteResp)      = Succ(Mp(cancelWriteConf));

      ofsort Bool

            isNull(fsp)              = Mp(fsp) eq Mp(null);
            isReadInd(fsp)           = Mp(fsp) eq Mp(readInd);
            isWriteInd(fsp)          = Mp(fsp) eq Mp(writeInd);
            isDataInd(fsp)           = Mp(fsp) eq Mp(dataInd);
            isDataReq(fsp)           = Mp(fsp) eq Mp(dataReq);
            isDataEndInd(fsp)        = Mp(fsp) eq Mp(dataEndInd);
            isDataEndReq(fsp)        = Mp(fsp) eq Mp(dataEndReq);
            isXfrEndReadInd(fsp)     = Mp(fsp) eq Mp(xfrEndReadInd);
            isXfrEndWriteInd(fsp)    = Mp(fsp) eq Mp(xfrEndWriteInd);
            isXfrEndReadResp(fsp)    = Mp(fsp) eq Mp(xfrEndReadResp);
            isXfrEndWriteResp(fsp)   = Mp(fsp) eq Mp(xfrEndWriteResp);
            isCancelReadInd(fsp)     = Mp(fsp) eq Mp(cancelReadInd);
            isCancelReadReq(fsp)     = Mp(fsp) eq Mp(cancelReadReq);
            isCancelReadConf(fsp)    = Mp(fsp) eq Mp(cancelReadConf);
            isCancelReadResp(fsp)    = Mp(fsp) eq Mp(cancelReadResp);
            isCancelWriteInd(fsp)    = Mp(fsp) eq Mp(cancelWriteInd);
            isCancelWriteReq(fsp)    = Mp(fsp) eq Mp(cancelWriteReq);
            isCancelWriteConf(fsp)   = Mp(fsp) eq Mp(cancelWriteConf);
            isCancelWriteResp(fsp)   = Mp(fsp) eq Mp(cancelWriteResp);
endtype


type QueueElement is FileServicePrimitives,Boolean,ExtendedNaturalNumber

      sorts Element
      opns nullEl: -> Element
            makeEl: FSPrim,Nat -> Element
            number: Element -> Nat
            prim: Element -> FSPrim
            isNullEl: Element -> Bool

      eqns forall fsp:FSPrim,tn:Nat,q_el:Element
      ofsort FSPrim
            prim(makeEl(fsp,tn)) = fsp;
            prim(nullEl) = null;
      ofsort Nat
            number(nullEl) = 0;
            number(makeEl(fsp,tn)) = tn;
      ofsort Bool
            isNullEl(nullEl) = true;
            isNullEl(q_el) = isNull(prim(q_el));
            ofsort Element
            makeEl(null,0) = nullEl;
      endtype
```

Figure 23 (continued) - Responder service definition

```
type RequestQueue is QueueElement, Boolean, ExtendedNaturalNumber
      sorts  Queue
      opns  empty:      -> Queue
            add: Element,Queue -> Queue
            tailOf: Queue -> Queue
            head: Queue -> Element
            last: Queue -> Element
            isEmpty: Queue -> Bool
            removeLast: Queue -> Queue
            size: Queue -> Nat
            firstWriteInd: Queue -> Element
            removeReqs: Queue -> Queue
            reverse: Queue -> Queue

      eqns  forall  x,y:Element,q:Queue
      ofsort  Element
            head(empty) = nullEl;
            head(add(x,empty)) = x;
            head(add(x,add(y,q))) = head(add(y,q));
            last(empty) =nullEl;
            last(add(x,empty)) = x;
            last(add(x,q)) = x;
            firstWriteInd(empty) = nullEl;
            isWriteInd(prim(head(q)))
                  => firstWriteInd(q) = head(q);
            not(isWriteInd(prim(head(q))))
                  => firstWriteInd(q) = firstWriteInd(tailOf(q))

      ofsort  Queue
            tailOf(empty) = empty;
            tailOf(add(x,empty)) = empty;
            tailOf(add(x,q)) = add(x,tailOf(q));
            add(nullEl,q) = q;
            removeLast(empty) = empty;
            removeLast(add(x,q)) = q;
            removeReqs(empty) = empty;
            isWriteInd(prim(head(q)))
                  => removeReqs(q) = empty;
            not(isWriteInd(prim(head(q))))
                  => removeReqs(q) = reverse(add(head(q),removeReqs(tailOf(q))));

      ofsort  Bool
            isEmpty(empty) = true;
            isEmpty(add(x,q)) = false;

      ofsort  Nat
            size(empty) = 0;
            size(add(x,q)) = Succ(size(q));

endtype

type DegreeOfOverlap is Boolean
```

Figure 23 (continued) - Responder service definition

```
        sorts  Overlap
        opns   normal: -> Overlap
               consecutive: -> Overlap
               concurrent: -> Overlap
               isNormal: Overlap -> Bool
               isConsecutive: Overlap -> Bool
               isConcurrent: Overlap -> Bool

        eqns   forall  degree:Overlap
        ofsort  Bool
               isNormal(normal) = true;
               isNormal(consecutive) = false;
               isNormal(concurrent) = false;

               isConsecutive(normal) = false;
               isConsecutive(consecutive) = true;
               isConsecutive(concurrent) = false;

               isConcurrent(normal) = false;
               isConcurrent(consecutive) = false;
               isConcurrent(concurrent) = true;
endtype


type FunctionalUnits is Boolean
        sorts  FUnits
        opns   setRead: -> FUnits
               setWrite: -> FUnits
               setReadWrite: -> FUnits
               isAvailableRead: FUnits -> Bool
               isAvailableWrite: FUnits -> Bool

        eqns   forall x:FUnits
               ofsort Bool
               isAvailableRead(setRead) = true;
               isAvailableRead(setWrite) = false;
               isAvailableRead(setReadWrite) = true;

               isAvailableWrite(setRead) = false;
               isAvailableWrite(setWrite) = true;
               isAvailableWrite(setReadWrite) = true;
endtype


(*
```

Behaviour

All events occur at the gates **Rout, RIn** or **Local**. Events at the gate **RIn** represent receipt of file service primitives by the file service provider; events at the gate **Rout** represent file service primitives that are issued by the file service user by the file service provider. The events at the gate **Local** establish the bounds on the behaviour of the file service i.e. the degree of overlap, the functional units available and, if overlapped access is allowed, the maximum number of request that may be outstanding.

The sequence of primitives allowed in the external file service is specified by the behaviour the process **NormalIdle, ConsecutiveIdle** or **ConcurrentIdle** (chosen on the basis of the **degreeOfOverlap**, established by an event at the gate **Local**).

Figure 23 (continued) - Responder service definition

```
*)

behaviour

Local?degree:Overlap;
([isNormal(degree)]->
 Local?functionalUnits:FUnits;
  NormalIdle[Rout,Rin](functionalUnits)

[]

 [isConsecutive(degree)]->
 Local?functionalUnits:FUnits?maxRequests:Nat;
   ConsecutiveIdle[Rout,Rin](functionalUnits,maxRequests)

[]

 [isConcurrent(degree)]->
 Local?functionalUnits:FUnits?maxRequests:Nat;
   ConcurrentIdle[Rout,Rin](functionalUnits,maxRequests)
)

where

(*
process  NormalIdle

This process specifies the allowed behaviour when overlapped access is not in use. This point represents the completion of
the exchange of F-OPEN primitives.
*)

process  NormalIdle[Rout,Rin](functionalUnits:FUnits)  :exit:-

(* If the appropriate functional unit is available, the file service user may receive an F-READ indication or an F-WRITE indication. The request
becomes the current data transfer. *)

( [isAvailableRead(functionalUnits)]->
  Rout?fsp:FSPrim[isReadInd(fsp)];
   Read[Rout,Rin]
    (normal,functionalUnits,Succ(0),Succ(0),makeEl(fsp,Succ(0)),Succ(0),empty,empty,empty)

  []

  [isAvailableWrite(functionalUnits)]->
  Rout?fsp:FSPrim[isWriteInd(fsp)];
   Write[Rout,Rin]
   (normal,functionalUnits,Succ(0),Succ(0),makeEl(fsp,Succ(0)),Succ(0),empty,empty,empty)
)

>>

(* Upon completion of the previous bulk data transfer the file service user may receive an F-READ or F-WRITE indication (as specified by the process
NormalIdle) or may exit the specification (i.e. initiate an exchange of F-CLOSE primitives). This point in the specification represents the data transfer
idle state. *)
```

Figure 23 (continued) - Responder service definition

```
        ( NormalIdle[Rout,Rin](functionalUnits)

      []

         exit
       )

endproc (* NormalIdle*)
```

(*

process ConsecutiveIdle

This process specifies the allowed behaviour when consecutive overlapped access is in use. This point represents the completion of the exchange of F-OPEN primitives.

*)

```
process ConsecutiveIdle[Rout,Rin](functionalUnits:FUnits,maxRequests:Nat)   :exit:=

(* If the appropriate functional unit is available, the file service user may receive an F-READ indication or an F-WRITE indication. The request
becomes the current data transfer. *)

 ( [isAvailableRead(functionalUnits)]->
   Rout?fsp:FSPrim?tn:Nat[isReadInd(fsp)];
    Read[Rout,Rin]
     (consecutive,functionalUnits,maxRequests,tn,makeEl(fsp,tn),tn,empty,empty,empty)

   []

    [isAvailableWrite(functionalUnits)]->
   Rout?fsp:FSPrim?tn:Nat[isWriteInd(fsp)];
    Write[Rout,Rin]
     (consecutive,functionalUnits,maxRequests,tn,makeEl(fsp,tn),tn,empty,empty,empty)
 )

>>

(* Upon completion of all outstanding data transfers the file service user may exit the specification (i.e. initiate an exchange of F-CLOSE primitives)
or may receive an F-READ or F-WRITE indication (as specified by the process ConsecutiveIdle) . This point in the specification represents the
data transfer idle state for consecutive access. *)

        ( ConsecutiveIdle[Rout,Rin](functionalUnits,maxRequests)

      []

         exit
       )

endproc (* ConsecutiveIdle*)
```

(*

process ConcurrentIdle

This process specifies the allowed behaviour when concurrent overlapped access is in use. This point represents the completion of the exchange of F-OPEN primitives.

Figure 23 (continued) - Responder service definition

```
*)

process  ConcurrentIdle[Rout,Rin](functionalUnits:FUnits,maxRequests:Nat)   :exit:-

(* The file service user may receive an F-READ or F-WRITE indication. *)

    Rout?fsp:FSPrim?tn:Nat[((isAvailableRead(functionalUnits)    and    isReadInd(fsp))    or
(isAvailableWrite(functionalUnits) and isWriteInd(fsp)))-and-(tn-eq-Succ(0))];

(

(* The request for a data transfer becomes the current read or write data transfer as appropriate. Read data transfers will be performed concurrently
with any write data transfers that are subsequently requested. *)

( ( [isAvailableRead(functionalUnits) and isReadInd(fsp)]->
    Read[Rout,Rin](concurrent,functionalUnits,maxRequests,tn,makeEl(fsp,tn),tn,empty,empty,empty)

     >>      ReadIdle[Rout,Rin](functionalUnits,maxRequests)
  )

    []

  ( [isAvailableWrite(functionalUnits) and isWriteInd(fsp)]->
    Write[Rout,Rin](concurrent,functionalUnits,maxRequests,tn,makeEl(fsp,tn),tn,empty,empty,empty)

     >>      WriteIdle[Rout,Rin](functionalUnits,maxRequests)
  )
)

 |||

(* The file service user may receive an F-READ or F-WRITE indication depending on the type of the data transfer that was first requested. *)

  (  [isAvailableWrite(functionalUnits) and isReadInd(fsp)]->
      WriteIdle[Rout,Rin](functionalUnits,maxRequests)

    []

    [isAvailableRead(functionalUnits) and isWriteInd(fsp)]->
     ReadIdle[Rout,Rin](functionalUnits,maxRequests)
  )
)

>>

(* The file service user has the opportunity to exit the specification. Alternatively, further data transfers may be requested, as specified by process
ConcurrentIdle. This represents the data transfer idle state. *)

        (  ConcurrentIdle[Rout,Rin](functionalUnits,maxRequests)

      []

         exit
       )
```

Figure 23 (continued) - Responder service definition

```
where

(*
process  ReadIdle

This process specifies the behaviour for the progress of read data transfers.
*)

 process  ReadIdle[Rout,Rin](functionalUnits:FUnits,maxRequests:Nat)  :exit:=

(* The file service user may receive an F-READ indication, this request will become the current read data transfer. *)

 ( Rout?fsp:FSPrim?tn:Nat[isReadInd(fsp)];
   Read[Rout,Rin]
   (concurrent,functionalUnits,maxRequests,tn,makeEl(fsp,tn),tn,empty,empty,empty)

  >>  ReadIdle[Rout,Rin](functionalUnits,maxRequests)
 )

  []

   exit

 endproc  (*ReadIdle*)

(*
process  WriteIdle

The process specifies the behaviour for the progress of write data transfers.
*)

 process  WriteIdle[Rout,Rin](functionalUnits:FUnits,maxRequests:Nat)  :exit:=

(* The file service user may receive an F-WRITE indication, this request will become the current write data transfer. *)

  ( Rout?fsp:FSPrim?tn:Nat[isWriteInd(fsp)];
    Write[Rout,Rin](concurrent,functionalUnits,maxRequests,tn,makeEl(fsp,tn),tn,empty,empty,empty)

  >>  WriteIdle[Rout,Rin](functionalUnits,maxRequests)
 )

 []

   exit

 endproc  (*WriteIdle*)

endproc  (*ConcurrentIdle*)


(*
process  Read

This process specifies the behaviour for the current read data transfer in the case where there are no outstanding requests
on the requestQ.
*)
```

Figure 23 (continued) - Responder service definition

```
process   Read[Rout,Rin]
            (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

(* An F-DATA request may be issued by the file service user.*)

Rin?fsp:FSPrim[isDataReq(fsp)];
 Read[Rout,Rin]
  (degree,functionalUnits,maxRequests,tnCounter,current,reqOutstanding,reqQ,dataEndQ,xfrEndQ)

[]

(* An F-DATA-END request may be issued by the file service user. The type and transfer number of the current read data transfer is added to the
dataEndQ. *)

Rin?fsp:FSPrim[isDataEndReq(fsp)];
   ReadDataXfrIdle[Rout,Rin]
  (degree,functionalUnits,maxRequests,tnCounter,
  nullEl,reqOutstanding,reqQ,add(current,dataEndQ),xfrEndQ)

[]

(* If overlapped access is in use and the number of outstanding requests has not exceeded the maximum, then the file service user may receive an
F-READ indication. If consecutive access is in use and the Write functional unit has been negotiated then the file service user may receive an F-
WRITE indication. The type and transfer number of the request is added to the requestQ and the number of outstanding requests is incremented. The
subsequent behaviour is dependant on the type of the request. *)

[not(isNormal(degree)) and (reqOutstanding lt maxRequests)]->
(   Rout?fsp:FSPrim?tn:Nat[isReadInd(fsp) and (tn eq (tnCounter+Succ(0)))];
    ReadRead[Rout,Rin]
    (degree,functionalUnits,maxRequests,tn,
    current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)

  []

  [isAvailableWrite(functionalUnits) and isConsecutive(degree)]->
  Rout?fsp:FSPrim?tn:Nat[isWriteInd(fsp) and (tn eq (tnCounter+Succ(0)))];
    ReadWrite[Rout,Rin]
    (degree,functionalUnits,maxRequests,tn,
    current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)
)

[]

(* The file service user may receive an F-TRANSFER-END indication for the data transfer at the head of the dataEndQ. The data transfer is
removed from the dataEndQ and added to the xfrEndQ. *)

[not(isEmpty(dataEndQ))]->
Rout?fsp:FSPrim?tn:Nat[isXfrEndReadInd(fsp) and (tn eq number(head(dataEndQ)))];
 Read[Rout,Rin]
  (degree,functionalUnits,maxRequests,tnCounter,
  current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

[]
```

Figure 23 (continued) - Responder service definition

```
(* An F-TRANSFER-END response may be issued by the file service user for the data transfer at the head of the xfrEndQ. The data transfer is
removed from the xfrEndQ. *)

 [not(isEmpty(xfrEndQ))]->
 Rin?fsp:FSPrim?tn:Nat[isXfrEndReadResp(fsp) and (tn eq number(head(xfrEndQ)))];
  Read[Rout,Rin]
   (degree,functionalUnits,maxRequests,tnCounter,
   current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))

[]

(* The initiating file service user may receive an F-CANCEL indication for the data transfer at the head of the dataEndQ. If this is empty then the file
service user may receive an F-CANCEL indication for the current data transfer. If the dataEndQ is empty then an F-CANCEL request may be issued
by the file service user for the current data transfer. *)

Interrupt[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endproc (* Read *)


(*
process ReadRead

This process specifies the behaviour in the case where the current transfer is a read and the next requested data transfer
is also a read. The behaviour is only allowed if consecutive or concurrent access is in use. *)

process  ReadRead[Rout,Rin](degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

(* An F-DATA request may be issued by the file service user for the current data transfer. *)

 Rin?fsp:FSPrim[isDataReq(fsp)];
   ReadRead[Rout,Rin]
   (degree,functionalUnits,maxRequests,tnCounter,current,reqOutstanding,reqQ,dataEndQ,xfrEndQ)

[]

(* An F-DATA-END request may be issued by the file service user for the current data transfer. The current data transfer is added to the
dataEndQ; the request at the head of the dataEndQ becomes the current data transfer. The subsequent behaviour is determined on the basis
of the request that will now be at the head of the requestQ. *)

 Rin?fsp:FSPrim[isDataEndReq(fsp)];
 ( [isWriteInd(prim(head(tailOf(reqQ))))]->
    ReadWrite[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,
     head(reqQ),reqOutstanding,tailOf(reqQ),add(current,dataEndQ),xfrEndQ)

   []

   [isReadInd(prim(head(tailOf(reqQ))))]->
    ReadRead[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,
     head(reqQ),reqOutstanding,tailOf(reqQ),add(current,dataEndQ),xfrEndQ)

   []

   [isEmpty(tailOf(reqQ))]->
```

Figure 23 (continued) - Responder service definition

```
   Read[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,
    head(reqQ),reqOutstanding,tailOf(reqQ),add(current,dataEndQ),xfrEndQ)
 )

[]

(* Further requests for read data transfers may be requested if the number of requests outstanding does not exceed the maximum. Write requests may
also be requested only if consecutive access is in use and the write functional unit is available. The type and transfer number of the request is added
to the requestQ*)

 [reqOutstanding lt maxRequests] ->
 ( Rout?fsp:FSPrim?tn:Nat[isReadInd(fsp) and (tn eq (tnCounter+Succ(0)))];
    ReadRead[Rout,Rin]
    (degree,functionalUnits,maxRequests,tn,
    current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)

  []

   [isAvailableWrite(functionalUnits) and isConsecutive(degree)]->
   Rout?fsp:FSPrim?tn:Nat[isWriteInd(fsp) and (tn eq (tnCounter+Succ(0)))];
     ReadRead[Rout,Rin]
     (degree,functionalUnits,maxRequests,tn,
    current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)
 )

[]

(* The file service user may receive an F-TRANSFER-END request for the data transfer at the head of the dataEndQ. The head of the dataEndQ
is removed and added to the xfrEndQ. *)

 [not(isEmpty(dataEndQ))]->
 Rout?fsp:FSPrim?tn:Nat[isXfrEndReadInd(fsp) and (tn eq number(head(dataEndQ)))];
  ReadRead[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,
   current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

[]

(* An F-TRANSFER-END response may be issued by the file service user for the data transfer at the head of the xfrEndQ. The head of the xfrEndQ
is removed.*)

 [not(isEmpty(xfrEndQ))]->
 Rin?fsp:FSPrim?tn:Nat[isXfrEndReadResp(fsp) and (tn eq number(head(xfrEndQ)))];
  ReadRead[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,
   current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))

[]

(* The initiating file service user may receive an F-CANCEL for the data transfer at the head of the dataEndQ. If this is empty then the file service
user may receive an F-CANCEL indication for the current read data transfer. If the dataEndQ is empty then an F-CANCEL request may be issued
by the file service user for the current read data transfer. *)

Interrupt[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endprog (* ReadRead *)
```

Figure 23 (continued) - Responder service definition

```
(*
process  ReadWrite

This process specifies the behaviour in the case where the current transfer is a read and the next requested data transfer
is a write. The behaviour is only allowed if consecutive access is in use. *)

process  ReadWrite[Rout,Rin](degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

[not(isNullEl(current))]->

(* An F-DATA request may issued by the file service user for the current data transfer. *)


( Rin?fsp:FSPrim[isDataReq(fsp)];
    ReadWrite[Rout,Rin]
   (degree,functionalUnits,maxRequests,tnCounter,current,reqOutstanding,reqQ,dataEndQ,xfrEndQ)

  []

(* An F-DATA-END request may be issued by the file service user for the current read data transfer. The current read data transfer is added to the
dataEndQ and the current data transfer becomes null. *)

  Rin?fsp:FSPrim[isDataEndReq(fsp)];
   ReadWrite[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,
    nullEl,reqOutstanding,reqQ,add(current,dataEndQ),xfrEndQ)
)

[]

(* The file service user may receive an F-TRANSFER-END indication for the (read) data transfer that is at the head of the dataEndQ. The head of
the dataEndQ is removed and added to the xfrEndQ.

If the dataEndQ is empty and there is no current data transfer, then the head of the requestQ becomes the current data transfer. The
subsequent behaviour is determined on the basis of the type of the request that is now at the head of the requestQ. If however, there is a current
data transfer, or the dataEndQ is not empty, then the behaviour is specified by process ReadWrite. *)

[not(isEmpty(dataEndQ))]->
Rout?fsp:FSPrim?tn:Nat[isXfrEndReadInd(fsp) and (tn eq number(head(dataEndQ)))];
( [isEmpty(tailOf(dataEndQ)) and isNullEl(current)]->
  ( [isReadInd(prim(head(tailOf(reqQ))))]->
     WriteRead[Rout,Rin]
      (degree,functionalUnits,maxRequests,tnCounter,
     head(reqQ),reqOutstanding,tailOf(reqQ),tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

    []

    [isWriteInd(prim(head(tailOf(reqQ))))]->
      WriteWrite[Rout,Rin]
      (degree,functionalUnits,maxRequests,tnCounter,
     head(reqQ),reqOutstanding,tailOf(reqQ),tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

    []

    [isEmpty(tailOf(reqQ))]->
```

Figure 23 (continued) - Responder service definition

```
    Write[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,
     head(reqQ),reqOutstanding,tailOf(reqQ),tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))
  )

  []

  [not(isEmpty(tailOf(dataEndQ))) or not(isNullEl(current))]->
    ReadWrite[Rout,Rin]
     (degree,functionalUnits,maxRequests,tnCounter,
    current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))
)

[]

(* The file service user may receive an F-WRITE or F-READ indication. The type and transfer number of the request is added to the requestQ *)

[reqOutstanding lt maxRequests]->
Rout?fsp:FSPrim?tn:Nat[(isReadInd(fsp) or isWriteInd(fsp)) and (tn eq (tnCounter+Succ(0)))];
  ReadWrite[Rout,Rin]
   (degree,functionalUnits,maxRequests,tn,
  current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)

[]

(* An F-TRANSFER-END response may be issued by the file service user for the data transfer at the head of the xfrEndQ. The head of the xfrEndQ is then removed. *)

[not(isEmpty(xfrEndQ))]->
Rin?fsp:FSPrim?tn:Nat[isXfrEndReadResp(fsp) and (tn eq number(head(xfrEndQ)))];
  ReadWrite[Rout,Rin]
   (degree,functionalUnits,maxRequests,tnCounter,
  current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))

[]

(* If the dataEndQ is empty then the file service user may receive an F-CANCEL indication for the current read data transfer and an F-CANCEL request may be issued by the file service user for the current read data transfer. If the dataEndQ is not empty then the file service user may receive an F-CANCEL indication for the read data transfer at the head of the dataEndQ. *)

Interrupt[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endproc (* ReadWrite *)


(*
process ReadDataXfrIdle

The process specifies the behaviour where there are no data transfer requests on the requestQ and there is no current data transfer in progress. *)

process ReadDataXfrIdle[Rout,Rin](degree:Overlap,functionalUnits:FUnits,
          maxRequests,tnCounter:Nat,current:Element,reqOutstanding:Nat,reqQ,dataEndQ,
          xfrEndQ:Queue) :exit:=

(* The file service user may receive an F-TRANSFER-END indication for the (read) data transfer at the head of the dataEndQ. The head of the dataEndQ is removed and added to the xfrEndQ. The subsequent behaviour is dependant on whether the dataEndQ is empty. *)
```

Figure 23 (continued) - Responder service definition

```
Rout?fsp:FSPrim?tn:Nat[isXfrEndReadInd(fsp) and (tn eq number(head(dataEndQ)))];
( [isEmpty(tailOf(dataEndQ))]->
    ReadXfrEndIdle[Rout,Rin]
    (degree,functionalUnits,maxRequests,tnCounter,
   current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

 []

 [not(isEmpty(tailOf(dataEndQ)))]->
    ReadDataXfrIdle[Rout,Rin]
    (degree,functionalUnits,maxRequests,tnCounter,
      current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))
)

[]

(* If overlapped access is in use then the file service user may receive an F-READ indication. If consecutive access is in use then the user may
receive an F-WRITE indication. A request for a read data transfer will become the current data transfer with subsequent behaviour specified in
process Read; a request for a write data transfer is added to the requestQ and subsequent behaviour is specified by process ReadWrite. *)

[not(isNormal(degree)) and (reqOutstanding lt maxRequests)]->
( Rout?fsp:FSPrim?tn:Nat[isReadInd(fsp) and (tn eq (tnCounter+Succ(0)))];
   Read[Rout,Rin]
    (degree,functionalUnits,maxRequests,tn,
    makeEl(fsp,tn),Succ(reqOutstanding),reqQ,dataEndQ,xfrEndQ)

 []

 [isAvailableWrite(functionalUnits) and isConsecutive(degree)]->
 Rout?fsp:FSPrim?tn:Nat[isWriteInd(fsp) and (tn eq (tnCounter+Succ(0)))];
    ReadWrite[Rout,Rin]
    (degree,functionalUnits,maxRequests,tn,
   current,reqOutstanding,add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)
)

[]

(* An F-TRANSFER-END response may be issued by the file service user for the data transfer at the head of the xfrEndQ. The data transfer is
removed from the head of the xfrEndQ. *)

[not(isEmpty(xfrEndQ))]->
Rin?fsp:FSPrim?tn:Nat[isXfrEndReadResp(fsp) and (tn eq number(head(xfrEndQ)))];
   ReadDataXfrIdle[Rout,Rin]
   (degree,functionalUnits,maxRequests,tnCounter,
  current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))

[]

(* The file service user may receive an F-CANCEL indication for the read data transfer at the head of the dataEndQ. *)

Interrupt[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endprog (* ReadDataXfrIdle *)
```

Figure 23 (continued) - Responder service definition

```
(*
process   ReadXfrEndIdle

The process specifies the behaviour when there is no current data transfer and the dataEndQ is empty. *)

process     ReadXfrEndIdle[Rout,Rin]
              (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
              current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:-

(* An F-TRANSFER-END response may be issued by the file service user for the data transfer at the head of the xfrEndQ. The data transfer is
removed from the xfrEndQ. If the xfrEndQ is empty then the process may exit. *)

 Rin?fsp:FSPrim?tn:Nat[isXfrEndReadResp(fsp) and (tn eq number(head(xfrEndQ)))];
 ([isEmpty(tailOf(xfrEndQ))]->
  exit

 []

  [not(isEmpty(tailOf(xfrEndQ)))]->
    ReadXfrEndIdle[Rout,Rin]
    (degree,functionalUnits,maxRequests,tnCounter,
    nullEl,pred(reqOutstanding),empty,empty,tailOf(xfrEndQ))
 )

[]

(* If overlapped access is in use then the file service user may receive an F-READ indication. If consecutive access is in use then the user may
receive an F-WRITE indication. A request for a read or write data transfer will become the current data transfer. The subsequent behaviour is
dependant on the type of the request. *)

 [not(isNormal(degree)) and (reqOutstanding lt maxRequests)]->
 ( Rout?fsp:FSPrim?tn:Nat[isReadInd(fsp) and (tn eq (tnCounter+Succ(0)))];
    Read[Rout,Rin]
     (degree,functionalUnits,maxRequests,tn,
    makeEl(fsp,tn),Succ(reqOutstanding),reqQ,dataEndQ,xfrEndQ)

  []

  [isAvailableWrite(functionalUnits) and isConsecutive(degree)]->
  Rout?fsp:FSPrim?tn:Nat[isWriteInd(fsp) and (tn eq (tnCounter+Succ(0)))];
    Write[Rout,Rin]
     (degree,functionalUnits,maxRequests,tn,
    makeEl(fsp,tn),Succ(reqOutstanding),reqQ,dataEndQ,xfrEndQ)
 )

endproc (* ReadXfrEndIdle *)


(*
process  Write

This process specifies the behaviour for the current write data transfer in the case where there are no outstanding requests
on the requestQ.
*)
```

Figure 23 (continued) - Responder service definition

```
process  Write[Rout,Rin]
            (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

[not(isNullEl(current))]->

(* The file service user may receive an F-DATA indication. *)

( Rout?fsp:FSPrim[isDataInd(fsp)];
   Write[Rout,Rin]
  (degree,functionalUnits,maxRequests,tnCounter,current,reqOutstanding,reqQ,dataEndQ,xfrEndQ)

  []

(* The file service user may receive an F-DATA-END indication. The current data transfer is added to the dataEndQ and the current data transfe
becomes null. *)

  Rout?fsp:FSPrim[isDataEndInd(fsp)];
   Write[Rout,Rin]
    (degree,functionalUnits,maxRequests,tnCounter,
    nullEl,reqOutstanding,reqQ,add(current,dataEndQ),xfrEndQ)
)

[]

(* The file service user may receive an F-TRANSFER-END indication for the (write) data transfer at the head of the dataEndQ. The data transfe
is removed from the dataEndQ and added to the xfrEndQ. *)

[not(isEmpty(dataEndQ))]->
Rout?fsp:FSPrim?tn:Nat[isXfrEndWriteInd(fsp) and (tn eq number(head(dataEndQ)))];
   WriteDataXfrIdle[Rout,Rin]
  (degree,functionalUnits,maxRequests,tnCounter,
  current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

[]

(* If overlapped access is in use and the number of outstanding requests has not exceeded the maximum, then the file service user may receive an
F-WRITE indication. If consecutive access is in use and the Write functional unit has been negotiated then the file service user may receive an F-
READ indication. The type and transfer number of the request is added to the requestQ, and the number of outstanding requests is incremented.
The subsequent behaviour is dependant on the type of the request. *)

[not(isNormal(degree)) and (reqOutstanding lt maxRequests)]->
( [isAvailableRead(functionalUnits) and isConsecutive(degree)]->
  Rout?fsp:FSPrim?tn:Nat[isReadInd(fsp) and (tn eq (tnCounter+Succ(0)))];
    WriteRead[Rout,Rin]
    (degree,functionalUnits,maxRequests,tn,
   current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)

  []

  Rout?fsp:FSPrim?tn:Nat[isWriteInd(fsp) and (tn eq (tnCounter+Succ(0)))];
    WriteWrite[Rout,Rin]
    (degree,functionalUnits,maxRequests,tn,
   current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)
)
```

Figure 23 (continued) - Responder service definition

```
[]

(* An F-TRANSFER-END response may be issued by the file service user for the data transfer at the head of the xfrEndQ. The data transfer is
removed from the xfrEndQ. *)

 [not(isEmpty(xfrEndQ))]->
 Rin?fsp:FSPrim?tn:Nat[((isXfrEndReadResp(fsp) and isReadInd(prim(head(xfrEndQ))))
   or (isXfrEndWriteResp(fsp) and isWriteInd(prim(head(xfrEndQ)))))
   and (tn eq number(head(xfrEndQ)))];
  Write[Rout,Rin]
   (degree,functionalUnits,maxRequests,tnCounter,
   current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))

[]

(* The file service user may receive an F-CANCEL indication for the current write data transfer. An F-CANCEL request may be issued by the file
service user for the first write data transfer on the xfrEndQ. If there is no write data transfer on the xfrEndQ then an F-CANCEL request may be
issued by the file service user for the data transfer at the head of the dataEndQ, or if this is empty, the current data transfer. *)

Interrupt[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endprog (* Write *)


(*
process WriteWrite

This process specifies the behaviour in the case where the current data transfer is a write and the next requested data transfer
is also a write. The behaviour is only allowed if consecutive or concurrent access is in use. *)

process  WriteWrite[Rout,Rin]
           (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
           current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

 [not(isNullEl(current))]->
 (

(* The file service user may receive an F-DATA indication for the current data transfer. *)

   Rout?fsp:FSPrim[isDataInd(fsp)];
     WriteWrite[Rout,Rin]
    (degree,functionalUnits,maxRequests,tnCounter,current,reqOutstanding,reqQ,dataEndQ,xfrEndQ)

   []

(* The file service user may receive an F-DATA-END indication for the current data transfer. The current data transfer is added to the dataEndQ
and the current data transfer becomes null. *)

   Rout?fsp:FSPrim[isDataEndInd(fsp)];
    WriteWrite[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,
     nullEl,reqOutstanding,reqQ,add(current,dataEndQ),xfrEndQ)
 )

[]
```

Figure 23 (continued) - Responder service definition

```
(* The file service user may receive an F-TRANSFER-END indication for the data transfer at the head of the dataEndQ. The head of the
dataEndQ is removed and added to the xfrEndQ. The subsequent behaviour is dependant on the request at the head of the requestQ; the
current data transfer remains null. *)

[not(isEmpty(dataEndQ))]->
Rout?fsp:FSPrim?tn:Nat[isXfrEndWriteInd(fsp) and (tn eq number(head(dataEndQ)))];
( [isEmpty(tailOf(reqQ))]->
   Write[Rout,Rin]
    (degree,functionalUnits,maxRequests,tnCounter,
   head(reqQ),reqOutstanding,tailOf(reqQ),tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

  []

  [isReadInd(prim(head(tailOf(reqQ))))]->
    WriteRead[Rout,Rin]
    (degree,functionalUnits,maxRequests,tnCounter,
   head(reqQ),reqOutstanding,tailOf(reqQ),tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

  []

  [isWriteInd(prim(head(tailOf(reqQ))))]->
    WriteWrite[Rout,Rin]
    (degree,functionalUnits,maxRequests,tnCounter,
   head(reqQ),reqOutstanding,tailOf(reqQ),tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))
)

[]

(* Further requests for write data transfers may be requested if the number of requests outstanding does not exceed the maximum. Read requests may
be requested only if consecutive access is in use and the read functional unit is available. The type and transfer number of the request is added to
the requestQ *)

[reqOutstanding lt maxRequests]->
( [isAvailableRead(functionalUnits) and isConsecutive(degree)]->
  Rout?fsp:FSPrim?tn:Nat[isReadInd(fsp) and (tn eq (tnCounter+Succ(0)))];
    WriteWrite[Rout,Rin]
    (degree,functionalUnits,maxRequests,tn,
   current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)



  []

  Rout?fsp:FSPrim?tn:Nat[isWriteInd(fsp) and (tn eq (tnCounter+Succ(0)))];
    WriteWrite[Rout,Rin]
    (degree,functionalUnits,maxRequests,tn,
   current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)
)

[]

(* An F-TRANSFER-END response may be issued by the file service user for the data transfer at the head of the xfrEndQ. The data transfer is
removed from the xfrEndQ. *)

[not(isEmpty(xfrEndQ))]->
```

Figure 23 (continued) - Responder service definition

```
Rin?fsp:FSPrim?tn:Nat[((isXfrEndReadResp(fsp) and isReadInd(prim(head(xfrEndQ))))
  or (isXfrEndWriteResp(fsp) and isWriteInd(prim(head(xfrEndQ)))))
  and (tn eq number(head(xfrEndQ)))];
  WriteWrite[Rout,Rin]
  (degree,functionalUnits,maxRequests,tnCounter,
  current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))

[]
```

(* The file service user may receive an F-CANCEL indication for the **current** write data transfer. An F-CANCEL request may be issued by the file service user for the first write data transfer on the **xfrEndQ**. If there is no write data transfer on the **xfrEndQ** then an F-CANCEL request may be issued by the file service user for the data transfer at the head of the **dataEndQ**, or if this is empty, the **current** data transfer. *)

```
Interrupt[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endproc (* WriteWrite *)
```

(*

process WriteRead

This process specifies the behaviour in the case where the **current** data transfer is a write and the next requested data transfer on the **requestQ** is a read. *)

```
process   WriteRead[Rout,Rin]
            (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=

[not(isNullEl(current))]->
(
```

(* The file service user may receive an F-DATA indication. *)

```
Rout?fsp:FSPrim[isDataInd(fsp)];
    WriteRead[Rout,Rin]
   (degree,functionalUnits,maxRequests,tnCounter,current,reqOutstanding,reqQ,dataEndQ,xfrEndQ)

  []
```

(* The file service user may receive an F-DATA-END indication for the **current** data transfer. The data transfer is added to the **dataEndQ** and the **current** data transfer becomes null. *)

```
  Rout?fsp:FSPrim[isDataEndInd(fsp)];
    WriteRead[Rout,Rin]
    (degree,functionalUnits,maxRequests,tnCounter,
    nullEl,reqOutstanding,reqQ,add(current,dataEndQ),xfrEndQ)
)

[]
```

(* The file service user may receive an F-TRANSFER-END indication for the data transfer at the head of the **dataEndQ**. The data transfer is removed from the **dataEndQ** and added to the **xfrEndQ**. *)

```
[not(isEmpty(dataEndQ))]->
Rout?fsp:FSPrim?tn:Nat[isXfrEndWriteInd(fsp) and (tn eq number(head(dataEndQ)))];
 WriteRead[Rout,Rin]
```

Figure 23 (continued) - Responder service definition

```
  (degree,functionalUnits,maxRequests,tnCounter,
  current,reqOutstanding,reqQ,tailOf(dataEndQ),add(head(dataEndQ),xfrEndQ))

[]

(* An F-TRANSFER-END response may be issued by the file service user for the data transfer at the head of the xfrEndQ. The data transfer is
removed from the xfrEndQ. If the xfrEndQ and the dataEndQ are empty, and the current data transfer is null, then the data transfer at the head of
the requestQ becomes the current data transfer and is removed from the requestQ; the subsequent behaviour is dependant on the type of the
data transfer that is now at the head of the requestQ. *)

[not(isEmpty(xfrEndQ))]->
Rin?fsp:FSPrim?tn:Nat[((isXfrEndReadResp(fsp) and isReadInd(prim(head(xfrEndQ))))
  or (isXfrEndWriteResp(fsp) and isWriteInd(prim(head(xfrEndQ)))))
  and (tn eq number(head(xfrEndQ)))];
( [isEmpty(tailOf(xfrEndQ)) and isEmpty(dataEndQ) and isNullEl(current)]->
  ( [isEmpty(tailOf(reqQ))]->
    Read[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,
     head(reqQ),pred(reqOutstanding),tailOf(reqQ),dataEndQ,tailOf(xfrEndQ))

    []

    [isReadInd(prim(head(tailOf(reqQ))))]->
     ReadRead[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,
      head(reqQ),pred(reqOutstanding),tailOf(reqQ),dataEndQ,tailOf(xfrEndQ))

    []

    [isWriteInd(prim(head(tailOf(reqQ))))]->
      ReadWrite[Rout,Rin]
       (degree,functionalUnits,maxRequests,tnCounter,
      head(reqQ),pred(reqOutstanding),tailOf(reqQ),dataEndQ,tailOf(xfrEndQ))

  )

 []

  [not(isEmpty(tailOf(xfrEndQ))) or not(isEmpty(dataEndQ)) or not(isNullEl(current))]->
    WriteRead[Rout,Rin]
    (degree,functionalUnits,maxRequests,tnCounter,
   current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))
)

[]

(* The file service user may receive an F-WRITE or F-READ indication. The type and transfer number of the request is added to the requestQ. *)

[reqOutstanding lt maxRequests]->
Rout?fsp:FSPrim?tn:Nat[(isReadInd(fsp) or isWriteInd(fsp)) and (tn eq (tnCounter+Succ(0)))];
  WriteRead[Rout,Rin]
   (degree,functionalUnits,maxRequests,tn,
  current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)

[]

(* The file service user may receive an F-CANCEL indication for the current write data transfer. An F-CANCEL request may be issued by the file
```

Figure 23 (continued) - Responder service definition

service user for the first write data transfer on the **xfrEndQ**. If there is no write data transfer on the **xfrEndQ** then an F-CANCEL request may be issued by the file service user for the data transfer at the head of the **dataEndQ**, or if this is empty, the **current** data transfer. *)

```
Interrupt[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endproc (* WriteRead *)
```

(*

process WriteDataXfrIdle

This process specifies the behaviour when the **current** data transfer is null, the **dataEndQ** is empty and the **xfrEndQ** is not empty. *)

```
process     WriteDataXfrIdle[Rout,Rin]
             (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
             current:Element,reqOutstanding:Nat,reqQ,dataEndQ,xfrEndQ:Queue) :exit:=
```

(* An F-TRANSFER-END response may be issued by the file service user for the data transfer at the head of the xfrEndQ. The data transfer is removed from the xfrEndQ. The subsequent behaviour is dependant on whether or not the xfrEndQ is now empty. *)

```
(( [isReadInd(prim(head(xfrEndQ)))]->
   Rin?fsp:FSPrim?tn:Nat[isXfrEndReadResp(fsp) and (tn eq number(head(xfrEndQ)))];
    exit

  []

   [isWriteInd(prim(head(xfrEndQ)))]->
   Rin?fsp:FSPrim?tn:Nat[isXfrEndWriteResp(fsp) and (tn eq number(head(xfrEndQ)))];
    exit
 )

 >>

 ( [isEmpty(tailOf(xfrEndQ))]->
    exit

  []

   [not(isEmpty(tailOf(xfrEndQ)))]->
     WriteDataXfrIdle[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,
     current,pred(reqOutstanding),reqQ,dataEndQ,tailOf(xfrEndQ))
 )
)

[]
```

(* Further requests for write data transfers may be requested if consecutive or concurrent access is in use and the number of requests outstanding does not exceed the maximum. Read requests may be requested only if consecutive access is in use and the read functional unit is available. The subsequent behaviour is dependant on the type of the request. *)

```
[not(isNormal(degree)) and (reqOutstanding lt maxRequests)]->
( [isAvailableRead(functionalUnits) and isConsecutive(degree)]->
  Rout?fsp:FSPrim?tn:Nat[isReadInd(fsp) and (tn eq (tnCounter+Succ(0)))];
    WriteRead[Rout,Rin]
    (degree,functionalUnits,maxRequests,tn,
```

Figure 23 (continued) - Responder service definition

```
    current,Succ(reqOutstanding),add(makeEl(fsp,tn),reqQ),dataEndQ,xfrEndQ)

  []

  Rout?fsp:FSPrim?tn:Nat[isWriteInd(fsp) and (tn eq (tnCounter+Succ(0)))];
    Write[Rout,Rin]
     (degree,functionalUnits,maxRequests,tn,
    makeEl(fsp,tn),Succ(reqOutstanding),reqQ,dataEndQ,xfrEndQ)
)

[]

(* An F-CANCEL request may be issued by the file service user for the first write request on the xfrEndQ. *)

Interrupt[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

endproc (* WriteDataXfrIdle *)


(*
process Interrupt

This process specifies the behaviour when a data transfer request is cancelled. *)

process   Interrupt[Rout,Rin]
            (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

  InitiatorCancel[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

[]

  ResponderCancel[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

where


(*
process InitiatorCancel

This process specifies the predicates and behaviour for the cancellation of a data transfer by the initiating file service user.
*)

  process   InitiatorCancel[Rout,Rin]
            (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

 [isAvailableRead(functionalUnits)]->
  ReadCancel[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

 []

 [isAvailableWrite(functionalUnits)]->
  WriteCancel[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

 where
```

Figure 23 (continued) - Responder service definition

```
(*
process  ReadCancel

This process specifies the predicates and the behaviour for the cancellation of a read data transfer by the file service user.
*)

   process  ReadCancel[Rout,Rin](degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

(* The file service user may receive an F-CANCEL indication for the read data transfer at the head of the dataEndQ. *)

   [isReadInd(prim(head(dataEndQ)))]->
   Rout?fsp:FSPrim?tn:Nat[isCancelReadInd(fsp) and (tn eq number(head(dataEndQ)))];
    CancelPending[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),xfrEndQ)

   []

(* The file service user may receive an F-CANCEL indication for the current data transfer is the xfrEndQ is empty. *)

   [isEmpty(xfrEndQ) and isReadInd(prim(current))]->
   Rout?fsp:FSPrim?tn:Nat[isCancelReadInd(fsp) and (tn eq number(current))];
    CancelPending[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),xfrEndQ)

  endproc (* ReadCancel *)


(*
process  WriteCancel

This process specifies the predicates and the behaviour for the cancellation of a write data transfer by the file service user.
*)

   process  WriteCancel[Rout,Rin]
            (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

(* The file service user may receive an F-CANCEL indication for the current write data transfer if the dataEndQ is empty. *)

   [isEmpty(dataEndQ) and isWriteInd(prim(current))]->
   Rout?fsp:FSPrim?tn:Nat[isCancelWriteInd(fsp) and (tn eq number(current))];
    CancelPending[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),xfrEndQ)

  endproc (* WriteCancel *)


(*
process  CancelPending

This process specifies the completion of a data transfer cancellation.
*)

   process  CancelPending[Rout,Rin]
            (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            cancelData:Element,xfrEndQ:Queue) :exit:=
```

Figure 23 (continued) - Responder service definition

```
(* An F-CANCEL response may be issued for the data transfer that is being cancelled. *)

  ( [isCancelReadInd(prim(cancelData))]->
    Rin?fsp:FSPrim?tn:Nat[isCancelReadResp(fsp) and (tn eq number(cancelData))];
     exit

   []

    [isCancelWriteInd(prim(cancelData))]->
    Rin?fsp:FSPrim?tn:Nat[isCancelWriteResp(fsp) and (tn eq number(cancelData))];
     exit
  )

  >>

(* If there are any data transfers on the xfrEndQ then these will be read data transfers, subsequent behaviour is therefore as specified in process
ReadXfrEndIdle, otherwise the file service user may exit the specification. *)

   ( [isEmpty(xfrEndQ)]->
      exit

    []

     [not(isEmpty(xfrEndQ))]->
       ReadXfrEndIdle[Rout,Rin]
      (degree,functionalUnits,maxRequests,tnCounter,nullEl,0,empty,empty,xfrEndQ)
   )

 endproc (* CancelPending *)

endproc (* InitiatorCancel *)


(*
process  ResponderCancel

This process specifies the predicates and behaviour for the cancellation of a data transfer by the responding file service user.
*)

  process  ResponderCancel[Rout,Rin]
           (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
           current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

 [isAvailableRead(functionalUnits)]->
  ReadCancel[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

 []

 [isAvailableWrite(functionalUnits)]->
  WriteCancel[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,current,dataEndQ,xfrEndQ)

 where


(*
process  ReadCancel
```

Figure 23 (continued) - Responder service definition

This process specifies the predicates and the behaviour for the cancellation of a read data transfer by the responding file service user. *)

```
   process  ReadCancel[Rout,Rin]
            (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
            current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

(* An F-CANCEL request may be issued by the file service user for the current data transfer if the dataEndQ is empty. *)

 [isEmpty(dataEndQ) and isReadInd(prim(current))]->
 Rin?fsp:FSPrim?tn:Nat[isCancelReadReq(fsp) and (tn eq number(current))];
  CancelPending[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),xfrEndQ)

 endproc (* ReadCancel *)


(*
process  WriteCancel

This process specifies the predicates and the behaviour for the cancellation of a write data transfer by the responding file service user. *)

   process  WriteCancel[Rout,Rin](degree:Overlap,functionalUnits:FUnits,maxRequests:Nat,
            tnCounter:Nat,current:Element,dataEndQ,xfrEndQ:Queue) :exit:=

(* If there are no write data transfers on the xfrEndQ then an F-CANCEL request may be issued by the file service user for the current write data transfer, if the dataEndQ is empty, or the write data transfer at the head of the dataEndQ. *)

 [isNull(prim(firstWriteInd(xfrEndQ)))]->
 ( [isEmpty(dataEndQ) and isWriteInd(prim(current))]->
  Rin?fsp:FSPrim?tn:Nat[isCancelWriteReq(fsp) and (tn eq number(current))];
   CancelPending[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),xfrEndQ)

  []

   [isNullEl(current) and isWriteInd(prim(head(dataEndQ)))]->
   Rin?fsp:FSPrim?tn:Nat[isCancelWriteReq(fsp) and (tn eq number(head(dataEndQ)))];
    CancelPending[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),xfrEndQ)
 )

 []

(* An F-CANCEL request may be issued by the file service user for the first write data transfer on the xfrEndQ. *)

  [not(isNullEl(firstWriteInd(xfrEndQ)))]->
  Rin?fsp:FSPrim?tn:Nat[isCancelWriteReq(fsp) and (tn eq number(firstWriteInd(xfrEndQ)))];
   CancelPending[Rout,Rin](degree,functionalUnits,maxRequests,tnCounter,makeEl(fsp,tn),removeReqs(xfrEndQ))

   endproc (* WriteCancel *)


(*
process  CancelPending

This process specifies the completion of a data transfer cancellation.
```

Figure 23 (continued) - Responder service definition

```
*)

 process CancelPending[Rout,Rin]
           (degree:Overlap,functionalUnits:FUnits,maxRequests,tnCounter:Nat,
           cancelData:Element,xfrEndQ:Queue) :exit:=

(* An F-CANCEL confirm may be issued for the data transfer that is being cancelled. *)

  ( [isCancelReadReq(prim(cancelData))]->
    Rout?fsp:FSPrim?tn:Nat[isCancelReadConf(fsp) and (tn eq number(cancelData))];
     exit

   []

    [isCancelWriteReq(prim(cancelData))]->
    Rout?fsp:FSPrim?tn:Nat[isCancelWriteConf(fsp) and (tn eq number(cancelData))];
     exit
   )

   >>

(* If there are any data transfers on the xfrEndQ then these will be read data transfers, subsequent behaviour is therefore as specified in process
ReadXfrEndIdle, otherwise the file service user may exit the specification. *)

   ( [isEmpty(xfrEndQ)]->
      exit

    []

     [not(isEmpty(xfrEndQ))]->
       ReadXfrEndIdle[Rout,Rin]
       (degree,functionalUnits,maxRequests,tnCounter,
       nullEl,size(xfrEndQ),empty,empty,removeLast(xfrEndQ))
   )

  endproc (* CancelPending *)

  endproc (* ResponderCancel *)

 endproc (* Interrupt *)

endspec (* Responder_External_Service *)
```

Figure 23 (continued) - Responder service definition

UDC 681.3 : 621.39

Descriptors: data processing, information interchange, network interconnection, open systems interconnection, communication procedure, files, transfer, access, management.

Price based on 80 pages

8571 PT*3 88 ■ 4851903 0536297 465 ■

INTERNATIONAL STANDARD

ISO 8571-3

First edition
1988-10-01

AMENDMENT 1
1992-12-15

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

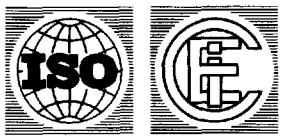
Part 3 :
File Service Definition

AMENDMENT 1 : Filestore Management

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

Partie 3: Définition du service de fichiers

AMENDEMENT 1 : Gestion du système de fichiers



Reference number
ISO 8571-3:1988/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 8571-3:1988 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology.*

ISO 8571-3 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*

© ISO/IEC 1992

All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

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

Part 3 :
File Service Definition

AMENDMENT 1 : Filestore Management

NOTE – This amendment has additional subclauses and tables to ISO 8571:1988 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.

0 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.

Section one: General

6.2 File service levels

(amend 1st paragraph's item (a), page 3)

a) the external file service (EFS), in which the user states its FTAM quality of service requirements, but has no awareness of error recovery, delegating such considerations to the service provider. Transfer of file data and other operations on the filestore are modeled in the external file service as a series of error-free operations. Thus within the external file service there is no visibility of recoverable errors or the error recovery actions;

6.3 Regimes of the file service

(amend 1st paragraph, page 4)

Four types of file service regime are defined:

a) the FTAM regime, existing while the application association is used for the FTAM protocol,with which a group of file object complete pathnames is associated;

b) the object selection regime during which a particular object is associated with the FTAM regime;

c) the file open regime during which a particular set of processing mode, presentation contexts and concurrency controls is in operation;

d) the data transfer regime during which a particular bulk data transfer specification and direction of transfer are in force.

(amend 3rd paragraph, page 4)

The file service provides for:

e) a sequence of object selection regimes in an FTAM regime;

f) a sequence of file open regimes in an object selection regime;

g) a sequence of data transfer regimes within a file open regime; the data transfer regimes may each be for either read or write data transfer. Write data transfer permits the operations insert, replace or extend.

7.2 Filestore management

(replace clause, page 4)

Eight services are associated with filestore management:

a) the change current name prefix service (see clause 14a.1) is used by the initiator to control the mapping of incomplete pathnames to complete pathnames during the current association;

b) the list file-directory service (see clause 14a.2) is used by the initiator to interrogate for the attributes of objects which correspond to a given attribute value assertion list in or under a file-directory;

c) the generalized selection service (see clause 14b.1) is used by the initiator to identify a group of pathnames of files with attributes corresponding to a given attribute value assertion list;

d) the group deletion service (see clause 14b.2) is used by the initiator to remove the file objects identified by pathnames in the generalized selection group such that the objects cease to exist.

e) the group move service (see clause 14b.3) is used by the initiator to transfer the file objects identified by pathnames in the generalized selection group to a destination directory.

f) the group copy service (see clause 14b.4) is used by the initiator to duplicate the file objects identified by pathnames in the generalized selection group to a destination directory.

g) the group list service (see clause 14b.5) is used by the initiator to interrogate the pathnames of the file objects in the generalized selection group.

h) the group change attribute service (see clause 14b.6) is used by the initiator to modify the attributes of file objects, identified by complete pathnames within the generalized selection group activity attribute.

(amend title clause 7.3, page 4)

7.3 Object selection regime control

(amend 1st paragraph, page 4)

(amend Figure 2, page 4)

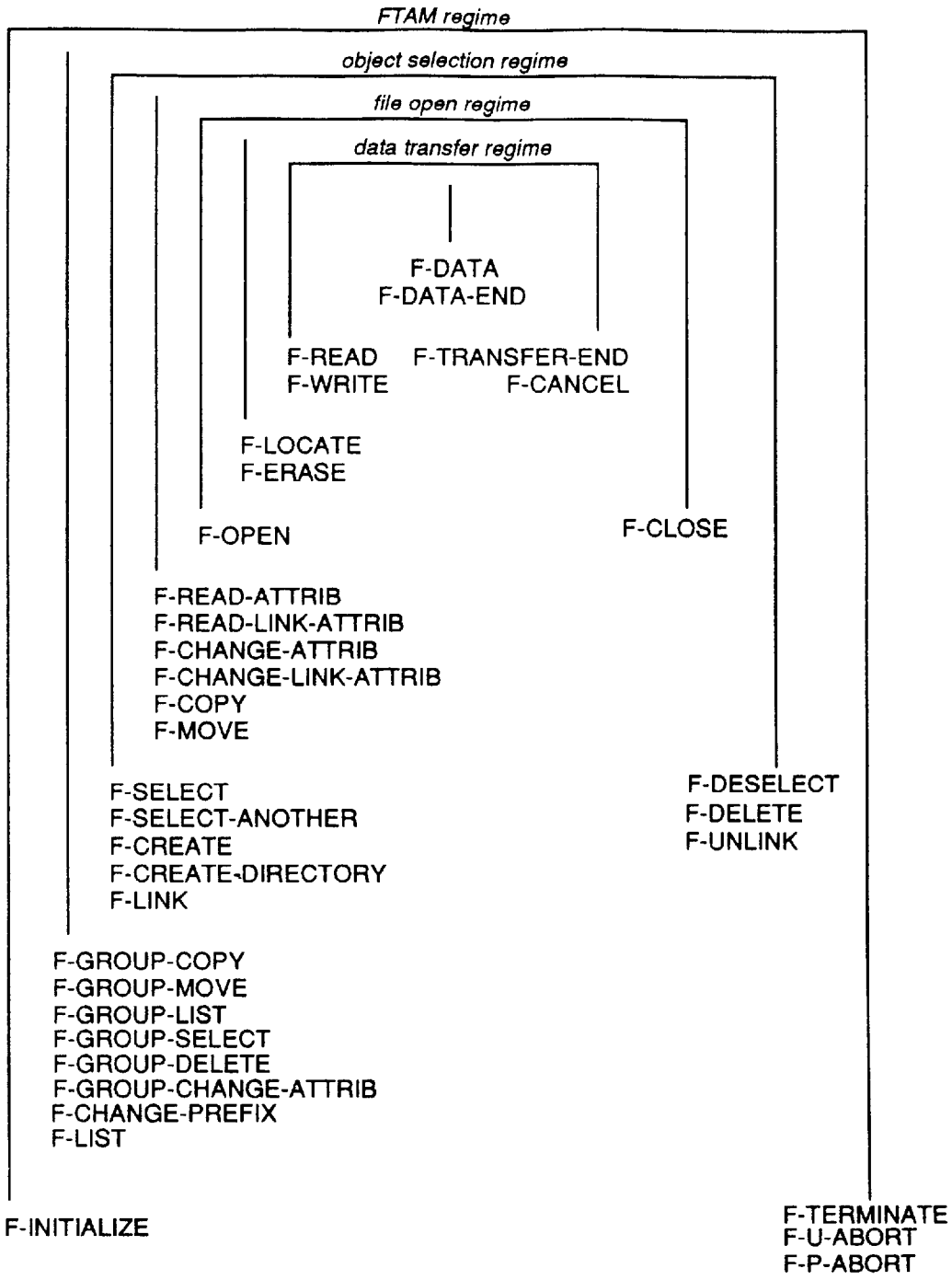


Figure 2 – File service regimes and related primitives

Eight services are associated with object selection regime control:

a) the object selection service (see clause 15.1) is used by the initiator to select a specific object by pathname and to bind the specified object to the FTAM regime;

aa) the select another service (see clause 15.1a) is used by the initiator to bind a previously unselected file from the generalized selection group to the FTAM regime;

b) the object deselection service (see clause 15.2) is used by the initiator to release the binding between the FTAM regime and the specified object;

c) the file creation service (see clause 15.3) is used by the initiator either

1) to create a specified file and to select the newly created file; or

2) depending on the override parameter of F–CREATE, to select an existing file;

and then to bind the specified file to the FTAM regime;

ca) the file-directory creation service (see clause 15.3a) is used by the initiator to create a specified file-directory object and bind the newly created file-directory to the FTAM regime;

cb) the reference creation service (see clause 15.3b) is used by the initiator to create a specified reference object and bind the existing object to which it is linked to the FTAM regime;

d) the object deletion service (see clause 15.4) is used by the initiator to release the binding between the FTAM regime and the specified object in such a way that the previously selected object ceased to exist;

e) the reference deletion service (see clause 15.5) is used to delete an existing reference, leaving the object to which it was linked intact in the filestore, and release the binding between the FTAM regime and the linked object.

(amend title to clause 7.4, page 5)

7.4 Object management

(amend 1st paragraph, page 5)

Six services are associated with object management:

a) the read attributes service (see clause 16.1) is used by the initiator to interrogate the object attributes of the selected object;

b) the change attributes service (see clause 16.2) is used by the initiator to modify the object attributes of the selected object;

c) the read reference attributes service (see clause 16.3) is used by the initiator to interrogate the object attributes of the reference identified by the current pathname activity attribute;

d) the change reference attributes service (see clause 16.4) is used by the initiator to modify the object attributes of the reference identified by the current pathname activity attribute;

e) the move object service (see clause 16.5) is used by the initiator to place the currently selected object into a specific file-directory by changing its primary pathname object attribute;

f) the copy object service (see clause 16.6) is used by the initiator to create a duplicate of the currently selected object in a specific file-directory.

NOTE – When accessing an object via a reference, the move object and copy object services operate only on the reference object, not the referent object.

7.6 Grouping control

(append after 1st paragraph, page 5)

The set of primitives contained within a group are constrained as follows:

a) FTAM regime control, filestore management and generalized filestore service primitives will never appear within a group;

b) recovery and restart service primitives will never appear within a group;

c) data transfer regime primitives, including F–DATA, F–DATA–END, F–LOCATE and F–ERASE, will never appear within a group;

d) if a regime is requested to be created and requested to be terminated within a single grouped sequence, then every regime requested to be created within the grouped sequence is also requested to be terminated within that grouped sequence;

e) primitives may be present within a group only if they are allowed within the negotiated service class, and the corresponding functional unit was

negotiated during the FTAM regime establishment;

f) any service from list 1 (below) must not appear in the same grouped sequence with a service from list 2:

1) F–CREATE,
F–CREATE–DIRECTORY

2) F–READ–LINK–ATTRIBUTES
F–CHANGE–LINK–ATTRIBUTES
F–UNLINK

Clause 12.2 and associated tables also place constraints on legal grouped sequences.

(append after clause 8.1.6, page 6)

8.1.6a Limited filestore management

The limited filestore management functional unit supports management of the current name prefix, and listing of objects within the filestore. In addition this functional unit supports the selection, deselection, and interrogation of directory and reference attributes.

8.1.6b Enhanced filestore management

The enhanced filestore management functional unit provides for the creation and deletion of file-directory and reference objects, and the modification of reference and directory object attributes.

8.1.6c Object manipulation functional unit

The object manipulation functional unit provides services to manage the position of objects within the filestore, and to duplicate objects within the filestore.

8.1.6d Group manipulation functional unit

The group manipulation functional unit provides for identifying a group of files within the filestore, interrogating the contents of that group, and for manipulation operations on that group of files.

8.2.1 File transfer class

(amend 1st paragraph, page 6)

ea) Optionally, the limited filestore management functional unit;

(amend 2nd paragraph, page 6)

In the file transfer service class, the use of the services is constrained so that there is a sequence of zero or more FTAM events on the application association. Each FTAM event requested by the initiator is a series of requests consisting of:

1) a single,optional, limited filestore management procedure to change current name prefix or list filestore;

2) a single grouped series of requests to establish a file open regime. (See clause 7.6 for valid grouped sequences);

3) a single bulk data transfer procedure, for either a read transfer or a write transfer. The processing mode parameter on the F–OPEN primitive is set to either a read or a valid write action, as defined in the constraint set, but not both.

4) a single grouped series of requests to release the file open and select regimes. (See clause 7.6 for valid grouped series.).

(delete note, page 6)

8.2.2 File access class

(amend 1st paragraph, pages 6 and 8)

The file access class consists of:

a) the kernel functional unit;

b) both of the read and write functional units;

c) the file access functional unit;

d) optionally, the grouping functional unit. If the grouping functional unit is successfully negotiated, its valid use in any instance by the initiator is optional but the acceptance by the responder is always mandatory.

e) optionally, the limited file management functional unit;

f) optionally, but only if the limited file management functional unit is present, the enhanced file management functional unit;

fa) optionally, the limited filestore management functional unit;

fb) optionally, but only if the limited filestore management functional unit is present, the enhanced filestore management functional unit;

fc) optionally, the object manipulation functional unit;

fd) optionally, but only if the object manipulation functional unit is present, the group manipulation functional unit;

g) optionally, the FADU locking functional unit;

h) optionally, in the internal file service, the recovery functional unit;

(append after entry U6, page 7)

Table 1 – Services and functional units of the External File Service

U6a Limited filestore management	change current name prefix list file-directory file-directory selection file-directory deselection reference selection reference deselection read file-directory attributes read reference attributes	O	O	O	O	O	14a.1 14a.2 15.1 15.2 15.1 15.2 16.1 16.3
U6b Enhanced filestore management	file-directory creation file-directory deletion reference creation reference deletion change file-directory attributes change reference attributes (requires 6a)		O	O	O	O	15.3a 15.4 15.3b 15.5 16.2 16.4
U6c Object manipulation	move object copy object		O	O	O	O	16.5 16.6
U6d Group manipulation	generalized selection generalized deletion group change attributes group move group copy group list select another		O	O	O	O	14b.1 14b.2 14b.7 14b.3 14b.5 14b.6 15.1a

i) optionally, in the internal file service, the restart data transfer functional unit.

(delete 2nd note, page 8)

8.2.3 File management class

(amend 1st paragraph, page 8)

The file management class consists of:

a) the kernel functional unit;

b) the limited file management functional unit;

c) optionally, the enhanced file management functional unit;

d) the grouping functional unit;

e) optionally, the limited filestore management functional unit;

f) optionally, but only if the limited filestore management functional unit is present, the enhanced filestore management functional unit;

g) optionally, the object manipulation functional unit;

h) optionally, but only if the object manipulation functional unit is present, the group manipulation functional unit.

(amend 2nd paragraph, page 8)

In the file management service class the use of the services is constrained so that there is a series of zero or more FTAM events on the application association. Each FTAM event requested by the initiator is a single grouped series of requests. (See clause 7.6 for valid sets of grouped primitives.) The file management class imposes the following further restrictions on grouped primitives:

a) The F–OPEN and F–CLOSE primitives are excluded from use;

b) if a grouped series of primitives establishes a select regime, it must also terminate the select regime.

8.2.4 File transfer and management class

(amend 1st paragraph, page 8)

The file transfer and management class consists of:

a) the kernel functional unit;

b) the grouping functional unit;

c) one or both of the read or write functional units;

d) the limited file management functional unit;

e) optionally, the enhanced file management functional unit;

ea) optionally, the limited filestore management functional unit;

eb) optionally, but only if the limited filestore management functional unit is present, the enhanced filestore management functional unit;

ec) optionally, the object manipulation functional unit;

ed) optionally, but only if the object manipulation functional unit is present, the group manipulation functional unit;

f) optionally, in the internal file service, the restart data transfer function unit;

g) optionally, in the internal file service, the recovery functional unit.

(amend 2nd paragraph, page 8)

In the file transfer and management service class, the use of the services is constrained so that there is a repeated series of FTAM events on the application association. Each FTAM event is either;

a transfer comprising:

1) a single, optional, limited filestore management procedure to change current name prefix or list filestore;

2) a single grouped series of requests requested by the initiator to establish a file open regime. (See clause 7.6 for valid grouped series).

3) a single bulk data transfer procedure, for either a read transfer or a write transfer. The processing mode parameter on the F–OPEN primitive is set to either a read or a valid write action, as defined in the constraint set, but not both.

4) a single grouped series of requests requested by the initiator to release the file open and select regimes. (See clause 7.6 for valid grouped series.)

or a single grouped series of requests requested by the initiator to effect management. (See clause 7.6 for valid grouped sequences.) The open regime will not be entered. If the select regime is established within a grouped series, then it will also be terminated within the grouped series of requests.

(delete the node, page 8)

(append after table 3, page 9)

limited filestore management enhanced filestore management	limited filestore management enhanced filestore management

Section two: Definition of file service primitives

12.2 Constraints on the issue of primitives

(amend 1st paragraph, pages 11 & 14; this includes the list of grouped sequences)

The primitives may be issued in any sequence consistent with the constraints given in tables 7 to 10. The sequences of primitives are defined under the individual services. Individual series of requests requested by the initiator may be interleaved to form grouped sequences. See clause 7.6 for valid groups.

(amend the notes, page 14)

NOTES

1) Not all sequences are allowed in the file transfer, file management, and file transfer and management classes (see clauses 8.2.1, 8.2.3, 8.2.4). In these classes, the threshold parameter is set so that the sequences either succeed or fail as a whole, i.e. set to the number of primitives between the begin group and end group primitives.

2) Other constraints will affect the ability of the file service user or file service provider to invoke the various procedures, such as flow control constraints on sending data or constraints on the ability of a file service user to accept spontaneous F-P-ABORT indications from the file service provider.

12.3.2 Conventions for tables 9 and 10

(amend 1st paragraph, page 14)

In tables 9 and 10, the entries indicate the functional units required for the succession to occur. The entries are:

Kernel	kernel functional unit
Grouping	grouping functional unit
Lmgt	limited management functional unit
Emgt	enhanced management functional unit
Recover	recover functional unit
Access	file access functional unit
G–Lmgt	grouping and limited file management functional units
G–Emgt	grouping and enhanced management functional units
Lfmg	limited filestore management functional unit
Efmg	enhanced filestore management functional unit
G–Lfmg	grouping and limited filestore management functional units
G–Efmg	grouping and enhanced filestore management functional units
Obmn	object manipulation functional unit
G–Obmn	grouping and object manipulation functional units
Grmn	group manipulation functional unit
G–Grmn	grouping and group manipulation functional units

(insert after 1st paragraph, page 14)

Tables 9 and 10 assume that completion of a previous file service event implies that all functional units required by the corresponding file service primitive were negotiated.

(insert after clause 12.3.2, page 14)

12.3.3 Notes from tables 9 and 10

[1] This transition is only allowed if no regimes other than the FTAM regime are currently established.

[2] This transition is only allowed if the select regime is the innermost regime established.

[3] This transition is only allowed if the open regime is the innermost regime established.

12.4 Confirmed services

(amend 2nd paragraph, page 20)

A request to establish a new regime (F–INITIALIZE, F–SELECT, F–SELECT–ANOTHER, F–CREATE, F–CREATE–DIRECTORY, F–LINK OR F–OPEN) may be rejected by use of a response with state result parameter indicating failure (see clause 13.1).

Table 6, page 11 modify as follows:

⋮	⋮	⋮	⋮
F-SELECT	Yes	Initiator	State result Action result Attributes Requested access Referent indicator Access passwords Concurrency control Shared ASE information Account Diagnostic

(append to table 6, pages 11 and 12)

Table 6 – File service primitives

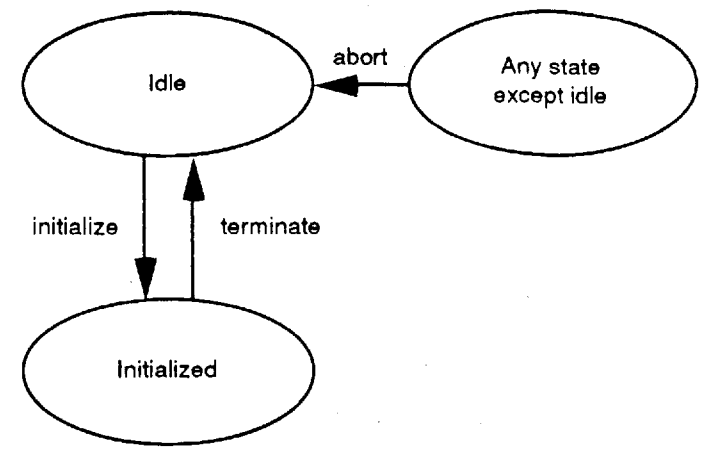
F-CHANGE-PREFIX	Yes	Initiator	Action result Reset Destination file-directory Access passwords Diagnostic
F-LIST	Yes	Initiator	Action result Attribute value assertions Scope Access passwords Attribute names Objects attributes list Diagnostic
F-GROUP-SELECT	Yes	Initiator	Action result State result Attribute value assertions Requested access Scope Access passwords Concurrency control Maximum set size Account Shared ASE information Diagnostic
F-GROUP-DELETE	Yes	Initiator	Action result Charging Request operation result Operation result Shared ASE information Diagnostic
F-GROUP-COPY	Yes	Initiator	Action result Destination file-directory Override Error action Create password Access passwords Attributes Request operation result Operation result Diagnostic
F-GROUP-MOVE	Yes	Initiator	Action result Destination file-directory Override Error action Create password Attributes Request operation result Operation result Diagnostic
F-GROUP-LIST	Yes	Initiator	Action result Attribute names Objects attributes list Diagnostic
F-GROUP-CHANGE-ATTRIB	Yes	Initiator	Action result Attributes Error action Shared ASE information Diagnostic
F-SELECT-ANOTHER	Yes	Initiator	State result Action result Last member indicator Shared ASE information referent indicator Diagnostic

(append to table 6, pages 11 and 12)

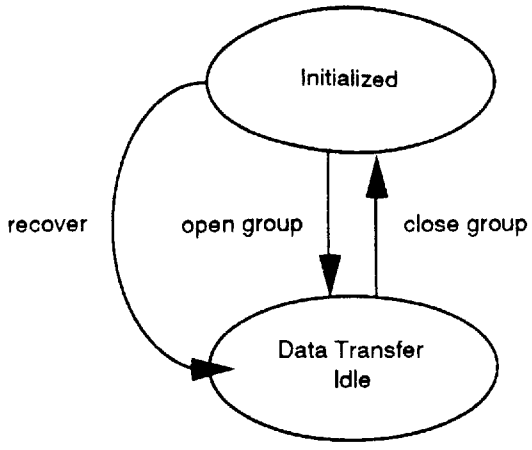
Table 6 (continued) – File service primitives

F-CREATE-DIRECTORY	Yes	Initiator	State result Action result Attributes Create password Requested access Shared ASE information Account Diagnostic
F-LINK	Yes	Initiator	State result Action result Attributes Target object Create password Requested access Access passwords Concurrency control Shared ASE information Account Diagnostic
F-UNLINK	Yes	Initiator	Action result Shared ASE information Charging Diagnostic
F-READ-LINK-ATTRIB	Yes	Initiator	Action result Attribute names Attributes Diagnostic
F-CHANGE-LINK-ATTRIB	Yes	Initiator	Action result Attributes Diagnostic
F-COPY	Yes	Initiator	Action result Destination file-directory Override Create password Access passwords Attributes Diagnostic
F-MOVE	Yes	Initiator	Action result Destination file-directory Override Create password Access passwords Attributes Diagnostic

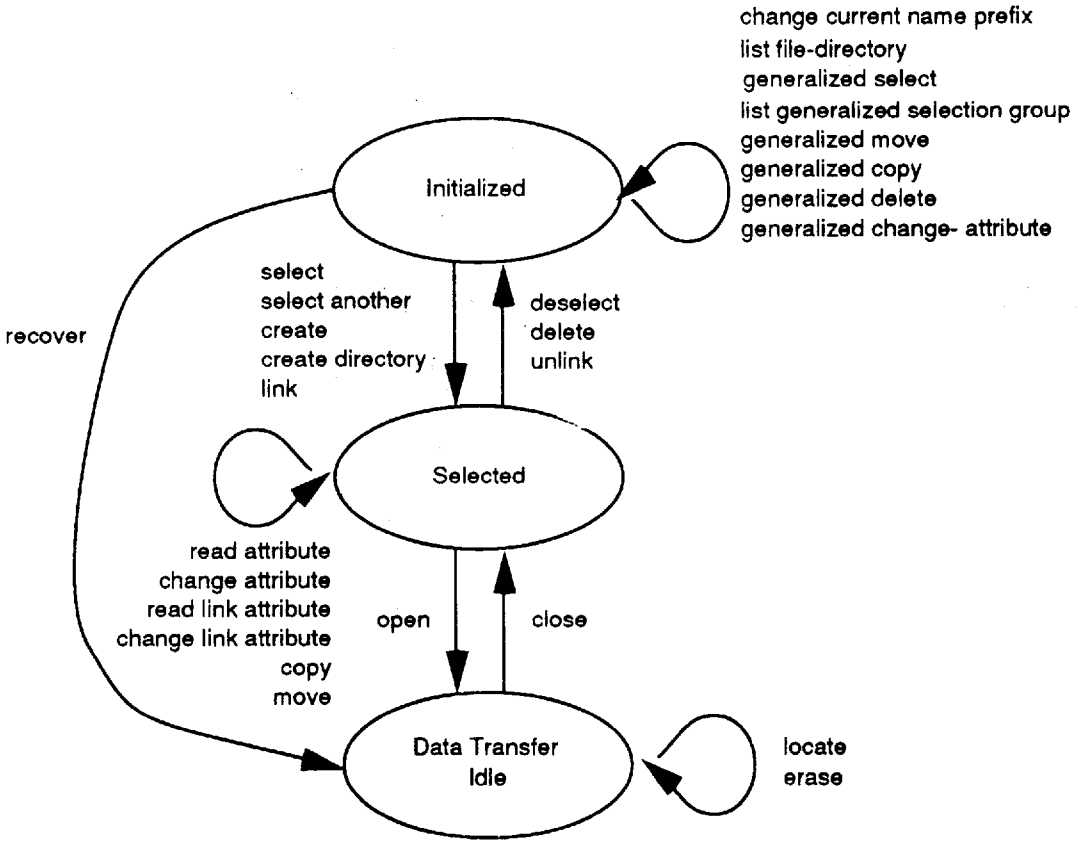
(amend Figure 3, page 13)



FTAM regime establishment



Regime establishment – file transfer class



File regime establishment – file access class

Figure 3 – Simplified State Diagram for successful activity (see Annex E)

(amend Tables 9 and 10, pages 16 to 19)

Table 9 – Sequence of service primitives for file service regimes – initiator

Previous file service event	F-CHANGE-PREFIX.request	F-LIST .request	F-GROUP-SELECT .request	F-GROUP-DELETE .request	F-GROUP-COPY.request	F-GROUP-MOVE.request
FTAM regime established	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-CHANGE-PREFIX.request						
F-CHANGE-PREFIX.confirm	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-LIST.request						
F-LIST.confirm	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-GROUP-SELECT.request						
F-GROUP-SELECT.confirm (+ve)	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-GROUP-SELECT.confirm (-ve)	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-GROUP-DELETE.request						
F-GROUP-DELETE.confirm	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-GROUP-COPY.request						
F-GROUP-COPY.confirm	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-GROUP-MOVE.request						
F-GROUP-MOVE.confirm	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-GROUP-LIST.request						
F-GROUP-LIST.confirm	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-GROUP-CHANGE-ATTRIBUTE.request						
F-GROUP-CHANGE-ATTRIBUTE.confirm	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-SELECT.request						
F-SELECT.confirm (+ve)						
F-SELECT.confirm (-ve)	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-SELECT-ANOTHER.request						
F-SELECT-ANOTHER.confirm (+ve)						
F-SELECT-ANOTHER.confirm (-ve)	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-CREATE.request						
F-CREATE.confirm (+ve)						
F-CREATE.confirm (-ve)	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-CREATE-DIRECTORY.request						
F-CREATE-DIRECTORY.confirm (+ve)						
F-CREATE-DIRECTORY.confirm (-ve)	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-LINK.request						
F-LINK.confirm (+ve)						
F-LINK.confirm (-ve)	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-DESELECT.request						
F-DESELECT.confirm	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-DELETE.request						
F-DELETE.confirm	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-UNLINK.request						
F-UNLINK.confirm	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-READ-ATTRIB.request						
F-READ-ATTRIB.confirm						
F-CHANGE-ATTRIB.request						
F-CHANGE-ATTRIB.confirm						
F-READ-LINK-ATTRIB.request						

8571 PT*3 88 ■ 4851903 0536311 665 ■
ISO 8571-3:1988/Amd.1:1992 (E)

Table 9 – Sequence of service primitives for file service regimes – initiator (continued)

Previous file service event	F-CHANGE-PREFIX.request	F-LIST .request	F-GROUP-SELECT .request	F-GROUP-DELETE .request	F-GROUP-COPY.request	F-GROUP-MOVE.request
F-READ-LINK-ATTRIB.confirm						
F-CHANGE-LINK-ATTRIB.request						
F-CHANGE-LINK-ATTRIB.confirm						
F-COPY.request						
F-COPY.confirm						
F-MOVE.request						
F-MOVE.confirm						
F-OPEN.request						
F-OPEN.confirm (+ve)						
F-OPEN.confirm (-ve)						
F-CLOSE.request						
F-CLOSE.confirm						
F-BEGIN-GROUP.request						
F-BEGIN-GROUP.confirm						
F-END-GROUP.request						
F-END-GROUP.confirm	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
F-RECOVER.request						
F-RECOVER.confirm (+ve)						
F-RECOVER.confirm (-ve)	Lfmg	Lfmg	Grmn	Grmn	Grmn	Grmn
Bulk data transfer						
F-LOCATE.request						
F-LOCATE.confirm						
F-ERASE.request						
F-ERASE.confirm						

Table 9 – Sequence of service primitives for file service regimes – initiator (continued)

Previous file service event	F-GROUP-CHANGE-ATTRIB .request	F-GROUP-LIST .request	F-SELECT .request	F-SELECT-ANOTHER .request	F-CREATE .request	F-CREATE-DIRECTORY .request	F-LINK .request
FTAM regime established	Grmn	Grmn	Kernel or Lfmg		Lmgt	Efmg	Efmg
F-CHANGE-PREFIX.request							
F-CHANGE-PREFIX.confirm	Grmn	Grmn	Lfmg		Lmgt	Efmg	Efmg
F-LIST.request							
F-LIST.confirm	Grmn	Grmn	Lfmg	Grmn	Lmgt	Efmg	Efmg
F-GROUP-SELECT.request							
F-GROUP-SELECT.confirm (+ve)	Grmn	Grmn	Kernel or Lfmg	Grmn	Lmgt	Efmg	Efmg
F-GROUP-SELECT.confirm (-ve)	Grmn	Grmn	Kernel or Lfmg	Grmn	Lmgt	Efmg	Efmg
F-GROUP-DELETE.request							
F-GROUP-DELETE.confirm	Grmn	Grmn	Kernel or Lfmg	Grmn	Lmgt	Efmg	Efmg
F-GROUP-COPY.request							
F-GROUP-COPY.confirm	Grmn	Grmn	Kernel or Lfmg	Grmn	Lmgt	Efmg	Efmg
F-GROUP-MOVE.request							
F-GROUP-MOVE.confirm	Grmn	Grmn	Kernel or Lfmg	Grmn	Lmgt	Efmg	Efmg
F-GROUP-LIST.request							
F-GROUP-LIST.confirm	Grmn	Grmn	Kernel or Lfmg	Grmn	Lmgt	Efmg	Efmg
F-SELECT.request							
F-SELECT.confirm (+ve)							
F-SELECT.confirm (-ve)	Grmn	Grmn	Kernel or Lfmg	Grmn	Lmgt	Efmg	Efmg
F-SELECT-ANOTHER.request							
F-SELECT-ANOTHER.confirm (+ve)							
F-SELECT-ANOTHER.confirm (-ve)	Grmn	Grmn	Kernel or Lfmg	Grmn	Lmgt	Efmg	Efmg
F-CREATE.request							
F-CREATE.confirm (+ve)							
F-CREATE.confirm (-ve)	Grmn	Grmn	Kernel or Lfmg	Grmn	Lmgt	Efmg	Efmg
F-CREATE-DIRECTORY.request							
F-CREATE-DIRECTORY.confirm (+ve)							
F-CREATE-DIRECTORY.confirm (-ve)	Grmn	Grmn	Efmg	Grmn	Lmgt	Efmg	Efmg
F-LINK.request							
F-LINK.confirm (+ve)							
F-LINK.confirm (-ve)	Grmn	Grmn	Efmg	Grmn	Lmgt	Efmg	Efmg
F-DESELECT.request							
F-DESELECT.confirm	Grmn	Grmn	Kernel or Lfmg	Grmn	Lmgt	Efmg	Efmg
F-DELETE.request							
F-DELETE.confirm	Grmn	Grmn	Kernel or Lfmg	Grmn	Lmgt	Efmg	Efmg
F-UNLINK.request							
F-UNLINK.confirm	Grmn	Grmn	Efmg	Grmn	Lmgt	Efmg	Efmg
F-READ-ATTRIB.request							
F-READ-ATTRIB.confirm							
F-CHANGE-ATTRIB.request							
F-CHANGE-ATTRIB.confirm							
F-READ-LINK-ATTRIB.request							

8571 PT*3 88 ■ 4851903 0536313 438 ■

Table 9 – Sequence of service primitives for file service regimes – initiator (continued)

Previous file service event	F-GROUP-CHANGE-ATTRIB.request	F-GROUP-LIST.request	F-SELECT .request	F-SELECT-ANOTHER .request	F-CREATE .request	F-CREATE-DIRECTORY .request	F-LINK .request
F-READ-LINK-ATTRIB.confirm							
F-CHANGE-LINK-ATTRIB.request							
F-CHANGE-LINK-ATTRIB.confirm							
F-COPY.request							
F-COPY.confirm							
F-MOVE.request							
F-MOVE.confirm							
F-OPEN.request							
F-OPEN.confirm (+ve)							
F-OPEN.confirm (-ve)							
F-CLOSE.request							
F-CLOSE.confirm							
F-BEGIN-GROUP.request			Kernel or G-Lfmg	Grmn	Lmgt	Efmg	Efmg
F-BEGIN-GROUP.confirm							
F-END-GROUP.request							
F-END-GROUP.confirm	Grmn	Grmn	Kernel or G-Lfmg	Grmn	Lmgt	Efmg	Efmg
F-RECOVER.request							
F-RECOVER.confirm (+ve)							
F-RECOVER.confirm (-ve)	Grmn	Grmn	Kernel or Lfmg	Grmn	Lmgt	Efmg	Efmg
Bulk data transfer							
F-LOCATE.request							
F-LOCATE.confirm							
F-ERASE.request							
F-ERASE.confirm							

Table 9 – Sequence of service primitives for file service regimes – initiator (continued)

Previous file service event	F-DESELECT .request	F-DELETE .request	F-UNLINK .request	F-READ-ATTRIB.request	F-CHANGE-ATTRIB .request	F-READ-LINK-ATTRIB .request
FTAM regime established						
F-CHANGE-PREFIX.request						
F-CHANGE-PREFIX.confirm						
F-LIST.request						
F-LIST.confirm						
F-GROUP-SELECT.request						
F-GROUP-SELECT.confirm (+ve)						
F-GROUP-SELECT.confirm (-ve)						
F-GROUP-DELETE.request						
F-GROUP-DELETE.confirm						
F-GROUP-COPY.request						
F-GROUP-COPY.confirm						
F-GROUP-MOVE.request						
F-GROUP-MOVE.confirm						
F-GROUP-LIST.request						
F-GROUP-LIST.confirm						
F-GROUP-CHANGE-ATTRIB.request						
F-GROUP-CHANGE-ATTRIB.confirm						
F-SELECT.request	Grouping|G-Lfmg	G-Lmgt|G-Efmg	G-Efmg	G-Lmgt|G-Lfmg	G-Lmgt|G-Lfmg	G-Lfmg
F-SELECT.confirm (+ve)	Kernel or Lfmg	Lmgt or Efmg	Efmg	Lmgt or Lfmg	Lmgt or Lfmg	Lfmg
F-SELECT.confirm (-ve)						
F-SELECT-ANOTHER.request	Grouping|G-Lfmg	G-Lmgt|G-Efmg	G-Efmg	G-Lmgt|G-Lfmg	G-Lmgt|G-Lfmg	G-Lfmg
F-SELECT-ANOTHER.confirm (+ve)	Kernel or Lfmg	Lmgt or Efmg	Efmg	Lmgt or Lfmg	Lmgt or Lfmg	Lfmg
F-SELECT-ANOTHER.confirm (-ve)						
F-CREATE.request	Grouping	G-Lmgt		G-Lmgt	G-Emgt	
F-CREATE.confirm (+ve)	Kernel	Lmgt		Lmgt	Emgt	
F-CREATE.confirm (-ve)						
F-CREATE-DIRECTORY.request	G-Efmg	G-Efmg		G-Efmg	G-Efmg	
F-CREATE-DIRECTORY.confirm (+ve)	Kernel or Efmg	Efmg		Efmg	Efmg	
F-CREATE-DIRECTORY.confirm (-ve)						
F-LINK.request	G-Efmg	G-Efmg	G-Efmg	G-Efmg	G-Efmg	G-Lfmg
F-LINK.confirm (+ve)	Efmg	Efmg	Efmg	Efmg	Efmg	Lfmg
F-LINK.confirm (-ve)						
F-DESELECT.request						
F-DESELECT.confirm						
F-DELETE.request						
F-DELETE.confirm						
F-UNLINK.request						
F-UNLINK.confirm						
F-READ-ATTRIB.request	Grouping or Lfmg	G-Lmgt|G-Lfmg	G-Efmg	G-Lmgt|G-Lfmg	G-Emgt|G-Efmg	G-Lfmg
F-READ-ATTRIB.confirm	Kernel or Lfmg	Lmgt or Lfmg	Efmg	Lmgt or Lfmg	Emgt or Efmg	Lfmg
F-CHANGE-ATTRIB.request	Grouping or Efmg	G-Lmgt|G-Efmg	G-Efmg	G-Lmgt|G-Efmg	G-Emgt|G-Efmg	G-Lfmg
F-CHANGE-ATTRIB.confirm	Kernel or Efmg	Lmgt or Efmg	Efmg	Lmgt or Efmg	Emgt or Efmg	Lfmg
F-READ-LINK-ATTRIB.request	Grouping	G-Efmg	G-Efmg	G-Lmgt|G-Efmg	G-Lfmg|G-Efmg	G-Lfmg

8571 PT*3 88 ■ 4851903 0536315 200 ■
ISO 8571-3:1988/Amd.1:1992 (E)

Table 9 – Sequence of service primitives for file service regimes – initiator (continued)

Previous file service event	F-DESELECT .request	F-DELETE .request	F-UNLINK .request	F-READ-ATTRIB.request .	F-CHANGE-ATTRIB .request	F-READ-LINK-ATTRIB .request
F-READ-LINK-ATTRIB.confirm	Lfmg	Efmg	Efmg	Lmgt|Lfmg	Efmg	Lfmg
F-CHANGE-LINK-ATTRIB.request	G-Efmg	G-Efmg	G-Efmg	G-Efmg	G-Efmg	G-Efmg
F-CHANGE-LINK-ATTRIB.confirm	Efmg	Efmg	Efmg	Efmg	Efmg	Efmg
F-COPY.request	G-Lfmg|Grouping	G-Lmgt|G-Efmg	G-Efmg	G-Lfmg|G-Lmgt	G-Efmg|G-Emgt	G-Lfmg
F-COPY.confirm	Lfmg|Kernel	Lmgt|Efmg	Efmg	Lmgt|Lfmg	Efmg|Emgt	Lfmg
F-MOVE.request	G-Lfmg|Grouping	G-Lmgt|G-Efmg	G-Efmg	G-Lfmg|G-Lmgt	G-Efmg|G-Emgt	G-Lfmg
F-MOVE.confirm	Lfmg|Kernel	Lmgt|Efmg	Efmg	Lmgt|Lfmg	Efmg|Emgt	Lfmg
F-OPEN.request						
F-OPEN.confirm (+ve)						
F-OPEN.confirm (-ve)	Kernel	Lmgt	Efmg	Lmgt	Emgt	Lfmg
F-CLOSE.request	Grouping	G-Lmgt	G-Efmg	G-Lmgt	G-Emgt	G-Lfmg
F-CLOSE.confirm	Kernel	Lmgt	Efmg	Lmgt	Emgt	Lfmg
F-BEGIN-GROUP.request	Lfmg|Kernel [2]	Efmg|Lmgt [2]	Efmg [2]	Lfmg|Lmgt[2]	Efmg|Emgt [2]	Lfmg [2]
F-BEGIN-GROUP.confirm						
F-END-GROUP.request						
F-END-GROUP.confirm	Lfmg|Kernel [2]	Efmg|Lmgt [2]	Efmg [2]	Lfmg|Lmgt [2]	Efmg|Emgt [2]	Lfmg [2]
F-RECOVER.request						
F-RECOVER.confirm (+ve)						
F-RECOVER.confirm (-ve)						
Bulk data transfer						
F-LOCATE.request						
F-LOCATE.confirm						
F-ERASE.request						
F-ERASE.confirm						

Table 9 – Sequence of service primitives for file service regimes – initiator (continued)

Previous file service event	F-CHANGE-LINK-ATTRIB .request	F-COPY .request	F-MOVE .request	F-OPEN .request	F-CLOSE .request	F-BEGIN-GROUP.request
FTAM regime established						Grouping
F-CHANGE-PREFIX.request						
F-CHANGE-PREFIX.confirm						Grouping
F-LIST.request						
F-LIST.confirm						Grouping
F-GROUP-SELECT.request						
F-GROUP-SELECT.confirm (+ve)						
F-GROUP-SELECT.confirm (-ve)						Grouping
F-GROUP-DELETE.request						
F-GROUP-DELETE.confirm						Grouping
F-GROUP-COPY.request						
F-GROUP-COPY.confirm						Grouping
F-GROUP-MOVE.request						
F-GROUP-MOVE.confirm						Grouping
F-GROUP-LIST.request						
F-GROUP-LIST.confirm						Grouping
F-GROUP-CHANGE-ATTRIB.request						
F-GROUP-CHANGE-ATTRIB.confirm						Grouping
F-SELECT.request	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-SELECT.confirm (+ve)	Efmg	Obmn	Obmn	Kernel		Grouping
F-SELECT.confirm (-ve)						Grouping
F-SELECT-ANOTHER.request	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-SELECT-ANOTHER.confirm (+ve)	Efmg	Obmn	Obmn	Kernel		Grouping
F-SELECT-ANOTHER.confirm (-ve)						Grouping
F-CREATE.request		G-Obmn	G-Obmn	Grouping		
F-CREATE.confirm (+ve)		Obmn	Obmn	Kernel		Grouping
F-CREATE.confirm (-ve)						Grouping
F-CREATE-DIRECTORY.request		G-Obmn	G-Obmn			
F-CREATE-DIRECTORY.confirm (+ve)		Obmn	Obmn			Grouping
F-CREATE-DIRECTORY.confirm (-ve)						Grouping
F-LINK.request	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-LINK.confirm (+ve)	Efmg	Obmn	Obmn	Kernel		Grouping
F-LINK.confirm (-ve)						Grouping
F-DESELECT.request						
F-DESELECT.confirm						Grouping
F-DELETE.request						
F-DELETE.confirm						Grouping
F-UNLINK.request						
F-UNLINK.confirm						Grouping
F-READ-ATTRIB.request	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-READ-ATTRIB.confirm	Efmg	Obmn	Obmn	Kernel		Grouping
F-CHANGE-ATTRIB.request	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-CHANGE-ATTRIB.confirm	Efmg	Obmn	Obmn	Kernel		Grouping
F-READ-LINK-ATTRIB.request	G-Efmg	G-Obmn	G-Obmn	Grouping		

8571 PT*3 88 ■ 4851903 0536317 083 ■
ISO 8571-3:1988/Amd.1:1992 (E)

Table 9 – Sequence of service primitives for file service regimes – initiator (continued)

Previous file service event	F-CHANGE-LINK-ATTRIB .request	F-COPY .request	F-MOVE .request	F-OPEN .request	F-CLOSE .request	F-BEGIN-GROUP.request
F-READ-LINK-ATTRIB.confirm	Efmg	Obmn	Obmn	Kernel		Grouping
F-CHANGE-LINK-ATTRIB.request	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-CHANGE-LINK-ATTRIB.confirm	Efmg	Obmn	Obmn	Kernel		Grouping
F-COPY.request	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-COPY.confirm	Efmg	Obmn	Obmn	Kernel		Grouping
F-MOVE.request	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-MOVE.confirm	Efmg	Obmn	Obmn	Kernel		Grouping
F-OPEN.request					Grouping	
F-OPEN.confirm (+ve)					Kernel	Grouping
F-OPEN.confirm (-ve)	Efmg	Obmn	Obmn	Kernel		Grouping
F-CLOSE.request	G-Efmg	G-Obmn	G-Obmn			
F-CLOSE.confirm	Efmg	Obmn	Obmn	Kernel	Kernel	Grouping
F-BEGIN-GROUP.request	Efmg [2]	Obmn [2]	Obmn [2]		Kernel [3]	
F-BEGIN-GROUP.confirm						
F-END-GROUP.request						
F-END-GROUP.confirm	Efmg [2]	Obmn [2]	Obmn [2]	Kernel [2]	Kernel [3]	Grouping
F-RECOVER.request						
F-RECOVER.confirm (+ve)					Kernel	Grouping
F-RECOVER.confirm (-ve)						Grouping
Bulk data transfer					Kernel	Grouping
F-LOCATE.request						
F-LOCATE.confirm					Kernel	Grouping
F-ERASE.request						
F-ERASE.confirm					Kernel	Grouping

Table 9 – Sequence of service primitives for file service regimes – initiator (continued)

Previous file service event	F-END-GROUP .request	F-RECOVER .request	Bulk data transfer	F-LOCATE .request	F-ERASE .request	Terminate FTAM Regime
FTAM regime established		Recover				Kernel
F-CHANGE-PREFIX.request						
F-CHANGE-PREFIX.confirm		Recover				Kernel
F-LIST.request						
F-LIST.confirm		Recover				Kernel
F-GROUP-SELECT.request						
F-GROUP-SELECT.confirm (+ve)						
F-GROUP-SELECT.confirm (-ve)		Recover				Kernel
F-GROUP-DELETE.request						
F-GROUP-DELETE.confirm		Recover				Kernel
F-GROUP-COPY.request						
F-GROUP-COPY.confirm						
F-GROUP-MOVE.request						
F-GROUP-MOVE.confirm						
F-GROUP-LIST.request						
F-GROUP-LIST.confirm						
F-GROUP-CHANGE-ATTRIB.request						
F-GROUP-CHANGE-ATTRIB.confirm						
F-SELECT.request	Grouping					
F-SELECT.confirm (+ve)						
F-SELECT.confirm (-ve)		Recover				Kernel
F-SELECT-ANOTHER.request	G-Grmn					
F-SELECT-ANOTHER.confirm (+ve)						
F-SELECT-ANOTHER.confirm (-ve)						
F-CREATE.request	Grouping					
F-CREATE.confirm (+ve)						
F-CREATE.confirm (-ve)		Recover				Kernel
F-CREATE-DIRECTORY.request	G-Efmg					
F-CREATE-DIRECTORY.confirm (+ve)						
F-CREATE-DIRECTORY.confirm (-ve)		Recover				Kernel
F-LINK.request	G-Efmg					
F-LINK.confirm (+ve)						
F-LINK.confirm (-ve)		Recover				Kernel
F-DESELECT.request	Grouping					
F-DESELECT.confirm		Recover				Kernel
F-DELETE.request	Grouping					
F-DELETE.confirm		Recover				Kernel
F-UNLINK.request	G-Efmg					
F-UNLINK.confirm		Recover				Kernel
F-READ-ATTRIB.request	Grouping					
F-READ-ATTRIB.confirm						
F-CHANGE-ATTRIB.request	Grouping					
F-CHANGE-ATTRIB.confirm						
F-READ-LINK-ATTRIB.request	G-Efmg					

8571 PT*3 88 ■ 4851903 0536319 956 ■

Table 9 – Sequence of service primitives for file service regimes – initiator (continued)

Previous file service event	F-END-GROUP .request	F-RECOVER .request	Bulk data transfer	F-LOCATE .request	F-ERASE .request	Terminate FTAM Regime
F-READ-LINK-ATTRIB.confirm						
F-CHANGE-LINK-ATTRIB.request	G-Efmg					
F-CHANGE-LINK-ATTRIB.confirm						
F-COPY.request	G-Obmn					
F-COPY.confirm						
F-MOVE.request	G-Obmn					
F-MOVE.confirm						
F-OPEN.request	Grouping					
F-OPEN.confirm (+ve)			Kernel	Access	Access	
F-OPEN.confirm (-ve)						
F-CLOSE.request						
F-CLOSE.confirm						
F-BEGIN-GROUP.request						
F-BEGIN-GROUP.confirm						
F-END-GROUP.request						
F-END-GROUP.confirm		Recover [1]	Kernel [3]	Access [3]	Access [3]	Kernel [1]
F-RECOVER.request						
F-RECOVER.confirm (+ve)			Kernel	Access	Access	
F-RECOVER.confirm (-ve)		Recover				Kernel
Bulk data transfer			Kernel	Access	Access	
F-LOCATE.request						
F-LOCATE.confirm			Kernel	Access	Access	
F-ERASE.request						
F-ERASE.confirm			Kernel	Access	Access	

Table 10 – Sequence of service primitives for file service regimes – responder

Previous file service event	F-CHANGE-PREFIX .response	F-LIST .response	F-GROUP-SELECT .response	F-GROUP-DELETE response	F-GROUP-COPY.response	F-GROUP-MOVE.response
FTAM regime established						
F-CHANGE-PREFIX.indication	Lfmg					
F-CHANGE-PREFIX.response						
F-LIST.indication		Lfmg				
F-LIST.response						
F-GROUP-SELECT.indication			Grmn			
F-GROUP-SELECT.response (+ve)						
F-GROUP-SELECT.response (-ve)						
F-GROUP-DELETE.indication				Grmn		
F-GROUP-DELETE.response						
F-GROUP-COPY.indication					Grmn	
F-GROUP-COPY.response						
F-GROUP-MOVE.indication						Grmn
F-GROUP-MOVE.response						
F-GROUP-LIST.indication						
F-GROUP-LIST.response						
F-GROUP-CHANGE-ATTRIB.indication						
F-GROUP-CHANGE-ATTRIB.response						
F-SELECT.indication						
F-SELECT.response (+ve)						
F-SELECT.response (-ve)						
F-SELECT-ANOTHER.indication						
F-SELECT-ANOTHER.response (+ve)						
F-SELECT-ANOTHER.response (-ve)						
F-CREATE.indication						
F-CREATE.response (+ve)						
F-CREATE.response (-ve)						
F-CREATE-DIRECTORY.indication						
F-CREATE-DIRECTORY.response (+ve)						
F-CREATE-DIRECTORY.response (-ve)						
F-LINK.indication						
F-LINK.response (+ve)						
F-LINK.response (-ve)						
F-DESELECT.indication						
F-DESELECT.response						
F-DELETE.indication						
F-DELETE.response						
F-UNLINK.indication						
F-UNLINK.response						
F-READ-ATTRIB.indication						
F-READ-ATTRIB.response						
F-CHANGE-ATTRIB.indication						
F-CHANGE-ATTRIB.response						
F-READ-LINK-ATTRIB.indication						

8571 PT*3 88 ■ 4851903 0536321 504 ■
ISO 8571-3:1988/Amd.1:1992 (E)

Table 10 – Sequence of service primitives for file service regimes – responder (continued)

Previous file service event	F-CHANGE-PREFIX .response	F-LIST .response	F-GROUP-SELECT .response	F-GROUP-DELETE .response	F-GROUP-COPY.response	F-GROUP-MOVE.response
F-READ-LINK-ATTRIB.response						
F-CHANGE-LINK-ATTRIB.indication						
F-CHANGE-LINK-ATTRIB.response						
F-COPY.indication						
F-COPY.response						
F-MOVE.indication						
F-MOVE.response						
F-OPEN.indication						
F-OPEN.response (+ve)						
F-OPEN.response (-ve)						
F-CLOSE.indication						
F-CLOSE.response						
F-BEGIN-GROUP.indication						
F-BEGIN-GROUP.response						
F-END-GROUP.indication						
F-END-GROUP.response						
F-RECOVER.indication						
F-RECOVER.response (+ve)						
F-RECOVER.response (-ve)						
Bulk data transfer						
F-LOCATE.indication						
F-LOCATE.response						
F-ERASE.indication						
F-ERASE.response						

Table 10 – Sequence of service primitives for file service regimes – responder (continued)

Previous file service event	F-GROUP-LIST.response	F-GROUP-CHANGE-ATTRIB.response	F-SELECT .response	F-SELECT-ANOTHER .response	F-CREATE .response	F-CREATE-DIRECTORY .response	F-LINK .response
FTAM regime established							
F-CHANGE-PREFIX.indication							
F-CHANGE-PREFIX.response							
F-LIST.indication							
F-LIST.response							
F-GROUP-SELECT.indication							
F-GROUP-SELECT.response (+ve)							
F-GROUP-SELECT.response (-ve)							
F-GROUP-DELETE.indication							
F-GROUP-DELETE.response							
F-GROUP-COPY.indication							
F-GROUP-COPY.response							
F-GROUP-MOVE.indication							
F-GROUP-MOVE.response							
F-GROUP-LIST.indication	Grmn						
F-GROUP-LIST.response							
F-GROUP-CHANGE-ATTRIB.indication		Grmn					
F-GROUP-CHANGE-ATTRIB.response							
F-SELECT.indication			Kernel|Lfmg				
F-SELECT.response (+ve)							
F-SELECT.response (-ve)							
F-SELECT-ANOTHER.indication				Grmn			
F-SELECT-ANOTHER.response (+ve)							
F-SELECT-ANOTHER.response (-ve)							
F-CREATE.indication					Lmgt		
F-CREATE.response (+ve)							
F-CREATE.response (-ve)							
F-CREATE-DIRECTORY.indication						Efmg	
F-CREATE-DIRECTORY.response (+ve)							
F-CREATE-DIRECTORY.response (-ve)							
F-LINK.indication							Efmg
F-LINK.response (+ve)							
F-LINK.response (-ve)							
F-DESELECT.indication							
F-DESELECT.response							
F-DELETE.indication							
F-DELETE.response							
F-UNLINK.indication							
F-UNLINK.response							
F-READ-ATTRIB.indication							
F-READ-ATTRIB.response							
F-CHANGE-ATTRIB.indication							
F-CHANGE-ATTRIB.response							
F-READ-LINK-ATTRIB.indication							

8571 PT*3 86 ■ 4851903 0536323 387 ■
ISO 8571-3:1988/Amd.1:1992 (E)

Table 10 – Sequence of service primitives for file service regimes – responder (continued)

Previous file service event	F-GROUP-CHANGE-ATTRIB.response	F-GROUP-LIST.response	F-SELECT .response	F-SELECT-ANOTHER .response	F-CREATE .response	F-CREATE-DIRECTORY .response	F-LINK .response
F-READ-LINK-ATTRIB.response							
F-CHANGE-LINK-ATTRIB.indication							
F-CHANGE-LINK-ATTRIB.response							
F-COPY.indication							
F-COPY.response							
F-MOVE.indication							
F-MOVE.response							
F-OPEN.indication							
F-OPEN.response (+ve)							
F-OPEN.response (-ve)							
F-CLOSE.indication							
F-CLOSE.response							
F-BEGIN-GROUP.indication							
F-BEGIN-GROUP.response			Grouping	G-Grmn	G-Lmgt	G-Efmg	G-Efmg
F-END-GROUP.indication							
F-END-GROUP.response							
F-RECOVER.indication							
F-RECOVER.response (+ve)							
F-RECOVER.response (-ve)							
Bulk data transfer							
F-LOCATE.indication							
F-LOCATE.response							
F-ERASE.indication							
F-ERASE.response							

Table 10 – Sequence of service primitives for file service regimes – responder (continued)

Previous file service event	F-DESELECT .response	F-DELETE .response	F-UNLINK .response	F-READ-ATTRIB .response	F-CHANGE-ATTRIB .response	F-READ-LINK-ATTRIB .response
FTAM regime established						
F-CHANGE-PREFIX.indication						
F-CHANGE-PREFIX.response						
F-LIST.indication						
F-LIST.response						
F-GROUP-SELECT.indication						
F-GROUP-SELECT.response (+ve)						
F-GROUP-SELECT.response (-ve)						
F-GROUP-DELETE.indication						
F-GROUP-DELETE.response						
F-GROUP-COPY.indication						
F-GROUP-COPY.response						
F-GROUP-MOVE.indication						
F-GROUP-MOVE.response						
F-GROUP-LIST.indication						
F-GROUP-LIST.response						
F-GROUP-CHANGE-ATTRIB.indication						
F-GROUP-CHANGE-ATTRIB.response						
F-SELECT.indication						
F-SELECT.response (+ve)	Grouping|G-Lfmg	G-Lmgt|G-Efmg	G-Efmg	G-Lmgt|G-Lfmg	G-Emgt|G-Efmg	G-Lfmg
F-SELECT.response (-ve)						
F-SELECT-ANOTHER.indication						
F-SELECT-ANOTHER.response (+ve)	Grouping|G-Lfmg	G-Lmgt|G-Efmg	G-Efmg	G-Lmgt|G-Lfmg	G-Emgt|G-Efmg	G-Lfmg
F-SELECT-ANOTHER.response (-ve)						
F-CREATE.indication						
F-CREATE.response (+ve)	Grouping	G-Lmgt		G-Lmgt	G-Emgt	
F-CREATE.response (-ve)						
F-CREATE-DIRECTORY.indication						
F-CREATE-DIRECTORY.response(+ve)	Grouping|G-Efmg	G-Efmg		G-Efmg	G-Efmg	
F-CREATE-DIRECTORY.response (-ve)						
F-LINK.indication						
F-LINK.response (+ve)	Grouping|G-Efmg	G-Efmg	G-Efmg	G-Efmg	G-Efmg	G-Efmg
F-LINK.response (-ve)						
F-DESELECT.indication	Kernel|Lfmg					
F-DESELECT.response						
F-DELETE.indication		Lmgt|Efmg				
F-DELETE.response						
F-UNLINK.indication			Efmg			
F-UNLINK.response						
F-READ-ATTRIB.indication				Lmgt|Lfmg		
F-READ-ATTRIB.response	Grouping|G-Lfmg	G-Lmgt|G-Efmg	G-Efmg		G-Emgt|G-Efmg	G-Lfmg
F-CHANGE-ATTRIB.indication					Emgt|Efmg	
F-CHANGE-ATTRIB.response	Grouping|G-Efmg	G-Lmgt|G-Efmg	G-Efmg	G-Lmgt|G-Efmg	G-Emgt|G-Efmg	G-Efmg
F-READ-LINK-ATTRIB.indication						Lfmg

8571 PT*3 88 ■ 4851903 0536325 15T ■

Table 10 – Sequence of service primitives for file service regimes – responder (continued)

Previous file service event	F-DESELECT .response	F-DELETE .response	F-UNLINK .response	F-READ-ATTRIB .response	F-CHANGE-ATTRIB .response	F-READ-LINK-ATTRIB .response
F-READ-LINK-ATTRIB.response	G-Lfmg|G-Efmg	G-Lmgt|G-Lfmg	G-Efmg	G-Lmgt|G-Lfmg	G-Emgt|G-Efmg	G-Lfmg
F-CHANGE-LINK-ATTRIB.indication						
F-CHANGE-LINK-ATTRIB.response	G-Lfmg|G-Efmg	G-Efmg	G-Efmg	G-Efmg	G-Emgt	G-Efmg
F-COPY.indication						
F-COPY.response	G-Lfmg|Grouping	G-Lmgt|G-Efmg	G-Efmg	G-Lmgt|G-Lfmg	G-Emgt|G-Efmg	G-Lfmg
F-MOVE.indication						
F-MOVE.response	G-Lfmg|Grouping	G-Lmgt|G-Efmg	G-Efmg	G-Lmgt|G-Lfmg	G-Emgt|G-Efmg	G-Lfmg
F-OPEN.indication						
F-OPEN.response (+ve)						
F-OPEN.response (-ve)						
F-CLOSE.indication						
F-CLOSE.response	Grouping	G-Lmgt	G-Efmg	G-Lmgt	G-Emgt	G-Lfmg
F-BEGIN-GROUP.indication						
F-BEGIN-GROUP.response	G-Lfmg|Group[2]	G-Lmgt[2]|G-Efmg	G-Efmg [2]	G-Lmgt[6]|G-lfmg	G-emgt[6]|G-efmg	G-Lfmg [2]
F-END-GROUP.indication						
F-END-GROUP.response						
F-RECOVER.indication						
F-RECOVER.response (+ve)						
F-RECOVER.response (-ve)						
Bulk data transfer						
F-LOCATE.indication						
F-LOCATE.response						
F-ERASE.indication						
F-ERASE.response						

Table 10 – Sequence of service primitives for file service regimes – responder (continued)

Previous file service event	F-CHANGE-LINK-ATTRIB .response	F-COPY .response	F-MOVE .response	F-OPEN .response	F-CLOSE .response	F-BEGIN-GROUP .response
FTAM regime established						
F-CHANGE-PREFIX.indication						
F-CHANGE-PREFIX.response						
F-LIST.indication						
F-LIST.response						
F-GROUP-SELECT.indication						
F-GROUP-SELECT.response (+ve)						
F-GROUP-SELECT.response (-ve)						
F-GROUP-DELETE.indication						
F-GROUP-DELETE.response						
F-GROUP-COPY.indication						
F-GROUP-COPY.response						
F-GROUP-MOVE.indication						
F-GROUP-MOVE.response						
F-GROUP-LIST.indication						
F-GROUP-LIST.response						
F-GROUP-CHANGE-ATTRIB.indication						
F-GROUP-CHANGE-ATTRIB.response						
F-SELECT.indication						
F-SELECT.response (+ve)	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-SELECT.response (-ve)						
F-SELECT-ANOTHER.indication						
F-SELECT-ANOTHER.response (+ve)	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-SELECT-ANOTHER.response (-ve)						
F-CREATE.indication						
F-CREATE.response (+ve)		G-Obmn	G-Obmn	Grouping		
F-CREATE.response (-ve)						
F-CREATE-DIRECTORY.indication						
F-CREATE-DIRECTORY.response (+ve)		G-Obmn	G-Obmn			
F-CREATE-DIRECTORY.response (-ve)						
F-LINK.indication						
F-LINK.response (+ve)	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-LINK.response (-ve)						
F-DESELECT.indication						
F-DESELECT.response						
F-DELETE.indication						
F-DELETE.response						
F-UNLINK.indication						
F-UNLINK.response						
F-READ-ATTRIB.indication						
F-READ-ATTRIB.response	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-CHANGE-ATTRIB.indication						
F-CHANGE-ATTRIB.response	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-READ-LINK-ATTRIB.indication						

8571 PT*3 88 ■ 4851903 0536327 T22 ■
ISO 8571-3:1988/Amd.1:1992 (E)

Table 10 – Sequence of service primitives for file service regimes – responder (continued)

Previous file service event	F-CHANGE-LINK-ATTRIB .response	F-COPY .response	F-MOVE .response	F-OPEN .response	F-CLOSE .response	F-BEGIN-GROUP .response
F-READ-LINK-ATTRIB.response	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-CHANGE-LINK-ATTRIB.indication	Efmg					
F-CHANGE-LINK-ATTRIB.response	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-COPY.indication		Obmn				
F-COPY.response	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-MOVE.indication			Obmn			
F-MOVE.response	G-Efmg	G-Obmn	G-Obmn	Grouping		
F-OPEN.indication				Kernel		
F-OPEN.response (+ve)					Grouping	
F-OPEN.response (-ve)						
F-CLOSE.indication					Kernel	
F-CLOSE.response	G-Efmg	G-Obmn	G-Obmn			
F-BEGIN-GROUP.indication						
F-BEGIN-GROUP.response	G-Efmg [2]	G-Obmn [2]	G-Obmn [2]		Grouping [3]	
F-END-GROUP.indication						Grouping
F-END-GROUP.response						
F-RECOVER.indication						
F-RECOVER.response (+ve)						
F-RECOVER.response (-ve)						
Bulk data transfer						
F-LOCATE.indication						
F-LOCATE.response						
F-ERASE.indication						
F-ERASE.response						

Table 10 – Sequence of service primitives for file service regimes – responder (continued)

Previous file service event	F-END-GROUP .response	F-RECOVER .response	Bulk data transfer	F-LOCATE .response	F-ERASE .response	Terminate FTAM Regime
FTAM regime established						
F-CHANGE-PREFIX.indication						
F-CHANGE-PREFIX.response						
F-LIST.indication						
F-LIST.response						
F-GROUP-SELECT.indication						
F-GROUP-SELECT.response (+ve)						
F-GROUP-SELECT.response (-ve)						
F-GROUP-DELETE.indication						
F-GROUP-DELETE.response						
F-GROUP-COPY.indication						
F-GROUP-COPY.response						
F-GROUP-MOVE.indication						
F-GROUP-MOVE.response						
F-GROUP-LIST.indication						
F-GROUP-LIST.response						
F-GROUP-CHANGE-ATTRIB.indication						
F-GROUP-CHANGE-ATTRIB.response						
F-SELECT.indication						
F-SELECT.response (+ve)	Grouping					
F-SELECT.response (-ve)	Grouping					
F-SELECT-ANOTHER.indication						
F-SELECT-ANOTHER.response (+ve)	Grouping					
F-SELECT-ANOTHER.response (-ve)	Grouping					
F-CREATE.indication						
F-CREATE.response (+ve)	Grouping					
F-CREATE.response (-ve)	Grouping					
F-CREATE-DIRECTORY.indication						
F-CREATE-DIRECTORY.response (+ve)	Grouping					
F-CREATE-DIRECTORY.response (-ve)	Grouping					
F-LINK.indication						
F-LINK.response (+ve)	Grouping					
F-LINK.response (-ve)	Grouping					
F-DESELECT.indication						
F-DESELECT.response	Grouping					
F-DELETE.indication						
F-DELETE.response	Grouping					
F-UNLINK.indication						
F-UNLINK.response	Grouping					
F-READ-ATTRIB.indication						
F-READ-ATTRIB.response	Grouping					
F-CHANGE-ATTRIB.indication						
F-CHANGE-ATTRIB.response	Grouping					
F-READ-LINK-ATTRIB.indication						

8571 PT*3 88 ■ 4851903 0536329 8T5 ■
ISO 8571-3:1988/Amd.1:1992 (E)

8571 PT*3 88 ■ 4851903 0536330 517 ■

Table 10 – Sequence of service primitives for file service regimes – responder (concluded)

Previous file service event	F-END-GROUP .response	F-RECOVER .response	Bulk data transfer	F-LOCATE .response	F-ERASE .response	Terminate FTAM Regime
F-READ-LINK-ATTRIB.response	Grouping					
F-CHANGE-LINK-ATTRIB.indication						
F-CHANGE-LINK-ATTRIB.response	Grouping					
F-COPY.indication						
F-COPY.response	Grouping					
F-MOVE.indication						
F-MOVE.response	Grouping					
F-OPEN.indication						
F-OPEN.response (+ve)	Grouping					
F-OPEN.response (-ve)	Grouping					
F-CLOSE.indication						
F-CLOSE.response						
F-BEGIN-GROUP.indication						
F-BEGIN-GROUP.response						
F-END-GROUP.indication						
F-END-GROUP.response						
F-RECOVER.indication		Recover				
F-RECOVER.response (+ve)						
F-RECOVER.response (-ve)						
Bulk data transfer						
F-LOCATE.indication				Access		
F-LOCATE.response						
F-ERASE.indication					Access	
F-ERASE.response						

13 Common file service parameters

13.5 Attributes

(amend 1st paragraph, page 20)

The attributes parameter conveys a list of object attribute names and object attribute values associated with the object. It also may convey a pathname used to access the object. F–INITIALIZE negotiates the attribute groups which are available for the duration of the FTAM regime. Subsequent primitives only use those attributes which were negotiated.

NOTE – The pathname element (not to be confused with the primary pathname element) of the attributes parameter is not an object attribute. An object may be accessible by several pathnames. The pathname element conveys the actual pathname identifying the object, and is used in conjunction with the current pathname activity attribute. It is used to change the object name attribute, and may effect the primary pathname attribute.

The object name attribute may be derived from the last graphicstring element in the pathname vector. If the object is being accessed by a reference, the object name will be that of the reference object, not the target object.

(amend 2nd paragraph, page 20)

On primitives other than F–SELECT issued by the initiator, the parameter conveys new values to be assigned to the object attributes. On primitives issued by the responder, the parameter reports the current values of the object attributes, or indications that there is no value available.

(amend 3rd paragraph, page 20)

The pathname element does not always correspond to the primary pathname object attribute. For existing objects, this element identifies a given object by any valid pathname, and sets the current pathname activity identifier. The pathname element on F–SELECT identifies the object to be selected, or on F–CREATE the pathname of the object to be created and selected. In the request and indication primitives it indicates the object required, and in the response and confirm primitives it indicates the object actually selected.

NOTES

1) If, for example, the object pathname requested gave a generic name or a generation name, the name selected might differ from that requested.

2) The pathname specified on create actions will not necessarily be the primary pathname of the object being created.

(amend 4th paragraph, page 20)

The object attributes, the range of values they may take and the actions to be taken are defined in the virtual filestore definition (ISO 8571–2). Annex B specifies which object attributes may be manipulated by each primitive.

13.6 Requested access

(amend 1st paragraph, page 20)

The requested access parameter indicates how objects affected by the operation will be accessed. The value gives, as a vector, the actions to be performed on objects affected by the service. The elements of the vector correspond to the read, insert, replace, extend, erase, read attribute, change attribute, and delete object actions, and each element indicates whether the action is required or not.

(amend 2nd paragraph's note, page 20)

NOTE – When objects are targeted for selection, either individually by a select regime establishment or recovery actions, or by a generalized selection regime, the requested access parameter is used to state the maximum facilities the current user will require during the whole select regime. These access requirements may be password protected by the access control object attribute which is matched to the access passwords parameter (see clause 13.7).

(amend 3rd paragraph, page 21)

The requested access may be further qualified by any concurrency control requested, which specifies how multiple users are to be restricted when requiring access to the same file object. The requested access parameter includes only those actions to be performed by the user; any action not included is not available later in the select regime.

(amend 4th paragraph, page 21)

The relationship between attributes and actions is detailed in ISO 8571–2. This parameter is used to set the current access request activity attribute, or in a generalized selection service, the generalized access request activity attribute.

13.7 Access passwords

(amend 1st paragraph, page 21)

The access passwords parameter provides passwords associated with the actions specified in the requested access parameter. Passwords for both the object's access controls and the specified pathname's path access controls may be specified. This parameter is only available if the security attribute group has been negotiated. The range of values the parameter takes is equal to that defined for the current access passwords activity attribute. This parameter is used to set the current access passwords activity attribute, or in a generalized selection service, the generalized access passwords activity attribute.

13.8 Concurrency control

(append after 1st paragraph, page 21)

NOTE – Concurrency control in this standard is only defined for actions on file object contents.

(append after clause 13.13, page 22)

13.14 Attribute names

The attribute names parameter indicates which object attributes from the kernel or negotiated attribute groups, given in the virtual filestore definition, are to be read. The parameter is a list, each element of which names an object attribute defined for any object type, as in ISO 8571–2. This parameter is not used to set any activity attributes.

13.15 Attribute value assertions

The attribute value assertions parameter consists of an attribute value assertion list. Its value is a set of legally related lists of attribute patterns that are used for matching objects. When used in conjunction with the generalized selection mechanism, the results of this parameter are used to set the generalized selection group activity attributes.

13.16 Create password

The create password is an optional parameter which may be required to establish that the user identified by the initiator identity parameter of the FTAM regime establishment service has permission to create objects in the current filestore. This parameter is not used to set any activity attributes.

13.17 Destination file-directory

The destination file-directory parameter is used to identify a target destination for one or more operations. The range of values this parameter may take is that for a valid pathname. The pathname must identify an existing file-directory object within the current filestore. This parameter does not set any activity attributes except when used with the change current name prefix service, in which case it affects the current name prefix activity attribute.

13.18 Objects attributes list

The objects attributes list parameter consists of a set of object attribute lists, as defined in clause 13.5. This parameter is returned as a part of a response or confirm primitive. It does not set any activity attributes.

13.19 Override

The override parameter defined the action to be taken if the named object already exists. The values of the parameter are:

a) fail the create action if the object already exists;

b) select the object if it already exists;

c) delete the object if it already exists and create a new object using the old object's attributes (effectively erase any contents of the existing object and select it);

d) delete the object if it already exists and create a new object using the initial attributes provided on the filestore service primitive.

The values (c) and (d) are dependent upon the provision of a delete password in the access passwords parameter where required by the access control attribute of the existing object. Not every creation service supports every override option. This parameter is not used to set any activity attributes.

13.20 Error Action

The error action parameter is used by services dealing with more than one object at a time via the generalized selection group. It carries instructions as to how to proceed when an error is detected by the responder. This parameter may take on the following values:

a) terminate – this value indicates that if an error is detected during the service, that all processing should cease, the action result should indicate the error encountered, and suitable diagnostics be returned. Any actions taken up to the point of error detection are not rolled back.

b) continue – this value indicates that if an error is detected during the service, that the service continue with the next object in the generalized selection list. Upon completion of the service, the action result should contain an informational message noting that one or more errors had occurred, and one or more diagnostic messages should be generated for each object for which an error was encountered.

13.21 Last member indicator

The Last Member indicator parameter is used by the responder on F-SELECT-ANOTHER primitive, to indicate that the currently selected object is last in the group of objects selected through generalized selection. This parameter can be set by the responder to indicate to the initiator that the next select request may result in a selection of an object with a previously selected pathname from the pathname group.

13.22 Scope

The scope parameter is used by the initiator to indicate the scope of objects for retrieving, on F-LIST and F-GROUP-SELECT services. It specifies the root file-directory of the retrieval scope, which may be a subtree in the whole filestore, by means of a complete or incomplete pathname (a reference object linked to a file-directory is allowed, in which case the referenced file-directory is a root of the retrieval scope subtree). It also indicates whether the retrieval scope is direct child objects of the specified root file-directory or all descendants within the whole subtree.

13.23 Referent Indicator

The referent indicator parameter is used by the responder in F-SELECT-ANOTHER (and F-SELECT) primitives to indicate that the currently selected object has been selected such that the last object in the pathname is a reference. This parameter may be set to true by the responder to indicate that the current pathname used is a reference. Note that the parameter is optional in all cases to allow for responders to conceal the existence of references for security purposes.

13.24 Request operation result

The request operation result parameter is used by the initiator on the F-GROUP-DELETE, F-GROUP-MOVE and F-GROUP-COPY primitives to indicate that result of the operations within the grouped services be returned. The value of this parameter is set by the initiator to request either an overall summary of the operation or the list of original pathnames successfully acted upon in the service.

13.25 Operation result

The operation result parameter is used by the responder on F-GROUP-DELETE, F-GROUP-MOVE and F-GROUP-COPY primitives to return the result of the operations within these group services. The value will be a choice of either the number of the files successfully acted upon or the complete list of the original pathnames.

14.1.2.12 Functional units

(amend 1st paragraph, page 24)

The requested functional units parameter negotiates the set of file service functional units, excluding the kernel, to be available from the negotiated service class on the application association (see clauses 8.1 and 10.3). On the request and indication primitives the parameter carries the full functional unit capability of the initiator. On the response and confirm primitives it carries all the functional units to be available on the association. When the required service class permits, the set may contain the optional functional units "read", "write", "file access", "limited file management", "enhanced file management", "limited filestore management", "enhanced filestore management", "object manipulation", "group manipulation", "grouping", and "FADU locking". For use by the internal file service, if the service class permits, it may also contain the "recovery" and "restart" functional units. The recovery and restart functional units are only visible in the IFS and are thus never explicitly requested in the external file service. They are indirectly requested via the FTAM quality of service parameter (see clause 10.2). This parameter is not used to set any activity attributes

(insert prior to clause 15, page 27)

14a Filestore services

These services provide filestore management capability to the file service user. They are not concerned with the establishment of regimes. The services provide facilities for interrogation of pathnames and attributes of objects in the filestore, and modification of the current name prefix activity attribute.

14a.1 Change current name prefix

14a.1.1 Function

This service changes the responder's value of the current name prefix activity attribute by specifying a pathname of a file-directory object within the virtual filestore. This primitive may only be issued outside of any object selection regime. Upon receiving the F–CHANGE–PREFIX indication, the responder verifies that the indicated file-directory exists, the initiator has "pass" access through the pathname specified, and the initiator has at least one object access permission ("read", "insert", "replace", "extend", "erase", "read attribute", "change attribute", "delete", or "link") when using that pathname. If these three conditions are met, the responder changes its current name prefix activity attribute to that pathname. The result of the operation is then returned to the initiator by means of the F–CHANGE–PREFIX response primitive's action result parameter.

14a.1.2 Types of primitives and parameters

Table 14a indicates the types of primitives and the parameters needed for changing the current name prefix.

14a.1.2.1 Action result

The action result parameter is defined in clause 13.2.

14a.1.2.2 Destination file-directory

The destination file-directory parameter is defined in clause 13.17. It identifies an existing file-directory object to which the current name prefix activity attribute is to be set. The responding application may optionally return a value confirming the newly set current name prefix.

14a.1.2.3 Reset

The reset parameter is a boolean value indicating whether a destination file-directory in the format of an incomplete pathname should be resolved from the existing current name prefix, or from the current name prefix assigned at the creation of the FTAM regime. When the destination file directory is specified in the form of a complete pathname the reset parameter is not significant. A value of true implies that the current name prefix should be reset to the current name prefix assigned at the creation of the FTAM regime prior to attempting to set it from the supplied destination file-directory parameter.

Note that a reset value of true and a destination file-directory value of an incomplete pathname containing no elements resets the current name prefix to the value assigned at the creation of the FTAM regime.

14a.1.2.4 Access passwords

The access passwords parameter is defined in clause 13.7. This parameter is used to establish access rights to the specified directory by the filestore user identified on the F–INITIALIZE service primitive. This parameter does not set any activity attributes, nor is any access assumed in any later services based on of the value of this parameter.

14a.1.2.5 Diagnostic

The diagnostic parameter is defined in clause 13.13.

14a.2 List file-directory

Table 14a – F–CHANGE–PREFIX parameters

Parameter	F-CHANGE-PREFIX request	F-CHANGE-PREFIX indication	F-CHANGE-PREFIX response	F-CHANGE-PREFIX confirm
Action result			Mandatory	Mandatory
Destination file-directory	Mandatory	Mandatory(=)	Optional	Optional(=)
Reset	Mandatory	Mandatory(=)		
Access passwords	Optional	Optional(=)		
Diagnostic			Optional	Optional

Table 14b – F–LIST parameters

Parameter	F-LIST request	F-LIST indication	F-LIST response	F-LIST confirm
Action result			Mandatory	Mandatory
Attribute value assertions	Mandatory	Mandatory(=)		
Access passwords	Optional	Optional(=)		
Attribute names	Mandatory	Mandatory(=)		
Scope	Mandatory	Mandatory(=)		
Objects attributes list			Conditional	Conditional(=)
Diagnostic			Optional	Optional

14a.2.1 Function

This service enumerates objects whose attributes satisfy the attribute value assertions given. The request and indication service primitives convey an attribute value assertion list. The action fails if

a) no object exists as specified by the attribute value assertion list;

b) the attribute value assertion list references attributes from attribute groups that were not successfully negotiated.

Upon receiving the F–LIST indication, the filestore provider searches the filestore for objects conforming to the attribute value assertion list, and reads the requested attributes of each object identified. The results are then reported by issuing the F–LIST response primitive.

NOTES

1) According to the definition of the attribute value assertion list search mechanism (ISO 8571-2, clause 9a), objects to which this filestore user does not have read-attribute access are not reported, as well as any objects under file-directories to which the filestore user does not have pass and read access permission.

2) This service does not detect duplications of objects due to multiple pathnames. If the same object is accessible by multiple pathnames, and more than one of the pathnames conform to the attribute value assertion list, then the object will be reported on multiple times, once by each conforming pathname.

3) It should not be considered a requirement that the entire real filestore be searched for objects conforming to an attribute value assertion list. Real implementations may incorporate search optimizations, such as limiting the scope of the search to file-directories described in the attribute value assertion list.

14a.2.2 Types of primitives and parameters

Table 14b indicates the types of primitives and the parameters needed for listing the contents of directories.

14a.2.2.1 Action result

The action result parameter is defined in clause 13.2.

14a.2.2.2 Attribute value assertions

The attribute value assertions parameter is defined in clause 13.15. It is used to restrict the list of objects to those which match the attribute value assertion list.

14a.2.2.3 Access passwords

The access passwords parameter is defined in clause 13.7. It is used to establish read-attribute access permission to the objects being listed, and read access permission to the file-directories containing them.

14a.2.2.4 Attribute names

The attribute names parameter is defined in clause 13.14.

14a.2.2.5 Objects attributes list

The objects attributes list parameter is defined in clause 13.18. It is used to return the list of objects conforming to the attribute value assertion list, and their corresponding attributes. The presence of the object attributes list parameter is conditional on the inclusion in the attribute names parameter of at least one attribute with a returnable value.

14a.2.2.6 Scope

This parameter is described in clause 13.22.

14a.2.2.7 Diagnostic

The diagnostic parameter is defined in clause 13.13.

14b Generalized filestore services

These services provide the mechanism for identifying a group of object pathnames by attribute value assertions, then operating on the objects identified by the group as a whole. They can be moved to a different location within the filestore, duplicated within the filestore, their attributes can be modified, or the pathnames of the group can be interrogated.

Members of the identified group can also be operated on individually under the object select regime.

14b.1 Generalized selection

14b.1.1 Function

The generalized selection service identifies a group of pathnames matching a given attribute value assertions list. Only existing objects of type 'file' and references to objects of type 'file' are considered for generalized selection. The group of pathnames identified consists of all pathnames:

a) whose object attributes satisfy the given attribute value assertion list; and

b) for which the initiating user demonstrates authorization for access to the identified object. Generalized selection fails if there are no objects that satisfy the given attribute value assertions.

NOTES

1) According to the definition of the attribute value assertion list search mechanism (ISO 8571-2, clause 9a), objects to which this filestore user does not have read-attribute access are not included, as well as any objects under file-directories to which the filestore user does not have pass and read access permission.

2) This service does not detect duplications of objects due to multiple pathnames. If the same object is accessible by multiple pathnames, and more than one of the pathnames conform to the attribute value assertion list, then the object will be included multiple times, once by each conforming pathname.

3) It should not be considered a requirement that the entire real filestore be searched for objects conforming to an attribute value assertion list. Real implementations may incorporate search optimizations, such as limiting the scope of the search to file-directories described in the attribute value assertion list.

These primitives may only be issued within an FTAM regime. No other regimes may be currently established when issuing this service. The filestore provider performs the generalized selection action after receiving the F–GROUP–SELECT indication and before issuing an F–GROUP–SELECT response primitive with an action result indicating success.

14b.1.2 Types of primitives and parameters

Table 14c indicates the types of primitives and the parameters needed for generalized selection.

14b.1.2.1 Action result

The action result parameter is defined in clause 13.2.

14b.1.2.2 State result

The state result parameter is defined in clause 13.1.

14b.1.2.3 Attribute value assertions

The attribute value assertions parameter is defined in clause 13.15. This attribute value assertion list identifies the group of pathnames used to set the generalized select group activity attribute.

14b.1.2.4 Requested access

The Requested access parameter is defined in clause 13.6. This parameter is used to identify actions that may be performed on the objects identified by the pathnames considered for the generalized select group. When taken in conjunction with the access passwords parameter, pathnames included in the group must identify objects for which the initiating filestore user has permissions for the access requested.

This parameter also sets the generalized access request activity attribute, used to establish select regimes on objects taken from the generalized select group.

14b.1.2.5 Access passwords

The access passwords parameter is defined in clause 13.7. It's presence is conditional on the security attribute group being successfully negotiated. This parameter is used to verify access permission to the objects identified by the generalized select group. It also sets the generalized access passwords activity attribute, used to establish select regimes on objects taken from the generalized select group.

14b.1.2.6 Concurrency control

The concurrency control parameter is defined in clause 13.8. No concurrency control is set during the establishment of the generalized selection regime.

Table 14c – F–GROUP–SELECT parameters

Parameter	F-GROUP-SELECT request	F-GROUP-SELECT indication	F-GROUP-SELECT response	F-GROUP-SELECT confirm
Action result			Mandatory	Mandatory
Attribute value assertions	Mandatory	Mandatory(=)		
Requested access	Mandatory	Mandatory(=)		
Access passwords	Conditional	Conditional(=)		
Concurrency control	Optional	Optional(=)		
Maximum set size	Optional	Optional(=)		
Scope	Mandatory	Mandatory(=)		
Account	Optional	Optional(=)		
Shared ASE information	Optional	Optional(=)	Optional	Optional(=)
Diagnostic			Optional	Optional

This parameter is only used to set the generalized concurrency control activity attribute, used to establish select regimes on objects taken from the generalized select group.

14b.1.2.7 Maximum set size

The maximum set size parameter gives the maximum number of pathnames in the generalized select group that the initiator will handle. Its value is an integer. If no value is supplied, then no restriction is to be placed on the group size. If the number of pathnames in the generalized select group exceeds this bound, then the generalized selection regime is not established, no generalized select group is established, and an error is reported to the initiator.

14b.1.2.8 Account

The account parameter is defined in clause 13.3. The value given overrides any value previously set by the F–INITIALIZE request for the duration of the generalized selection regime. This account may be explicitly overridden in nested regimes.

14b.1.2.9 Shared ASE information

The shared ASE information parameter is defined in clause 13.10.

14b.1.2.10 Diagnostic

The diagnostic parameter is defined in clause 13.13.

14b.1.2.11 Scope

This parameter is described in clause 13.22.

14b.2 Generalized deletion

14b.2.1 Function

Table 14d – F–GROUP–DELETE parameters

Parameter	F-GROUP-DELETE request	F-GROUP-DELETE indication	F-GROUP-DELETE response	F-GROUP-DELETE confirm
Action result			Mandatory	Mandatory
Charging			Conditional	Conditional(=)
Request operation result	Optional	Optional(=)		
Operation result			Conditional	Conditional(=)
Shared ASE information	Optional	Optional(=)	Optional	Optional(=)
Diagnostic			Optional	Optional

The F–GROUP–DELETE service removes the file objects and resets to null the generalized select group activity attribute value. The file objects are removed in such a way that the objects cease to exist and hence are not available for selection.

NOTE – Implementations should be aware that this is a destructive service and will delete all the files whose pathnames are in the current selection group from the filestore.

This primitive may only be used within the FTAM regime.

The filestore provider performs the generalized deletion action after receiving the F–GROUP–DELETE indication primitive, and before issuing the F–GROUP–DELETE response primitive.

14b.2.2 Types of primitives and parameters

Table 14d indicates the types of primitives and the parameters needed for generalized deletion.

14b.2.2.1 Action result

The action result parameter is defined in clause 13.2.

14b.2.2.2 Charging

The charging parameter is defined in clause 13.4. Charges are only reported against an overriding account set up at regime establishment.

14b.2.2.3 Request operation result

The request opertaion result parameter is defined in 13.24. This parameter is optional.

14b.2.2.4 Operation result

The operation result parameter is defined in 13.25. This parmeter is conditional on the presence of the request operation result parameter.

14b.2.2.5 Shared ASE information

The shared ASE information parameter is defined in clause 13.10.

14b.2.2.6 Diagnostic

The diagnostic parameter is defined in clause 13.13.

14b.3 Group move

14b.3.1 Function

The F–GROUP–MOVE service moves the file objects identified by the generalized select group by changing their location in the filestore structure and resets to null the generalized select group activity attribute value.. The destination is specified by the destination file-directory parameter. This parameter changes the primary pathname object attribute of all objects in the group. Other object attributes may also be changed by use of the attributes parameter.

Moving an object consists of entering a parenthood relationship for the given object into a named file-directory, removing the parenthood relationship for that object from the old parent file-directory for that object, and updating the primary pathname attribute of the object to reflect the new primary pathname.

When references to files exist in the generalized selection group, the file object is moved to the new location, not the reference.

Prior to moving each object, the filestore user identified in the F–INITIALIZE service (along with the access passwords in the generalized access passwords activity attribute) must have change-attribute permission to the object via the corresponding pathname. Also, the filestore user (along with the create password) must have permission to create objects in this filestore, and insert permission in the destination file-directory.

The F–GROUP–MOVE service is not an atomic action. Move operations are performed one at a time, until either an error occurs or the list of pathnames is exhausted. The error action parameter determines whether or not an error terminates the service. A shared ASE information parameter is provided for use should the use of some other ASE such as CCR be desired to form an atomic action from this service.

14b.3.2 Types of primitives and parameters

Table 14e indicates the types of primitives and the parameters needed for group move operations.

14b.3.2.1 Action result

The action result parameter is defined in clause 13.2.

14b.3.2.2 Destination file-directory

The destination file-directory parameter is defined in clause 13.17. It identifies the file-directory that should maintain the parenthood relationship to the objects identified by the pathnames in the generalized select group.

14b.3.2.3 Override

The override parameter is defined in clause 13.19 and only applies to the create part of the action. Only the following override options are supported for the F–GROUP–MOVE service:

a) fail the action if the file already exists;

b) delete the object if it already exists and create a new object using the initial attributes provided on the filestore service primitive.

14b.3.2.4 Error action

The error action parameter is described in clause 13.20.

14b.3.2.5 Request operation result

The request operation result parameter is defined in 13.24. This parameter is optional.

14.b.3.2.6 Operation result

The operation result parameter is defined in 13.25. This parameter is conditional on the presence of the request operation result parameter.

14b.3.2.7 Create password

The create password parameter is described in clause 13.16.

14b.3.2.8 Access passwords

The access passwords parameter is described in clause 13.7. It is used to establish delete access to any objects which may be overridden by this service. The presence of the access passwords parameter is conditional on the successful negotiation of the security attribute group during the establishment of the FTAM regime.

14b.3.2.9 Attributes

The attributes parameter is described in clause 13.5. This parameter is used to set any additional attributes (other than primary pathname) on the objects being moved.

14b.3.2.10 Diagnostic

The diagnostic parameter is described in clause 13.13.

14b.4 Group copy

14b.4.1 Function

The F–GROUP–COPY service copies the file objects identified by the generalized select group by creating new objects with duplicate contents in the specified destination location in the filestore structure and the value of the generalized select group activity attribute remains unchanged. The destination is specified by the destination file-directory parameter. Object attributes may also be changed by use of the attributes parameter.

When references to files exist in the generalized selection group, the file object is duplicated into the new location, not the reference.

Prior to copying each object, the filestore user identified in the F–INITIALIZE service (along with the access passwords in the generalized access passwords activity attribute) must have read-attributes and read permission to the object via the corresponding pathname. Also, the filestore user (along with the create password) must have permission to create objects in this filestore, and insert permission in the destination file-directory.

The F–GROUP–COPY service is not an atomic action. Copy operations are performed one at a time, until either an error occurs or the list of pathnames is exhausted. The error action parameter determines whether or not an error terminates the service. A shared ASE information parameter is provided for use

Table 14e – F–GROUP–MOVE parameters

Parameter	F-GROUP-MOVE request	F-GROUP-MOVE indication	F-GROUP-MOVE response	F-GROUP-MOVE confirm
Action result			Mandatory	Mandatory
Destination file-directory	Mandatory	Mandatory(=)	Optional	Optional(=)
Override	Mandatory	Mandatory(=)		
Error action	Mandatory	Mandatory(=)		
Request operation result	Optional	Optional(=)		
Operation result			Conditional	Conditional(=)
Create password	Optional	Optional(=)		
Access passwords	Conditional	Conditional(=)		
Attributes	Optional	Optional(=)	Optional	Optional(=)
Diagnostic			Optional	Optional

Table 14f – F–GROUP–COPY parameters

Parameter	F-GROUP-COPY request	F-GROUP-COPY indication	F-GROUP-COPY response	F-GROUP-COPY confirm
Action result			Mandatory	Mandatory
Destination file-directory	Mandatory	Mandatory(=)	Optional	Optional(=)
Override	Mandatory	Mandatory(=)		
Error action	Mandatory	Mandatory(=)		
Request operation result	Optional	Optional(=)		
Operation result			Conditional	Conditional(=)
Create password	Optional	Optional(=)		
Access passwords	Conditional	Conditional(=)		
Attributes	Optional	Optional(=)		
Diagnostic			Optional	Optional

should the use of some other ASE such as CCR be desired to form an atomic action from this service.

14b.4.2 Types of primitives and parameters

Table 14f indicates the types of primitives and the parameters needed for group copy operations.

14b.4.2.1 Action result

The action result parameter is defined in clause 13.2.

14b.4.2.2 Destination file-directory

The destination file-directory parameter is defined in clause 13.17. It identifies the file-directory in which the duplicate objects are to be created.

14b.4.2.3 Override

The override parameter is defined in clause 13.19 and only applies to the create part of the action. Only the following override options are supported for the F–GROUP–COPY service:

a) fail the action if the file already exists;

b) delete the object if it already exists and create a new object using the initial attributes provided on the filestore service primitive.

14b.4.2.4 Error action

The error action parameter is described in clause 13.20.

14b.4.2.5 Request operation result

The request operation result parameter is defined in 13.24. This parameter is optional.

14.b.4.2.6 Operation result

The operation result parameter is defined in 13.25. This parameter is conditional on the presence of the request operation result parameter.

14b.4.2.7 Create password

The create password parameter is described in clause 13.16.

14b.4.2.8 Access passwords

The access passwords parameter is described in clause 13.7. It is used to establish delete access to any objects which may be overridden by this service. The presence of the access passwords parameter is conditional on the successful negotiation of the security attribute group during the establishment of the FTAM regime.

14b.4.2.9 Attributes

The attributes parameter is described in clause 13.5. This parameter is used to set any additional attributes (other than primary pathname) on the object being copied.

14b.4.2.10 Diagnostic

The diagnostic parameter is described in clause 13.13.

14b.5 Group list

14b.5.1 Function

The F–GROUP–LIST service returns the set of pathnames maintained by the filestore provider in the generalized select group activity attribute, and selected object attributes of the represented objects.

Table 14g – F–GROUP–LIST

Parameter	F-GROUP-LIST request	F-GROUP-LIST indication	F-GROUP-LIST response	F-GROUP-LIST confirm
Action result	Mandatory	Mandatory(=)	Mandatory	Mandatory
Attribute names				
Objects attributes list			Conditional	Conditional(=)
Diagnostic			Optional	Optional

Upon receiving the F–GROUP–LIST indication, the filestore provider reads the requested attributes of each object identified by pathname in the generalized select group. Any pathname for which access permission is denied is removed from the list without notification of the filestore user. The results are then reported by issuing the F–GROUP–LIST response primitive.

14b.5.2 Types of primitives and parameters

Table 14g indicates the types of primitives and the parameters needed for listing the generalized selection group.

14b.5.2.1 Action result

The action result parameter is described in clause 13.2.

14b.5.2.2 Attribute names

The attribute names parameter is described in clause 13.14.

14b.5.2.3 Objects attributes list

The objects attributes list parameter is described in clause 13.18. The presence of the object attributes list parameter is conditional on the inclusion in the attribute names parameter of at least one attribute with a returnable value.

14b.5.2.4 Diagnostic

The diagnostic parameter is described in clause 13.13.

14b.6 Group change attribute

14.b.6.1 Function

The F-GROUP-CHANGE-ATTRIB service modifies the attributes of the file objects in the generalized select group. The primitive conveys a list of attribute names and values.

The filestore provider performs the change attribute action and sets the two object attributes, date and time of last attribute modifications and identity of last attribute modifier, after receiving the F-GROUP-CHANGE-ATTRIB indication, and before issuing the F-GROUP-CHANGE-ATTRIB response primitive. If any change causes an action result other than success to be generated, none of the requested changes against the file are made. The action result for that file is "unsuccessful".

The F-GROUP-CHANGE-ATTRIB service is not an atomic action. Change attribute operations are performed one at a time, until either an error occurs or the list of pathnames is exhausted. The error action parameter determines whether or not an error terminates the service. A shared ASE information parameter is provided, should some other ASE, such as CCR, desire to form an atomic action using this service.

14b.6.2 Type of primitives and parameters

Table 14h indicates the types of primitives and the parameters needed to change attributes of grouped files.

14b.6.2.1 Action result

The action result parameter is defined in clause 13.2.

14b.6.2.2 Error Action

The error action parameter is defined in clause 13.20.

14b.6.2.3 Attribute

The attribute parameter, defined in 13.5, indictes on the request and indication primitives which object attributes given in the virtual filestore definition are to be changed and what the new values are. The parameter is a list, each element of which names a file attribute defined in ISO 8571-2 and provides a new value for it.

14b.6.2.4 Request operation result

The request opertaion result parameter is defined in 13.24. This parameter is optional.

14b.6.2.5 Operation result

The operation result parameter is defined in 13.25. This parmeter is conditional on the presence of the re-

Table 14h – F–GROUP–CHANGE–ATTRIB parameters

Parameter	F-GROUP-CHANGE-ATTRIB request	F-GROUP-CHANGE-ATTRIB indication	F-GROUP-CHANGE-ATTRIB response	F-GROUP-CHANGE-ATTRIB confirm
Action result			Mandatory	Mandatory
Attributes	Mandatory	Mandatory(=)		
Error action	Mandatory	Mandatory(=)		
Shared ASE information	Optional	Optional(=)	Optional	Optional(=)
Request operation result	Optional	Optional(=)		
Operation result			Conditional	Conditional(=)
Diagnostic			Optional	Optional(=)

quest operation result parameter.

14b.6.2.6 Shared ASE information

The shared ASE information is defined in clause 13.10.

14b.6.2.7 Diagnostic

The diagnostic parameter is described in clause 13.13.

Append the following parameter to Table 15

Parameter	F-SELECT request	F-SELECT indication	F-SELECT response	F-SELECT confirm
.			.	.
.			.	.
.			.	.
Referent indicator			Optional	Optional

(amend title to clause 15, page 27)

15 Object selection regime control

(amend 1st paragraph)

These services control the regime which binds an identified object to an FTAM regime. The object selection service is pertinent to files only, unless the limited filestore management is available. This is because selection of objects other than files is permitted if and only if the limited filestore management functional unit is available.

Different services are used depending on whether the object exists before the selection or not, the kind of object to be selected or created, and on whether it is to exist after the selection or not.

(amend title to clause 15.1, page 27)

15.1 Object selection service

15.1.1 Function

(amend 1st paragraph, page 27)

This service selects an object which already exists, by specifying an existing pathname in the attributes parameter to identify the object unambiguously. These primitives may only be issued if there is no current object selection within the current FTAM regime. The filestore provider performs the select object action after receiving the F–SELECT indication and before issuing an F–SELECT response primitive with an action result indicating success.

The object selection service does not affect, and is not affected by, any existing activity attributes that may have been established by a generalized select.

15.1.2 Types of primitives and parameters

(amend 1st paragraph, page 27)

Table 15 indicates the types of primitives and the parameters needed for object selection.

15.1.2.3 Attributes

(amend 1st paragraph, pages 27 to 28)

The attributes parameter is described in clause 13.5. On F–SELECT the parameter carries only a single object pathname, which may or may not be a primary pathname. The pathname is used to identify the object in the object selection process. On the F–SELECT response and confirm primitives only the pathname is returned which may not be identical to that issued on the request and indication.

15.1.2.8 Account

(amend 1st paragraph, page 28)

The account parameter is described in clause 13.3. The value given overrides any value previously set by the F–INITIALIZE or F–GROUP–SELECT requests for the duration of this select regime.

(append after subclause 15.1.2.9, page 28)

15.1.2.9 Referent indicator

The Referent indicator parameter is described in 13.23, and can be used if and only if the limited filestore management functional unit is available.

(append after clause 15.1, page 28)

15.1a Select another

15.1a.1 Function

The F–SELECT–ANOTHER service selects an object from the set of previously unselected pathnames in the generalized select group, setting the current access request, current concurrency control, current access passwords, and current access control activity attributes from the generalized access request, generalized concurrency control, generalized access passwords, and generalized access control activity attributes, respectively.

If no pathnames exist in the generalized select group, then an error indication is returned. A filestore user may cycle through the set of pathnames multiple times. A last member indicator parameter is returned on the F–SELECT–ANOTHER response and confirm primitives, to indicate that the currently selected object is the last pathname in the group.

Table 15a – F–SELECT–ANOTHER parameters

Parameter	F-SELECT-ANOTHER request	F-SELECT-ANOTHER indication	F-SELECT-ANOTHER response	F-SELECT-ANOTHER confirm
Action result			Mandatory	Mandatory
State result			Mandatory	Mandatory
Last member indicator			Optional	Optional
Shared ASE information	Optional	Optional(=)	Optional	Optional(=)
Diagnostic			Optional	Optional
Referent indicator			Optional	Optional

NOTES

1) No order is implied in the generalized select group, and there is no guarantee that pathnames will be chosen in the same order as a previous pass through the group, or as listed by an F–GROUP–LIST service.

2) No concurrency control is implied by the generalized selection service. Objects may be removed from the generalized select group due to object deletion or changes in access control during a given FTAM regime without notification. Once an object is selected by the F–SELECT–ANOTHER service, the concurrency control specified in the generalized concurrency control activity attribute is in force for that object only during the life of this select regime.

15.1a.2 Types of primitives and parameters

Table 15a indicates the types of primitives and the parameters needed for selecting another pathname from the generalized selection group.

15.1a.2.1 Action result

The action result parameter is described in clause 13.2.

15.1a.2.2 State result

The state result parameter is described in clause 13.1.

15.1a.2.3 Last Member indicator

The Last Member indicator is described in clause 13.21.

15.1a.2.4 Referent indicator

The referent indicator is described in clause 13.23.

15.1a.2.5 Shared ASE information

The shared ASE information parameter is described in clause]13.10.

15.1a.2.6 Diagnostic

The diagnostic parameter is described in clause 13.13.

(amend title to clause 15.2, page 28)

15.2 Object deselection service

15.2.1 Function

(amend 1st paragraph, page 28)

The F–DESELECT service releases the binding between the select regime and the object. The object continues to exist and is available for subsequent selection. The primitives may only be used while an object is selected. The object select regime is always terminated following an F–DESELECT response or confirm for all values of the action result parameter

(amend 2nd paragraph, page 28)

The filestore provider performs the deselect object action after receiving the F–DESELECT indication primitive, and before issuing the F–DESELECT response primitive. In the external service all concurrency control is released when the object is deselected, except when performed within a CCR atomic action. The current account activity attribute is reset to that of the FTAM regime.

15.2.2 Types of primitives and parameters

(amend 1st paragraph, page 28)

Table 16 indicates the types of primitives and the parameters needed for object deselection.

15.2.2.2 Charging

Table 17a – F–CREATE–DIRECTORY parameters

Parameter	F-CREATE-DIRECTORY request	F-CREATE-DIRECTORY indication	F-CREATE-DIRECTORY response	F-CREATE-DIRECTORY confirm
Action result			Mandatory	Mandatory
State result			Mandatory	Mandatory
Attributes	Mandatory	Mandatory(=)	Mandatory	Mandatory(=)
Create password	Optional	Optional(=)		
Requested access	Mandatory	Mandatory(=)		
Shared ASE information	Optional	Optional(=)	Optional	Optional(=)
Account	Optional	Optional(=)		
Diagnostic			Optional	Optional

(amend 1st paragraph)

The charging parameter is described in clause 13.4. Charges are only reported on deselection against an overriding account set up at object selection.

15.3.2.3 Override

(amend 1st paragraph, page 29)

The override parameter is defined in clause 13.19.

15.3.2.5 Create password

(amend 1st paragraph, page 29)

The create password parameter is defined in clause 13.16.

(add after clause 15.3, page 30)

15.3a File-directory creation

15.3a.1 Function

The F–CREATE–DIRECTORY service causes a file-directory to be created and establishes a selection of the newly created file-directory. If an object with the specified pathname already exists then the request is rejected. The file-directory create service may only be used if there is no currently selected object.

The filestore provider performs the create file-directory action after receiving the F–CREATE–DIRECTORY indication primitive, and before issuing the F–CREATE–DIRECTORY response primitive with action result parameters indicating success.

15.3a.2 Types of primitives and parameters

Table 17a indicates the types of primitives and the parameters needed for creating a file-directory.

15.3a.2.1 Action result

The action result parameter is described in clause 13.2.

15.3a.2.2 State result

The state result parameter is described in clause 13.1.

15.3a.2.3 Attributes

The attributes parameter is described in clause 13.5. For those attribute groups negotiated on F–INITIALIZE, the attribute values given are associated with the newly created file-directory. If, for any particular attribute a value is not given, a default is determined locally by the responder; this value may be "no value available". For attributes other than pathname and permitted actions, the responder may change a value proposed by the initiator to "no value available" but may not assign any different value to it. The responder does report any local modification to the values of the pathname or permitted actions attributes. In addition to the attributes in the attribute groups negotiated on F–INITIALIZE the attributes in the kernel attribute group are also associated with the newly created file-directory.

15.3a.2.4 Create password

The create password parameter is described in clause 13.16.

15.3a.2.5 Requested access

The requested access parameter is described in clause 13.6. The parameter is used to establish the access available in the select regime following the create action. The requested access during the select regime need not be the full capability of the file-directory. Hence the access requested must be a subset of those capabilities established with the attributes parameter. The established capabilities include all responder provided defaults which are

Table 17b – F–LINK parameters

Parameter	F-LINK request	F-LINK indication	F-LINK response	F-LINK confirm
Action result			Mandatory	Mandatory
State result			Mandatory	Mandatory
Attributes	Mandatory	Mandatory(=)	Mandatory	Mandatory(=)
Target object	Mandatory	Mandatory(=)	Mandatory	Mandatory(=)
Create password	Optional	Optional(=)		
Requested access	Mandatory	Mandatory(=)		
Access passwords	Conditional	Conditional(=)		
Concurrency control	Optional	Optional(=)		
Shared ASE information	Optional	Optional(=)	Optional	Optional(=)
Account	Optional	Optional(=)		
Diagnostic			Optional	Optional

within the kernel attribute group or the other attribute groups negotiated at F–INITIALIZE.

If the requested access parameter is incompatible with the newly created file-directory, the select will fail even though the file-directory was created.

15.3a.2.6 Shared ASE information

The shared ASE information parameter is described in clause 13.10.

15.3a.2.7 Account

The account parameter is described in clause 13.3. The value given overrides any value previously set on the F–INITIALIZE or F–GROUP–SELECT request for the duration of this selection.

15.3a.2.8 Diagnostic

The diagnostic parameter is described in clause 13.13.

15.3b Reference creation

15.3b.1 Function

The F–LINK service causes a reference to be created and establishes a selection of the referenced object. If an object with the specified reference pathname already exists then the request is rejected. This service also fails if the object to which the reference is to be created does not exist. The create reference service may only be used if there is not a currently selected object.

The filestore provider performs the create reference action after receiving the F–LINK indication primitive, and before issuing the F–LINK response primitive with action result parameters indicating success.

15.3b.2 Types of primitives and parameters

Table 17b indicates the types of primitives and the parameters needed for creating a reference.

15.3b.2.1 Action result

The action result parameter is described in clause 13.2.

15.3b.2.2 State result

The state result parameter is described in clause 13.1.

15.3b.2.3 Attributes

The attributes parameter is described in clause 13.5. For those attribute groups negotiated on F–INITIALIZE, the attribute values given are associated with the newly created reference. If, for any particular attribute a value is not given, a default is determined locally by the responder; this value may be "no value available". For attributes other than pathname and permitted actions, the responder may change a value proposed by the initiator to "no value available" but may not assign any different value to it. The responder does report any local modification to the values of the pathname or permitted actions attributes. In addition to the attributes in the attribute groups negotiated on F–INITIALIZE the attributes in the kernel attribute group are also associated with the newly created reference.

15.3b.2.4 Target object

The target object parameter identifies an existing file or file-directory object within the filestore by pathname. This is the object to which the reference will be linked. The filestore user must have been granted link permission for this object for the service to succeed.

15.3b.2.5 Create password

The create password parameter is described in clause 13.16.

15.3b.2.6 Requested access

The requested access parameter is described in clause 13.6. The parameter is used to establish the access available to the target object in the select regime following the create reference action. The requested access during the select regime must be a subset of the access permitted to this filestore user by the target object.

If the requested access parameter is incompatible with the existing target object, the select will fail even though the reference was created.

15.3b.2.7 Access passwords

The access passwords parameter is described in clause 13.7. It is conditional on the security group being selected. If the link member of the set of access passwords does not match any of the non-empty password strings in the link member of the target object's access control attribute, the reference is not created and the select regime is not established. If the remaining members of the set of access passwords do not match any of the non-empty password strings in the access control attribute of the target object, the reference may be created, but the select regime is not established. This parameter is used to set the current access passwords activity attribute.

15.3b.2.8 Concurrency control

The concurrency control parameter is described in clause 13.8. This parameter is used to establish concurrency controls on the target object during the select regime. It is only valid when the reference links to a file object. If the required concurrency control is not available, the select regime is not established, although the reference may be created.

15.3b.2.9 Shared ASE information

The shared ASE information parameter is described in clause 13.10.

15.3b.2.10 Account

The account parameter is described in clause 13.3. The value given overrides any value previously set on the F–INITIALIZE or F–GROUP–SELECT request for the duration of this selection.

15.3b.2.11 Diagnostic

The diagnostic parameter is described in clause 13.13.

(amend title to clause 15.4, page 30)

15.4 Object deletion service

15.4.1 Function

(amend 1st paragraph, page 30)

The F–DELETE service releases an existing object select regime in such a way that the selected object ceases to exist, and is not available for reselection. The primitives may only be issued while an object is selected. The select regime is always released following an F–DELETE response or confirm, for all possible values of the action result parameter. File-directories may not be deleted while they maintain any parenthood relationships. In this case, the select regime is terminated, but the file-directory is not deleted. The action result parameter will reflect this condition. If access to the selected object was by reference, then both the referenced object and the reference are deleted.

(amend 2nd paragraph, page 30)

The filestore provider performs the delete object action after receiving the F–DELETE indication, and before issuing the F–DELETE response primitive, with an action result parameter of success. The delete object action can be performed only if the initiating entity has the "delete" access control permission (see clause 13.6). This access control permission is established by the requested access, concurrency control and access passwords parameters on the F–SELECT, F–CREATE, F–CREATE–DIRECTORY, or F–LINK primitives, or by the generalized requested access, generalized concurrency control, and generalized access passwords activity attributes on an F–SELECT–ANOTHER primitive, whichever established the object selection regime.

15.4.2 Types of primitives and parameters

(amend 1st paragraph, page 30)

Table 18 indicates the types of primitives and the parameters needed for object deletion.

15.4.2.1 Action result

(amend 1st paragraph, page 30)

The action result parameter, defined in 13.2, indicates success or failure of the deletion action. The object is

Table 18a – F–UNLINK parameters

Parameter	F-UNLINK request	F-UNLINK indication	F-UNLINK response	F-UNLINK confirm
Action result	Optional	Optional(=)	Mandatory	Mandatory
Shared ASE information			Optional	Optional(=)
Charging			Conditional	Conditional(=)
Diagnostic			Optional	Optional

always deselected whatever the value of the action result parameter.

15.4.2.3 Charging

(amend 1st paragraph)

The charging parameter is defined in 13.4. Charges are only reported on deselection against an overriding account set up at object selection.

(append after clause 15.4, page 30)

15.5 Reference deletion

15.5.1 Function

The F–UNLINK service releases an existing object select regime in such a way that the reference object identified by the current pathname activity attribute ceases to exist, and is not available for selection. The object linked by the reference continues to exist, as well as any other references to the same object. The primitives may only be issued while an object is selected and the current pathname activity attribute identifies a reference object. The select regime is always released following an F–UNLINK response or confirm, for all possible values of the action result parameter.

The filestore provider performs the delete reference action after receiving the F–UNLINK indication, and before issuing the F–UNLINK response primitive, with an action result parameter of success. The delete reference action can be performed only if the initiating entity has the "delete" access control permission (see clause 13.6) for the reference object.

15.5.2 Types of primitives and parameters

Table 18a indicates the types of primitives and the parameters needed for deleting a reference.

15.5.2.1 Action result

The action result parameter is described in clause 13.2.

15.5.2.2 Shared ASE information

The shared ASE information parameter is described in clause 13.10.

15.5.2.3 Charging

The charging parameter is described in clause 13.4. Charging is only reported against an overriding account for this regime.

15.5.2.4 Diagnostic

The diagnostic parameter is described in clause 13.13.

(amend title of clause 16, page 31)

16 Object management

(amend 1st paragraph, page 31)

These services provide object management capability to the file service user. They are not concerned with the establishment of regimes. The services provide facilities for interrogation and modification, where permitted, of object attributes. Managed objects are of type file only, unless the limited filestore management functional unit is available. This is because the selection of objects other than files is permitted if and only if the limited filestore management functional unit is available.

16.1.1 Function

(amend 1st paragraph)

The F–READ–ATTRIB service interrogates the object attributes of the selected object. The request and indication convey a list of the object attribute names for which values are to be read. The response and confirm then convey the corresponding returned values. Attribute names of attributes not supported by the object type are ignored. This allows attributes to be read from objects whose type is unknown.

16.1.2 Types of primitives and parameters

(amend 1st paragraph, page 31)

Table 19 indicates the types of primitives and the parameters needed to read object attributes.

16.1.2.2 Attribute names

(amend 1st paragraph, page 31)

The attribute names parameter is described in clause 13.14.

16.2.1 Function

(amend 1st paragraph, page 31)

The F–CHANGE–ATTRIB service modified the attributes of the selected object. The primitives convey a list of object attribute names and values.

(amend 2nd paragraph, page 31)

The filestore provider performs the change attribute action and sets the two object attributes, date and time of last attribute modification and identity of last attribute modifier, after receiving the F–CHANGE–ATTRIB indication, and before issuing the F–CHANGE–ATTRIB response primitive. If any change causes an action result other than success to be generated, none of the requested changes are made; an "unsuccessful" action result parameter is returned.

16.2.2 Types of primitives and parameters

(amend 1st paragraph, page 31)

Table 20 indicates the types of primitive and the parameters needed to change object attributes.

16.2.2.2 Attributes

(amend 1st paragraph, page 31)

The attributes parameter, defined in 13.5, indicates on the request and indication primitives which object attributes given in the virtual filestore definition are to be changed and what the new values are to be. The parameter is a list, each element of which names an object attribute defined in ISO 8571–2 and provides a new value for it.

(amend 2nd paragraph, page 31)

The presence of the attributes parameter on the response and confirm is conditional on the inclusion in the attributes parameter of at least one attribute name and value. Values for attributes not specifically requested are not changed or returned. Values for attributes not in the kernel or negotiated attribute group are not changed or returned. Values for attributes not supported by the selected object type are not changed or returned. ISO 8571–2 defines the actions permitted on attributes.

(append after clause 16.2, page 31)

16.3 Read reference attributes

16.3.1 Function

The F–READ–LINK–ATTRIB service interrogates the attributes of the reference identified by the current pathname activity attribute. The request and indication convey a list of the reference attribute names for which values are to be read. The response and confirm then convey the corresponding returned values.

The filestore provider performs the read reference attribute action after receiving the F–READ–LINK–ATTRIB indication, and before issuing the F–READ–LINK–ATTRIB response primitive.

16.3.2 Types of primitives and parameters

Table 19a indicates the types of primitives and the parameters needed for reading reference attributes.

16.3.2.1 Action result

The action result parameter is described in clause 13.2.

Table 19a – F–READ–LINK–ATTRIB parameters

Parameter	F-READ-LINK-ATTRIB request	F-READ-LINK-ATTRIB indication	F-READ-LINK-ATTRIB response	F-READ-LINK-ATTRIB confirm
Action result	Mandatory	Mandatory(=)	Mandatory	Mandatory
Attribute names				
Attributes			Conditional	Conditional(=)
Diagnostic			Optional	Optional

16.3.2.2 Attribute names

The attribute names parameter is described in clause 13.14. The attribute names allowed are constrained to those defined for reference objects.

16.3.2.3 Attributes

The attributes parameter is described in clause 13.5. The presence of the attributes parameter is conditional on the inclusion in the attribute names parameter of at least one attribute with a returnable value. Values for attributes not specifically requested are not returned. Values for attributes not in the kernel attribute group or a negotiated attribute group are not returned. The value of password attributes are not returned. The responder may return "no value available" for partially supported attributes. ISO 8571–2 details the attributes which are and are not returned for reference objects.

16.3.2.4 Diagnostic

The diagnostic parameter is described in clause 13.13.

16.4 Change reference attributes

16.4.1 Function

The F–CHANGE–LINK–ATTRIB service modifies the attributes of the reference identified by the current pathname activity attribute. The primitives convey a list of reference attribute names and values.

The filestore provider performs the change reference attribute action and sets the two object attributes of the indicated reference, date and time of last attribute modification and identity of last attribute modifier, after receiving the F–CHANGE–LINK–ATTRIB indication, and before issuing the F–CHANGE–LINK–ATTRIB response primitive. If any change causes an action result other than success to be generated, none of the requested changes are made; an "unsuccessful" action result parameter is returned.

16.4.2 Types of primitives and parameters

Table 20a indicates the types of primitives and the parameters needed for changing reference attributes.

16.4.2.1 Action result

The action result parameter is described in clause 13.2.

16.4.2.2 Attributes

The attributes parameter is described in clause 13.5. It indicates on the request and indication primitives which reference attributes given in the virtual filestore definition are to be changed and what the new values are to be. The parameter is a list, each element of which names a reference attribute defined in ISO 8571–2 and provides a new value for it.

The presence of the attributes parameter on the response and confirm is conditional on the inclusion in the attributes parameter of at least one attribute name and value. Values for attributes not specifically requested are not changed or returned. Values for attributes not in the kernel or negotiated attribute groups are not changed or returned. ISO 8571–2

Table 20a – F–CHANGE–LINK–ATTRIB parameters

Parameter	F-CHANGE-LINK-ATTRIB request	F-CHANGE-LINK-ATTRIB indication	F-CHANGE-LINK-ATTRIB response	F-CHANGE-LINK-ATTRIB confirm
Action result	Mandatory	Mandatory(=)	Mandatory	Mandatory
Attributes			Conditional	Conditional(=)
Diagnostic			Optional	Optional

Table 20b – F–MOVE parameters

Parameter	F-MOVE request	F-MOVE indication	F-MOVE response	F-MOVE confirm
Action result			Mandatory	Mandatory
Destination file-directory	Mandatory	Mandatory(=)	Optional	Optional(=)
Override	Mandatory	Mandatory(=)		
Create password	Optional	Optional(=)		
Access passwords	Conditionial	Conditional(=)		
Attributes	Optional	Optional(=)	Optional	Optional(=)
Diagnostic			Optional	Optional

defines the actions permitted on attributes.

16.4.2.3 Diagnostic

The diagnostic parameter is described in clause 13.13.

16.5 Move object

16.5.1 Function

The F–MOVE service moves the object selected by changing its location in the filestore structure. The destination is specified by the destination file-directory parameter. This parameter changes the primary pathname object attribute of the selected object. Other object parameters may also be changed by use of the attributes parameter.

Moving an object consists of entering a parenthood relationship for the given object into a named file-directory, removing the parenthood relationship for that object from the old parent file-directory for that object, and updating the primary pathname attribute of the object to reflect the new primary pathname. If the object selected is a reference, then only the reference object is moved, not the referenced object.

Prior to moving the object, the filestore user identified in the F–INITIALIZE service must have change-attribute permission to the object via the corresponding pathname. Also, the filestore user must have permission to create objects in this filestore, and insert permission in the destination file-directory.

16.5.2 Types of primitives and parameters

Table 20b indicates the types of primitives and the parameters needed for moving an object within the filestore.

16.5.2.1 Action result

The action result parameter is defined in clause 13.2.

16.5.2.2 Destination file-directory

The destination file-directory parameter is defined in clause 13.17. It identifies the file-directory that should now maintain the parenthood relationship to the selected object

16.5.2.3 Override

The override parameter is defined in clause 13.19 and only applies to the create part of the action. Only the following override options are supported for the F–MOVE service:

a) fail the action if the object already exists;

b) delete any existing object that may exist with the same pathname and create a new object using the initial attributes provided on the filestore service primitive.

16.5.2.4 Create password

The create password parameter is described in clause 13.16.

16.5.2.5 Access passwords

The access passwords parameter is described in clause 13.7. It is used to establish delete access to any object which may be overridden by this service. This parameter is conditional on the security attribute group being successfully negotiated during establishment of the FTAM regime.

16.5.2.6 Attributes

The attributes parameter is described in clause 13.5. This parameter is used to change any additional attributes (other than primary pathname) on the object being moved.

16.5.2.7 Diagnostic

Table 20c – F–COPY parameters

Parameter	F-COPY request	F-COPY indication	F-COPY response	F-COPY confirm
Action result			Mandatory	Mandatory
Destination file-directory	Mandatory	Mandatory(=)	Optional	Optional(=)
Override	Mandatory	Mandatory(=)		
Create password	Optional	Optional(=)		
Access passwords	Conditionial	Conditional(=)		
Attributes	Optional	Optional(=)	Optional	Optional(=)
Diagnostic			Optional	Optional

The diagnostic parameter is described in clause 13.13.

16.6 Copy object

16.6.1 Function

The F–COPY service copies the currently selected object by creating a new object with duplicate contents in the specified destination location in the filestore structure. The destination is specified by the destination file-directory parameter. The new object's parameters may also be set by use of the attributes parameter.

If the object selected is a reference, then only the reference is duplicated, not the referenced object.

If the object selected is a directory, then the copy will only succeed if the directory has no children.

Prior to copying the object, the filestore user identified in the F–INITIALIZE service must have read-attributes and read permission to the object via the corresponding pathname. Also, the filestore user must have permission to create objects in this filestore, and insert permission in the destination file-directory

16.6.2 Types of primitives and parameters

Table 20c indicates the types of primitives and the parameters needed for duplicating an object within the filestore.

16.6.2.1 Action result

The action result parameter is defined in clause 13.2.

16.6.2.2 Destination file-directory

The destination file-directory parameter is defined in clause 13.17. It identifies the file-directory in which the duplicate of the selected object should be placed.

16.6.2.3 Override

The override parameter is defined in clause 13.19 and only applies to the create part of the action. Only the following override options are supported for the F–COPY service:

a) fail the action if the object already exists;

b) delete any existing object that may exist with the same pathname and create a new object using the initial attributes provided on the filestore service primitive.

16.6.2.4 Create password

The create password parameter is described in clause 13.16.

16.6.2.5 Access passwords

The access passwords parameter is described in clause 13.7. It is used to establish delete access to any object which may be overridden by this service. This parameter is conditional on the security attribute group being successfully negotiated during the establishment of the FTAM regime.

16.6.2.6 Attributes

The attributes parameter is described in clause 13.5. This parameter is used to set any additional attributes (other than primary pathname) on the object being copied.

16.6.2.7 Diagnostic

The diagnostic parameter is described in clause 13.13.

Annex A
(normative)

Diagnostic parameter values

A.6 Applicability of the diagnostic parameter

(amend 1st paragraph, page 48)

The diagnostic parameter occurs on the following services:

F-INITIALIZE
F-U-ABORT
F-P-ABORT
F-CHANGE-PREFIX
F-LIST
F-GROUP-SELECT
F-GROUP-DELETE
F-GROUP-MOVE
F-GROUP-COPY
F-GROUP-LIST
F-GROUP-CHANGE-ATTRIB
F-SELECT
F-SELECT-ANOTHER
F-DESELECT
F-CREATE
F-CREATE-DIRECTORY
F-LINK
F-DELETE
F-UNLINK
F-READ-ATTRIB
F-CHANGE-ATTRIB
F-READ-LINK-ATTRIB
F-CHANGE-LINK-ATTRIB
F-MOVE
F-COPY
F-OPEN
F-CLOSE
F-RECOVER
F-LOCATE
F-ERASE
F-DATA-END
F-TRANSFER-END
F-CANCEL

(insert after Table 45, page 50)

Table 45a – Filestore management related diagnostics

Type	Identifier	Observer	Source	Reason
012	7000	5	5	Access control inconsistent
012	7001	5	5	Access control not available
012	7002	5	5	Path access control inconsistent
012	7003	5	5	Path access control not available
2	7004	5	5	Invalid access passwords
2	7005	5	5	Invalid path access passwords
0	7006	4,5	4,5	Object name truncated
0	7007	4,5	4,5	Path member name(s) truncated
2	7008	5	5	Target file-directory does not exist
2	7009	5	5	Target object is not a file-directory
2	7010	5	5	No access to specified file-directory
2	7011	5	5	No access via specified path
12	7013	5	5	Ambiguous object specification
0	7014	5	5	No extension attribute sets are supported
0	7015	5	5	Extension attribute set(s) not supported
12	7016	5	5	Attribute value assertion too compex
2	7017	5	5	Attribute value assertion does not match any objects
2	7018	5	5	Inconsistent attribute value assertions
2	7019	5	5	Invalid attribute value assertion
2	7020	5	5	Extension attribute set member not recognized

Table 45b – Generalized selection related diagnostics

Type	Identifier	Observer	Source	Reason
012	8000	5	5	Access control not available
012	8001	5	5	Access control inconsistent
2	8002	5	5	Invalid access passwords
2	8003	5	5	Invalid path access passwords
012	8004	5	5	Path access control not available
012	8005	5	5	Path access control inconsistent
12	8006	5	5	Selection attributes not matched
2	8007	5	5	Attribute value assertion does not match any objects
2	8008	5	5	Inconsistent attribute value assertions
2	8009	5	5	Invalid attribute value assertion(s)
12	8010	5	5	Attribute value assertion too compex
12	8011	5	5	Maximum set size exceeded
02	8012	5	5	No extension attribute sets supported
0	8013	5	5	Extension attribute set(s) not supported
2	8014	5	5	Extension attribute set member not recognized
2	8015	5	5	Invalid account

Table 45c – Filestore structure management related diagnostics

Type	Identifier	Observer	Source	Reason
12	9000	5	5	Target file-directory not found
2	9001	5	5	Initial attributes not possible
2	9002	4	2	Bad attribute name
12	9003	5	5	Object already exists
12	9004	5	5	Object cannot be created
12	9005	5	5	Object cannot be deleted
012	9006	5	5	Access control not available
012	9007	5	5	Access control inconsistent
012	9008	5	5	Path access control not available
012	9009	5	5	Path access control inconsistent
2	9010	5	5	No access via specified path
0	9011	5	5	Initial attributes altered
0	9012	5	5	Override deleted and recreated object with new attributes
12	9013	5	5	Create override - not possible
2	9014	5	1	Invalid create password
2	9015	5	1	Invalid delete password on override
2	9016	5	1	Bad attribute value
02	9018	5	5	No extension attributes supported
0	9019	5	5	Extension attribute set not supported
2	9020	5	5	Extension attribute set member not recognized
2	9021	5	5	No pathnames remaining in generalized select group
0	9022	5	5	Attribute name(s) inconsistent with object type

(amend Table 46, page 50)

Table 46 – Selection related diagnostics

Type	Identifier	Observer	Source	Reason
12	3000	5	5	Named object not found
12	3001	5	5	Selection attributes not matched
2	3002	5	5	Initial attributes not possible
2	3003	4	2	Bad attribute name
12	3004	5	5	Non-existent object
12	3005	5	5	Object already exists
12	3006	5	5	Object cannot be created
12	3007	5	5	Object cannot be deleted
02	3008	5	5	Concurrency control not available
02	3009	5	5	Concurrency control not supported
02	3010	5	5	Concurrency control not possible
01	3011	5	5	More restrictive lock
12	3012	5	5	Object busy
12	3013	5	5	Object not available
012	3015	5	5	Access control not available
012	3016	5	5	Access control inconsistent
0	3017	4,5	4,5	Object name truncated
0	3018	5	5	Initial attributes altered

Table 46 – Selection related diagnostics (continued)

Type	Identifier	Observer	Source	Reason
12	3019	5	1	Bad account
0	3020	5	5	Override selected existing file
0	3021	5	5	Override deleted and recreated file with old attributes
0	3022	5	5	Override deleted and recreated object with new attributes
12	3023	5	5	Create override - not possible
12	3024	5	5	Ambiguous object specification
2	3025	5	1	Invalid create password
2	3026	5	1	Invalid delete password on override
2	3027	5	1	Bad attribute value
2	3028	5	1	Requested access violates permitted actions
2	3029	5	1	Functional unit not available for requested access
01	3030	5	5	Object created but not selected
012	3031	5	5	Path access control not available
012	3032	5	5	Path access control not supported
012	3033	5	5	Path access control inconsistent
2	3034	5	5	Invalid access passwords
2	3035	5	1	Invalid path access passwords
2	3036	5	5	No access via specified path
0	3037	5	5	Concurrency control not supported on object type
0	3039	5	5	Last pathname in generalized select group
1	3040	5	5	No more pathnames in generalized select group
0	3041	4,5	4,5	Path member name(s) truncated
02	3042	5	5	No extension attribute sets supported
0	3043	5	5	Extension attribute set(s) not supported
2	3044	5	5	Extension attribute set member not recognized
2	3045	5	5	Initial attributes inconsistent with object type
12	3047	5	5	Ambiguous referent object specification
2	3048	5	5	Reference created but failed to select referent
2	3049	5	5	Target referent is a reference object
2	3050	5	5	File-directory not empty
2	3051	5	5	Referenced object not deleted
2	3052	5	5	Reference object not deleted
2	3053	5	5	Object cannot be unlinked

(amend Table 47, page 51)

Table 47 – File management related diagnostics

Type	Identifier	Observer	Source	Reason
02	4009	5	5	No extension attributes supported
0	4010	5	5	Extension attribute set not supported
2	4011	5	5	Extension attribute set member not recognized
0	4012	5	5	Attribute name(s) inconsistent with object type
2	4013	5	5	Object is not a reference
0	4014	4,5	4,5	Object name truncated
0	4015	4,5	4,5	Referenced object name truncated
0	4016	4,5	4,5	Path member name(s) truncated
12	4017	5	5	Target file-directory not found
2	4018	5	5	Initial attributes not possible
12	4019	5	5	Object already exists
12	4020	5	5	Object cannot be created
12	4021	5	5	Object cannot be deleted
012	4022	5	5	Access control not available
012	4023	5	5	Access control inconsistent
012	4024	5	5	Path access control not available
012	4025	5	5	Path access control inconsistent
2	4026	5	5	No access via specified path
0	4027	5	5	Initial attributes altered
0	4028	5	5	Override deleted and recreated object with new attributes
12	4029	5	5	Create override - not possible
2	4030	5	1	Invalid create password
2	4031	5	1	Invalid delete password on override
2	4032	5	1	Bad attribute value

Annex B
(normative)

Relation of attributes to primitives

(amend 1st paragraph, page 53)

Tables 50 through 51 indicate which of the attributes defined in the virtual filestore are affected by the various primitives defined in the file service.

(insert after 1st paragraph, page 53)

Table 50 shows the affected object attributes of a given specified object. Table 50a shows the affected attributes of the parent file-directory containing the specified object. Table 50b shows the affected attributes of a linking reference through which the object may have been specified.

Notes on Tables 50 to 50b:

1) Any password values should not be returned.
2) affects attribute of reference object when target object is accessed via reference.
3) Destination file-directory's attribute.

(amend 2nd paragraph, page 53)

Primitives which do not occur in the tables 50 through 51 do not affect the virtual filestore attributes directly. However, the deselect, delete, unlink, and group delete primitives may restore a previous value of the current account attribute.

(amend list item (e), 2nd paragraph, page 53)

e) set; the attribute value is set on the basis of a value transferred on a parameter of the request and indication primitives, or to a local default if there is no value transferred;

(amend Table 50, page 53)

Table 50 – Object attributes

Attribute Name	F-CHANGE-PREFIX	F-LIST	F-GROUP-SELECT
access control	compare	compare & return [1]	compare
availability		compare & return	compare
contents type		compare & return	compare
date & time of creation		compare & return	compare
date & time of last attribute modification		compare & return	compare
date & time of last modification		compare & return	compare
date & time of last read access		compare & return	compare
future object size		compare & return	compare
identity of creator		compare & return	compare
identity of last attribute modifier		compare & return	compare
identity of last modifier		compare & return	compare
identity of last reader		compare & return	compare
legal qualifications		compare & return	compare
linked object type		compare & return	compare
linked object		compare & return	compare
object name		compare & return	compare
object size		compare & return	compare
object type		compare & return	compare
path access control		compare & return [1]	compare
permitted actions		compare & return	compare
primary pathname		compare & return	compare
private use		compare & return	compare
storage account		compare & return	compare
child object			

Table 50 – Object attributes (continued)

Attribute Name	F-GROUP-COPY	F-GROUP-MOVE	F-GROUP-LIST	F-GROUP-CHANGE-ATTRIB	F-SELECT
access control	compare & set	compare & change	compare & return [1]	compare & change	compare
availability	set	change	return	change	
contents type			return		
date & time of creation	implicit		return		
date & time of last attribute modification	implicit	implicit	return	implicit	
date & time of last modification	implicit		return		
date & time of last read access	implicit		return		
future object size	set	change	return	change	
identity of creator	set		return		
identity of last attribute modifier	implicit	implicit	return	implicit	
identity of last modifier	implicit		return		
identity of last reader	implicit		return		
legal qualifications	set	change	return	change	
linked object type	implicit		return		
linked object	implicit		return		
object name	compare & set	compare &change	compare &return	compare & change	compare & return
object size	implicit		return		
object type	implicit		return		
path access control	set	change	return [1]	change	
permitted actions	compare	compare	compare & return	compare	compare
primary pathname	implicit	implicit	return	change	
private use	set	change	return	change	
storage account	set	change	return	change	
child object					

Table 50 – Object attributes (continued)

Attribute Name	F-CREATE	F-CREATE-DIRECTORY	F-LINK	F-DELETE
access control	set & return [1]	set & return [1]	compare	compare
availability	set & return	set & return		
contents type	set & return			
date & time of creation	implicit	implicit		
date & time of last attribute modification	implicit	implicit		
date & time of last modification	implicit	implicit		
date & time of last read access	implicit	implicit		
future object size	set & return	set & return		
identity of creator	set & return	set & return		
identity of last attribute modifier	implicit	implicit		
identity of last modifier	implicit	implicit		
identity of last reader	implicit	implicit		
legal qualifications	set & return	set & return		
linked object type				
linked object				
object name	set & return	set & return		
object size	implicit	implicit		
object type	implicit	implicit	return	
path access control		set & return [1]		compare
permitted actions	set & return	set & return	compare	
primary pathname	implicit	implicit	return	
private use	set & return			
storage account	set & return	set & return		
child object				

Table 50 – Object attributes (continued)

Attribute Name	F-UNLINK	F-READ-ATTRIB	F-READ-LINK-ATTRIB	F-CHANGE-ATTRIB
access control		compare & return [1]		compare, change, & return [1]
availability		return		change & return
contents type		return		
date & time of creation		return		
date & time of last attribute modification		return		implicit
date & time of last modification		return		
date & time of last read access		return		
future object size		return		change & return
identity of creator		return		
identity of last attribute modifier		return		implicit
identity of last modifier		return		
identity of last reader		return		
legal qualifications		return		change & return
linked object type				
linked object				
object name		return [2]		change & return [2]
object size		return		
object type		return		
path access control		return [1]		change & return [1]
permitted actions		return		
primary pathname		return		change & return
private use		return		change & return
storage account		return		change & return
child object		return		

Table 50 – Object attributes (continued)

Attribute Name	F-CHANGE-LINK-ATTRIB	F-COPY	F-MOVE	F-CLOSE & F-ABORT
access control		compare & set	compare & change	
availability		set	change	
contents type		implicit		
date & time of creation		implicit		
date & time of last attribute modification		implicit		
date & time of last modification		implicit		implicit
date & time of last read access		implicit		implicit
future object size		set	change	
identity of creator		implicit		
identity of last attribute modifier		implicit		
identity of last modifier		implicit		implicit
identity of last reader		implicit		implicit
legal qualifications		set	change	
linked object type				
linked object				
object name		set	change	
object size		set		implicit
object type		set		
path access control		set	change	
permitted actions		set		
primary pathname		implicit	implicit	
private use		set	change	
storage account		set	change	
child object				

Table 50a – Parent file-directory attributes

Attribute Name	F-CHANGE-PREFIX	F-LIST	F-GROUP-SELECT
access control	compare	compare	compare
date & time of creation			
date & time of last attribute modification			
date & time of last modification			
date & time of last read access		implicit	implicit
future object size			
identity of creator			
identity of last attribute modifier			
identity of last modifier			
identity of last reader		implicit	implicit
legal qualifications			
object name			
object size			
object type			
path access control	compare	compare	compare
permitted actions			
primary pathname			
private use			
storage account			

Table 50a – Parent file-directory attributes (continued)

Attribute Name	F-GROUP-COPY	F-GROUP-MOVE	F-GROUP-LIST	F-GROUP-CHANGE-ATTRIB	F-SELECT
access control	compare [3]	compare [3]			
date & time of creation					
date & time of last attribute modification					
date & time of last modification	implicit [3]	implicit [3]			
date & time of last read access					
future object size					
identity of creator					
identity of last attribute modifier					
identity of last modifier	implicit [3]	implicit, implicit [3]			
identity of last reader					
legal qualifications					
object name					
object size	implicit [3]	implicit, implicit [3]			
object type					
path access control	compare, compare [3]	compare, compare [3]		compare,	
permitted actions					
primary pathname					
private use					
storage account					

Table 50a – Parent file-directory attributes (continued)

Attribute Name	F-CREATE	F-CREATE-DIRECTORY	F-LINK	F-DELETE
access control	compare	compare	compare	compare
date & time of creation				
date & time of last attribute modification				
date & time of last modification	implicit	implicit	implicit	implicit
date & time of last read access				
future object size				
identity of creator				
identity of last attribute modifier				
identity of last modifier	implicit	implicit	implicit	implicit
identity of last reader				
legal qualifications				
object name				
object size	implicit	implicit	implicit	implicit
object type				
path access control	compare	compare	compare [3]	compare
permitted actions				
primary pathname				
private use				
storage account				

Table 50a – Parent file-directory attributes (continued)

Attribute Name	F-UNLINK	F-READ-ATTRIB	F-READ-LINK-ATTRIB	F-CHANGE-ATTRIB
access control	compare			
date & time of creation				
date & time of last attribute modification				
date & time of last modification	implicit			
date & time of last *read access*				
future object size				
identity of creator				
identity of last attribute modifier				
identity of last modifier	implicit			
identity of last reader				
legal qualifications				
object name				
object size	implicit			
object type				
path access control	compare	compare	compare	compare
permitted actions				
primary pathname				
private use				
storage account				

Table 50a – Parent file-directory attributes (continued)

Attribute Name	F-CHANGE-LINK-ATTRIB	F-COPY	F-MOVE	F-CLOSE & F-ABORT
access control		compare [3]	compare, compare [3]	
date & time of creation				
date & time of last attribute modification				
date & time of last modification		implicit [3]	implicit, implicit [3]	
date & time of last read access				
future object size				
identity of creator				
identity of last attribute modifier				
identity of last modifier		implicit [3]	implicit, implicit [3]	
identity of last reader				
legal qualifications				
object name				
object size		implicit [3]	implicit, implicit [3]	
object type				
path access control	compare	compare	compare	
permitted actions				
primary pathname				
private use				
storage account				

Table 50b – Reference attributes

Attribute Name	F-CHANGE-PREFIX	F-LIST	F-GROUP-SELECT
access control		compare, return [1]	compare
date & time of creation		compare, return	compare
date & time of last attribute modification		compare, return	compare
future object size		compare, return	compare
identity of creator		compare, return	compare
identity of last attribute modifier		compare, return	compare
legal qualifications		compare, return	compare
linked object type		compare, return	compare
linked object		compare, return	compare
object name		compare, return	compare
object size		compare, return	compare
object type		compare, return	compare
path access control		compare, return [1]	compare
permitted actions		compare, return	compare
primary pathname		compare, return	compare
private use		compare, return	compare
storage account		compare, return	compare

Table 50b – Reference attributes (continued)

Attribute Name	F-GROUP-COPY	F-GROUP-MOVE	F-GROUP-LIST	F-GROUP-CHANGE-ATTRIB	F-SELECT
access control	compare	compare	compare & return[1]		
date & time of creation	implicit		return		
date & time of last attribute modification	implicit		return		
future object size	set	change	return		
identity of creator	implicit		return		
identity of last attribute modifier	implicit		return		
legal qualifications	set	change	return		
linked object type			return		
linked object			return		
object name	set	change	return	change	return
object size	implicit		return		
object type			return		
path access control	compare, set	compare, change	return [1]	compare	compare
permitted actions	set		return		
primary pathname	implicit	implicit	return	implicit	
private use	set	change	return		
storage account	set	change	return		

Table 50b – Reference attributes (continued)

Attribute Name	F-CREATE	F-CREATE-DIRECTORY	F-LINK	F-DELETE
access control			set & return [1]	
date & time of creation			implicit	
date & time of last attribute modification			implicit	
future object size			set & return	
identity of creator			implicit	
identity of last attribute modifier			implicit	
legal qualifications			set & return	
linked object type			implicit	
linked object			set & return	
object name			set & return	
object size			implicit	
object type			implicit	
path access control			set & return [1]	
permitted actions			set & return	
primary pathname			implicit	
private use			set & return	
storage account			set & return	

Table 50b – Reference attributes (continued)

Attribute Name	F-UNLINK	F-READ-ATTRIB	F-READ-LINK-ATTRIB	F-CHANGE-ATTRIB
access control	compare		compare & return [1]	
date & time of creation			return	
date & time of last attribute modification				
future object size			return	
identity of creator			return	
identity of last attribute modifier			return	
legal qualifications			return	
linked object type			return	
linked object			return	
object name		return	return	change & return
object size			return	
object type			return	
path access control		compare	return [1]	compare
permitted actions				
primary pathname			return	implicit
private use			return	
storage account			return	

Table 50b – Reference attributes (continued)

Attribute Name	F-CHANGE-LINK-ATTRIB	F-COPY	F-MOVE	F-CLOSE & F-ABORT
access control	compare, change, & return [1]	compare & set	compare & change	
date & time of creation		implicit		
date & time of last attribute modification	implicit	implicit		
future object size	change & return	set	change	
identity of creator		implicit		
identity of last attribute modifier	implicit	implicit		
legal qualifications	change & return	set	change	
linked object type	implicit	implicit		
linked object	change & return	implicit		
object name	change & return	set	change	
object size	implicit	implicit		
object type		implicit		
path access control	change & return [1]	set	change	
permitted actions		implicit		
primary pathname	implicit	implicit	implicit	
private use	change & return	set	change	
storage account	change & return	set	change	

Page 54, replace table 51 by the following table

Table 51 – Activity attributes

Attribute Name	F-INITIALIZE	F-CHANGE-PREFIX	F-GROUP-SELECT	F-SELECT
active contents type				
active legal qualification				implicit
current access control				implicit
current access passwords		change		change
current access request				change
current account	set		change	change
current calling application entity title	set			
current concurrency control				set
current initiator identity	set			
current location				
current locking style				
current name prefix	implicit	change & return		
current object type				set
current pathname				set
current processing mode				
current responding application entity title	implicit & return			
generalized access passwords			set	
generalized access request			set	
generalized concurrency control			set	
generalized selection group			set	

Table 51 – Activity attributes (continued)

Attribute Name	F-SELECT-ANOTHER	F-CREATE	F-CREATE-DIRECTORY	F-LINK
active contents type				
active legal qualification	implicit	set	set	set
current access control	implicit	implicit	implicit	implicit
current access passwords	implicit	set	set	set
current access request	implicit	set	set	set
current account		change	change	change
current calling application entity title				
current concurrency control	implicit	set	set	set
current initiator identity				
current location				
current locking style				
current name prefix				
current object type	implicit	implicit	implicit	implicit
current pathname	implicit	set	set	set
current processing mode				
current responding application entity title				
generalized access passwords				
generalized access request				
generalized concurrency control				
generalized selection group				

Table 51 – Activity attributes (continued)

Attribute Name	F-CHANGE-ATTRIB	F-CHANGE-LINK-ATTRIB	F-COPY	F-MOVE
active contents type				
active legal qualification	change	change	change	change
current access control	change	change	change	change
current access passwords				
current access request				
current account				
current calling application entity title				
current concurrency control				
current initiator identity				
current location				
current locking style				
current name prefix				
current object type		change		
current pathname	change	change	change	change
current processing mode				
current responding application entity title				
generalized access passwords				
generalized access request				
generalized concurrency control				
generalized selection group				

Table 51 – Activity attributes (continued)

Attribute Name	F-OPEN	F-READ & F-WRITE	F-LOCATE & F-ERASE	F-RECOVER
active contents type	set			set
active legal qualification				set
current access control				set
current access passwords				set
current access request	compare			set
current account				
current calling application entity title				
current concurrency control	change			set
current initiator identity				
current location	set	change	change	set
current locking style	set			set
current name prefix				change
current object type				set
current pathname				set
current processing mode	set			set
current responding application entity title				
generalized access passwords				set
generalized access request				set
generalized concurrency control				set
generalized selection group				set

Annex E
(normative)

State transition diagrams

(amend 3rd paragraph, page 59)

Each of the file regime control state transition diagrams are divided into parts. The transitions produced by single primitives are shown in the first set of diagrams, and the transitions produced by grouped sequences are shown in the second set of diagrams.

(amend after last paragraph, before note, page 59)

With the availability of the filestore management functional units, additional group sequences are allowed as defined in clause 7.

(insert after Figure 12, Page 61),

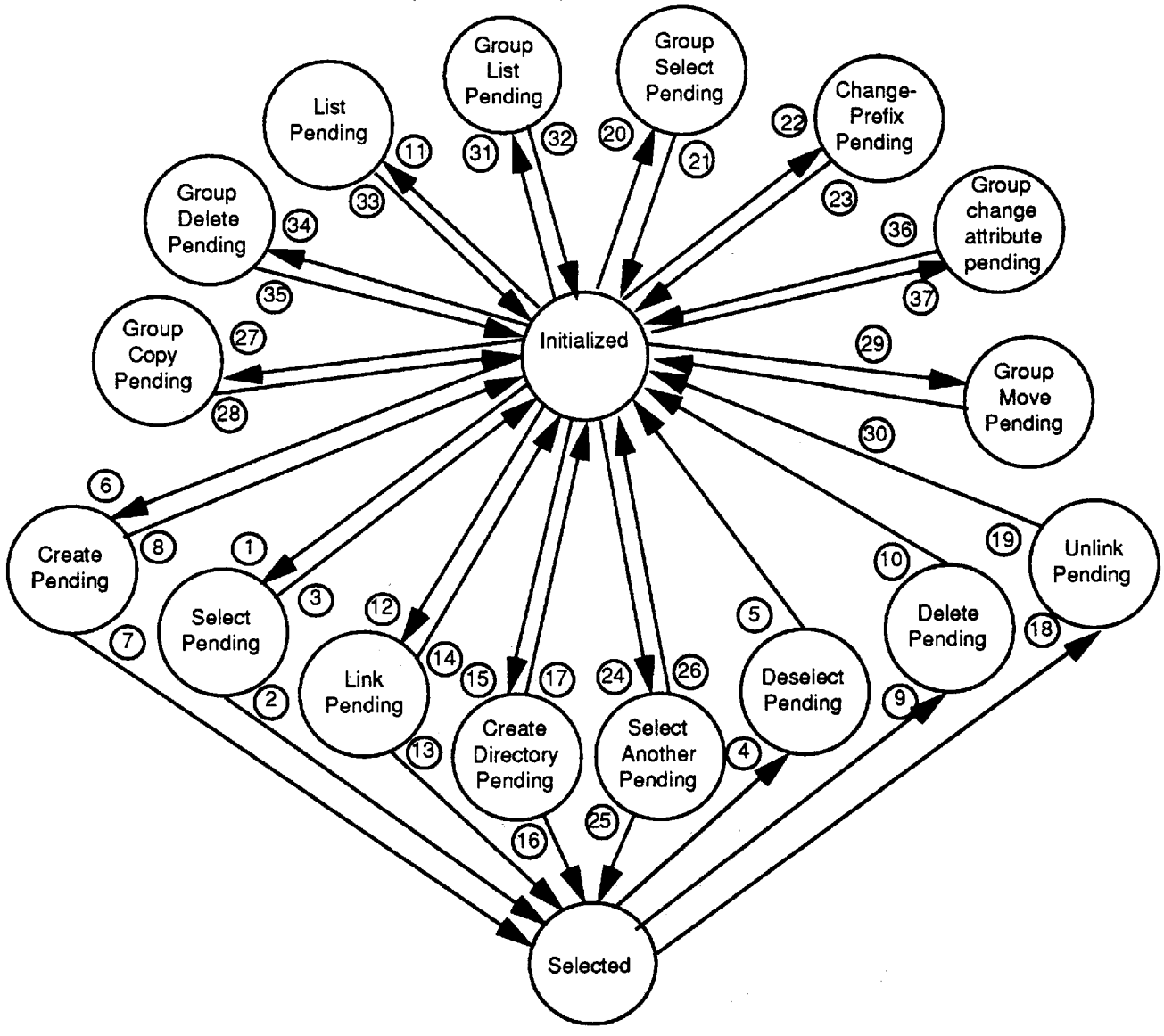


Figure 12a - State Transition Diagram of the Select Regime Establishment Service with Generalized Select (Initiator)

1 F-SELECT request
2 F-SELECT confirm (+)
3 F-SELECT confirm (-)
4 F-DESELECT request
5 F-DESELECT confirm
6 F-CREATE request
7 F-CREATE confirm (+)
8 F-CREATE confirm (-)
9 F-DELETE request
10 F-DELETE confirm
11 F-LIST request
12 F-LINK request
13 F-LINK confirm (+)
14 F-LINK confirm (-)
15 F-CREATE-DIRECTORY request
16 F-CREATE-DIRECTORY confirm (+)
17 F-CREATE-DIRECTORY confirm (-)
18 F-UNLINK request
19 F-UNLINK confirm
20 F-GROUP-SELECT request
21 F-GROUP-SELECT confirm
22 F-CHANGE-PREFIX request
23 F-CHANGE-PREFIX confirm
24 F-SELECT-ANOTHER request
25 F-SELECT-ANOTHER confirm (+)
26 F-SELECT-ANOTHER confirm (-)
27 F-GROUP-COPY request
28 F-GROUP-COPY confirm
29 F-GROUP-MOVE request
30 F-GROUP-MOVE confirm
31 F-GROUP-LIST request
32 F-GROUP-LIST confirm
33 F-LIST confirm
34 F-GROUP-DELETE request
35 F-GROUP-DELETE confirm
36 F-GROUP-CHANGE-ATTRIB req
37 F-GROUP-CHANGE-ATTRIB conf

(amend Figure 13, page 62)

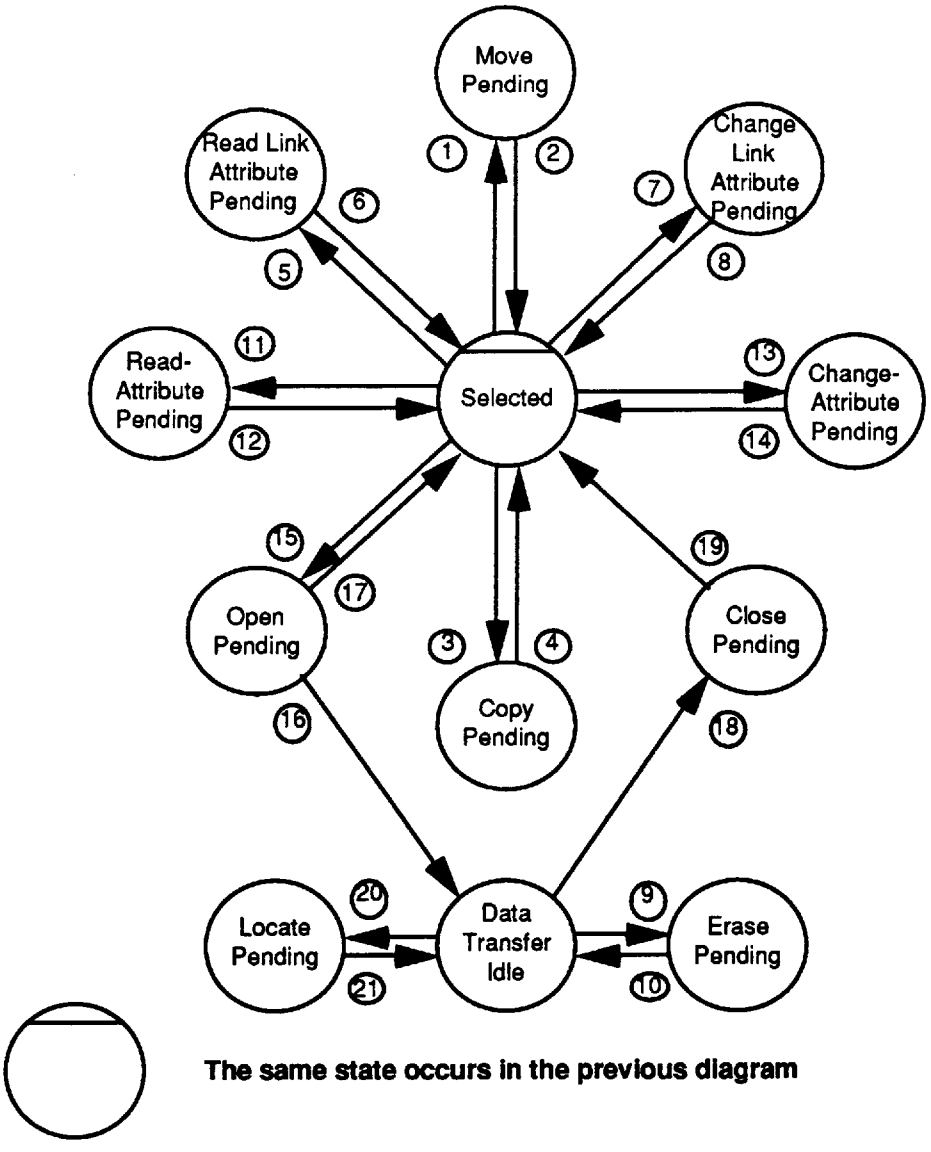


Figure 13 - State Transition Diagram of the File Regime Establishment Service (Initiator)

1	F-MOVE request	8	F-CHANGE-LINK-ATTRIB confirm	15	F-OPEN request
2	F-MOVE confirm	9	F-ERASE request	16	F-OPEN confirm (+)
3	F-COPY request	10	F-ERASE confirm	17	F-OPEN confirm (-)
4	F-COPY confirm	11	F-READ-ATTRIB request	18	F-CLOSE request
5	F-READ-LINK-ATTRIB request	12	F-READ-ATTRIB confirm	19	F-CLOSE confirm
6	F-READ-LINK-ATTRIB confirm	13	F-CHANGE-ATTRIB request	20	F-LOCATE request
7	F-CHANGE-LINK-ATTRIB request	14	F-CHANGE-ATTRIB confirm	21	F-LOCATE confirm

(insert after Figure 14, page 63)

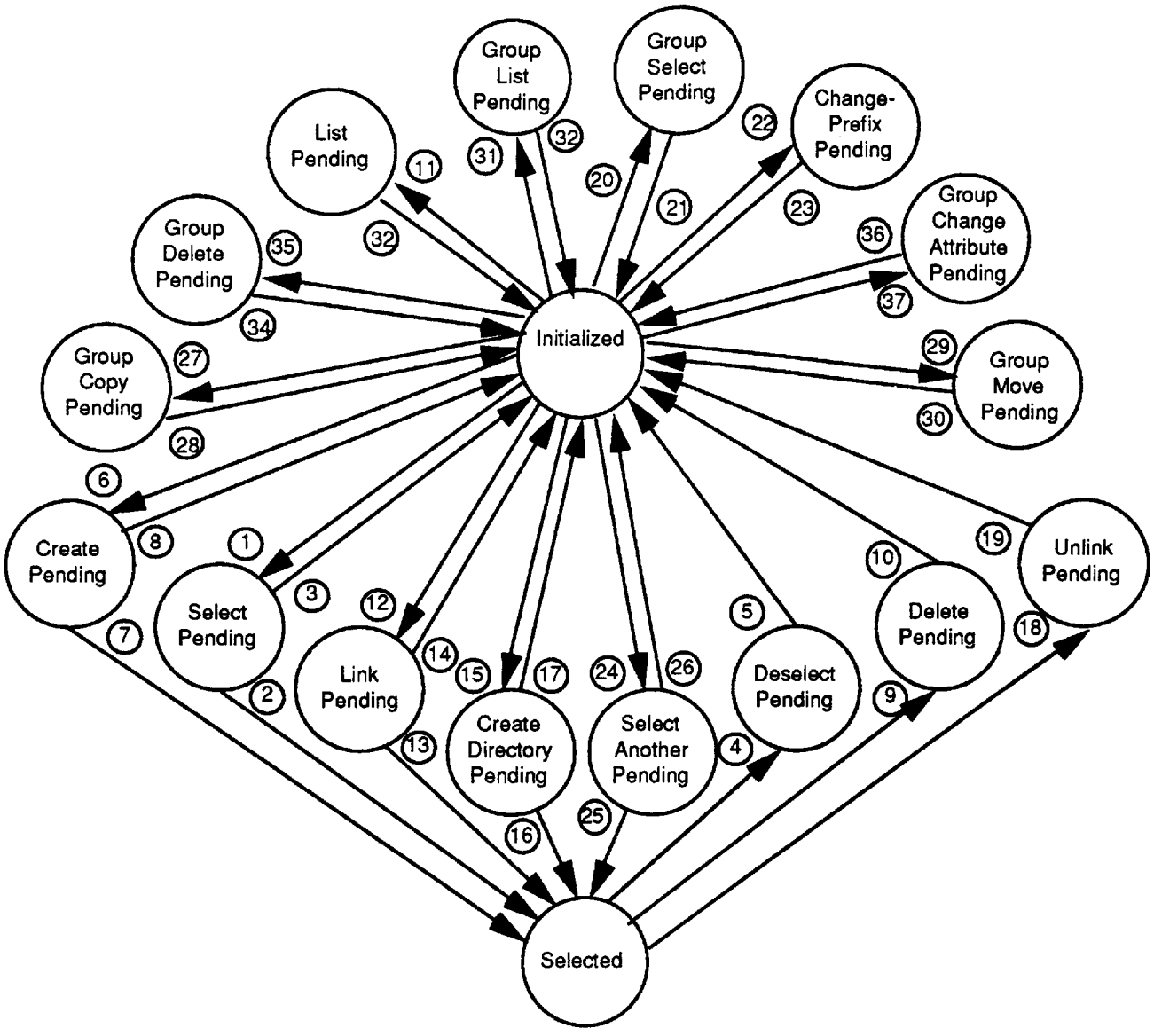


Figure 14a - State Transition Diagram of the Select Regime Establishment Service with Generalized Select (Responder)

1 F-SELECT indication
2 F-SELECT response (+)
3 F-SELECT response (-)
4 F-DESELECT indication
5 F-DESELECT response
6 F-CREATE indication
7 F-CREATE response (+)
8 F-CREATE response (-)
9 F-DELETE indication
10 F-DELETE response
11 F-LISTindication
12 F-LINK indication
13 F-LINK response (+)
14 F-LINK response (-)
15 F-CREATE-DIRECTORY indication
16 F-CREATE-DIRECTORY response (+)
17 F-CREATE-DIRECTORY response (-)
18 F-UNLINK indication
19 F-UNLINK response
20 F-GROUP-SELECT indication
21 F-GROUP-SELECT response
22 F-CHANGE-PREFIX indication
23 F-CHANGE-PREFIX response
24 F-SELECT-ANOTHER indication
25 F-SELECT-ANOTHER indication (+)
26 F-SELECT-ANOTHER indication (-)
27 F-GROUP-COPY indication
28 F-GROUP-COPY response
29 F-GROUP-MOVE indication
30 F-GROUP-MOVE response
31 F-GROUP-LIST indication
32 F-GROUP-LIST response
33 F-LIST response
34 F-GROUP-DELETE indication
35 F-GROUP-DELETE response
36 F-GROUP-CHANGE-ATTRIB ind
37 F-GROUP-CHANGE-ATTRIB resp

(amend Figure 15, page 64)

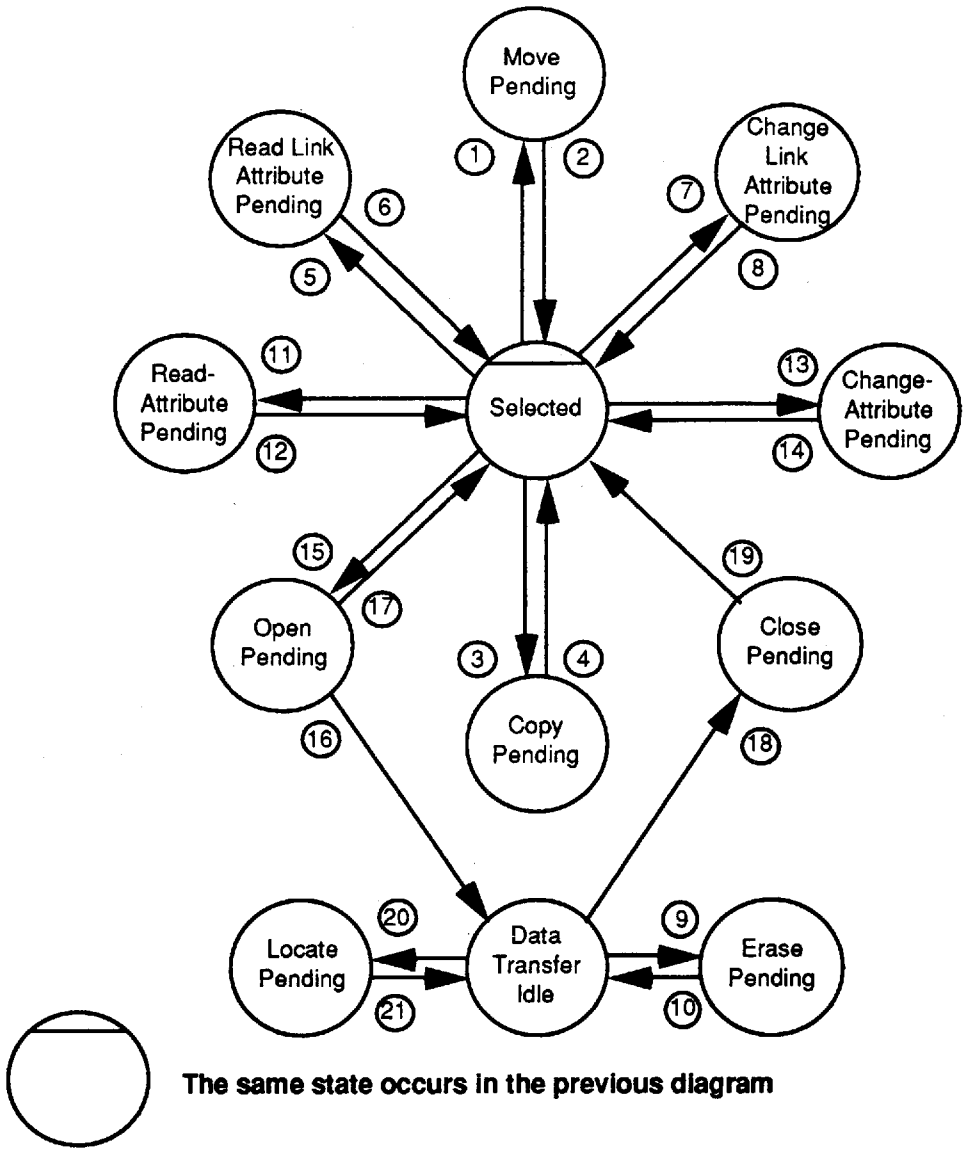


Figure 15 - State Transition Diagram of the File Regime Establishment Service (Responder)

1	F-MOVE indication	8	F-CHANGE-LINK-ATTRIB response	15	F-OPEN indication
2	F-MOVE response	9	F-ERASE indication	16	F-OPEN response (+)
3	F-COPY indication	10	F-ERASE response	17	F-OPEN response (-)
4	F-COPY response	11	F-READ-ATTRIB indication	18	F-CLOSE indication
5	F-READ-LINK-ATTRIB indication	12	F-READ-ATTRIB response	19	F-CLOSE response
6	F-READ-LINK-ATTRIB response	13	F-CHANGE-ATTRIB indication	20	F-LOCATE indication
7	F-CHANGE-LINK-ATTRIB indication	14	F-CHANGE-ATTRIB response	21	F-LOCATE response

ISO 8571-3:1988/Amd.1:1992 (E)

UDC 681.3.01

Descriptors: data processing, information interchange, network interconnection, open systems interconnection, communication procedure, files, management.

Price based on 84 pages

8571 PT*3 88 ■ 4851903 0513152 848 ■

INTERNATIONAL STANDARD ISO 8571-3 : 1988
TECHNICAL CORRIGENDUM 2

Published 1992-10-15

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ELECTROTECHNIQUE INTERNATIONALE

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

Part 3:
File Service Definition

TECHNICAL CORRIGENDUM 2

Systèmes de traitement de l'information — Interconnexion de systèmes ouverts — Gestion, accès et transfert de fichier —

Partie 3: Définition du service de transfert de fichier

RECTIFICATIF TECHNIQUE 2

Technical corrigendum 2 to International Standard ISO 8571-3 : 1988 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

Page 33

Subclause 17.1.2.4

Insert a note after item c) as follows:

"NOTE — Item a) is present to accommodate document types which have not yet been defined by this part of ISO 8571."

Page 53

Annex B

Table 50, at the intersection of the row titled "contents type" and the column titled "F-CREATE", change the entry from "set" to "set & return".

UDC 681.3 : 621.39 **Ref. No. ISO 8571-3 : 1988/Cor.2 : 1992 (E)**

Descriptors: data processing, information interchange, network interconnection, open systems interconnection, files, communication procedure, management.

© **ISO/IEC 1992**

Printed in Switzerland

8571 PT*3 88 ■ 4851903 0105933 8 ■

INTERNATIONAL STANDARD ISO 8571-3 : 1988
TECHNICAL CORRIGENDUM 1

Published 1991-06-01

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

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

Part 3:
File Service Definition

TECHNICAL CORRIGENDUM 1

Systèmes de traitement de l'information — Interconnexion de systèmes ouverts — Gestion, accès et transfert de fichier — Partie 3: Définition du service de transfert de fichier

RECTIFICATIF TECHNIQUE 1

Technical corrigendum 1 to International Standard ISO 8571-3 : 1988 was prepared by ISO/IEC JTC 1, *Information technology.*

Page 2

Clause 2

Delete reference to ISO 9804, ISO 9805, and the footnote and insert the following:

"ISO/IEC 9804, *Information technology — Open Systems Interconnection — Service definition for the Commitment, Concurrency and Recovery service element.*

ISO/IEC 9805, *Information technology — Open Systems Interconnection — Protocol specification for the Commitment, Concurrency and Recovery service element."*

UDC 681.3.621.39 **Ref. No. ISO 8571-3 : 1988/Cor.1 : 1991 (E)**

Descriptors: data processing, information interchange, network interconnection, open systems interconnection, files, communication procedure, management.

© **ISO/IEC 1991**

Printed in Switzerland

Page 6

Subclause 8.2.1

Second paragraph, line 4, delete "is a sequence of:" and insert "requested by the initiator is a series of requests consisting of:"

Subparagraphs 1) and 3), lines 1, delete "grouped sequence" and insert "grouped series of requests"

Page 8

Subclause 8.2.3

Second paragraph, line 4, delete "is a sequence of:" and insert "requested by the initiator is a series of requests consisting of:"

Subclause 8.2.4

Second paragraph, line 4, delete "is either;" and insert "requested by the initiator is either:"

Subparagraph 1), line 1, delete "grouped sequence" and insert "grouped series of requests" and line 2, delete "sequence" and insert "series"

Subparagraph 3), line 1, delete "grouped sequence" and insert "grouped series of requests" and line 9, delete "or a single grouped sequence to effect file management" and insert "or a single grouped series of requests to effect file management"

Page 14

Subclause 12.2

Line 4, delete "primitive sequences" and insert "series of requests requested by the initiator"

Page 21

Subclause 13.8

Delete the text of 13.8 and insert the following:

"The concurrency control parameter provides a mechanism for the initiator to ensure that an initiator has a consistent view of the file by restricting shared access. The value is a vector whose elements indicate, for each of the actions specified in the requested access parameter, independent control of the possible set of actions on the selected or newly created file for duration of the select regime. Those actions specified in the concurrency control parameter, and not requested by the initiator in the requested access parameter, fall outside of the possible actions on the file and hence are not controllable by the initiator. Thus, the only locks useable for the available actions (those specified by the initiator in the requested access parameter) are *not required, shared, exclusive* and *no access.*

"For every file section or file open regime, each lock is either defaulted or specified in one of the following categories:

a) not required — I will not perform the operation — others may;

b) shared — I may perform the operation — so can others;

c) exclusive — I can perform the operation — others cannot;

d) no access; no one may perform the action.

"Independent control is available on the following actions: read, insert, replace, extend, erase, read attribute, change attribute and delete file.

"If the FADU locking functional unit has not been negotiated or the *Enable FADU locking* parameter on the F-OPEN request was set off the scope of the concurrency lock remains in place for the duration of the regime that the lock was requested (SELECT or OPEN).

"If the FADU locking is requested, the *FADU locking* functional unit is negotiated and the *Enable FADU locking* parameter on the F-OPEN request was set on, concurrency controls specified at the time the file is opened are not applied immediately (although any concurrency control requested at file selection remains in force). Instead the control requested is applied for the duration of each file access requested. In addition, individual file access data units (FADUs) can have a lock applied for some period within the file open regime, bounded by a pair of file access actions which are marked by the *Enable FADU locking* parameter.

"This parameter is used to set the current concurrency control activity attribute."

Page 21

Subclause 13.9

Delete the text of 13.9 and insert the following:

"The lock on a file access data unit can have two states — on or off. While the lock is off, the actions specified as shared for the file as a whole can be performed concurrently with other similar accesses. However when the lock is on as a result of setting the *Enable FADU locking* parameter, that user takes *exclusive* control (or *no access* in place of actions requested as *not required*) of the whole file access data unit concerned.

"Locks may be turned on or off in association with the F-LOCATE, F-READ, or F-WRITE actions; requests to turn a lock on are honoured before the associated action is performed, while requests to turn them off are honoured afterwards.

"This parameter is not used to set any activity attribute."

Page 23

Subclause 14.1.2.4

Line 2, after "entity." add "This parameter could form part of the access control mechanism and if it is not supplied, certain accesses could fail."

Subclause 14.1.2.5

Line 3, after "failure." add "This parameter is mandatory if the recovery functional unit is available on the association."

Page 24

Table 11

Change the entries as follows:

Parameter	F-INITIALIZE request	F-INITIALIZE indication	F-INITIALIZE response	F-INITIALIZE confirm
. . .	. . .	. . .	. . .	. . .
Calling Application Title	Optional	Optional (=)		
Responding Application Title			Conditional	Conditional (=)
Called Presentation Address	Optional	Optional (=)		
Calling Presentation Address	Optional	Optional (=)		
Responding Presentation Address			Optional	Optional (=)
. . .	. . .	. . .	. . .	. . .
Functional Units	Mandatory	Mandatory	Mandatory	Mandatory (=)
. . .	. . .	. . .	. . .	. . .

Add the following note:

"NOTE — Based on the availability of a directory service facility within the Application Process on the local system, the three presentation address parameters are optional as they are derived from their corresponding Application Title parameters by the directory function within the Application Process."

Page 29

Table 17

Change parameter "Concurrency Control" as follows:

Parameter	F-CREATE request	F-CREATE indication	F-CREATE response	F-CREATE confirm
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
Concurrency Control	Optional	Optional (=)		
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

Subclause 15.3.2.4

Line 12, delete "attributes." and insert "attributes, and also to any attributes that have their values reduced to "no value available"."

Add the following note:

"NOTE — When the *override* parameter is used with values *b* (select old file) or *c* (delete old file and create with old attributes) and the named file already exists, the *initial attributes* parameter is ignored as no new attributes are required."

Page 33

Subclause 17.1.2.5

Delete the last paragraph and insert the following:

"If the FADU locking is requested, the *FADU locking* functional unit is negotiated and the *Enable FADU locking* parameter on the F-OPEN request was set on, concurrency controls specified at the time the file is opened are not applied immediately (although any concurrency control requested at file selection remains in force). Instead, the control requested is applied for the duration of each file access requested."

Subclause 17.1.2.7

Delete the second and third paragraphs and insert the following:

"If FADU locking is available, concurrency controls not available may not fail until the specific action is requested within the OPEN regime. As these locks are not applied until the specific file access action is requested. This parameter is not used to set any activity attribute."

Subclause 17.1.2.10

Line 5 and b), delete "at start of file" and insert "at start of transfer"

Page 34

Clause 18

Delete the third paragraph and insert the following:

"If the threshold number of primitives is not processed before failure effecting regime creation is detected, a negative response is made for the first primitive in the group which has a state result parameter. The state result parameter of this primitive will indicate a failure. The diagnostic on the response of this primitive is set to indicate "transient error" or "permanent error", as appropriate, with an error identifier of "subsequent error" as the first component of its diagnostic, and subsequent diagnostic components, in the sequence of diagnostics, carrying the actual detailed diagnostic information for the failure."

Page 49

Annex A, table 43

Change the "Identifiers" as follows:

Identifier	Observer	Source
0	1,2,4,5	0-5
2	1,2,4,5	0-5
8	1,2,4,5	0-5
9	1,2,4,5	0-5

Annex A, table 44

Change the "Identifiers" as follows:

Identifier	Observer	Source
1011	2,4	3
1012	2,4	2
1013	1,2,4,5	0-5
1014	1,2,4,5	0-5

Page 50

Annex A, table 45

Change the "Identifiers" as follows:

Identifier	Observer	Source
2000	5	1
2014	2,4	2
2019	5	1

Page 51

Annex A, table 48

Change the "Identifiers" as follows:

Identifier	Observer	Source
5014	5	1
"	2,4	2,4

Page 53

Annex B, table 50

Change the Attribute Name "permitted actions" as follows:

Attribute Name	F-SELECT	F-CREATE
permitted actions	compare	set & return

Change the Attribute Name "identity of creator" as follows:

Attribute Name	F-SELECT	F-CREATE
identity of creator		implicit

Page 55

Annex C

Delete annex C and insert the following:

"Annex C
(informative)

File transfer with commitment control

NOTE – ISO/IEC 9804 and ISO/IEC 9805 (CCR) have recently been published but do not correspond to the text on which annex C was originally based. A normative revision of this annex is in the course of preparation and will describe the relationship between corresponding CCR and FTAM services based on ISO/IEC 9804 and ISO/IEC 9805."

Page 59

Annex E

Fifth paragraph, delete lines 1 and 2 beginning "The valid grouped sequences..." and insert the following:

"The valid grouped sequences for the initiator, the *positive* valid grouped sequences for the responder, and the notation used in figures 14 and 16, are the following:"

8571 PT 3-88 4851903 0063560 3 Q-60-04

INTERNATIONAL STANDARD

ISO 8571-3
First edition
1988-10-01

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION
МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ

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

Part 3 :
File Service Definition

Systèmes de traitement de l'information — Interconnexion de systèmes ouverts — Gestion, accès et transfert de fichier —

Partie 3 : Définition du service de transfert de fichier

Reference number
ISO 8571-3 : 1988 (E)

D548

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

Draft International Standards adopted by the technical committees are circulated to the member bodies for approval before their acceptance as International Standards by the ISO Council. They are approved in accordance with ISO procedures requiring at least 75 % approval by the member bodies voting.

International Standard ISO 8571-3 was prepared by Technical Committee ISO/TC 97, *Information processing systems.*

Users should note that all International Standards undergo revision from time to time and that any reference made herein to any other International Standard implies its latest edition, unless otherwise stated.

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*

Annexes A, B, C, D and E form an integral part of this International Standard.

© International Organization for Standardization, 1988 •

Printed in Switzerland

Contents

Page

0 Introduction 1

1 Scope and field of application 1

2 References 1

3 Definitions 2

4 Abbreviations 2

5 Conventions 2

Section one: General

6 Model of the file service 3

6.1 File service provider and file service users 3

6.2 File service levels 3

6.3 Regimes of the file service 4

7 Services of the file service 4

7.1 FTAM regime control 4

7.2 Filestore management 4

7.3 File selection regime control 4

7.4 File management 5

7.5 File open regime control 5

7.6 Grouping control 5

7.7 Access to file content 5

7.8 Bulk data transfer 5

7.9 Recovery 5

7.10 Checkpointing and restarting 5

8 Functional units and service classes 5

8.1 Functional units 5

8.2 Service classes 6

8.3 Application Entity roles 8

9 Levels of file service 8

10 Negotiation of service class, FTAM QoS and functional units 9

10.1 Service Class 9

10.2 FTAM Quality of Service 9

10.3 Functional units 9

Section two : Definition of file service primitives

11 File service primitives 11

12 Sequences of primitives 11

12.1 Normal sequences 11

12.2 Constraints on the issue of primitives 11

12.3 Conventions 14

12.4 Confirmed Services 20

13 Common file service parameters 20

13.1 State result 20

13.2 Action result 20

13.3 Account 20

13.4 Charging 20

13.5 Attributes 20

13.6 Requested access 20

13.7 Access Passwords 21

13.8 Concurrency Control 21

13.9 FADU Lock 21

13.10 Shared ASE information 21

13.11 Activity Identifier 21

13.12 File Access Data Unit Identity 21

13.13 Diagnostic 22

14 FTAM regime control 23

14.1 FTAM regime establishment service 23

14.2 FTAM regime termination service (orderly) 26

14.3 FTAM regime termination service (abrupt) 26

15 File selection regime control 27

15.1 File selection service 27

15.2 File deselection service 28

15.3 File creation service 28

15.4 File deletion service 30

16 File management 31

16.1 Read attribute service 31

16.2 Change attribute service 31

17 File open regime control 31

17.1 File open service 32

17.2 File close service 33

18 Grouping control 34
18.1 Beginning of grouping service 34
18.2 End of grouping service 35
19 Recovery (Internal service only) 35
19.1 Regime recovery service 35
20 Access to file contents 36
20.1 Bulk data transfer service 36
20.2 Locate file access data unit service 36
20.3 Erase file access data unit service 37
Section three: Definition of bulk data transfer primitives
21 Bulk data transfer service primitives 38
22 Sequences of bulk data transfer primitives 38
22.1 Normal sequences 38
22.2 Constraints on issue of primitives 38
23 Common bulk data transfer parameters 44
23.1 Bulk Data Transfer Specification 44
23.2 Checkpoint Identifier 44
24 Bulk data transfer 44
24.1 Read bulk data service 44
24.2 Write bulk data service 44
24.3 Data unit transfer service 44
24.4 End of data transfer service 45
24.5 End of transfer service 45
24.6 Cancel data transfer service 45
24.7 Sequence of primitives on write 46
24.8 Sequence of primitives on read 46
25 Checkpointing and restart (Internal BDT Service Only) 46
25.1 Checkpointing service 46
25.2 Restarting data transfer service 47

Annexes

A Diagnostic parameter values 48

B Relation of attributes to primitives 53

C File transfer with commitment control 55

D Reference to FTAM control information 58

E State transition diagrams 59

Figures

1 Service Levels 3

2 File service regimes and related primitives 4

3 Simplified State Diagram for successful activity (see Annex E) 13

4 Confirmed service 20

5 F-U-ABORT service 27

6 F-P-ABORT service 27

7 F-P-ABORT collision 27

8 Simplified State Diagram for Bulk Data Transfer (see Annex E) 39

9 Sequence of primitives on write 46

10 Sequence of primitives on read 47

11 State Transition Diagram for Association Establishment (Initiator) 60

12 State Transition Diagram for Association Establishment (Responder) 61

13 State Transition Diagram of the File Regime Establishment Service (Initiator) 62

14 State Transition Diagram for Grouped Sequences (Initiator) 63

15 State Transition Diagram of the File Regime Establishment Service (Responder) 64

16 State Transition Diagram for Grouped Sequences (Responder) 65

17 State Transition Diagram for the Bulk Data Transfer Service (Initiator) 66

18 State Transition Diagram for the Bulk Data Transfer Service (Responder) 67

Tables

1 Services and functional units of the External File Service 7
2 Services and functional units of the Internal File Service 7
3 Functional units in the file services 9
4 Service Class Combinations 9
5 Service Class Negotiation 10
6 File service primitives 11
7 Sequence of service primitives for FTAM regime establishment — initiator 15
8 Sequence of service primitives for FTAM regime establishment — responder 15
9 Sequence of service primitives for file service regimes — initiator 16
10 Sequence of service primitives for file service regimes — responder 18
11 F-INITIALIZE parameters 24
12 F-TERMINATE parameters 26
13 F-U-ABORT parameters 26
14 F-P-ABORT parameters 26
15 F-SELECT parameters 27
16 F-DESELECT parameters 28
17 F-CREATE parameters 29
18 F-DELETE parameters 30
19 F-READ-ATTRIB parameters 31
20 F-CHANGE-ATTRIB parameters 32
21 F-OPEN parameters 32
22 F-CLOSE 34
23 F-BEGIN-GROUP parameters 35
24 F-RECOVER parameters 35
25 BDT read sub-parameters 36
26 BDT write sub-parameters 36
27 Access contexts 36
28 F-LOCATE parameters 37
29 F-ERASE parameters 37
30 Bulk data transfer service primitives 38
31 Sequence of service primitives for bulk data transfer — initiator 40
32 Sequence of service primitives for bulk data transfer — responder 42
33 F-READ parameters 44

34 F-WRITE parameters 44

35 F-DATA parameters 45

36 F-DATA-END parameters 45

37 F-TRANSFER-END parameters 45

38 F-CANCEL parameters 46

39 F-CHECK parameters 47

40 F-RESTART parameters 47

41 Error types 48

42 Sources and observers of errors 48

43 General FTAM diagnostics 49

44 Protocol and supporting service related diagnostics 49

45 Association related diagnostics 50

46 Selection related diagnostics 50

47 File management related diagnostics 51

48 Access related diagnostics 51

49 Recovery related diagnostics 52

50 File attributes 53

51 Activity attributes 54

52 File service primitives associated with CCR primitives 56

53 FTAM Primitives with Shared ASE Information parameters 56

54 Composite FTAM actions 57

INTERNATIONAL STANDARD ISO 8571-3 : 1988 (E)

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

Part 3 :
File Service Definition

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 a File Service and specifies a File 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, which may be stored within open systems or passed between application processes.

ISO 8571 defines a basic file service. It provides sufficient facilities to support file transfer, and establishes a framework for file access and file management. ISO 8571 does not specify the interfaces to a file transfer or access facility within the local system.

It is recognised that, with respect to Communication Quality of Service, (described in 14.1.2.16), work is still in progress to provide an integrated treatment of quality of service across all of the layers of the OSI Reference Model and to ensure that the individual treatments in each layer service satisfy overall quality of service objectives in a consistent manner. As a consequence, an addendum may be added to this International Standard at a later time which reflects further quality of service developments and integration.

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

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

Annex A - Diagnostic parameter values
Annex B - Relation of attributes to primitives
Annex C - File transfer with commitment control
Annex D - Reference to FTAM control information
Annex E - State transition diagrams

1 Scope and field of application

This part of ISO 8571 defines in an abstract way the externally visible file transfer, access and management service within the OSI Application Layer in terms of:

a) the primitive actions and events of the service;

b) the parameter data associated with each primitive action and event;

c) the relationship between, and the valid sequences of, these actions and events.

The service defined in ISO 8571-3 is that which is provided by the OSI file transfer, access and management protocol ISO 8571-4 in conjunction with the Association Control Service Elements ISO 8649 and with the Presentation service ISO 8822.

ISO 8571-3 does not specify individual implementations or products, nor does it constrain the implementation of entities and interfaces within a computer system. There is, therefore, no conformance to this part of ISO 8571.

2 References

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

ISO/TR 8509, *Information Processing Systems - Open Systems Interconnection - Service Conventions.*

ISO 8571, *Information Processing Systems - Open Systems Interconnection - File transfer, access and management.*
- *Part 1: General introduction.*
- *Part 2: Virtual Filestore definition.*
- *Part 4: File Protocol specification.*

ISO 8649, *Information Processing Systems -Open Systems Interconnection - Service definition for the Association Control Service Element.*

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

ISO 8831, *Information Processing Systems -Open Systems Interconnection - Job Transfer and Manipulation Concepts and Services.*

ISO 9804, *Information Processing Systems -Open Systems Interconnection - Definition of Application Service Elements - Commitment, Concurrency and Recovery.* [1)]

ISO 9805, *Information Processing Systems -Open Systems Interconnection - Specification of protocols for Application Service Elements - Commitment, Concurrency and Recovery.* [1)]

3 Definitions

Terms used in ISO 8571-3 are defined in ISO 8571-1.

4 Abbreviations

Abbreviations used in ISO 8571-3 are defined in ISO 8571-1.

5 Conventions

ISO 8571-3 uses the descriptive conventions in the OSI Service Conventions in ISO/TR 8509.

[1)] At present at the stage of draft; publication anticipated in due course.

Section one: General

6 Model of the file service

6.1 File service provider and file service users

ISO 8571-3 uses the abstract model for a service defined in the OSI service conventions in ISO TR 8509 (see note 1). The model defines the interactions between the two file service users and the file service provider. Information is passed between a file service user and the file service provider by file service primitives which may carry parameters.

One of the file service users is defined as the initiator and the other is defined as the responder (see ISO 8571-1).

The responder is an application-entity handling a virtual filestore. The virtual filestore has the properties defined in ISO 8571-2, and may be realized in a real system by a real filestore or by an application process. Attributes of the virtual filestore manipulated by the service primitives defined in ISO 8571-3 are listed in annex B.

The file service defines a single activity between an initiator and a responder (see note 2).

NOTES

1 ISO/TR 8509 defines a model for the service provided by a layer of the OSI Reference Model. The file service does not correspond to such a layer (it is a division within the application layer) but the model used is identical in all other respects.

2 At any one time, an application entity may be involved in more than one instance of the file service activity and each instance is based on a separate application association.

6.2 File service levels

Two levels of file service are defined:

a) the external file service (EFS), in which the user states its FTAM quality of service requirements, but has no awareness of error recovery, delegating such considerations to the service provider. Transfer of file data is modelled in the external file service as a series of error-free operations. Thus within the external file service there is no visibility of recoverable errors or the error recovery actions;

b) the internal file service (IFS) used by the error recovery protocol machine. This service includes primitives giving its users facilities for error recovery and control of the checkpointing mechanisms. The protocol specification which relates the external to the internal file service therefore consists of a standard set of procedures for error recovery and the protocol machine which executes these procedures is the user of the internal file service. The choice of the error recovery procedures to be used is based on the analysis of cost from the FTAM and communications quality of service requested in the External Service and local management information.

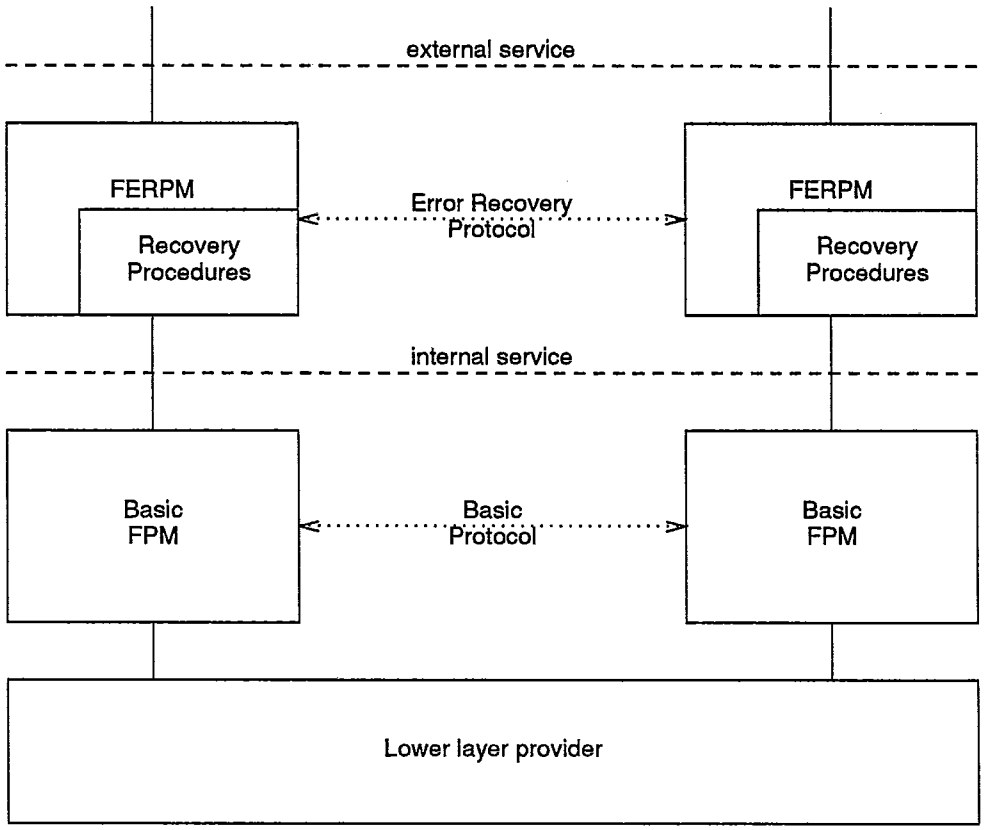


Figure 1 — Service Levels

The relationship between the internal and external file service services is shown diagrammatically in figure 1.

6.3 Regimes of the file service

Four types of file service regime are defined:

a) the FTAM regime which exists while the application association is used for the FTAM protocol.

b) the file selection regime during which a particular file is associated with the FTAM regime;

c) the file open regime during which a particular set of processing mode, presentation contexts and concurrency controls is in operation;

d) the data transfer regime during which a particular bulk data transfer specification and direction of transfer are in force.

There is at most one instance of each type of regime at any one time.

The file service provides for:

e) a sequence of file selection regimes in an FTAM regime;

f) a sequence of file open regimes in a file selection regime;

g) a sequence of data transfer regimes within a file open regime; the data transfer regimes may each be for either read or write data transfer. Write data transfer permits the operations insert, replace or extend.

Termination of a regime implies termination of all regimes nested within that regime. The nesting of regimes is shown in figure 2.

7 Services of the file service

This clause provides a short description of the services of the file service. The services and the primitives by which they are invoked are defined in sections two and three. For each service, the user of the service (the application entity that invokes the sequence of primitives) is stated. The external and internal file service levels are defined in 6.2.

7.1 FTAM regime control

Three services are associated with FTAM regime control:

a) the FTAM regime establishment service (see 14.1) is used by the initiator to create and bind an FTAM regime to the application association linking the two file service users;

b) the FTAM regime termination service (orderly) (see 14.2) is used by the initiator to dissolve the FTAM regime and unbind it from the application association between the file service users and the file service provider;

c) the FTAM regime termination service (abrupt) (see 14.3) is used by either of the service users or the service provider to dissolve the FTAM regime and its binding to the application association unconditionally.

7.2 Filestore management

ISO 8571-3 does not define any filestore management operations.

NOTE - Such operations may be included in future addenda to ISO 8571-3.

7.3 File selection regime control

Four services are associated with file selection regime control:

a) the file selection service (see 15.1) is used by the initiator to select an existing file and to bind the specified file to the FTAM regime;

b) the file deselection service (see 15.2) is used by the initiator to release the binding between the FTAM regime and the specified file;

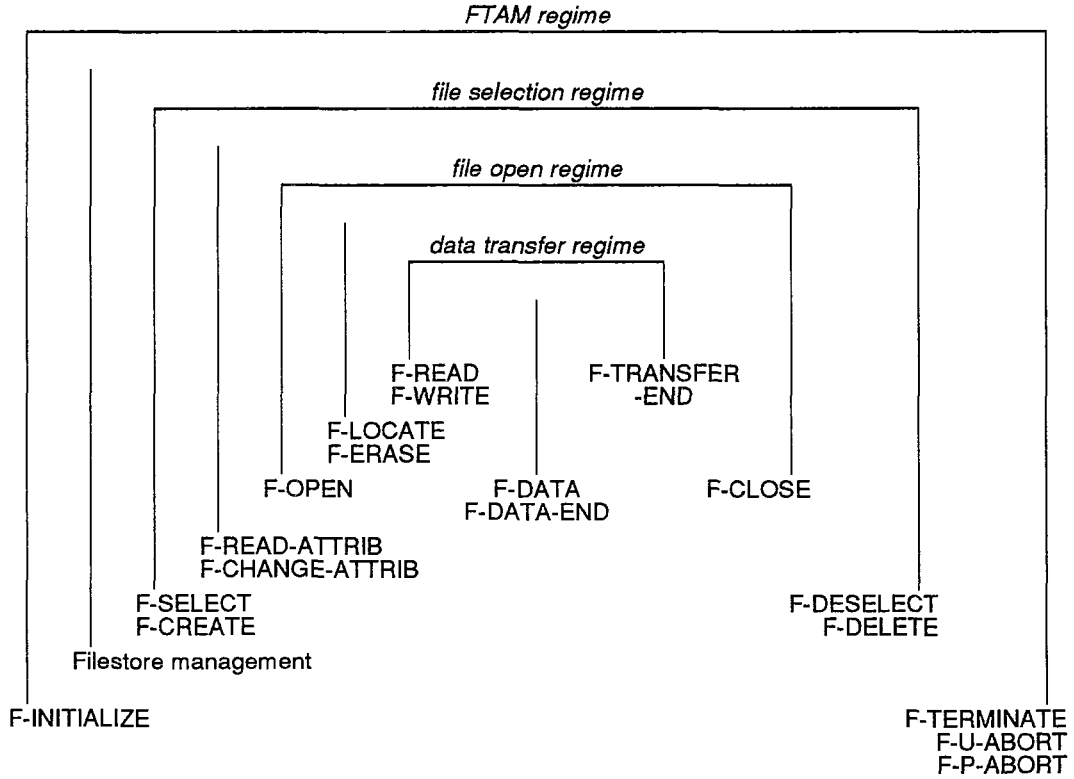


Figure 2 — File service regimes and related primitives

c) the file creation service (see 15.3) is used by the initiator either

1) to create a specified file and to select the newly created file; or

2) depending on the override parameter of F-CREATE, to select an existing file;

and then to bind the specified file to the FTAM regime;

d) the file deletion service (see 15.4) is used by the initiator to release the binding between the FTAM regime and the specified file in such a way that the previously selected file ceases to exist.

7.4 File management

Two services are associated with file management:

a) the read attributes service (see 16.1) is used by the initiator to interrogate the file attributes of the selected file;

b) the change attributes service (see 16.2) is used by the initiator to modify the file attributes of the selected file.

7.5 File open regime control

Two services are associated with file open regime control:

a) the file open service (see 17.1) is used by the initiator to establish the processing mode, presentation contexts and concurrency controls for data transfer or access;

b) the file close service (see 17.2) is used by the initiator to release the context established by the file open service.

7.6 Grouping control

Two services are associated with grouping control:

a) the beginning of grouping service (see 18.1) is used by the initiator to indicate the start of a set of primitives which are to be processed and responded to as a group;

b) the end of grouping service (see 18.2) is used by the initiator to indicate the end of a set of grouped primitives which are to be processed and responded to as a group.

7.7 Access to file content

The transfer of the FADUs of a file is performed by a bulk data transfer procedure, which forms a self contained unit. The services which make up this procedural unit are described in 7.9 and 7.10. There are two additional services associated with file access:

a) the locate file access data unit service (see 20.2) is used by the initiator to specify the identity of a file access data unit which is to be located by the responder;

b) the erase file access data unit service (see 20.3) is used by the initiator to remove a file access data unit from the file.

7.8 Bulk data transfer

Bulk data transfer refers to the transfer, optionally with checkpointing, of single file access data units (see 20.1). There are six additional services associated with different stages in bulk data transfer:

a) the read bulk data service (see 24.1) is used by the initiator to initiate a bulk data transfer from the responder (in the role of sender), to the initiator (in the role of receiver);

b) the write bulk data service (see 24.2) is used by the initiator to initiate a bulk data transfer from the initiator (in the role of sender) to the responder (in the role of receiver);

c) the data unit transfer service (see 24.3) is used by the sender to transmit bulk data;

d) the end of data transfer service (see 24.4) is used by the sender to indicate completion of the data transfer;

e) the end of transfer service (see 24.5) is used by the initiator to confirm that the data transfer is complete;

f) the cancel data transfer service (see 24.6) is used by either the sender or the receiver to cancel a data transfer activity.

7.9 Recovery

One service is associated with recovery. This service is visible only in the internal file service.

The regime recovery service (see 19.1) is used by the initiator to recreate the open regime after a failure within the open regime. Errors occurring outside the open regime are not recoverable by this service.

NOTE - Concurrency controls remain in force during a recovery attempt, or if so specified by another ASE. Otherwise, concurrency controls are released on a permanent error (see 13.2).

7.10 Checkpointing and restarting

Two services are associated with checkpointing and restarting. These services are visible only in the internal file service:

a) the checkpointing service (see 25.1) is used by the sender of data to establish marks in the flow of data for the purpose of subsequent recovery or restart;

b) the restarting data transfer service (see 25.2) is used by the sender or the receiver of data to interrupt a transfer in progress and negotiate a point at which it is to be restarted.

8 Functional units and service classes

Functional units and file service classes are logical groupings of related services defined in ISO 8571-3 for the purpose of:

a) negotiation of the file service user's requirements during FTAM regime establishment;

b) reference by other International Standards.

NOTE - The constraint set applicable to the file restricts the functionality given in 8.1 and 8.2.

8.1 Functional units

The services associated with each functional unit are specified in tables 1 and 2.

8.1.1 Kernel functional unit

The kernel functional unit supports the basic file services for the establishment and release of the FTAM and file selection regimes.

8.1.2 Read functional unit

The read functional unit supports establishment and release of the open regime and the transfer of data from the responder to the initiator.

8.1.3 Write functional unit

The write functional unit supports establishment and release of the open regime and the transfer of data from the initiator to the responder.

8.1.4 File access functional unit

The file access functional unit allows an FADU in the file access structure to be located for file access and allows an FADU to be manipulated within the file access structure.

8.1.5 Limited file management functional unit

The limited file management functional unit supports file management for the creation and deletion of files and for the interrogation of file attributes.

8.1.6 Enhanced file management functional unit

The enhanced file management functional unit extends the capabilities of the limited file management functional unit to include the modification of file attributes.

8.1.7 Grouping functional unit

The grouping functional unit allows several regimes to be established in one exchange, by combining several independent primitives into a group for procedural purposes.

8.1.8 FADU Locking functional unit

The FADU locking functional unit allows the invocation of concurrency control locks on a per FADU basis in addition to a file basis.

8.1.9 Recovery functional unit

The recovery functional unit allows the initiator to recreate an open regime which has been destroyed by some failure. The recovery may be immediate or deferred on the existing, or different a, association.

8.1.10 Restart data transfer functional unit

The restart data transfer functional unit allows a data transfer to be interrupted and restarted immediately at a negotiated point within the current transfer.

8.1.11 Service classes and functional units

Table 1 shows which of the functional units are mandatory and which optional in each of the service classes: transfer, access, management, transfer and management, and unconstrained. Service classes are defined in 8.2.

The recovery and restart functional units are never explicitly visible in the external file service. Where they are shown as optional in table 2 it is in the internal file service that they are optional. The external file service and the internal file service are defined in 6.2.

8.2 Service classes

Five file service classes are defined in terms of combinations of functional units:

a) file transfer class (see 8.2.1);

b) file access class (see 8.2.2);

c) file management class (see 8.2.3);

d) file transfer and management class (see 8.2.4);

e) unconstrained class (see 8.2.5).

8.2.1 File transfer class

The file transfer class consists of;

a) the kernel functional unit;

b) the grouping functional unit;

c) one or both of the read or write functional units;

d) optionally, the limited file management functional unit;

e) optionally, but only if the limited file management functional unit is present, the enhanced file management functional unit;

f) optionally, in the internal file service, the recovery functional unit.

g) optionally, in the internal file service, the restart data transfer functional unit.

In the file transfer service class, the use of the services is constrained so that there is a sequence of zero or more FTAM events on the application association. Each FTAM event is a sequence of:

1) a single grouped sequence to establish a file open regime. This sequence consists of:

- an F-BEGIN-GROUP primitive;
- an F-SELECT or F-CREATE primitive;
- optionally, an F-READ-ATTRIB primitive;
- optionally, an F-CHANGE-ATTRIB primitive;
- an F-OPEN primitive;
- an F-END-GROUP primitive.

2) a single bulk data transfer procedure, for either a read transfer or a write transfer. The processing mode parameter on the F-OPEN primitive is set to either a read or a valid write action, as defined in the constraint set, but not both.

3) a single grouped sequence to release the file open and select regimes. This sequence consists of:

- an F-BEGIN-GROUP primitive;
- an F-CLOSE primitive;
- optionally, an F-READ-ATTRIB primitive;
- optionally, an F-CHANGE-ATTRIB primitive;
- an F-DESELECT or F-DELETE primitive;
- an F-END-GROUP primitive.

The threshold parameter is set equal to the number of primitives between F-BEGIN-GROUP and F-END-GROUP. The threshold parameter is defined in clause 18.

NOTE - Each of the primitives shown as optional in these sequences may only be present if the corresponding functional unit was negotiated during the FTAM regime establishment.

8.2.2 File access class

The file access class consists of:

a) the kernel functional unit;

b) both of the read and write functional units;

c) the file access functional unit;

d) optionally, the grouping functional unit. If the grouping functional unit is successfully negotiated, its valid use in any instance by the initiator is optional but its acceptance by the responder is always mandatory.

Table 1 — Services and functional units of the External File Service

Functional Unit	Services	Service classes					Clause
		T	A	M	TM	U	
U1 Kernel	FTAM regime establishment FTAM regime termination (orderly) FTAM regime termination (abrupt) File selection File deselection	M	M	M	M	M	14.1 14.2 14.3 15.1 15.2
U2 Read	Read bulk data Data unit transfer End of data transfer End of transfer Cancel data transfer File open File close	*	M		•	O	24.1 24.3 24.4 24.5 24.6 17.1 17.2
U3 Write	Write bulk data Data unit transfer End of data transfer End of transfer Cancel data transfer File open File close	•	M		•	O	24.2 24.3 24.4 24.5 24.6 17.1 17.2
U4 File access	Locate FADU Erase FADU (requires U2 or U3)		M			O	20.2 20.3
U5 Limited file management	File creation File deletion Read attributes	O	O	M	M	O	15.3 15.4 16.1
U6 Enhanced file management	Change attributes (requires U5)	O	O	O	O	O	16.2
U7 Grouping	Beginning of grouping End of grouping	M	O	M	M	O	18.1 18.2
U8 FADU locking	FADU Locking (requires U2 or U3) and U4		O			O	

Table 2 — Services and functional units of the Internal File Service

Functional Unit	Services	Service classes					Clause
		T	A	M	TM	U	
U9 Recovery	Regime Recovery Checkpointing Cancel data transfer (for recoverable errors)	O	O		O	O	19.1 25.1 24.6
U10 Restart data transfer	Restarting data transfer Checkpointing Cancel data transfer (for recoverable errors)	O	O		O	O	25.2 25.1 24.6

Key to tables 1 and 2

Service class abbreviations

T = File transfer class
A = File access class
M = File management class
TM = File transfer and management class
U = Unconstrained class

Abbreviations within service class

M = Mandatory
O = Optional
• = At least one of U2 or U3
blank = not permitted

e) optionally, the limited file management functional unit;

f) optionally, but only if the limited file management functional unit is present, the enhanced file management functional unit;

g) optionally, the FADU locking functional unit;

h) optionally, in the internal file service, the recovery functional unit.

i) optionally, in the internal file service, the restart data transfer functional unit.

NOTES

1 The threshold constraints applicable to the file transfer class do not apply to the file access class.

2 The constraints on grouping and sequence of events defined in 8.2.1 do not apply to the file access service class.

8.2.3 File management class

The file management class consists of:

a) the kernel functional unit;

b) the limited file management functional unit;

c) optionally, the enhanced file management functional unit;

d) the grouping functional unit.

In the file management service class the use of the services is constrained so that there is a sequence of zero or more FTAM events on the application association. Each FTAM event is a sequence of:

- an F-BEGIN-GROUP primitive;
- an F-SELECT or F-CREATE primitive;
- optionally, an F-READ-ATTRIB primitive;
- optionally, an F-CHANGE-ATTRIB primitive;
- an F-DESELECT or F-DELETE primitive;
- an F-END-GROUP primitive.

The threshold parameter, defined in clause 18, is set equal to the number of primitives between F-BEGIN-GROUP and F-END-GROUP.

8.2.4 File transfer and management class

The file transfer and management class consists of:

a) the kernel functional unit;

b) the grouping functional unit;

c) one or both of the read or write functional units;

d) the limited file management functional unit;

e) optionally, the enhanced file management functional unit;

f) optionally, in the internal file service, the recovery functional unit;

g) optionally, in the internal file service, the restart data transfer functional unit.

In the file transfer and management service class, the use of the services is constrained so that there is a repeated sequence of FTAM events on the application association. Each FTAM event is either;

a transfer comprising:

1) a single grouped sequence to establish a file open regime. This sequence consists of:

- an F-BEGIN-GROUP primitive;
- an F-SELECT or F-CREATE primitive;
- optionally, an F-READ-ATTRIB primitive;
- optionally, an F-CHANGE-ATTRIB primitive;
- an F-OPEN primitive;
- an F-END-GROUP primitive.

2) a single bulk data transfer procedure, for either a read transfer or a write transfer. The processing mode parameter on the F-OPEN primitive is set to either read or a valid write action, as defined in the constraint set, but not both.

3) a single grouped sequence to release the file open and select regimes. This sequence consists of:

- an F-BEGIN-GROUP primitive;
- an F-CLOSE primitive;
- optionally, an F-READ-ATTRIB primitive;
- optionally, an F-CHANGE-ATTRIB primitive;
- an F-DESELECT or F-DELETE primitive;
- an F-END-GROUP primitive.

or a single grouped sequence to effect file management consisting of:

- an F-BEGIN-GROUP primitive;
- an F-SELECT or F-CREATE primitive;
- optionally, an F-READ-ATTRIB primitive;
- optionally, an F-CHANGE-ATTRIB primitive;
- an F-DESELECT or F-DELETE primitive;
- an F-END-GROUP primitive.

The threshold parameter, defined in clause 18, is set equal to the number of primitives between F-BEGIN and F-END-GROUP.

NOTE - The primitives in the functional units may only be present in these sequences if the corresponding functional units were negotiated during the FTAM regime establishment.

8.2.5 Unconstrained class

The unconstrained class consists of :

a) the kernel functional unit;

b) optionally, any other functional units.

NOTES

1 The threshold parameter constraints do not apply to the unconstrained class.

2 The unconstrained service class is provided for definition of non standard applications. No restrictions are imposed by ISO 8571 on this class.

8.3 Application Entity roles

An application entity keeps the same role (either initiator or responder) for the duration of the FTAM regime. This specific role applies to each of the available functional units.

9 Levels of file service

Two levels of file service are defined in 6.2. They are:

a) the external file service (EFS), in which the user states its FTAM quality of service requirements, but has no awareness of error recovery.

b) the internal file service (IFS) used by the error recovery protocol machine. This service includes primitives giving its users facilities for error recovery and control of the checkpointing mechanisms.

The functional units providing visible services in the external file service and in the internal file service are defined in table 3.

The external file service is supported by the error recovery protocol, which may be null, and indirectly by the basic protocol specified in ISO 8571-4. The internal file service is supported by the basic protocol.

Selection of a named functional unit in the external service implies selection of the same named functional unit in the internal file service. In the internal file service the recovery and restart functional units are optional. The primitives of the restart and recovery functional units will not be visible to the external file service user.

Table 3 — Functional units in the file services

External file service	Internal file service
Kernel	Kernel
Read	Read
Write	Write
File access	File access
Limited file management	Limited file management
Enhanced file management	Enhanced file management
Grouping	Grouping
FADU locking	FADU locking
	Recovery
	Restart data transfer

10 Negotiation of service class, FTAM QoS and functional units

Service class and FTAM quality of service are negotiated independently on the F-INITIALIZE exchange. Each of these negotiations contributes to the final selection of functional units available on the association.

The kernel functional unit is always available and is not included in the negotiations.

The availability of the functional units, read, write, file access, limited file management, enhanced file management, grouping and FADU locking is controlled by the service class negotiated (see Tables 1 and 2).

The proposal of either, or both, of the restart or recovery functional units is as a result of a local decision based on the underlying service and the FTAM quality of service negotiated between the two file service users.

10.1 Service Class

The initiator states in the service class parameter of the F-INITIALIZE request primitive the service classes required. The valid combinations are defined in table 4. Each of the combinations listed may have included with it the unconstrained class.

Table 4 — Service Class Combinations

Abbreviation	Service Class Capability
T	Transfer class
M	Management class
A	Access class
T,A	Transfer class, Access class
T,M,TM	Transfer class, Management class, Transfer and Management class
A,T,M,TM	Access class, Transfer class, Management class or Transfer and Management class

During negotiation the service provider removes any service classes it is unable to support and signals the remaining set to the responder on the F-INITIALIZE indication.

The responder removes from the service class list all classes it is unable to support. It then selects from the remaining classes the highest service class it is able to support. It returns this on the service class parameter of the F-INITIALIZE response primitive. Support of both the transfer class and the management class requires support of the transfer and management class.

The order of service classes is defined as, highest to lowest; access, transfer and management, transfer, management, unconstrained.

The agreed service class is signalled to the initiator on the F-INITIALIZE confirm primitive. This negotiation of service classes always results in a single service class being agreed. Table 5 details the outcome of the negotiation.

The functional unit constraints of each service class are defined in tables 1 and 2. The functional unit negotiation is defined in 10.3.

10.2 FTAM Quality of Service

The initiator states on the F-INITIALIZE request primitive the FTAM quality of service indicating the error classes the application is susceptible to.

The service provider adds to the functional units included in the functional units parameter neither, one or both of the recovery and restart functional units, according to the file service user's requested FTAM quality of service, local knowledge and which of the error recovery and restart functional units it is able to support. The service provider indicates to the responder, on the FTAM quality of service parameter of the F-INITIALIZE indication primitive, the value requested by the initiator and possibly reduced by the service provider.

The responding file protocol machine uses the initiator's FTAM quality of service, local knowledge and capability to determine whether it is able to support the requested FTAM quality of service. If the responder is able to provide the requested FTAM quality of service it returns the parameter unchanged. The recovery and/or restart functional units are removed from the set of functional units parameter if the responding file protocol machine is unable to support them.

If the responder is unable to support the requested FTAM quality of service, it returns on the F-INITIALIZE response primitive the value it is able to support. The service provider may reduce the value of the FTAM quality of service if the resulting functional units do not provide the capability required. The final value of the FTAM quality of service is signalled to the file service user on the F-INITIALIZE confirm primitive. The negotiation of this parameter does not, in itself, prevent the establishment of an FTAM regime. If the quality of service is below that acceptable to the file service user the regime may be terminated by the issue of an F-TERMINATE request primitive by the file service user.

10.3 Functional units

The initiator states on the F-INITIALIZE request primitive its requirement in terms of functional units. The complete list proposed comprises all the mandatory functional units, except the kernel, in the service classes proposed in the service class parameter plus additional optional functional

units. The recovery and restart functional units are only inserted by the service provider.

The service provider removes from this set any functional units it is unable to support and signals the remaining set to the responder on the F-INITIALIZE indication primitive. The responder removes from this set any functional units it is unable to support and signals the remaining set to the service provider on the F-INITIALIZE response primitive.

The service provider signals the same set to the initiator on the F-INITIALIZE confirm primitive. This negotiated set of functional units is then available for use in the established FTAM regime.

Table 5 — Service Class Negotiation

Initiator capability	Responder capability					
	T	M	A	T,A	T,M,TM	A,T,M,TM
T	T	—	—	T	T	T
M	—	M	—	—	M	M
A	—	—	A	A	—	A
T,A	T	—	A	A	T	A
T,M,TM	T	M	—	T	TM	TM
A,T,M,TM	T	M	A	A	TM	A

Key to Table 5

In table 5 the symbols have the meaning defined in table 4. Additionally the — symbol has the following meaning:

a) if both initiator and responder capabilities included the unconstrained service class then that class is the result of the negotiation.

b) if either or both lack the unconstrained class capability then the responder rejects the FTAM regime establishment attempt with a state result parameter indicating failure and, optionally, an appropriate diagnostic value.

Section two : Definition of file service primitives

11 File service primitives

Each of the services constituting the file service is achieved by invoking a sequence of file service primitives.

Table 6 lists for each service;

a) the primitives associated with the service;

b) the parameters associated with the primitives;

c) the file service user that is permitted to issue the request primitive;

In table 6, parameters and primitives which are only visible in the internal service are enclosed in square brackets.

The semantics of the primitives and their parameters are defined in clauses 13 to 20.

The primitives of the bulk data transfer service are listed in clause 21 and defined in clauses 23 to 25.

12 Sequences of primitives

This clause defines the constraints on the permissible sequences for the EFS in which the primitives defined in clauses 14 to 20 can occur. The individual primitives in a service may occur only in the sequences given as part of the primitive definitions. The F-RECOVER, F-RESTART and F-CHECK service primitives are not issued in the external file service.

12.1 Normal sequences

The normal progress of use of the file service is illustrated by the state transition diagram shown in figure 3, applicable separately to each application entity. Full state transition diagrams are included in annex E.

12.2 Constraints on the issue of primitives

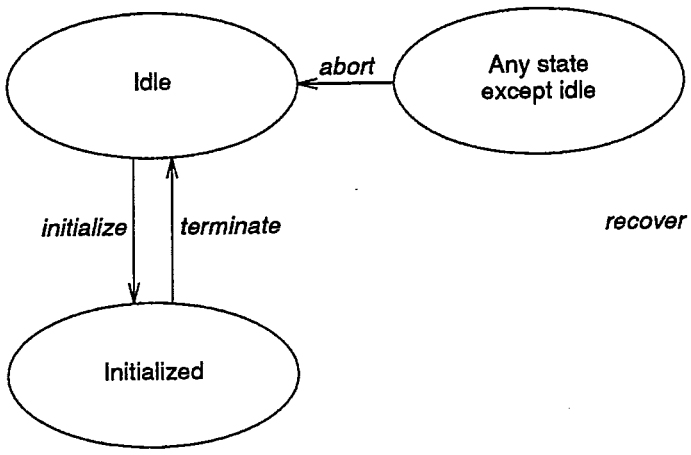
The primitives may be issued in any sequence consistent with the constraints given in tables 7 to 10. The sequences

Table 6 — File service primitives

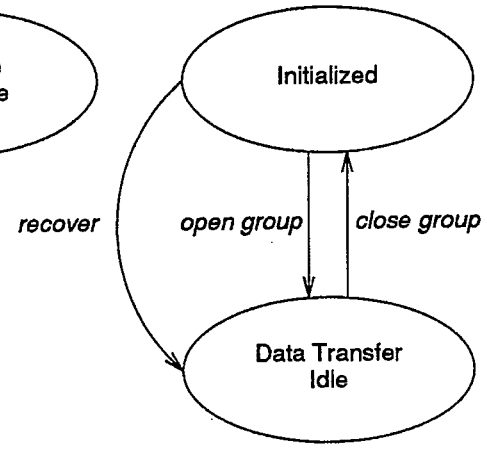
Primitive	Confirmed	Request by	Parameters
F-INITIALIZE	Yes	Initiator	State Result Action Result Called application title Calling application title Responding application title Called presentation address Calling presentation address Responding presentation address Presentation context management Application context name Service class Functional units Attribute groups Shared ASE information FTAM QoS Communication QoS Contents type list Initiator Identity Account Filestore password Diagnostic [Checkpoint window]
F-TERMINATE	Yes	Initiator	Shared ASE information Charging
F-U-ABORT	No	Either	Action Result Diagnostic
F-P-ABORT	No	Provider	Action result Diagnostic
F-SELECT	Yes	Initiator	State Result Action Result Attributes Requested Access Access passwords Concurrency control Shared ASE information Account Diagnostic

Table 6 (continued) — File service primitives

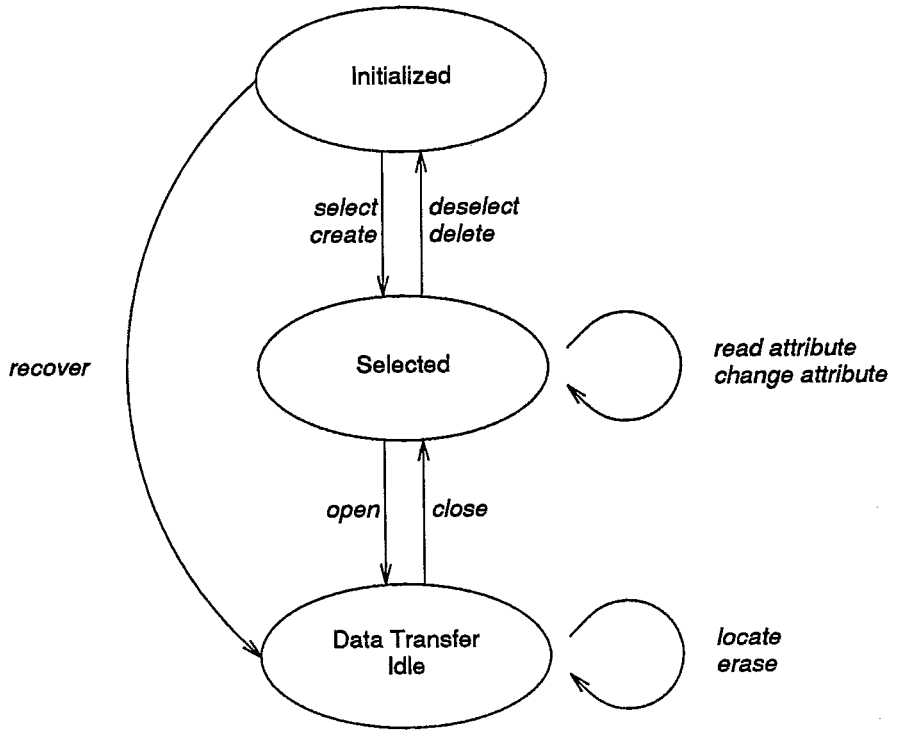
Primitive	Confirmed	Request by	Parameters
F-DESELECT	Yes	Initiator	Action Result Charging Shared ASE information Diagnostic
F-CREATE	Yes	Initiator	State Result Action Result Override Initial Attributes Create password Requested Access Access passwords Concurrency control Shared ASE information Account Diagnostic
F-DELETE	Yes	Initiator	Action Result Shared ASE information Charging Diagnostic
F-READ-ATTRIB	Yes	Initiator	Action Result Attribute names Attributes Diagnostic
F-CHANGE-ATTRIB	Yes	Initiator	Action Result Attributes Diagnostic
F-OPEN	Yes	Initiator	State Result Action result Processing mode Contents type Concurrency control Shared ASE information Enable FADU Locking Diagnostic [Activity identifier] [Recovery mode]
F-CLOSE	Yes	Initiator	Action Result Shared ASE information Diagnostic
F-BEGIN-GROUP	Yes	Initiator	Threshold
F-END-GROUP	Yes	Initiator	-
[F-RECOVER]	Yes	Initiator	[State Result Action result Activity identifier Bulk transfer number Requested Access Access passwords Contents type Recovery point Diagnostic]
F-LOCATE	Yes	Initiator	Action Result FADU identity FADU Lock Diagnostic
F-ERASE	Yes	Initiator	Action Result FADU identity Diagnostic



FTAM regime establishment



Regime establishment — file transfer class



File regime establishment — file access class

Figure 3 — Simplified State Diagram for successful activity (see Annex E)

of primitives are defined under the individual services. Individual primitive sequences may be interleaved to form the following grouped sequences, which are expressed using the notation defined in annex E.

a) F-BEGIN-GROUP
(F-SELECT|F-CREATE)
[F-READ-ATTRIB] [F-CHANGE-ATTRIB]
F-OPEN
F-END-GROUP

b) F-BEGIN-GROUP
F-CLOSE
[F-READ-ATTRIB] [F-CHANGE-ATTRIB]
(F-DESELECT|F-DELETE)
F-END-GROUP

c) F-BEGIN-GROUP
(F-SELECT|F-CREATE)
[F-READ-ATTRIB] [F-CHANGE-ATTRIB]
(F-DESELECT|F-DELETE)
F-END-GROUP

d) F-BEGIN-GROUP
(F-SELECT|F-CREATE)
[F-READ-ATTRIB] [F-CHANGE-ATTRIB]
F-END-GROUP

e) F-BEGIN-GROUP
[F-READ-ATTRIB] [F-CHANGE-ATTRIB]
(F-DESELECT|F-DELETE)
F-END-GROUP

NOTES

1 Not all the above sequences are allowed in the file transfer, file management and file transfer and management classes (see 8.2.1, 8.2.3, 8.2.4). In the file transfer class, only sequences (a) and (b),are valid, and the threshold parameter is set so that the sequences either succeed or fail as a whole, i.e. set to the number of primitives between the begin group and end group primitives. In the file management class only sequence (c) is valid. (see 8.2.3). In the file transfer and management class only the sequences (a), (b) and (c) are valid.

2 Other constraints will affect the ability of the file service user or file service provider to invoke the various procedures, such as flow control constraints on sending data or constraints on the ability of a file service user to accept spontaneous F-P-ABORT indications from the file service provider.

12.3 Conventions

The following conventions apply to entries in tables 7 to 10.

12.3.1 Conventions for tables 7 and 8

In tables 7 and 8, the entry "yes" implies that the succession given can occur. The column "start of file regimes" indicates entry to tables 9 and 10. The row "end of file regimes" indicates exit from tables 9 and 10, and the row "within file regimes" indicates any other entry in tables 9 and 10.

12.3.2 Conventions for tables 9 and 10

In tables 9 and 10, the entries indicate the functional units required for the succession to occur. The entries are:

Kernel	kernel functional unit
Grouping	grouping functional unit
Lmgt	limited file management functional unit
Emgt	enhanced file management functional unit
Recover	recover functional unit
Access	file access functional unit
G-Lmgt	grouping and limited file management functional units
G-Emgt	grouping and enhanced file management functional units
Lmgt-Rec	limited file management and recovery functional units
Rec-Acc	recovery and file access functional units
G-Rec	grouping and recovery functional units
G-Acc	grouping and file access functional units

The row "start file regimes" indicates entry from tables 7 and 8, and the column "end file regimes" indicates return to tables 7 and 8. The row and column "bulk data transfer" indicate use of the bulk data transfer service defined in section three.

In tables 7 to 10, the value of the state result parameter is indicated by:

(+ve) positive response or confirm; the primitive carries a state result indicating a successful transition;

(-ve) negative response or confirm; the primitive carries a state result indicating an unsuccessful transition or regime establishment attempt.

Table 7 — Sequence of service primitives for FTAM regime establishment — initiator

Previous file service event	May next issue			
	F-INITIALIZE request	F-TERMINATE request	F-U-ABORT request	Start File Selection Regimes
Idle	Yes			
F-INITIALIZE request			Yes	
F-INITIALIZE confirm (+ve)			Yes	Yes
F-INITIALIZE confirm (-ve)	Yes			
F-TERMINATE request			Yes	
F-TERMINATE confirm	Yes			
F-U-ABORT request	Yes			
F-U-ABORT indication	Yes			
F-P-ABORT indication	Yes			
Within file selection regimes			Yes	
End file selection regimes		Yes	Yes	Yes

Table 8 — Sequence of service primitives for FTAM regime establishment — responder

Previous file service event	May next issue			
	F-INITIALIZE response	F-TERMINATE response	F-U-ABORT request	Start File Selection Regimes
Idle				
F-INITIALIZE indication	Yes		Yes	
F-INITIALIZE response (+ve)			Yes	Yes
F-INITIALIZE response (-ve)				
F-TERMINATE indication		Yes	Yes	
F-TERMINATE response				
F-U-ABORT request				
F-U-ABORT indication				
F-P-ABORT indication				
Within file selection regimes			Yes	
End file selection regimes			Yes	Yes

Table 9 — Sequence of service primitives for file service regimes — initiator

Previous file service event	May next issue					
	F-SELECT request	F-DESELECT request	F-CREATE request	F-DELETE request	F-READ -ATTRIB request	F-CHANGE -ATTRIB request
Start file selection regimes	Kernel		Lmgt			
F-SELECT request		Grouping		G-Lmgt	G-Lmgt	G-Emgt
F-SELECT confirm (+ve)		Kernel		Lmgt	Lmgt	Emgt
F-SELECT confirm (-ve)	Kernel		Lmgt			
F-DESELECT request						
F-DESELECT confirm	Kernel		Lmgt			
F-CREATE request		Grouping		G-Lmgt	G-Lmgt	G-Emgt
F-CREATE confirm (+ve)		Lmgt		Lmgt	Lmgt	Emgt
F-CREATE confirm (-ve)	Lmgt		Lmgt			
F-DELETE request						
F-DELETE confirm	Lmgt		Lmgt			
F-READ-ATTRIB request		G-Lmgt		G-Lmgt		G-Emgt
F-READ-ATTRIB confirm		Lmgt		Lmgt	Lmgt	Emgt
F-CHANGE-ATTRIB request		G-Emgt		G-Emgt		
F-CHANGE-ATTRIB confirm		Emgt		Emgt	Emgt	Emgt
F-OPEN request						
F-OPEN confirm (+ve)						
F-OPEN confirm (-ve)		Kernel		Lmgt	Lmgt	Emgt
F-CLOSE request		Grouping		G-Lmgt	G-Lmgt	G-Emgt
F-CLOSE confirm		Kernel		Lmgt	Lmgt	Emgt
F-BEGIN-GROUP request	Grouping	Grouping	G-Lmgt	G-Lmgt	G-Lmgt	G-Emgt
F-BEGIN-GROUP confirm						
F-END-GROUP request						
F-END-GROUP confirm	Grouping	Grouping	G-Lmgt	G-Lmgt	G-Lmgt	G-Emgt
F-RECOVER request						
F-RECOVER confirm (+ve)						
F-RECOVER confirm (-ve)	Recover		Lmgt			
Bulk data transfer						
F-LOCATE request						
F-LOCATE confirm						
F-ERASE request						
F-ERASE confirm						

May next issue								
F-OPEN request	F-CLOSE request	F-BEGIN -GROUP request	F-END -GROUP request	F-RECOVER request	Bulk Data Transfer	F-LOCATE request	F-ERASE request	End file selection regimes
		Grouping		Recover				Kernel
Grouping								
Kernel		Grouping						
		Grouping		Recover				Kernel
			Grouping					
		Grouping		Recover				Kernel
G-Lmgt								
Lmgt		G-Lmgt						
		G-Lmgt		Lmgt-Rec				Lmgt
			G-Lmgt					
		G-Lmgt		Lmgt-Rec				Lmgt
G-Lmgt			G-Lmgt					
Lmgt		G-Lmgt						
G-Emgt			G-Emgt					
Emgt		G-Emgt						
			Grouping					
	Kernel	Grouping			Kernel	Access	Access	
Kernel		Grouping						
Kernel		Grouping						
	Grouping							
Grouping	Grouping	Grouping		G-Rec	Grouping	Access	Access	Grouping
	Recover	G-Rec			Recover	Rec-Acc	Rec-Acc	
		G-Rec		Recover				Recover
	Kernel	Grouping			Kernel	Access	Access	
	Access	G-Acc			Access	Access	Access	
	Access	G-Acc			Access	Access	Access	

Table 10 — Sequence of service primitives for file service regimes — responder

Previous file service event	May next issue					
	F-SELECT response	F-DESELECT response	F-CREATE response	F-DELETE response	F-READ -ATTRIB response	F-CHANGE -ATTRIB response
Start file selection regimes						
F-SELECT indication	Kernel					
F-SELECT response (+ve)		Grouping		G-Lmgt	G-Lmgt	G-Emgt
F-SELECT response (-ve)						
F-DESELECT indication		Kernel				
F-DESELECT response						
F-CREATE indication			Lmgt			
F-CREATE response (+ve)		G-Lmgt		G-Lmgt	G-Lmgt	G-Emgt
F-CREATE response (-ve)						
F-DELETE indication				Lmgt		
F-DELETE response						
F-READ-ATTRIB indication					Lmgt	
F-READ-ATTRIB response		G-Lmgt		G-Lmgt		G-Emgt
F-CHANGE-ATTRIB indication						Emgt
F-CHANGE-ATTRIB response		G-Emgt		G-Emgt		
F-OPEN indication						
F-OPEN response (+ve)						
F-OPEN response (-ve)						
F-CLOSE indication						
F-CLOSE response		Grouping		G-Lmgt	G-Lmgt	G-Emgt
F-BEGIN-GROUP indication						
F-BEGIN-GROUP response	Grouping	Grouping	G-Lmgt	G-Lmgt	G-Lmgt	G-Emgt
F-END-GROUP indication						
F-END-GROUP response						
F-RECOVER indication						
F-RECOVER response (+ve)						
F-RECOVER response (-ve)						
Bulk data transfer						
F-LOCATE indication						
F-LOCATE response						
F-ERASE indication						
F-ERASE response						

May next issue								
F-OPEN response	F-CLOSE response	F-BEGIN -GROUP response	F-END -GROUP response	F-RECOVER response	Bulk Data Transfer	F-LOCATE response	F-ERASE response	End file selection regimes
								Kernel
Grouping								
			Grouping					Kernel
			Grouping					Kernel
G-Lmgt								
			G-Lmgt					Lmgt
			G-Lmgt					Lmgt
G-Lmgt			G-Lmgt					
G-Emgt			G-Emgt					
Kernel								
			Grouping					
			Grouping					
	Kernel							
	Grouping							
		Grouping						
								Grouping
				Recover				
								Recover
						Access		
							Access	

12.4 Confirmed Services

For all confirmed services the sequence of events in a successful exchange is shown in figure 4, where F-XXX indicates the name of the service element.

A request to establish a new regime (F-INITIALIZE, F-SELECT, F-CREATE or F-OPEN) may be rejected by use of a response with state result parameter indicating failure (see 13.1).

13 Common file service parameters

File service parameters which apply to several primitives are defined in this clause and referenced by the primitive definitions. Parameters which apply to only one primitive are defined in association with the primitive concerned. Annex B relates the primitives to any attributes they modify.

13.1 State result

The state result parameter conveys information relating to the service state machine. This parameter is returned on the response and confirm primitives of services which can have failed to change regimes as required by the request and indication primitives. The state result parameter is not present on primitives which do not cause a state change, neither is it present on primitives which force state changes but cannot fail. (for example F-DESELECT). The values of the state result are "success" or "failure". This parameter is not used to set any activity attribute.

13.2 Action result

The action result parameter conveys information which summarises that available in the diagnostic parameter (See 13.13). The value is never less severe than the most severe diagnostic value. In the IFS the valid values of the action result parameter are "success", "transient error" or "permanent error". In the external service the only valid values are success or permanent error. The term "unsuccessful" is used in ISO 8571-3 to indicate either transient or permanent error. When a response or confirm primitive includes a state result indicating failure the action result is set to "transient error" or "permanent error". A "successful" action result may be accompanied by a diagnostic of an informative error type. This parameter is not used to set any activity attribute.

13.3 Account

The account parameter identifies the account to which costs incurred in the regime which is being established are to be charged. The range of values the parameter may take is equal to that defined for the current account activity attribute. This parameter is used to set the current account activity attribute, for the duration of the regime being entered. If the parameter is not present, no value is assigned to the activity attribute, which is then either unset or retains some previous value based on local considerations. The current account activity attribute reverts to its previous value at the end of a regime.

13.4 Charging

The charging parameter conveys information on the costs attributed to the account during the regime which is being released. The value of the parameter is a list of triples; the elements of each triple are: a GraphicString resource identifier, a GraphicString charging unit and an integer charge value. The charging parameter is present at the end of a regime, only if the account parameter was present at the beginning of that regime. It is not mandatory to return a charge if that charge is zero. The resource identifier and charging unit values are implementation dependent. The return of a charge may be associated with the restoration of the current account activity attribute to its previous value.

13.5 Attributes

The attributes parameter conveys a list of file attribute names and file attribute values associated with the file. F-INITIALIZE negotiates the attribute groups which are available for the duration of the FTAM regime. Subsequent primitives only use those attributes which were negotiated.

On primitives other than F-SELECT issued by the initiator, the parameter conveys new values to be assigned to the file attributes. On primitives issued by the responder, the parameter reports the current values of the file attributes, or indicates that there is no value available.

The filename attribute value on F-SELECT identifies the file to be selected, or on F-CREATE the name of the file to be created and selected. In the request and indication primitives it indicates the file required, and in the response and confirm primitives it indicates the file actually selected.

NOTE - If, for example, the filename requested gave a generic name or a generation name, the name selected might differ from that requested.

The file attributes, the range of values they may take and the actions to be taken are defined in the virtual filestore definition (ISO 8571-2). Annex B specifies which file attributes may be manipulated by each primitive. This parameter is not used to set any activity attribute.

13.6 Requested access

The requested access parameter indicates the basis on which the file is being selected or recovered. The value gives, as a vector, the actions to be performed during the selection. The elements of the vector correspond to the read, insert, replace, extend, erase, read attribute, change attribute and delete file actions, and each element indicates whether the action is required or not.

The value of the parameter on request and indication primitives indicates the initiator's requirements.

NOTE - When a file is selected by use of F-SELECT, F-CREATE or F-RECOVER the requested access parameter is used to state the maximum facilities the current user will require during the whole select regime. These access requirements may be password protected by the access control file attribute which is matched to the access passwords parameter (see 13.7).

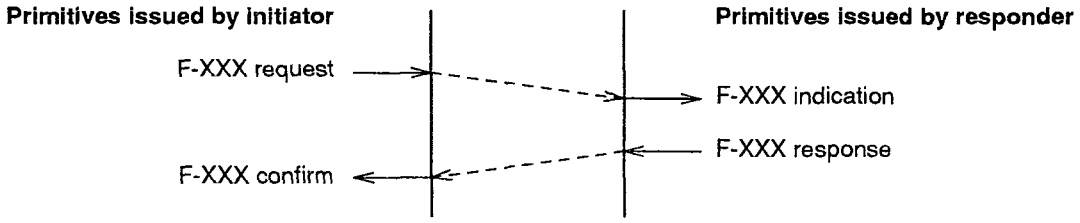


Figure 4 — Confirmed service

The requested access is further qualified by the concurrency control parameter which specifies how multiple users are to be restricted when requiring access to the same file. The requested access parameter includes only those actions to be performed by the user, any action not included is not available later in the select regime.

The relationship between attributes and actions is detailed in ISO 8571-2. This parameter is used to set the current access request activity attribute.

13.7 Access Passwords

The access passwords parameter provides passwords associated with the actions specified in the requested access parameter. This parameter is only available if the security attribute group has been negotiated. The range of values the parameter takes is equal to that defined for the current access passwords activity attribute. This parameter is used to set the current access passwords activity attribute.

13.8 Concurrency Control

The concurrency control parameter indicates the relation of the file select, or file open, regime to other activities on the same file. The value is a vector whose elements indicate, for each of the actions specified in the requested access parameter elements (see 13.6), which access locks are required. The locks define the access available to the user and the access available to any other user. The concurrency control parameter qualifies the requested access for the regime being entered. The shared and exclusive locks are only permitted for the actions negotiated as available by the requested access parameter. For those actions not available the only locks permitted are no access and not required.

The locks available are as follows:

a) not required - I will not perform the operation - others may;

b) shared - I may perform the operation - so may others;

c) exclusive - I may perform the operation - others may not;

d) no access - no one may perform the operation.

They are available on the following actions: read, insert, replace, extend, erase, read attribute, change attribute and delete file.

If FADU locking has not been invoked by the Enable FADU Lock parameter, the locks have a scope of the open regime (see 17.1.2.7).

If FADU locking is negotiated, the locks have the scope of the data transfer regime and values of "not required" are assumed elsewhere in the open regime, except where specifically overridden by explicit FADU locks.

This parameter is used to set the current concurrency control activity attribute.

13.9 FADU Lock

This parameter may be used to set individual FADU locks on or off, and if it is absent the state of the locks is unchanged. Switching the locks changes the value agreed at select and/or open from "not required" to "no access" and from "shared" to "exclusive" until the lock is explicitly switched off, or the FADU is erased, or the file is closed. Switching off the lock returns the FADU to its previous state for both reading and writing.

If for a BDT operation the lock is being set on, then the action of FADU locking is performed before the transfer. If the FADU lock is being set off then the transfer is preformed before the FADU lock is released.

The FADU lock parameter is available if;

a) the FADU locking functional unit is negotiated; and

b) the Enable FADU locking parameter on the F-OPEN request was set on

This parameter is not used to set any activity attribute.

13.10 Shared ASE information

The shared ASE Information parameter allows information of other ASEs to be associated with FTAM primitives. The information to be conveyed and the symbiotic relationship established between FTAM and other ASEs is defined by the application context. One example of such a combination is given in Annex C, which defines the combination of FTAM and CCR for the purpose of file transfer. Other such combinations may be established by other standards or by the registration of application contexts. This parameter is not used to set any activity attribute.

13.11 Activity Identifier

The activity identifier parameter is only visible in the internal file service, and then only if the recovery functional unit has been negotiated on F-INITIALIZE (see 14.1.2.12). It gives an unambiguous identifier for the file activity to be performed within the open regime. A different activity identifier value is allocated to each activity involving a particular pair of initiating and responding entities. This identifier is used in re-establishing the data transfer regime after errors.

An activity identifier may be reused following receipt of an F-CLOSE confirm of an open regime having the same activity identifier. The responder loses all knowledge of an activity identifier on issue of the F-CLOSE response. An F-U-ABORT or F-P-ABORT between these events followed by an F-RECOVER will result in a characteristic error being generated. The value of the activity identifier is an integer. This parameter is not used to set any activity attribute.

13.12 File Access Data Unit Identity

The file access data unit identity specifies the target FADU to which a series of one or more specific file service operations is related. The parameter may take one of the values specified in ISO 8571-2. In addition in access context FL, the file access data unit identity is qualified by an integer level number of the requested DUs, relative to the root of the addressed FADU. This parameter is used to set the current location activity attribute.

NOTE - Further constraints may be imposed by the constraint set, the permitted actions file attribute (see ISO 8571-2) and the access contexts to be used if reading the file.

The FADU addressed by the file access data unit identity depends on the operation to be performed.

a) LOCATE: The FADU identity address the FADU to be located.

b) READ: The FADU identity addresses the FADU(s) to be read. If read in an access context returning structuring information (HA, HN, FA, FL and FS), the node descriptor returned contains the node name, which need not necessarily be identical to the FADU identity.

c) INSERT, REPLACE and EXTEND: The location of each node to be inserted, replaced or extended is

determined in a way specified for that action in the constraint set in use (see ISO 8571-2), based on either:

1) the FADU identity, if no structuring information is transmitted (i.e. only the data unit contents are transferred);

2) the FADU identity and the node name in the first node descriptor of each FADU if structuring information is transmitted.

If performed in an access context requiring structuring information (HA, FA and FS) the node name in the first node descriptor of an FADU transmitted in the transferred data is identical to the node name stored in the virtual filestore, otherwise the operation will fail.

d) ERASE: The FADU identity addresses the FADU to be erased.

13.13 Diagnostic

The diagnostic parameter conveys detailed information on the failure of a requested action. The diagnostic parameter amplifies the information given in the action result parameter (see 13.2). Three diagnostic types are defined to distinguish between the provision of information to qualify a successful action, supplemental detail to a transient error (in the IFS only) and supplemental detail to a permanent error. The possible values for the diagnostic parameter are defined in annex A. This parameter is not used to set any activity attribute.

14 FTAM regime control

The services described in this clause control the FTAM regime characterizing the file service association. There are groups of primitives concerned with regime establishment, orderly regime termination and abrupt regime termination.

In the following tables the following definitions apply:

Mandatory — The parameter is present on all occurrences of the primitives.

Optional — If the option is present on the request it is also present on the indication. If the parameter is present on the response it is also present on the confirm.

Dependent — If the response and confirm are dependent they are both present if, and only if, the parameter is present on the request and indication.

Conditional — Indicates that the availability of the parameter is conditional on a successful fulfillment or negotiation of another parameter or functional unit in a previous primitive. If the condition is fulfilled then the presence of the parameter is mandatory.

14.1 FTAM regime establishment service

14.1.1 Function

The establishment of an FTAM regime is the first phase in any instance of file activity. The F-INITIALIZE request primitive is issued by the file transfer initiator.

The F-INITIALIZE primitives are used only to create an FTAM regime and may not be issued within an existing regime.

14.1.2 Types of primitives and parameters

Table 11 indicates the types of primitives and parameters needed for FTAM regime establishment.

14.1.2.1 State result

The state result, defined in 13.1, indicates whether the FTAM regime has been established.

14.1.2.2 Action result

The Action result parameter is defined in 13.2.

14.1.2.3 Called Application Title

The called application title is the title used to identify the filestore . The value is an application entity title. This parameter is not used to set any activity attribute.

14.1.2.4 Calling Application Title

The calling application title is the title of the initiating FTAM entity. The value is an application entity title. This parameter is used to set the current calling application entity title activity attribute.

14.1.2.5 Responding Application Title

The responding application title is that title returned by the responder and that which will be used for re-establishing the association after failure. The value is an application entity title. This parameter is used to set the current responding application entity title activity attribute at the initiator.

NOTE - Application Entity Titles are used by the file service users to provide one another with naming information which is stable on a longer timescale than any particular FTAM regime. They are used, for example, in error recovery and access control.

14.1.2.6 Called Presentation Address

The called presentation address is the address used by the calling service user to identify the PSAP to which the association is to be established. The value is a PSAP-address. This parameter is not used to set any activity attribute

14.1.2.7 Calling Presentation Address

The calling presentation address is the address from which the association is established. The value is a PSAP-address. This parameter is not used to set any activity attribute.

14.1.2.8 Responding Presentation Address

The responding presentation address is the address which is used in re-establishing the association after failure. The value is a PSAP-address. This parameter is not used to set any activity attribute.

NOTES

1 The responding address is not necessarily textually identical to the called address. It may differ, for example, if generic addressing or redirection are in use.

2 Presentation Addresses are used by the service users to specify the addressing requirements of the particular FTAM regime being initialized. The relation between a presentation Address and an Application Entity Title is determined by the specification of the Application Entity as a whole.

14.1.2.9 Presentation Context Management

The requested presentation context management parameter indicates whether or not the Context management functional unit in the Presentation service is to be used during the FTAM open and recovery procedures. The value is a boolean. The responder may refuse an initiator proposal to use presentation context management, even if the facility is available on the underlying presentation connection, and still establish an FTAM regime. The responder may not indicate use of presentation context management if the initiator has not done so. This parameter is not used to set any activity attribute.

14.1.2.10 Application Context Name

The application context name parameter carries a name used to represent the properties of the association as a whole. The initiator proposes a name which may be accepted, and returned by the responder, or the responder may return a different name. In either case the name returned by the responder is the application context name applicable to the established association.

NOTE - This parameter is in general specific to an application. However ISO 8571-4 defines a name for use when the primary intent is to transfer files as an activity in its own right.

14.1.2.11 Service Class

The service class parameter on the request and indication takes one of the values defined in 10.1. This represents the capability of the initiator. (see 8.2 and 10.1). The service class set proposed by the initiator is reduced, by the responder, to a single class (see 10.1) which is returned on the response and confirm primitives. If no service class is acceptable to the responder the FTAM regime

Table 11 — F-INITIALIZE parameters

Parameter	F-INITIALIZE request	F-INITIALIZE indication	F-INITIALIZE response	F-INITIALIZE confirm
State result			Mandatory	Mandatory
Action result			Mandatory	Mandatory
Called Application Title	Mandatory	Mandatory (=)		
Calling Application Title	Mandatory	Mandatory (=)		
Responding Application Title			Mandatory	Mandatory (=)
Called Presentation Address	Mandatory	Mandatory (=)		
Calling Presentation Address	Mandatory	Mandatory (=)		
Responding Presentation Address			Mandatory	Mandatory (=)
Presentation Context Management	Mandatory	Mandatory	Mandatory	Mandatory (=)
Application Context Name	Optional	Optional (=)	Optional	Optional (=)
Service Class	Mandatory	Mandatory	Mandatory	Mandatory (=)
Functional Units	Mandatory	Mandatory (=)	Mandatory	Mandatory (=)
Attribute Groups	Mandatory	Mandatory	Mandatory	Mandatory (=)
Shared ASE Information	Optional	Optional (=)	Optional	Optional (=)
FTAM Quality of Service	Mandatory	Mandatory	Mandatory	Mandatory
Communications Quality of Service	Mandatory	Mandatory	Mandatory	Mandatory (=)
Contents Type List	Conditional	Conditional	Dependent	Dependent (=)
Initiator Identity	Optional	Optional (=)		
Account	Optional	Optional (=)		
Filestore Password	Optional	Optional (=)		
Diagnostic			Optional	Optional
Additional Parameters in the Internal File Service				
Checkpoint Window	Conditional	Conditional (=)	Conditional	Conditional (=)

(=) signifies that the value is not modified by the file service provider.

establishment fails. This parameter is not used to set any activity attributes.

14.1.2.12 Functional Units

The requested functional units parameter negotiates the set of file service functional units, excluding the kernel, to be available from the negotiated service class on the application association (see 8.1 and 10.3). On the request and indication primitives the parameter carries the full functional unit capability of the initiator. On the response and confirm primitives it carries all the functional units to be available on the association. When the required service class permits, the set may contain the optional functional units "read", "write", "file access", "limited file management", "enhanced file management", "grouping" and "FADU locking". For use by the internal file service, if the service class permits, it may also contain the "recovery" and "restart" functional units. The recovery and restart functional units are only visible in the IFS and are thus never explicitly requested in the external file service, they are indirectly requested via the FTAM quality of service parameter (see 10.2). This parameter is not used to set any activity attributes.

14.1.2.13 Attribute Groups

The requested attribute group parameter negotiates the set of optional file attribute groups to be available on the application association. Specification of the attribute groups is defined in ISO 8571-2. The set may be empty or it may contain some combination of the attribute groups "storage", "security" or "private" (see ISO 8571-2). The responder may reduce the attribute groups proposed by the initiator,

within the limits of ISO 8571-2. The kernel attribute group is not proposed or negotiated, hence negotiation of the attribute groups will not prevent the establishment of an FTAM regime. This parameter is not used to set any activity attributes.

14.1.2.14 Shared ASE information

The shared ASE information parameter is defined in 13.10.

14.1.2.15 FTAM Quality of Service

The FTAM quality of service parameter is used to convey information relating to the error susceptibility of the external file service user. The detailed negotiation rules are specified in clause 10. The values of this parameter indicate the application is susceptible to errors of one of the following types:

a) not susceptible to errors. No error recovery to be provided.

b) errors which damage the data transfer regime.

c) errors which damage the open or data transfer regimes.

d) errors which damage the select, open or data transfer regimes, or errors which lose the association.

This parameter is not used to set any activity attributes.

14.1.2.16 Communication Quality of Service

The requested communication quality of service parameter conveys the quality of service to be negotiated on the association. On the request primitive it indicates the quality of service requested by the initiator. On the indication primitive it indicates the quality of service requested, reduced as necessary to signal that achievable, by the service provider. On the response and confirm primitives it indicates the quality of service achieved. The values taken by the Communication Quality of Service parameter are as defined in ISO 8649. This parameter is not used to set any activity attributes.

NOTES

1 This parameter is referenced, via the ASE association control service definition (ISO 8649), to the Presentation Service Definition (ISO 8822), then to the Session Service Definition (ISO 8326).

2 As an International Standard ISO 8571 has no intrinsic basis on which to select or manipulate values of any aspect of communications quality of service (CQoS).

14.1.2.17 Contents Type List

The requested contents type list parameter carries a list of document types and/or abstract syntaxes. This parameter is mandatory (in the transfer class, the transfer and management class and the access class), if the presentation context management functional unit is not being negotiated. This parameter allows the establishment of the necessary presentation contexts at the time of establishing the FTAM regime. The initiator proposes a list of elements which are each either document type names or abstract syntax names from which a unique list of the required abstract syntaxes is constructed. The latter is used to construct the presentation context definition list parameter in the ACSE A-ASSOCIATE service primitive. The service provider reduces the value of the contents type list to remove any contents types which require abstract syntaxes rejected by the presentation service provider. The responder further reduces the list to remove any contents types it will not support, and any consequential reduction in the required abstract syntax list is made. The Contents Type List response returns the list of those agreed document type names and abstract syntax names for which there is a supported abstract syntax and so is used to build the presentation context definition result list by the service provider. This parameter is not used to set any activity attribute.

14.1.2.18 Initiator Identity

The initiator identity parameter identifies the calling user. The value of the optional initiator identity parameter is a GraphicString. The range of values the parameter may take is equal to that defined for the current initiator identity activity attribute. If the parameter is not present, the current initiator identity activity attribute remains unset. If this parameter value, or its omission, is unacceptable to the responder, the responder generates a state result indicating failure to establish the FTAM regime and/or an action result indicating permanent error with an optional diagnostic detailing the reason. This parameter is used to set the current initiator identity activity attribute.

14.1.2.19 Account

The account parameter is defined in 13.3. The account given is charged for all costs incurred by the FTAM regime, on the application association; this excludes nested costs associated with file selection regimes where an explicit overriding account parameter is given when the regime is established. The value of this optional parameter is a GraphicString. If this parameter value, or its omission, is unacceptable to the responder, the responder generates a state result indicating failure to establish the FTAM regime and/or an action result indicating permanent error with an optional diagnostic detailing the reason. This parameter is used to set the current account activity attribute.

14.1.2.20 Filestore Password

The filestore password parameter conveys a password which is used to authenticate the initiator to the responder. This parameter is a GraphicString or an OCTET STRING used by the responder to authenticate the initiator identity parameter. If this parameter value, or its omission, is unacceptable to the responder, the responder generates a state result indicating failure to establish the FTAM regime and/or an action result indicating permanent error with an optional diagnostic detailing the reason. This parameter is not used to set any activity attributes.

14.1.2.21 Diagnostic

The diagnostic parameter is defined in 13.13.

14.1.2.22 Checkpoint Window

The requested checkpoint window parameter indicates, for each direction of transmission, the maximum number of checkpoints which may remain unacknowledged. This parameter is conditional upon the recovery or restart functional units having been selected in the IFS, in which case it is mandatory. Checkpoints are only inserted by the sender. Values of this parameter never cause diagnostics on F-INITIALIZE, but may be the reason for subsequent termination. The continued progress of the service is only guaranteed if the entity acting as receiver gives acknowledgements within this limit. The window size is stated independently by each entity as the maximum value for when that entity is the sending entity. There is no negotiation.

Table 12 — F-TERMINATE parameters

Parameter	F-TERMINATE request	F-TERMINATE indication	F-TERMINATE response	F-TERMINATE confirm
Shared ASE Information	Optional	Optional (=)	Optional	Optional (=)
Charging			Conditional	Conditional

(=) signifies that the value is not modified by the file service provider

The stated value from each entity, as the sending entity, is maintained by the corresponding entity for use when it is the receiving entity. The values for each direction of data transfer are not necessarily the same. The parameter is an integer. The parameter is not used to set any activity attributes.

14.2 FTAM regime termination service (orderly)

14.2.1 Function

An FTAM regime may be terminated by an exchange of F-TERMINATE primitives. This primitive will only be issued when there are no actions in progress. The F-TERMINATE request primitive can be issued by the file transfer initiator (the issuer of the F-INITIALIZE request) at any time after the receipt of an F-INITIALIZE confirm primitive, providing no file is selected. The issue of an F-TERMINATE request does not imply the success of any previous activity. Indications of success or failure are given on the completion of each activity.

14.2.2 Types of primitives and parameters

Table 12 indicates the types of primitive and the parameters needed for orderly FTAM regime termination.

14.2.2.1 Shared ASE Information

The shared ASE information parameter is defined in 13.10.

14.2.2.2 Charging

The charging parameter is defined in 13.4. The parameter reflects costs incurred during the FTAM regime, on the basis of the account parameter quoted when the FTAM regime was established. This process excludes charges assigned to accounts within nested regimes. The presence of this parameter is conditional on the account parameter having been provided by the initiator on the F-INITIALIZE primitive.

14.3 FTAM regime termination service (abrupt)

14.3.1 Function

Either external file service user may issue an F-U-ABORT request primitive at any time after an F-INITIALIZE request primitive has been issued, or an F-INITIALIZE indication primitive has been received. The file service provider may issue an F-P-ABORT primitive at any time after an F-INITIALIZE request primitive has been received, or an F-INITIALIZE indication primitive issued. The F-U-ABORT or F-P-ABORT primitives terminate the FTAM regime unconditionally, abandoning any file activity that was in progress and leaving the selected file in an undefined state. If error recovery is to be performed, the responsibility for initiating the recovery lies with the initiator. Once an F-U-ABORT request or F-P-ABORT request primitive has been issued, the FTAM regime is terminated; the request cannot be rejected.

NOTE - In the external file service data transfer phase the use of F-CANCEL is preferred over that of F-U-ABORT. In the external file service if the F-U-ABORT fails to reach the peer entity because of an underlying service failure the issuer of the F-U-ABORT will release locks while the peer entity will receive an F-P-ABORT from the lower layer failure and will retain locks pending a recovery action. No mechanisms are defined in ISO 8571 to resolve this difference of view of the file status.

The filestore provider performs the local close file and deselect file actions on receipt of an F-U-ABORT indication or an F-P-ABORT indication if the file is open, and the local deselect file action if the file is closed but selected.

NOTE - Abrupt termination may lose information about charges which are levied. The initiator will not know which unconfirmed services have affected the file. An internal service abort for a transient error will not be visible in the external service, and concurrency controls will stay in place while recovery is in progress. An abort in the external service will leave the initiator in doubt about the state of the file, with concurrency controls released.

14.3.2 Types of primitives and parameters

Tables 13 and 14 indicate the types of primitives and parameters needed for abrupt FTAM regime termination.

Table 13 — F-U-ABORT parameters

Parameter	F-U-ABORT request	F-U-ABORT indication
Action Result	Mandatory	Mandatory
Diagnostic	Optional	Optional

Table 14 — F-P-ABORT parameters

Parameter	F-P-ABORT indication
Action Result	Mandatory
Diagnostic	Optional

14.3.2.1 Action Result

The action result parameter is defined in 13.2.

14.3.2.2 Diagnostic

The diagnostic parameter is defined in 13.13.

14.3.3 Sequence of primitives

The sequence of events in a user initiated abrupt termination is defined in figure 5.

The sequence in a service provider initiated abrupt termination is defined in the time sequence diagram shown in figure 6.

Collision of an F-P-ABORT and an F-U-ABORT can result in the loss of the F-U-ABORT indication shown in figure 7.

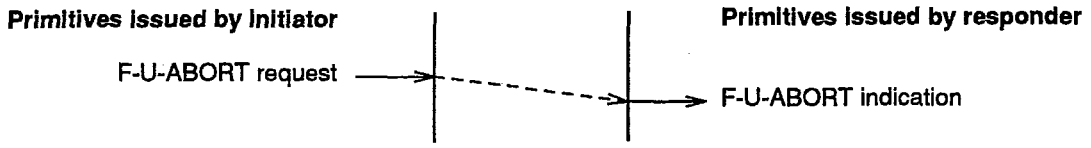


Figure 5 — F-U-ABORT service



Figure 6 — F-P-ABORT service

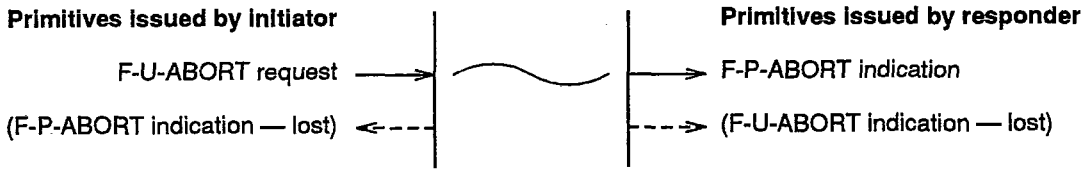


Figure 7 — F-P-ABORT collision

15 File selection regime control

These services control the regime which binds an identified file to an FTAM regime. Different services are used depending on whether the file exists before the selection or not, and on whether it is to exist after the selection or not.

15.1 File selection service

15.1.1 Function

This service selects a file which already exists, by specifying a filename attribute to identify the file unambiguously. These primitives may only be issued if there is no current file selection within the current FTAM regime. The filestore provider performs the select file action after receiving the F-SELECT indication and before issuing an F-SELECT response primitive with an action result indicating success.

15.1.2 Types of primitives and parameters

Table 15 indicates the types of primitives and the parameters needed for file selection.

15.1.2.1 State result

The state result parameter is defined in 13.1. The parameter indicates whether the select regime has been successfully established.

15.1.2.2 Action result

The action result parameter is defined in 13.2.

15.1.2.3 Attributes

The attributes parameter is defined in 13.5. On F-SELECT the parameter carries only a single file name. The file name is the only file attribute used to identify the file in the file

Table 15 — F-SELECT parameters

Parameter	F-SELECT request	F-SELECT indication	F-SELECT response	F-SELECT confirm
State result			Mandatory	Mandatory
Action result			Mandatory	Mandatory
Attributes	Mandatory	Mandatory (=)	Mandatory	Mandatory (=)
Requested Access	Mandatory	Mandatory (=)		
Access Passwords	Conditional	Conditional (=)		
Concurrency Control	Optional	Optional (=)		
Shared ASE Information	Optional	Optional (=)	Optional	Optional (=)
Account	Optional	Optional (=)		
Diagnostic			Optional	Optional

(=) signifies that the value is not modified by the file service provider.

selection process. On the F-SELECT response and confirm primitives only the filename is returned which may not be identical to that issued on the request and indication.

15.1.2.4 Requested Access

The requested access parameter is defined in 13.6. If the requested access is not permitted by the responder the select regime is not established.

15.1.2.5 Access Passwords

The access passwords parameter is defined in 13.7. It is conditional on the security group being selected. If the members of the set of access passwords do not match any of the non-empty password strings in the access control condition (see ISO 8571-2), the select regime is not established. This parameter is used to set the current access passwords activity attribute.

15.1.2.6 Concurrency Control

The concurrency control parameter is defined in 13.8. If the required concurrency control is not available the select regime is not established.

15.1.2.7 Shared ASE Information

The shared ASE information parameter is defined in 13.10.

15.1.2.8 Account

The account parameter is defined in 13.3. The value given overrides any value previously set by the F-INITIALIZE request for the duration of this select regime.

15.1.2.9 Diagnostic

The diagnostic parameter is defined in 13.13.

15.2 File deselection service

15.2.1 Function

The F-DESELECT service releases the binding between the file select regime and the file. The file continues to exist and is available for subsequent selection. The primitives may only be used while a file is selected. The file select regime is always terminated following an F-DESELECT response or confirm for all values of the action result parameter.

The filestore provider performs the deselect file action after receiving the F-DESELECT indication primitive, and before issuing the F-DESELECT response primitive. In the external service all concurrency control is released when the file is deselected, except when performed within a CCR atomic action. The current account activity attribute is reset to that of the FTAM regime.

15.2.2 Types of primitives and parameters

Table 16 indicates the types of primitives and the parameters needed for file deselection.

15.2.2.1 Action result

The action result parameter is defined in 13.2. The selection regime is terminated whatever the value of the action result parameter.

15.2.2.2 Charging

The charging parameter is defined in 13.4. Charges are only reported on deselection against an overriding account set up at file selection.

15.2.2.3 Shared ASE Information

The shared ASE information parameter is defined in 13.10.

15.2.2.4 Diagnostic

The diagnostic parameter is defined in 13.13.

15.3 File creation service

15.3.1 Function

The F-CREATE service generally causes a file to be created and establishes a selection of the newly created file. If the identified file exists then, depending on the override parameter, the request may be rejected, or the existing file may be selected, possibly after recreation. The create service may only be used if there is no currently selected file.

The filestore provider performs the create file action after receiving the F-CREATE indication primitive, and before issuing the F-CREATE response primitive with action result parameters indicating success. If the override parameter is appropriately set and the named file already exists, the responder performs the select action instead of the create action.

15.3.2 Types of primitives and parameters

Table 17 indicates the types of primitives and the parameters needed for file creation.

15.3.2.1 State result

The state result parameter is defined in 13.1. The parameter indicates the select regime is established, either for an existing file or a newly created one, or that the select has failed.

15.3.2.2 Action result

The action result parameter is defined in 13.2. The parameter indicates whether the create was successful or not. It is possible for the create to be successful and the

Table 16 — F-DESELECT parameters

Parameter	F-DESELECT request	F-DESELECT indication	F-DESELECT response	F-DESELECT confirm
Action result			Mandatory	Mandatory
Charging			Conditional	Conditional
Shared ASE Information	Optional	Optional (=)	Optional	Optional (=)
Diagnostic			Optional	Optional

(=) signifies that the value is not modified by the file service provider.

Table 17 — F-CREATE parameters

Parameter	F-CREATE request	F-CREATE indication	F-CREATE response	F-CREATE confirm
State result			Mandatory	Mandatory
Action result			Mandatory	Mandatory
Override	Mandatory	Mandatory (=)		
Initial Attributes	Mandatory	Mandatory (=)	Mandatory	Mandatory (=)
Create Password	Optional	Optional (=)		
Requested Access	Mandatory	Mandatory (=)		
Access Passwords	Conditional	Conditional (=)		
Concurrency Control	Conditional	Conditional (=)		
Shared ASE Information	Optional	Optional (=)	Optional	Optional (=)
Account	Optional	Optional (=)		
Diagnostic			Optional	Optional

(=) signifies that the value is not modified by the file service provider.

subsequent select to fail, in which case a new file is known to the filestore but it is not currently selected.

15.3.2.3 Override

The override parameter defines the action to be taken if the named file already exists. The values of the parameter are;

a) fail the create action if the file already exists;

b) select file if it already exists;

c) delete file if it already exists and create a new file using the old file's attributes (effectively delete the contents of the existing file and select it);

d) delete file if it already exists and create a new file using the initial attributes provided on the F-CREATE primitive.

The values (c) and (d) are dependent upon the provision of a delete password in the access passwords parameter where required by the access control attribute of the existing file. This parameter is not used to set any activity attributes.

15.3.2.4 Initial Attributes

The initial attributes parameter has the same form as the attributes parameter, which is defined in 13.5. For those attribute groups negotiated on F-INITIALIZE, the attribute values given, are associated with the newly created file. If, for any particular attribute a value is not given, a default is determined locally by the responder; this value may be "no value available". For attributes other than filename and permitted actions, the responder may change a value proposed by the initiator to "no value available" but may not assign any different value to it. The responder does report any local modification to the values of the filename or permitted actions attributes. In addition to the attributes in the attribute groups negotiated on F-INITIALIZE the attributes in the kernel attribute group are also associated with the newly created file. This parameter is used to set the current access passwords activity attribute if a new file has been created.

15.3.2.5 Create Password

The create password is an optional parameter which may be required to establish that the user identified by the initiator identity parameter of the FTAM regime establishment service has permission to create files in the current filestore. The value is an OCTET STRING or GraphicString. This parameter is not used to set any activity attributes.

15.3.2.6 Requested Access

The requested access parameter is defined in 13.6. The parameter is used to establish the access available in the select regime following the create action. The requested access during the select regime need not be the full capability of the file. Hence the access requested must be a subset of those capabilities established with the initial attributes parameter. The established capabilities include all responder provided defaults which are within the kernel attribute group or the other attribute groups negotiated at F-INITIALIZE.

If the requested access parameter is incompatible with the newly created file, the select will fail even though the file was created.

15.3.2.7 Access Passwords

The access passwords parameter is defined in 13.7. The access passwords are used to authenticate the requested access and, if appropriate, the requested concurrency control for the select regime. If the override parameter creates a new file or recreates a file with new attributes the access password parameter is only used to verify any delete actions; it is not used to verify other actions on the selected file since the appropriate passwords are included in the initial attributes parameter. This parameter is used to set the current access passwords activity attribute if the override parameter resulted in the selection of an existing file.

15.3.2.8 Concurrency Control

The concurrency control parameter is defined in 13.8.

15.3.2.9 Shared ASE Information

The shared ASE information parameter is defined in 13.10.

15.3.2.10 Account

The account parameter is defined in 13.3. The value given overrides any value previously set on the F-INITIALIZE request for the duration of this selection.

15.3.2.11 Diagnostic

The diagnostic parameter is defined in 13.13.

15.4 File deletion service

15.4.1 Function

The F-DELETE service releases an existing file select regime in such a way that the selected file ceases to exist, and is not available for reselection. The primitives may only be issued while a file is selected. The selected regime is always released following an F-DELETE response or confirm, for all possible values of the action result parameter.

The filestore provider performs the delete file action after receiving the F-DELETE indication, and before issuing the F-DELETE response primitive, with an action result parameter of success. The delete file action can be performed only if the initiating entity has the "delete file" access control permission (see 13.6). This access control permission is established by the requested access, concurrency control and access passwords parameters on the F-SELECT or F-CREATE primitives which established the file selection regime.

15.4.2 Types of primitives and parameters

Table 18 indicates the types of primitives and the parameters needed for file deletion.

15.4.2.1 Action Result

The action result parameter, defined in 13.2, indicates success or failure of the deletion action. The file is always deselected whatever the value of the action result parameter.

15.4.2.2 Shared ASE Information

The shared ASE information parameter is defined in 13.10.

15.4.2.3 Charging

The charging parameter is defined in 13.4. Charges are only reported on deselection against an overriding account set up at file selection.

15.4.2.4 Diagnostic

The diagnostic parameter is defined in 13.13.

Table 18 — F-DELETE parameters

Parameter	F-DELETE request	F-DELETE indication	F-DELETE response	F-DELETE confirm
Action Result			Mandatory	Mandatory
Shared ASE Information	Optional	Optional (=)	Optional	Optional (=)
Charging			Conditional	Conditional
Diagnostic			Optional	Optional

(=) signifies that the value is not modified by the file service provider.

16 File management

These services provide file management capability to the file service user. They are not concerned with the establishment of regimes. The services provide facilities for interrogation and modification, where permitted, of file attributes.

16.1 Read attribute service

16.1.1 Function

The F-READ-ATTRIB service interrogates the file attributes of the selected file. The request and indication convey a list of the file attribute names for which values are to be read. The response and confirm then convey the corresponding returned values.

The filestore provider performs the read attribute action after receiving the F-READ-ATTRIB indication, and before issuing the F-READ-ATTRIB response primitive.

16.1.2 Types of primitives and parameters

Table 19 indicates the types of primitive and the parameters needed to read file attributes.

16.1.2.1 Action Result

The action result parameter is defined in 13.2.

16.1.2.2 Attribute Names

The attribute names parameter indicates which file attributes from the kernel or negotiated attribute groups, given in the virtual filestore definition, are to be read. The parameter is a list, each element of which names a file attribute defined in ISO 8571-2. This parameter is not used to set any activity attributes.

16.1.2.3 Attributes

The attribute parameter is defined in 13.5. The presence of the attributes parameter is conditional on the inclusion in the attribute names parameter of at least one attribute with a returnable value. Values for attributes not specifically requested are not returned. Values for attributes not in the kernel attribute group or a negotiated attribute group are not returned. The value of password attributes are not returned. The responder may return "no value available" for partially supported attributes. ISO 8571-2 details the attributes which are and are not returned.

16.1.2.4 Diagnostic

The diagnostic parameter is defined in 13.13.

16.2 Change attribute service

16.2.1 Function

The F-CHANGE-ATTRIB service modifies the file attributes of the selected file. The primitives convey a list of file attribute names and values.

The filestore provider performs the change attribute action and sets the two file attributes, date and time of last attribute modification and identity of last attribute modifier, after receiving the F-CHANGE-ATTRIB indication, and before issuing the F-CHANGE-ATTRIB response primitive. If any change causes an action result other than success to be generated, none of the requested changes are made; an "unsuccessful" action result parameter is returned.

16.2.2 Types of primitives and parameters

Table 20 indicates the types of primitive and the parameters needed to change file attributes.

16.2.2.1 Action Result

The action result parameter is defined in 13.2. An unsuccessful action result indicates none of the changes has been performed.

16.2.2.2 Attributes

The attribute parameter, defined in 13.5, indicates on the request and indication primitives which file attributes given in the virtual filestore definition are to be changed and what the new values are to be. The parameter is a list, each element of which names a file attribute defined in ISO 8571-2 and provides a new value for it.

The presence of the attributes parameter on the response and confirm is conditional on the inclusion in the attributes parameter of at least one attribute name and value. Values for attributes not specifically requested are not changed or returned. Values for attributes not in the kernel or negotiated attribute group are not changed or returned. ISO 8571-2 defines the actions permitted on attributes.

16.2.2.3 Diagnostic

The diagnostic parameter is defined in 13.13.

17 File open regime control

These services establish or release the file open regime within which file data can be transferred. This regime establishes processing mode, presentation contexts, and concurrency control for the data transfer activity which is to be performed.

Table 19 — F-READ-ATTRIB parameters

Parameter	F-READ-ATTRIB request	F-READ-ATTRIB indication	F-READ-ATTRIB response	F-READ-ATTRIB confirm
Action Result			Mandatory	Mandatory
Attribute Names	Mandatory	Mandatory (=)		
Attributes			Conditional	Conditional (=)
Diagnostic			Optional	Optional

(=) signifies that the value is not modified by the file service provider.

Table 20 — F-CHANGE-ATTRIB parameters

Parameter	F-CHANGE-ATTRIB request	F-CHANGE-ATTRIB indication	F-CHANGE-ATTRIB response	F-CHANGE-ATTRIB confirm
Action Result			Mandatory	Mandatory
Attributes	Mandatory	Mandatory (=)	Conditional	Conditional (=)
Diagnostic			Optional	Optional

(=) signifies that the value is not modified by the file service provider.

17.1 File open service

17.1.1 Function

The F-OPEN service establishes an open regime. The regime establishment determines the abstract syntaxes and document types necessary to access the file and establish that the association has the capability of communicating them. It also establishes concurrency and possible processing modes.

The open service may only be used if a file has been selected, and no open regime currently exists.

The filestore entity performs the open file action after receiving the F-OPEN indication primitive, and before issuing the F-OPEN response primitive with an action result parameter indicating success.

The current location activity attribute is set by this service.

17.1.2 Types of primitives and parameters

Table 21 indicates the types of primitive and the parameters needed for file opening.

17.1.2.1 State Result

The state result parameter, defined in 13.1, indicates whether the open regime has been established or not.

17.1.2.2 Action Result

The action result parameter is defined in 13.2.

17.1.2.3 Processing Mode

The processing mode parameter establishes a subset of the valid actions negotiated in the select regime, for use within the open regime being established. The processing mode parameter indicates the valid actions to be performed as a result of access control and bulk data transfer requests; this determines the filestore actions which the responding entity can perform. The parameter value indicates whether read, insert, replace, extend or erase are to be permitted. The parameter is compared with the current access request activity attribute and the access constraints determined by the selected view of the document type referenced by the Contents Type parameter (see 17.1.2.4). If the requested processing mode is not available for the filestore, or the constraints set referenced by the contents type may potentially be violated, the open regime establishment fails. This parameter is used to set the current processing mode activity attribute.

17.1.2.4 Contents Type

The contents type parameter identifies the abstract data type of the contents of the file, including the structuring information. The contents type is a scalar set at file creation time. Its value is either a document type with optional parameters or an abstract syntax together with a constraint set name.

The request and indication primitives convey either the required contents type or indicate "contents type unknown". For the case of a document type the parameters may be present or they may be omitted if they are unknown.

Table 21 — F-OPEN parameters

Parameter	F-OPEN request	F-OPEN indication	F-OPEN response	F-OPEN confirm
State Result			Mandatory	Mandatory
Action Result			Mandatory	Mandatory
Processing Mode	Mandatory	Mandatory (=)		
Contents Type	Mandatory	Mandatory (=)	Mandatory	Mandatory (=)
Concurrency Control	Optional	Optional (=)	Dependent	Dependent (=)
Shared ASE Information	Optional	Optional (=)	Optional	Optional (=)
Enable FADU Locking	Conditional	Conditional (=)		
Diagnostic			Optional	Optional
Additional Parameters in the Internal File Service				
Activity Identifier	Conditional	Conditional (=)		
Recovery Mode	Conditional	Conditional	Conditional	Conditional (=)

(=) signifies that the value is not modified by the file service provider.

The response and confirm always indicate the valid file contents type. The initiator indicating on the request and indication primitives that the contents type is unknown does not prevent the file being opened. If the open regime establishment is successful the contents type parameter, on the response and confirm primitives, conveys the contents type file attribute from the file or, if a simplification or relaxation was requested that is returned. The initiator is then able to decide on further action.

If the contents type parameter on the request and indication primitives is a document type name it may take one of three forms:

a) a document type name only

b) a document type name with a null list of parameters

c) a document type name with a list of parameters.

the open only succeeds if:

d) the proposed document type name is identical to either the document type name in the contents type file attribute, or, if only read actions are allowed by the processing mode parameter, a simplification or relaxation defined in the document type referenced in the contents type file attribute;

and

e) if parameters were provided, they match exactly the parameters of the contents type file attribute. A null list of parameters match any parameters in the contents type file attribute.

If the contents type parameter on the request and indication primitives is a constraint set name and abstract syntax name pair then the open only succeeds if the pair exactly match the pair in the contents type file attribute. This parameter is used to set the active contents type activity attribute.

17.1.2.5 *Concurrency Control*

The concurrency control parameter, defined in 13.8, allows the concurrency control information, initially established at file creation or selection (see 15.3 and 15.1), to be modified when the file is opened. The concurrency control restrictions, set by the initiator, may be made more restrictive than that specified on the selection, but are not made less restrictive.

The F-OPEN fails if a file service user, having established shared access for an action on file selection, requests a more restrictive lock on F-OPEN and that lock is unavailable.

The sequence of locks, in ascending order of restrictiveness, is not required, shared, exclusive and no access.

If the FADU locking functional unit has been negotiated, and the Enable FADU locking parameter is set on (see 17.1.2.7), the concurrency controls established at file select and/or open are only in place during a data transfer regime. Outside a data transfer regime, an FADU is in the "not required" state unless explicitly locked by the FADU locking mechanism.

17.1.2.6 Shared ASE Information

The shared ASE information parameter is defined in 13.10.

17.1.2.7 Enable FADU Locking

Use of the enable FADU locking parameter modifies the semantics of concurrency control. The enable FADU locking parameter is available only if the FADU locking functional unit was negotiated on the FTAM regime establishment, in which case it is mandatory. The enable FADU locking parameter indicates whether locking is to be on a per FADU basis, as opposed to a file basis. The value of this parameter is a boolean which indicates whether FADU locking, in addition to whole file concurrency control locking, is required.

The enable FADU locking parameter is only available if:

a) the storage attribute group has been negotiated; and

b) the concurrency control parameter is present.

If FADU locking is selected the open will not fail for concurrency control reasons, however subsequent attempts to set FADU locks may fail. If FADU locking is not enabled the open may fail as described in 17.1.2.5. This parameter is used to set the current locking style activity attribute.

17.1.2.8 Diagnostic

The diagnostic parameter is defined in 13.13.

17.1.2.9 Activity Identifier

The activity identifier parameter is defined in 13.11; it is mandatory in the internal file service if the recovery functional unit has been negotiated.

17.1.2.10 Recovery Mode

The recovery mode parameter in the internal service indicates what error recovery facilities are to be available during the current open regime, and at what points in the file, data transfer can be resumed. The value is one of "none", "at start of file", or "at any active checkpoint". The recovery mode parameter is conditional and is only available in the internal file service, and then is mandatory if the restart or recovery functional units were negotiated on F-INITIALIZE. The value "none" allows the recovery facility to be made unavailable for a specific open regime. The recovery mode is negotiated by the initiator proposing one value and the responder returning that value, as agreement to it, or a value lower if the responder is not prepared to support the initiator's value. The value returned by the responder is that established for the open regime. The decreasing order of value is

a) any active checkpoint;

b) at the start of file;

c) none.

Whatever the value of this parameter it does not prevent the establishment of an open regime. If the value none is agreed then restart and recovery within the open regime is invalid. This parameter is not used to set any activity attribute.

17.2 File close service

17.2.1 Function

The F-CLOSE service releases an existing file open regime. The file open regime is always terminated following an F-CLOSE response and confirm primitive for all possible values of the action result parameter.

Table 22 — F-CLOSE

Parameter	F-CLOSE request	F-CLOSE indication	F-CLOSE response	F-CLOSE confirm
Action Result	Optional	Optional (=)	Mandatory	Mandatory
Shared ASE Information	Optional	Optional (=)	Optional	Optional (=)
Diagnostic	Optional	Optional (=)	Optional	Optional

(=) signifies that the value is not modified by the file service provider.

The filestore provider performs the close file action after receiving the F-CLOSE indication primitive, and before issuing the F-CLOSE response primitive.

The file attributes in the storage attribute group are updated as follows, dependent on the processing mode:

a) For read mode, the date and time of last read access and identity of last reader.

b) For insert, replace, extend and erase modes the date and time of last modification and the identity of last modifier.

17.2.2 Types of primitives and parameters

Table 22 indicates the types of primitives and the parameters needed for terminating a file open regime.

17.2.2.1 Action Result

The action result parameter is defined in 13.2. The open regime is terminated notwithstanding the value of the action result or diagnostic parameter. The action result parameter, and the associated diagnostic parameter, play different roles in the two file services. In the external file service the action result parameter is only present on the response and confirm primitives where it indicates either success or permanent error. In the internal file service the action result parameter may also be present on the request and indication where it is used to signal a transient error between the two error recovery protocol machines.

17.2.2.2 Shared ASE Information

The shared ASE information parameter is defined in 13.10.

17.2.2.3 Diagnostic

The diagnostic parameter is defined in 13.13. It is used to qualify the action result parameter on each primitive and is optionally present in each of the file services when the action result parameter is present. (see 17.2.2.1)

18 Grouping control

The grouping control mechanisms allow a number of regimes to be established or released in one interaction. The service user initiating the group of requests brackets them with grouping control primitives. The responding user analyses and retains its response to each of the requests until the group is complete or a state result failure is detected. It then returns its responses in a corresponding group.

A parameter of the F-BEGIN-GROUP request specifies a threshold value which is the number of primitives to be processed after the F-BEGIN-GROUP, without a "failure" state result (see 13.1), before any action is to be taken. Primitives within the group which do not convey state result parameters are always counted as being "successful". Action result failures do not affect the processing of the group.

If the threshold number of primitives is not processed before a state result failure is detected, a negative response is made for the group. This is signalled by all primitive responses after the begin group response, and before the state result failure response, having their action result parameter indicate "permanent error". The diagnostic on the response of these primitives is set to indicate "transient error" with an error identifier of "subsequent error". If the first response primitive has a state result of failure the diagnostic parameter carries normal detailed failure reasons.

Once a failure response is made on a state result parameter, no response is made to any further primitive within the group, except the F-END-GROUP primitive.

Once the threshold value has been equalled or exceeded by the success count, the successful actions are performed; all remaining primitives are attempted in order until a primitive generating state failure is encountered when no further primitives are processed; or F-END-GROUP is encountered.

18.1 Beginning of grouping service

18.1.1 Function

The F-BEGIN-GROUP service indicates the start of a set of grouped requests, which are to be processed and responded to as a group.

18.1.2 Types of primitives and parameters

Table 23 indicates the types of primitives and the parameters needed for beginning of group.

18.1.2.1 Threshold

The threshold parameter specifies the number of primitives within a group (i.e. after, but not including, the begin group) which are to be analysed without being unsuccessful before any part of the group can succeed. A primitive is categorized as successful or unsuccessful in terms of its state result parameters. Those primitives which only have an action result parameter are to be deemed successful for the purposes of the threshold count. For the groupings specified in the file transfer, file management and file transfer and management classes of ISO 8571 the threshold parameter value equals the number of primitives between (but not including) the begin group and end group primitives. This parameter is not used to set any activity attributes.

Table 23 — F-BEGIN-GROUP parameters

Parameter	F-BEGIN-GROUP request	F-BEGIN-GROUP indication	F-BEGIN-GROUP response	F-BEGIN-GROUP confirm
Threshold	Mandatory	Mandatory (=)		

(=) signifies that the value is not modified by the file service provider.

18.2 End of grouping service

The F-END-GROUP service indicates the end of a set of concatenated requests started by an F-BEGIN-GROUP. There is always an F-END-GROUP to match each F-BEGIN-GROUP, unless the application association is broken by an F-U-ABORT or an F-P-ABORT. The types of primitives defined are F-END-GROUP request, F-END-GROUP indication, F-END-GROUP response and F-END-GROUP confirm. The primitives do not carry parameters.

19 Recovery (Internal service only)

19.1 Regime recovery service

19.1.1 Function

The F-RECOVER service allows the recreation of an open regime after a failure. It allows the filestore entity to access records of suspended activities by reference to a previously established activity identifier. The regimes are recreated on the basis of information, including activity attribute values, stored by the initiator and the responder in secure storage. This body of information is called a docket.

Regime recovery is visible to the internal service user only, and then only if the recovery functional unit was negotiated on the F-INITIALIZE primitives.

NOTE - The mechanisms for recovery in the external service are specified as part of the error recovery protocol specified in ISO 8571-4.

19.1.2 Types of primitives and parameters

Table 24 indicates the types of primitives and the parameters needed for recovery. The parameters are used to identify the activity to be recovered and docket to be associated with the activity. These parameters and the docket information are subjected to the select and open algorithms in order to recover the open regime.

19.1.2.1 State Result

The state result parameter is defined in 13.1.

19.1.2.2 Action Result

The action result parameter is defined in 13.2.

19.1.2.3 Activity Identifier

The activity identifier parameter is defined in 13.11.

19.1.2.4 Bulk Transfer Number

The bulk transfer number parameter indicates, by reference to the number of a bulk data transfer procedure, which bulk data transfer is to be recovered. The recovery position within the bulk data transfer is defined by the recovery point parameter (see 19.1.2.8). The first data transfer within an open regime is numbered 1, and subsequent transfers are consecutively numbered. Read and write transfers are numbered from a single sequence.

19.1.2.5 Requested Access

The requested access parameter is defined in 13.6. The requested access and access password parameters confirm the authenticity of the initiator attempting to recover the activity. If the requested access does not completely satisfy the responder the regime establishment fails with a permanent error.

19.1.2.6 Access Passwords

The access passwords parameter is defined in 13.7.

Table 24 — F-RECOVER parameters

Parameter	F-RECOVER request	F-RECOVER indication	F-RECOVER response	F-RECOVER confirm
State Result			Mandatory	Mandatory
Action Result			Mandatory	Mandatory
Activity Identifier	Mandatory	Mandatory (=)		
Bulk Transfer Number	Mandatory	Mandatory (=)		
Requested Access	Mandatory	Mandatory (=)		
Access Passwords	Optional	Optional (=)		
Contents type			Mandatory	Mandatory
Recovery Point	Conditional	Conditional (=)	Conditional	Conditional (=)
Diagnostic			Optional	Optional

(=) signifies that the value is not modified by the file service provider.

19.1.2.7 Contents Type

The contents type parameter, defined in 17.1.2.4, returns the file contents type name to the initiator.

19.1.2.8 Recovery Point

The recovery point parameter indicates that recovery is to a point before the start of the bulk data transfer (value zero), or to a checkpoint within the bulk data transfer, or to a point following its completion (see note). The recovery point is within the bulk data transfer identified by the bulk data transfer number parameter (see 19.1.2.4). The recovery point is determined by the entity which was receiving data at the time of the failure. Consequently the parameter is conditional and is required to be present either on the request or the response primitive.

NOTE - Recovery to a point following a bulk data transfer completion is defined for the case where the receiver has received an F-DATA-END indication but the F-TRANSFER-END exchange has not completed. In such a situation it would be unnecessary to go back to the last confirmed checkpoint. The only requirement is for both sender and receiver to have an agreed understanding of the completion of the transfer. A value one larger than the highest checkpoint number issued by the sender is used to indicate recovery after bulk data transfer.

19.1.2.9 Diagnostic

The diagnostic parameter is defined in 13.13.

20 Access to file contents

20.1 Bulk data transfer service

The access to all or part of the content of a file is performed by the bulk data transfer procedures defined in section three.

These procedures start and finish in a single data transfer idle state, and so can be considered as a self-contained procedural unit, which is primitive in the definition of the remainder of the file service.

In using the bulk data transfer procedures, the following FTAM specific definitions apply:

a) Table 25 defines the sub-parameters in the Bulk Data Transfer Specification parameter for a read transfer:

Table 25 — BDT read sub-parameters

Sub-parameter	Status	Defined in
File Access Data Unit Identity	Mandatory	13.12
Access Context	Mandatory	20.1.2
FADU Lock	Optional	13.9

b) Table 26 defines the sub parameters in the Bulk Data Transfer Specification parameter for a write transfer:

Table 26 — BDT write sub-parameters

Sub-parameter	Status	Defined in
File Access Data Unit Operation	Mandatory	20.1.1
File Access Data Unit Identity	Mandatory	13.12
FADU Lock	Optional	13.9

The filestore provider performs the locate action after receiving an F-READ or F-WRITE indication. It then initiates a read, insert, replace or extend action as appropriate for the FADU requested or for each FADU received. The read action is completed before the issue of an F-DATA-END request and the write action or actions are completed before the issue of an F-TRANSFER-END response primitive.

The file remains open after a sequence of F-CANCEL primitives, although the result of interrupted operations is not defined. Further F-READ or F-WRITE operations, not necessarily related to any previous read or write attempt, may be attempted after the completion of the sequence of F-CANCEL primitives has disposed of any previous activity.

20.1.1 File Access Data Unit Operation

The file access data unit operation parameter indicates the action to be taken by the filestore provider on receipt of the data transferred. The values are defined in the constraint set in use.

20.1.2 Access Context

The access context parameter specifies a view of the file access structure, for read operations, which is to be used for a read transfer during the data transfer regime.

The valid values of the access context parameter are dependent on the constraint set in use.

The value of the parameter is given in table 27.

Table 27 — Access contexts

Access Context	Description
HA	Hierarchical All Data Units
HN	Hierarchical No Data Units
FA	Flat All Data Units
FL	Flat One Level Data Units
FS	Flat Single Data Unit
UA	Unstructured All Data Units
US	Unstructured Single Data Unit

The above access contexts are defined in ISO 8571-2.

20.2 Locate file access data unit service

20.2.1 Function

The F-LOCATE service specifies the identity of a file access data unit which is to be located by the filestore provider in preparation for file access.

20.2.2 Types of primitives and parameters

Table 28 indicates the types of primitives and the parameters needed for a locate interaction.

20.2.2.1 Action Result

The action result parameter is defined in 13.2.

20.2.2.2 File Access Data Unit Identity

The file access data unit identity parameter is defined for the request and indication primitives in 13.12. The identity in the request and indication may be any of the values defined in ISO 8571-2, including next, first or last.

The response and confirm primitives return an identity which is one of: begin, end, Node-name, Sequence of node-names or Node number.

The above identities are defined in ISO 8571-2.

This parameter is used to set the current location activity attribute.

Table 28 — F-LOCATE parameters

Parameter	F-LOCATE request	F-LOCATE indication	F-LOCATE response	F-LOCATE confirm
Action Result			Mandatory	Mandatory
File Access Data Unit Identity	Mandatory	Mandatory (=)	Optional	Optional (=)
FADU Lock	Optional	Optional (=)		
Diagnostic			Optional	Optional

(=) signifies that the value is not modified by the file service provider.

20.2.2.3 FADU Lock

The FADU lock parameter is defined in 13.9.

20.2.2.4 Diagnostic

The diagnostic parameter is defined in 13.13.

20.3 Erase file access data unit service

20.3.1 Function

The F-ERASE service specifies the identity of a file access data unit which is to be erased by the filestore provider. The filestore provider performs the erase action after receiving the F-ERASE indication, and before issuing the F-ERASE response primitive.

20.3.2 Types of primitives and parameters

Table 29 indicates the types of primitives and the parameters needed for an erase interaction.

20.3.2.1 Action Result

The action result parameter is defined in 13.2

20.3.2.2 File Access Data Unit Identity

The file access data unit identity parameter is defined in 13.12. See ISO 8571-2 for additional information.

20.3.2.3 Diagnostic

The diagnostic parameter is defined in 13.13.

Table 29 — F-ERASE parameters

Parameter	F-ERASE request	F-ERASE indication	F-ERASE response	F-ERASE confirm
Action Result			Mandatory	Mandatory
File Access Data Unit Identity	Mandatory	Mandatory (=)		
Diagnostic			Optional	Optional

(=) signifies that the value is not modified by the file service provider.

Section three: Definition of bulk data transfer primitives

21 Bulk data transfer service primitives

Each of the services constituting the bulk data transfer service is achieved by invoking a sequence of bulk data transfer service primitives.

Table 30 lists for each service;

a) the primitives associated with the service;

b) the parameters associated with the primitives;

c) the bulk data transfer service user that is permitted to issue the request primitive;

d) how failure of the service is notified to the issuer of the request.

In table 30 those parameters and primitives which occur only in the internal bulk data transfer service are enclosed in square brackets.

During a read operation the responder is the sender and the initiator the receiver of the file. During a write operation the initiator is the sender and the responder the receiver of the file.

The semantics of the primitives and their parameters are defined in clauses 23 to 25.

22 Sequences of bulk data transfer primitives

This clause gives the constraints on the valid sequences in which the primitives defined in clauses 24 and 25 can occur. The individual primitives in a service may occur only in the sequences given as part of the primitive definitions.

22.1 Normal sequences

The normal progress of use of the bulk data transfer service is illustrated by the state transition diagram shown in figure 8. Full state transition diagrams are contained in annex E.

22.2 Constraints on issue of primitives

The primitives may be issued in any sequence consistent with the constraints given in tables 31 and 32 The sequences given with the individual primitive definitions apply.

22.2.1 Key to tables 31 and 32

In tables 31 and 32, the entries indicate the functional units required for the succession to occur. The entries are:

Read	read functional unit
Write	write functional unit
Either	read or write functional unit

The row "start of bulk data" indicates entry from tables 9 and 10 of section two, and the column "end of bulk data" indicates return to tables 9 and 10.

Table 30 — Bulk data transfer service primitives

Primitive Name	Confirmed Service	Request by	Parameters	Failure Notification
F-READ	No	Initiator	Bulk data transfer specification	F-DATA-END action result
F-WRITE	No	Initiator	Bulk data transfer specification	F-CANCEL
F-DATA	No	Sender	Data value	F-CANCEL
F-DATA-END	No	Sender	Action result Diagnostic	F-CANCEL
F-TRANSFER-END	Yes	Initiator	Action result Shared ASE information Diagnostic	action result
F-CANCEL	Yes	Either	Action result Shared ASE information Diagnostic	none
[F-CHECK	Yes	Sender	Checkpoint identifier	F-CANCEL]
[F-RESTART	Yes	Either	Checkpoint identifier	F-CANCEL]

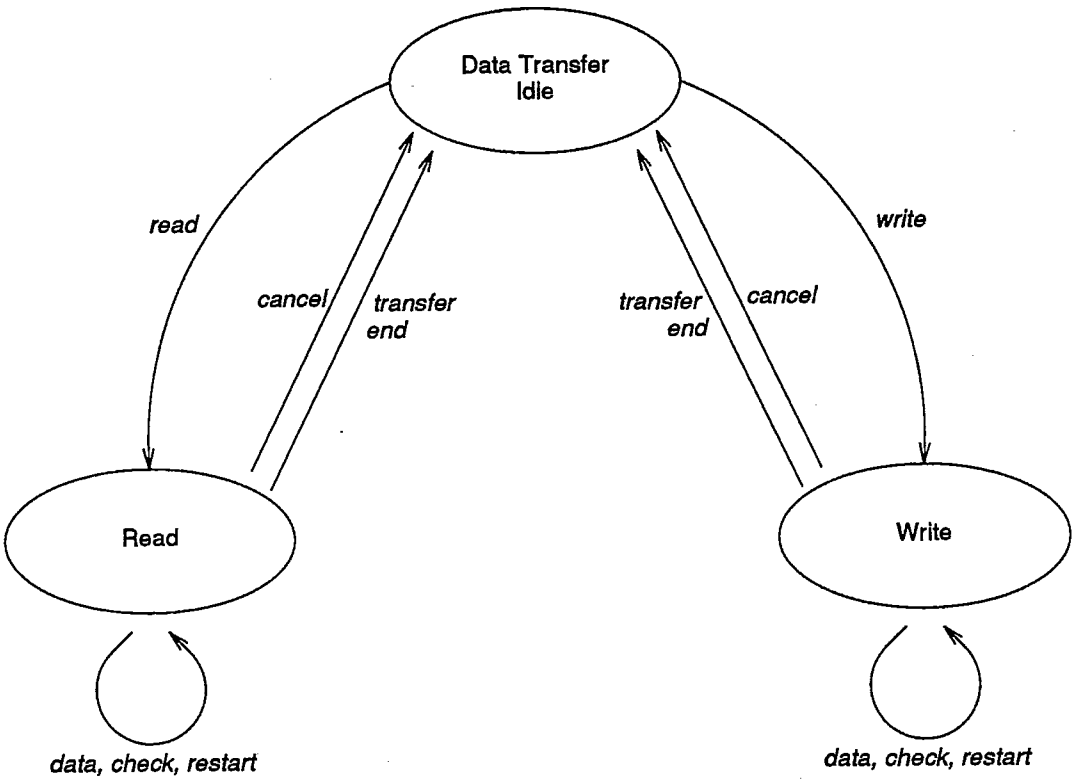


Figure 8 — Simplified State Diagram for Bulk Data Transfer (see Annex E)

Table 31 — Sequence of service primitives for bulk data transfer — initiator

Previous file service event	May next issue				
	F-READ request	F-WRITE request	F-DATA request	F-DATA-END request	F-TRANSFER -END request
Start file Bulk Data	Read	Write			
F-READ request					
F-WRITE request			Write	Write	
F-DATA request			Write	Write	
F-DATA indication					
F-DATA-END request					Write
F-DATA-END indication					Read
F-TRANSFER-END request					
F-TRANSFER-END confirm					
F-CANCEL request					
F-CANCEL indication					
F-CANCEL response					
F-CANCEL confirm					
F-CHECK request			Write	Write	
F-CHECK indication					
F-CHECK response					Read
F-CHECK confirm			Write	Write	Write
F-RESTART request					
F-RESTART indication					
F-RESTART response			Write	Write	
F-RESTART confirm			Write	Write	

May next issue						
F-CANCEL request	F-CANCEL response	F-CHECK request	F-CHECK response	F-RESTART request	F-RESTART response	End of Bulk Data
Read				Read		
Write		Write		Write		
Write		Write		Write		
Read			Read	Read		
Read			Read	Read		
						Either
	Either					
						Either
						Either
Write		Write		Write		
Read			Read	Read		
Read			Read	Read		
Write		Write		Write		
Either						
Either					Either	
Either		Write		Either		
Either		Write		Either		

Table 32 — Sequence of service primitives for bulk data transfer — responder

Previous file service event	May next issue			
	F-DATA request	F-DATA-END request	F-TRANSFER -END response	F-CANCEL request
Start file Bulk Data				
F-READ indication	Read	Read		Read
F-WRITE indication				Write
F-DATA request	Read	Read		Read
F-DATA indication				Write
F-DATA-END request				
F-DATA-END indication				Write
F-TRANSFER-END indication			Either	Write
F-TRANSFER-END response				
F-CANCEL request				
F-CANCEL indication				
F-CANCEL response				
F-CANCEL confirm				
F-CHECK request	Read	Read		Read
F-CHECK indication				Write
F-CHECK response			Write	Write
F-CHECK confirm	Read	Read		Read
F-RESTART request				Either
F-RESTART indication				Either
F-RESTART response	Read	Read		Either
F-RESTART confirm	Read	Read		Either

May next issue					
F-CANCEL response	F-CHECK request	F-CHECK response	F-RESTART request	F-RESTART response	End of Bulk Data
	Read		Read		
			Write		
	Read		Read		
		Write	Write		
		Write	Write		
			Write		
					Either
Either					
					Either
					Either
	Read		Read		
		Write	Write		
		Write	Write		
	Read		Read		
				Either	
	Read		Either		
	Read		Either		

23 Common bulk data transfer parameters

23.1 Bulk Data Transfer Specification

The Bulk Data Transfer Specification parameter identifies the data to be transferred and for read transfer specifies the access context in which the data transfer is to take place. It may also specify the actions to be taken to generate or use the data transferred. The detailed form of the parameter depends on whether the bulk data transfer to be performed is read or write.

NOTE - The details of the parameter for FTAM use are given in 20.1.

23.2 Checkpoint Identifier

The checkpoint identifier parameter gives an unambiguous identification to allow reference to the checkpoints. The value of the parameter is an integer. For the F-CHECK primitive, the value is between 1 and 999998 inclusive. The value supplied in the first F-CHECK request in a bulk data transfer, either read or write, is 1, and thereafter subsequent values increment by 1. For the F-RESTART primitive, the value is between 0 and 999998.

The value zero is reserved to indicate "beginning of FADU" in restart and recovery activities. The value one greater than the last checkpoint identifier issued by the sender in a series of FADUs before data end is signalled indicates the end of the bulk data transfer. The correlation of the checkpoint identifier range constraint with that of the Session serial number constraint is the responsibility of the application entity as a whole.

24 Bulk data transfer

This group of services performs the transfer of bulk data. The procedure begins with the service initiator issuing either an F-READ request or an F-WRITE request. This leads to the issue of a sequence of F-DATA requests followed by an F-DATA-END request by the sender of the data. The procedure is completed by the initiator issuing an F-TRANSFER-END request. The primitives are defined in 24.1 to 24.6, and the two valid sequences are defined in 24.7 and 24.8.

In the internal service, the checkpointing and recovery primitives defined in clause 25 may occur within or abutted to the sequence of F-DATA primitives.

24.1 Read bulk data service

24.1.1 Function

The F-READ service specifies a data transfer from the service responder (that is the sender) to the service initiator (that is the receiver). Only one F-READ procedure may be in progress at any time on a single application association. The direction of data flow established continues until the exchange of F-TRANSFER-END primitives. Rejection of an F-READ indication is by issue of an F-DATA-END with an action result indicating unsuccessful.

These primitives signal a transfer of control from the initiator to the sender. They mark a reversal of the service asymmetry for the duration of the data transfer.

24.1.2 Types of primitives and parameters

Table 33 indicates the types of primitives and the parameters needed for a read bulk data interaction.

24.1.2.1 Bulk Data Transfer Specification

The bulk data transfer specification is defined in 23.1.

Table 33 — F-READ parameters

Parameter	F-READ request	F-READ indication
Bulk Data Transfer Specification	Mandatory	Mandatory (=)

(=) signifies that the value is not modified by the service provider.

24.2 Write bulk data service

24.2.1 Function

The F-WRITE service specifies a data transfer from the service initiator (that is the sender) to the service provider (that is the receiver). Only one F-WRITE procedure may be in progress at any time on a single application association. The direction of data flow established continues until the exchange of F-TRANSFER-END primitives. An F-WRITE indication can be rejected by issuing an F-CANCEL request (see 24.6). If the transfer is rejected, no further F-DATA indication primitives are received by the responder.

24.2.2 Types of primitives and parameters

Table 34 indicates the types of primitives and the parameters needed for a write bulk data interaction.

Table 34 — F-WRITE parameters

Parameter	F-WRITE request	F-WRITE indication
Bulk Data Transfer Specification	Mandatory	Mandatory (=)

(=) signifies that the value is not modified by the service provider.

24.2.2.1 Bulk Data Transfer Specification

The bulk data transfer specification is defined in 23.1

24.3 Data unit transfer service

24.3.1 Function

The F-DATA service transfers data between the users of the service. The data are transferred as values of known data types, using the underlying Presentation Service mechanisms (see ISO 8822). The data transfer may be from either entity, depending on whether a read or a write transfer has been requested by the initiator.

24.3.2 Types of primitives and parameters

Table 35 indicates the types of primitives and the parameters needed for data value transfer.

24.3.2.1 Data Value

The data value parameter is a value of a known data type; the type identifies the abstract syntax applicable to the data value and the syntactic description of the data value within that abstract syntax.

NOTE - In the supporting protocol, a series of data values may be conveyed as an equivalent series of presentation data values in a single P-DATA primitive.

Table 35 — F-DATA parameters

Parameter	F-DATA request	F-DATA indication
Data Value	Mandatory	Mandatory (=)

(=) signifies that the value is not modified by the service provider.

24.4 End of data transfer service

24.4.1 Function

The completion of the data transfer is indicated by the F-DATA-END primitives. The sender issues an F-DATA-END request primitive when it has sent all the necessary data. The sender may issue an F-DATA-END request with an unsuccessful action result as a rejection of an F-READ indication.

24.4.2 Types of primitives and parameters

Table 36 indicates the types of primitives and the parameters needed to end a data transfer.

Table 36 — F-DATA-END parameters

Parameter	F-DATA-END request	F-DATA-END indication
Action Result	Mandatory	Mandatory
Diagnostic	Optional	Optional

24.4.2.1 Action Result

The action result is defined in 13.2. When the F-DATA-END primitive is used to reject a read indication the reason is carried by the action result and diagnostic. The initiator responds with an F-TRANSFER-END as in the non error case.

24.4.2.2 Diagnostic

The diagnostic parameter is defined in 13.13.

24.5 End of transfer service

24.5.1 Function

The completion of a transfer is indicated by an exchange of F-TRANSFER-END primitives. This exchange is initiated by the initiator after having issued or received an F-DATA-END primitive. Receipt of the F-TRANSFER-END indication or confirm as appropriate informs the sender that no further error recovery actions, involving that bulk data transfer, will be requested.

24.5.2 Types of primitives and parameters

Table 37 indicates the types of primitives and the parameters needed for data transfer end.

24.5.2.1 Action Result

The action result parameter is defined in 13.2.

24.5.2.2 Shared ASE Information

The shared ASE information parameter is defined in 13.10.

24.5.2.3 Diagnostic

The diagnostic parameter is defined in 13.13.

24.6 Cancel data transfer service

24.6.1 Function

Either of the service users may cancel a data transfer activity by issuing an F-CANCEL request primitive. The F-CANCEL primitive may be issued during data transfer after the issue or receipt of an F-READ or F-WRITE request or indication. At the end of data transfer, it may not be issued:

a) by the initiator acting as sender after issue of an F-DATA-END request;

b) by the responder acting as receiver after issue of an F-TRANSFER-END response;

c) by the responder acting as sender after the issue of an F-DATA-END request;

d) by the initiator acting as receiver after the issue of an F-TRANSFER-END request.

If either the cancel data transfer service or the end of data transfer service is used, the data transfer regime ceases. Where use of these services collides, the cancel data transfer service takes precedence.

After an F-CANCEL procedure the two users may have different views of the state of the activity. The F-CANCEL primitives interrupt any activity in progress (including an F-RESTART sequence) and any undelivered indications or confirms may be discarded.

The file remains open after a sequence of F-CANCEL primitives, although the result of interrupted operations is not defined. Further F-READ or F-WRITE operations, not necessarily related to any previous read or write attempt, may be attempted after the completion of the sequence of F-CANCEL primitives has disposed of any previous activity.

24.6.2 Types of primitives and parameters

Table 38 indicates the types of primitives and the parameters needed to cancel data transfer.

Table 37 — F-TRANSFER-END parameters

Parameter	F-TRANSFER-END request	F-TRANSFER-END indication	F-TRANSFER-END response	F-TRANSFER-END confirm
Action Result			Mandatory	Mandatory
Shared ASE Information	Optional	Optional (=)	Optional	Optional (=)
Diagnostic			Optional	Optional

(=) signifies that the value is not modified by the service provider.

Table 38 — F-CANCEL parameters

Parameter	F-CANCEL request	F-CANCEL indication	F-CANCEL response	F-CANCEL confirm
Action Result	Mandatory	Mandatory	Mandatory	Mandatory
Shared ASE Information	Optional	Optional (=)	Optional	Optional (=)
Diagnostic	Optional	Optional	Optional	Optional

(=) signifies that the value is not modified by the service provider.

24.6.2.1 Action Result

The action result parameter, defined in 13.2, and the diagnostic parameter are used to convey the reason for the cancel.

24.6.2.2 Shared ASE Information

The shared ASE information parameter is defined in 13.10.

24.6.2.3 Diagnostic

The diagnostic parameter is defined in 13.13.

24.7 Sequence of primitives on write

The sequence of events in a successful write operation is defined in the time sequence diagram in figure 9. The F-DATA primitives in the diagram stand for an arbitrary sequence of F-DATA primitives in the internal service.

24.8 Sequence of primitives on read

The sequence of events in a successful read operation is defined in the time sequence diagram in figure 10. The F-DATA primitives in the diagram stand for an arbitrary sequence of F-DATA primitives in the internal service.

25 Checkpointing and restart (Internal BDT Service Only)

The checkpointing service is only available if the restart or/and recovery functional units are negotiated on F-INITIALIZE. The restart service is only available if the restart functional unit is negotiated on F-INITIALIZE. The checkpointing and restart primitives may be issued while a data transfer activity is in progress (after an F-READ or F-WRITE and before an F-DATA-END, for the sender and F-TRANSFER-END for the receiver) to control the progress of the data transfer.

NOTE - The receipt of a restart indication after the issue of an F-DATA-END may result in the sender returning to the data transfer activity it has just left.

25.1 Checkpointing service

Checkpoint insertion is only visible in the internal service. The protocol error recovery procedures (see ISO 8571-4) specify how the primitives are used in supporting the external service.

25.1.1 Function

The F-CHECK group of primitives mark and acknowledge points in a sequence of F-DATA primitives. The receipt of the F-CHECK confirm informs the sender of data, that data before the point marked and acknowledged has been received and secured, such that no earlier point is requested in an F-RESTART primitive. The sender may continue to send F-DATA primitives or further checkpoints before receipt of any particular primitive has been acknowledged. The number of checkpoints which may be outstanding is negotiated when the FTAM regime in which the bulk data is to be transferred is established. However, all outstanding checkpoints are acknowledged before the F-TRANSFER-END request or response is issued.

25.1.2 Types of primitives and parameters

Table 39 indicates the types of primitives and parameters needed for checkpointing.

25.1.2.1 Checkpoint Identifier

The checkpoint identifier parameter is defined in 23.2.

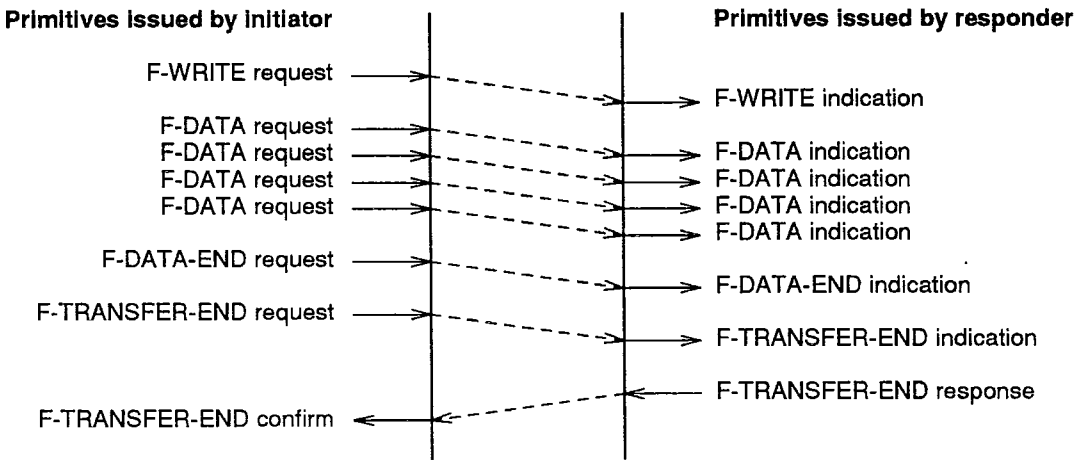


Figure 9 — Sequence of primitives on write

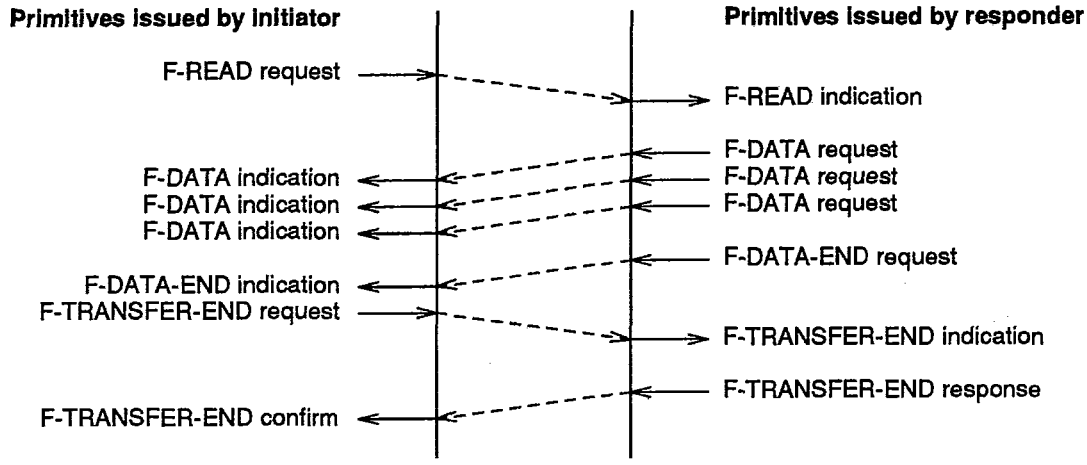


Figure 10 — Sequence of primitives on read

Table 39 — F-CHECK parameters

Parameter	F-CHECK request	F-CHECK indication	F-CHECK response	F-CHECK confirm
Checkpoint Identifier	Mandatory	Mandatory (=)	Mandatory	Mandatory (=)

(=) signifies that the value is not modified by the service provider.

25.2 Restarting data transfer service

Restarting of the data transfer is only visible in the internal service. The protocol error recovery procedures (see ISO 8571-4) specify how the primitives are to be used in supporting the external service.

25.2.1 Function

The F-RESTART group of primitives interrupts any bulk data transfer activity in progress, with possible loss of any undelivered indications or confirms. It negotiates a point at which data transfer is to be restarted. At the end of data transfer, it may not be issued:

a) by the initiator acting as sender after issue of an F-DATA-END request;

b) by the responder acting as receiver after issue of an F-TRANSFER-END response;

c) by the responder acting as sender after the issue of an F-DATA-END request;

d) by the initiator acting as receiver after the issue of an F-TRANSFER-END request.

If the receiver and the sender of data both issue an F-RESTART request, the service provider resolves the collision and issues a confirmation to each user giving the point at which the transfer is to be restarted (see ISO 8571-4). An F-RESTART indication is rejected by using an F-CANCEL request primitive.

25.2.2 Types of primitives and parameters

Table 40 indicates the types of primitives and the parameters needed to restart data transfer.

25.2.2.1 Checkpoint Identifier
The checkpoint identifier parameter is defined in 23.2. The use of the parameter depends on the direction of transfer. The issuer of the request identifies a checkpoint which is:

a) for the sender, the last point acknowledged;

b) for the receiver, the last point received and secured.

The issuer of the response identifies a checkpoint which is:

c) for the sender, equal to the value provided by the issuer of the request;

d) for the receiver, the last point received and secured.

In a collision, the value provided by the receiver is returned to both the sender and the receiver.

Table 40 — F-RESTART parameters

Parameter	F-RESTART request	F-RESTART indication	F-RESTART response	F-RESTART confirm
Checkpoint Identifier	Mandatory	Mandatory (=)	Mandatory	Mandatory (=)

(=) signifies that the value is not modified by the service provider.

Annex A
Diagnostic parameter values

(This annex forms part of the standard.)

A.1 Introduction

This annex defines the components which form the diagnostic parameter. It details for each component the valid values and meaning of those values.

A.2 Form of the diagnostic parameter

The value of the diagnostic parameter is structured into the following items:

a) an error type (see A.3), indicating "permanent error" or "transient error" or "informative". A permanent error occurs every time the sequence of events is repeated, and implies the failure of at least the present operation being performed. A transient error may not re-occur if the sequence is repeated but does imply the failure of the operation being performed. An informative error does not require recovery and does not affect the current state of the file service;

b) an error identifier (see A.4); the error identifiers categorize errors in terms of concepts defined in the virtual filestore definition or in terms of ISO 7498.

c) the type of entity which detected the error, called the observer (see A.5).

d) the type of entity which is believed to have caused the error called the presumed source (see A.5).

e) for transient errors, optionally, a suggested delay before recovery is attempted. The value is an integer *x* which indicates a delay time of 2 to power *x* seconds.

f) optionally, a text message in natural language giving further details of the cause of the error; it may include unstandardized concepts relating to the local system environment of the filestore provider. The values are of the type GraphicString.

A.3 Error type

Table 41 defines error type values for use in the diagnostic parameter. The values in the table identify classes of error of different severity.

A.4 Error identifiers

The diagnostics are classified into related groups and numbered accordingly.

A.5 Observer and source

The observer of the error and the presumed source are indicated from the categories defined in table 42.

A.6 Applicability of the diagnostic parameter

The diagnostic parameter occurs on the following services:

F-INITIALIZE
F-U-ABORT
F-P-ABORT
F-SELECT
F-DESELECT
F-CREATE
F-DELETE
F-READ-ATTRIB
F-CHANGE-ATTRIB
F-OPEN
F-CLOSE
F-RECOVER
F-LOCATE
F-ERASE
F-DATA-END
F-TRANSFER-END
F-CANCEL

Table 41 — Error types

Error type value	Error type
0	informative
1	transient error
2	permanent error

Table 42 — Sources and observers of errors

Observer and source identifier	Description	Qualification
0	no categorization possible	source only
1	initiating file service user	
2	initiating file protocol machine	
3	service supporting the file protocol machines	source only
4	the responding file protocol machine	
5	the responding file service user (filestore)	

Table 43 — General FTAM diagnostics

Type	Identifier	Observer	Source	Reason
12	0	0-5	0-5	No reason
012	1	5	5	Responder error (unspecific)
12	2	0-5	0-5	System shutdown
012	3	5	1	FTAM management problem (unspecific)
0 2	4	5	1	FTAM management, bad account
0 2	5	5	1	FTAM management, security not passed
0	6	5	5	Delay may be encountered
012	7	1,5	1	Initiator error (unspecific)
012	8	0-5	0-5	Subsequent error
012	9	0-5	0-5	Temporal insufficiency of resources
12	10	5	1	Access request violates VFS security
12	11	5	1	Access request violates local security

Table 44 — Protocol and supporting service related diagnostics

Type	Identifier	Observer	Source	Reason
2	1000	2	4	Conflicting parameter values
		4	2	"
2	1001	2	4	Unsupported parameter values
		4	2	"
2	1002	2	4	Mandatory parameter not set
		4	2	"
2	1003	2	4	Unsupported parameter
		4	2	"
2	1004	2	4	Duplicated parameter
		4	2	"
2	1005	2	4	Illegal parameter type
		4	2	"
2	1006	2	4	Unsupported parameter types
		4	2	"
2	1007	2	4	FTAM protocol error (unspecific)
		4	2	"
2	1008	2	4	FTAM protocol error, procedure error
		4	2	"
2	1009	2	4	FTAM protocol error, functional unit error
		4	2	"
2	1010	2	4	FTAM protocol error, corruption error
		4	2	"
2	1011	2-4	3	Lower layer failure
12	1012	3	2	Lower layer addressing error
12	1013	0-5	0-5	Timeout
12	1014	0-5	0-5	System shutdown
2	1015	2	4	Illegal grouping sequence
		4	2	"
2	1016	2	4	Grouping threshold violation
		4	2	"
2	1017	4	2	Specific PDU request inconsistent with the current requested access

Table 45 — Association related diagnostics

Type	Identifier	Observer	Source	Reason
2	2000	3,5	1	Association with user not allowed
	2001			(not assigned)
2	2002	4,5	1	Unsupported service class
0 2	2003	4,5	1	Unsupported functional unit
012	2004	5	1	Attribute group error (unspecific)
2	2005	5	1	Attribute group not supported
0 2	2006	5	1	Attribute group not allowed
0 2	2007	5	1	Bad account
012	2008	4	1	Association management (unspecific)
2	2009	4	1	Association management - bad address
12	2010	4	1	Association management - bad account
0 2	2011	4	2	Checkpoint window error - too large
0 2	2012	4	2	Checkpoint window error - too small
0 2	2013	4	2	Checkpoint window error - unsupported
012	2014	3	2	Communications QoS not supported
2	2015	4,5	1	Initiator identity unacceptable
0	2016	4	1	Context management refused
0	2017	4	1	Rollback not available
0	2018	5	1	Contents type list cut by responder
0	2019	3,5	1	Contents type list by Presentation service
2	2020	5	1	Invalid filestore password
2	2021	4,5	1	Incompatible service classes

Table 46 — Selection related diagnostics

Type	Identifier	Observer	Source	Reason
12	3000	5	5	Filename not found
12	3001	5	5	Selection attributes not matched
2	3002	5	5	Initial attributes not possible
2	3003	4	2	Bad attribute name
12	3004	5	5	Non-existent file
12	3005	5	5	File already exists
12	3006	5	5	File cannot be created
12	3007	5	5	File can not be deleted
0 2	3008	5	5	Concurrency control not available
0 2	3009	5	5	Concurrency control not supported
0 2	3010	5	5	Concurrency control not possible
01	3011	5	5	More restrictive lock
12	3012	5	5	File busy
12	3013	5	5	File not available
012	3014	5	5	Access control not available
012	3015	5	5	Access control not supported
012	3016	5	5	Access control inconsistent
0	3017	4,5	4,5	Filename truncated
0	3018	5	5	Initial attributes altered
12	3019	5	1	Bad account
0	3020	5	5	Override selected existing file
0	3021	5	5	Override deleted and recreated file with old attributes
0	3022	5	5	Create override deleted and recreate file with new attributes
12	3023	5	5	Create override - not possible
12	3024	5	5	Ambiguous file specification
2	3025	5	1	Invalid create password
2	3026	5	1	Invalid delete password on override
2	3027	5	1	Bad attribute value
2	3028	5	1	Requested access violates permitted actions
2	3029	5	1	Functional unit not available for requested access
01	3030	5	5	File created but not selected

Table 47 — File management related diagnostics

Type	Identifier	Observer	Source	Reason
012	4000	5	5	Attribute non - existent
12	4001	5	5	Attribute cannot be read
12	4002	5	5	Attribute cannot be changed
12	4003	4,5	4,5	Attribute not supported
2	4004	4	2	Bad attribute name
2	4005	5	1	Bad attribute value
0	4006	5	5	Attribute partially supported
0	4007	5	5	Additional set attribute value not distinct

Table 48 — Access related diagnostics

Type	Identifier	Observer	Source	Reason
12	5000	5	1	Bad FADU (unspecific)
2	5001	5	1	Bad FADU - size error
2	5002	5	1	Bad FADU - type error
2	5003	5	1	Bad FADU - poorly specified
2	5004	5	1	Bad FADU - bad location
01	5005	5	5	FADU does not exist
012	5006	5	5	FADU not available (unspecific)
12	5007	5	5	FADU not available for reading
12	5008	5	5	FADU not available for writing
12	5009	5	5	FADU not available for location
12	5010	5	5	FADU not available for erasure
12	5011	5	5	FADU cannot be inserted
12	5012	5	5	FADU cannot be replaced
012	5013	5	5	FADU cannot be located
2	5014	5	1	Bad data element type
		3	2,4	"
12	5015	5	5	Operation not available
12	5016	5	5	Operation not supported
0 2	5017	5	1	Operation inconsistent
012	5018	5	5	Concurrency control not available
0 2	5019	5	5	Concurrency control not supported
0 2	5020	5	1	Concurrency control inconsistent
012	5021	5	5	Processing mode not available
0 2	5022	5	5	Processing mode not supported
0 2	5023	5	1	Processing mode inconsistent
0 2	5024	5	5	Access context not available
0 2	5025	5	5	Access context not supported
12	5026	5	5	Bad write (unspecific)
12	5027	5	5	Bad read (unspecific)
012	5028	5	5	Local failure (unspecific)
012	5029	5	5	Local failure - filespace exhausted
012	5030	5	5	Local failure - data corrupted
012	5031	5	5	Local failure - device failure
2	5032	5	5	Future file size exceeded
0	5034	5	5	Future file size increased
0 2	5035	5	1	Functional unit invalid in processing mode
0 2	5036	5	1	Contents type inconsistent
0	5037	1	5	Contents type simplified
0	5038	5	5	Duplicate FADU name
12	5039	2	4	Damage to select/open regime
		4	2	"
12	5040	5	5	FADU locking not available on file
12	5041	5	5	FADU locked by another user

Table 49 — Recovery related diagnostics

Type	Identifier	Observer	Source	Reason
2	6000	4	2	Bad checkpoint (unspecific)
		2	4	"
2	6001	4	2	Activity not unique
2	6002	4	2	Checkpoint outside window
		2	4	"
2	6003	4	2	Activity no longer exists
2	6004	4	2	Activity not recognized
2	6005	4	4	No docket
2	6006	4	4	Corrupt docket
12	6007	4	4	File waiting restart
012	6008	4	2	Bad recovery point
2	6009	4	2	Non-existent recovery point
0 2	6010	4,5	2,1	Recovery mode not available
0 2	6011	4,5	2,1	Recovery mode inconsistent
0	6012	4,5	2,1	Recovery mode reduced
0 2	6013	5	1	Access control not available
0 2	6014	5	1	Access control not supported
0 2	6015	5	1	Access control inconsistent
0 2	6016	5	1	Contents type inconsistent
0	6017	5	1	Contents type simplified

Annex B
Relation of attributes to primitives

(This annex forms part of the standard.)

Tables 50 and 51 indicate which of the attributes defined in the virtual filestore are affected by the various primitives defined in the file service.

Primitives which do not occur in the tables 50 and 51 do not affect the virtual filestore attributes directly. However, the deselect primitive restores a previous value of the current account attribute.

Each entry in the table indicates the resultant effect of a particular primitive or group of primitives on one attribute. The table entries are:

a) blank; the attribute is not affected;

b) return; the attribute value is returned on the response and confirm primitives;

c) change; the attribute value is changed on the basis of a value transferred by the request and indication primitives; the value of a scalar attribute is replaced; the value of a vector attribute must be replaced completely; for set attributes individual elements may be changed; for creation, the initial value is assigned;

d) implicit; the attribute value is changed as a result of the use of the primitives, but no value is transferred; the new value results from the actions performed by the primitives;

e) set; the attribute value is set on the basis of a value transferred on a parameter of the request and indication primitives of F-CREATE, or to a local default if there is no value transferred;

f) compare; a value is transferred with the request and indication primitives, and the actions specified for the primitives are performed only if the given value matches the attribute value. The algorithm for matching values is part of the attribute definition.

Table 50 — File attributes

Attribute Name	F-SELECT	F-CREATE	F-READ -ATTRIB	F-CHANGE -ATTRIB	F-CLOSE & F-ABORT
filename	compare & return	set & return	return	change & return	
permitted actions	compare	set	return		
contents type		set	return		
storage account	compare	set	return	change & return	
date and time of creation		implicit	return		
date and time of last modification		implicit	return		implicit
date and time of last read access		implicit	return		implicit
date and time of last attribute modification		implicit	return	implicit	
identity of creator		set	return		
identity of last modifier		implicit	return		implicit
identity of last reader		implicit	return		implicit
identity of last attribute modifier		implicit	return	implicit	
file availability		set	return	change & return	
filesize		implicit	return		implicit
future filesize		set	return	change & return	
access control	compare	change	return (see note)	change & return (see note)	
legal qualifications		set	return	change & return	
private use		set	return	change & return	

NOTE - Any password values should not be returned

Table 51 — Activity attributes

Attribute Name	F-INITIALIZE	F-SELECT	F-CREATE	F-OPEN	F-READ & F-WRITE	F-LOCATE & F-ERASE	F-RECOVER
current access request		change	change	compare			change
current initiator identity	change						
current location				change	change	change	change
current processing mode				change			change
current calling application entity title	change						
current responding application entity title	change						
current account	change	change	change				
current concurrency control		change	change	change			
current locking style				change			change
current access passwords		change	change				change
active contents type				change			change
active legal qualifications				change			change

Annex C
File transfer with commitment control

(This annex forms part of the standard.)

C.1 Status

This annex defines mappings to the shared ASE information parameter of the commitment service for the file transfer service class. No general mapping is defined for the other file service classes.

This annex does not represent all possible or reasonable symbioses between FTAM and CCR application service elements. It is intended that, as the application layer structure definition matures, this annex will be expressed only in architectural terms of the relationships between FTAM and the properties of other application service elements which make up the general application entity.

ISO 8571 may be used in such a way that commitment and rollback mechanisms are not available, but if the services of ISO 9804 (CCR) are required to provide commitment mechanisms for FTAM activities, then each CCR primitive is issued only with FTAM primitives which carry a "shared ASE information" parameter. (This parameter is able to convey the full semantics and parameters of the permitted CCR primitives). Tables 52 and 53 list the FTAM primitives with a concurrency control parameter, and list the CCR primitives which can be issued with those FTAM primitives.

If a sequence of file service activities is to be considered as atomic, or if a file service activity is to be combined with other activities to form an atomic action, the service elements involved can act as a symbiotic ASEs to represent the start of the action, or for the commitment, or both. An atomic action begun in one file service activity may terminate on a different one.

NOTES

1 The CCR global recovery mechanism are distinguished from the recovery mechanisms in the error recovery protocol which supports the external file service. File protocol recovery involves only the two file error recovery protocol machines, and is not visible to the users of the external file service. CCR recovery is visible to all the participants in the atomic action.

2 When CCR is not applied during the select phase there are no recovery mechanisms for failures which occur outside the open regime. Such failures cause release of concurrency controls. In the external file service the initiator initiates recovery from an open regime failure, either by the file error recovery protocol procedures, or by the CCR procedures if selected; concurrency controls remain in place until recovery occurs.

C.2 Commitment structure

The procedure for establishing distributed commitment to change consists of two phases, each delimited by an exchange of primitives. These are:

Phase I C-BEGIN (request/indication)
or
C-RESTART (request/indication) (response/confirm)
C-READY (request/indication) - positive response
C-REFUSE (request/indication) - negative response

Phase II C-PREPARE (request/indication)
C-COMMIT (request/indication) - success
C-COMMIT (response/confirm).
or
C-ROLLBACK (request/indication) - failure
C-ROLLBACK (response/confirm)

The commitment primitives carry parameters, which are defined in the commitment concurrency and recovery service (ISO 9804). To carry these parameters, the file service provides a single transparent file service parameter (the shared ASE information parameter) to allow the transfer of commitment control information in the abstract syntax defined in ISO 9804 and ISO 9805. When use of CCR is proposed in the F-INITIALIZE, this abstract syntax name is added to the presentation context requirements, and the use of CCR is accepted only if the presentation context is established.

C.3 Association with the external file service

The service primitives in the external file service which are able to be associated with the primitives in the commitment service are indicated in table 52. The parameters of the CCR service are carried in the file service shared ASE information parameters. When CCR primitives other than C-PREPARE, C-READY or C-REFUSE request are able to be issued, they are issued with the next possible FTAM primitive. A C-READY or C-REFUSE is issued with the next possible FTAM primitive following a C-PREPARE if they have not already been issued.

The semantics of the C-BEGIN request/indication (and the corresponding procedures) are applied before those of the associated FTAM service primitive. The semantics of all other CCR primitives except C-RESTART are applied after the corresponding FTAM primitive.

A C-RESTART request/indication is applied before the corresponding primitive. If this is ROLLBACK, the ROLLBACK takes effect before the FTAM response is generated. If this is COMMIT or ACTION, the FTAM state is not affected.

An FTAM virtual filestore is always a CCR subordinate. Where FTAM is used by some other Application Service Element (ASE), the application entity containing an FTAM responder may also contain both a CCR subordinate and a CCR superior. An FTAM responder never offers commitment if any FTAM action result in the atomic action has been unsuccessful.

The shared ASE information parameter may be null, or may carry the CCR semantics and parameters of an associated CCR primitive. The sequence of CCR primitives is policed by CCR not FTAM. If the use of CCR has not been negotiated the shared ASE information parameter does not contain CCR information.

Table 52 — File service primitives associated with CCR primitives

CCR Primitive	Associated FTAM Primitives
C-BEGIN request/indication	F-OPEN or F-SELECT or F-CREATE request/indication
C-READY request/indication	F-CLOSE or F-TRANSFER-END response/confirm with an action result of success.
C-REFUSE request/indication	F-OPEN, F-SELECT, F-CREATE, F-CLOSE, F-CANCEL or F-TRANSFER-END response/confirm with an action result other than success.
C-PREPARE request/indication	F-CLOSE, F-TRANSFER-END request/indication
C-COMMIT request/indication	F-DESELECT, F-CLOSE request/indication
C-COMMIT response/confirm	F-DESELECT, F-CLOSE response/confirm
C-ROLLBACK request/indication	F-DESELECT, F-CLOSE request/indication
C-ROLLBACK response/confirm	F-DESELECT, F-CLOSE response/confirm with an action result other than success
C-RESTART request/indication	F-U-ABORT with F-OPEN, F-CREATE, F-SELECT request/indication (see note 1).
C-RESTART response/confirm	F-OPEN, F-CREATE, F-SELECT response/confirm

NOTES

1 Restart; the external file service is supported by mechanisms which contain all errors after which communication can be expected to continue. The definition of mapping for C-RESTART is to allow for failure elsewhere in the commitment tree, and the restart of the action is seen by the external service provider as a new activity. Further qualifying information is signalled in the shared ASE information parameter conveying the C-RESTART of the F-OPEN for this new activity.

2 The value taken by the Shared ASE Information parameter to convey CCR information is defined in ISO 9804 (CCR); it can contain additional diagnostic information.

3 Use of the shared ASE information parameter on the F-CREATE or F-SELECT primitives, rather than the parameter on the F-OPEN primitive, allows the creation of a file and the writing of its contents to be controlled as a single atomic action.

4 On F-DESELECT the request and response primitives carry commitment control independently.

Table 53 — FTAM Primitives with Shared ASE Information parameters

FTAM Primitives	Permitted CCR Primitives
F-SELECT request/indication	C-BEGIN request/indication C-RESTART request/indication
F-SELECT response/confirm	C-REFUSE request/indication C-RESTART response/confirm
F-CREATE request/indication	C-BEGIN request/indication C-RESTART request/indication
F-CREATE response/confirm	C-REFUSE request/indication C-RESTART response/confirm
F-OPEN request/indication	C-BEGIN request/indication C-RESTART request/indication
F-OPEN response/confirm	C-RESTART response/confirm C-REFUSE request/indication
F-TRANSFER-END request/indication	C-PREPARE request/indication
F-TRANSFER-END response/confirm	C-READY request/indication C-REFUSE request/indication
F-CANCEL response/confirm (or request/indication issued by the responder)	C-REFUSE request/indication
F-CLOSE request/indication	C-PREPARE request/indication C-COMMIT request/indication C-ROLLBACK request/indication
F-CLOSE response/confirm	C-READY request/indication C-REFUSE request/indication C-COMMIT response/confirm C-ROLLBACK response/confirm
F-DESELECT request/indication	C-COMMIT request/indication C-ROLLBACK request/indication
F-DESELECT response/confirm	C-COMMIT response/confirm C-ROLLBACK response/confirm

C.4 Making composite FTAM actions atomic

Table 54 lists the file operations defined in ISO 8831 (Job Transfer and Manipulation), and defines how they are achieved as atomic actions.

Table 54 — Composite FTAM actions

Operation	Achieved by
Write Normal	F-CREATE with override set to delete and re-create old file if it exists, followed by a single FTAM activity.
Write Old	F-SELECT, followed by single FTAM activity for a replace action.
Write New	F-CREATE without override and a single FTAM activity.
Add	F-CREATE with override set to select, followed by a single FTAM activity.
Append	F-SELECT, followed by a single FTAM activity.
Read Copy	F-SELECT, followed by a single FTAM activity.
Read Move	F-SELECT, followed by a single FTAM activity carrying C-BEGIN, terminating in F-DELETE. The CCR atomic action is completed after the FTAM activity by separate CCR service primitives.

If a C-PREPARE is received within a select regime, then a C-READY with the requested access activity attribute allowing file deletion implies a commitment to delete the file if F-DELETE is subsequently issued with a C-COMMIT.

Annex D
Reference to FTAM control information

(This annex forms part of the standard.)

This annex defines terms which are used in other standards to reference the definitions of the parameter information necessary to specify a file transfer activity. When a distributed application requires the transfer of a request that a file transfer be performed, the standards concerned include the description of the transfer by using these terms to reference the file service standard.

Two types of terms are defined, giving respectively the information necessary to start a transfer, and the information available on the outcome of the transfer attempt. In each case, the information is the information content of a sequence of file service primitives.

The information content of a primitive is the identity of the primitive together with the parameters carried by that primitive. It differs from the primitive itself in being quoted outside the file context, and in being without inherent sequence constraints.

The read transfer specification is the information content of the F-SELECT request primitive, plus the information content of the F-OPEN request primitive, plus the information content of the F-READ request primitive.

The write transfer specification is the information content of the F-CREATE request primitive, plus the information content of the F-OPEN request primitive, plus the information content of the F-WRITE request primitive.

The read transfer outcome is the information content of the F-SELECT confirm primitive, plus the information content of the F-OPEN confirm primitive, plus the information content of the F-TRANSFER-END confirm primitive or F-CANCEL response or confirm primitive.

The write transfer outcome is the information content of the F-CREATE confirm primitive, plus the information content of the F-OPEN confirm primitive, plus the information content of the F-TRANSFER-END confirm primitive or F-CANCEL response or confirm primitive.

NOTE - The failure of some primitives may prevent the issue of further primitives by implementations conforming to ISO 8571-4. In this case the outcome contains only the information from the primitive actually issued.

Annex E
State transition diagrams

(This annex forms part of the standard.)

The state transition descriptions in this annex define the sequences of service primitives in the external file service. The descriptions of the initiator and the responder are provided separately by state transition diagrams applicable at the initiating and responding file service access points.

For the purposes of these state transition diagrams, complete grouped sequences of primitives, bracketed by BEGIN-GROUP and END-GROUP primitives, constitute a single transition; the primitives in a grouped sequence are syntactic segments of a single communicated message.

Each of the file regime control state transition diagrams is divided into two parts. The transitions produced by single primitives are shown in the first diagram, and the transitions produced by grouped sequences are shown in the second diagram.

The grouped sequences are defined below, using the following notation:

a) Square brackets, "[" and "]", indicate optional primitives within a sequence;

b) Vertical bar, "|", indicates alternatives;

c) Round brackets, "(" and ")", have normal algebraic significance.

The valid grouped sequences, and the definition of the notation used in figures 14 and 16, are:

d) Group "A": F-BEGIN-GROUP
(F-SELECT|F-CREATE)
[F-READ-ATTRIB]
[F-CHANGE-ATTRIB]
F-OPEN F-END-GROUP

e) Group "B": F-BEGIN-GROUP F-CLOSE
[F-READ-ATTRIB]
[F-CHANGE-ATTRIB]
(F-DESELECT|F-DELETE)
F-END-GROUP

f) Group "C": F-BEGIN-GROUP
(F-SELECT|F-CREATE)
[F-READ-ATTRIB]
[F-CHANGE-ATTRIB]
(F-DESELECT|F-DELETE)
F-END-GROUP

g) Group "D": F-BEGIN-GROUP
(F-SELECT|F-CREATE)
[F-READ-ATTRIB]
[F-CHANGE-ATTRIB]
F-END-GROUP

h) Group "E": F-BEGIN-GROUP
[F-READ-ATTRIB]
[F-CHANGE-ATTRIB]
(F-DESELECT|F-DELETE)
F-END-GROUP

NOTE - In the file transfer service class, only groups "A" and "B" are valid. In the file management class only group "C" is valid. In the file transfer and management class only the groupings "A", "B" and "C" are valid. In these cases the threshold parameter is set so that the sequences either succeed or fail as a whole.

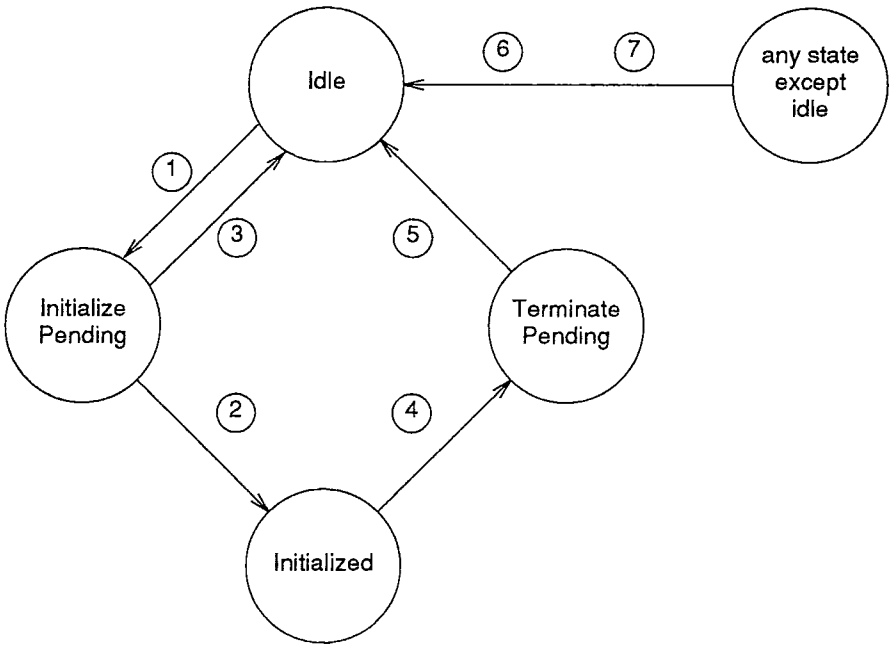


Figure 11 — State Transition Diagram for Association Establishment (Initiator)

Key — Transitions

1 F-INITIALIZE request
2 F-INITIALIZE confirm (positive)
3 F-INITIALIZE confirm (negative)
4 F-TERMINATE request
5 F-TERMINATE confirm
6 F-U-ABORT request or indication
7 F-P-ABORT indication

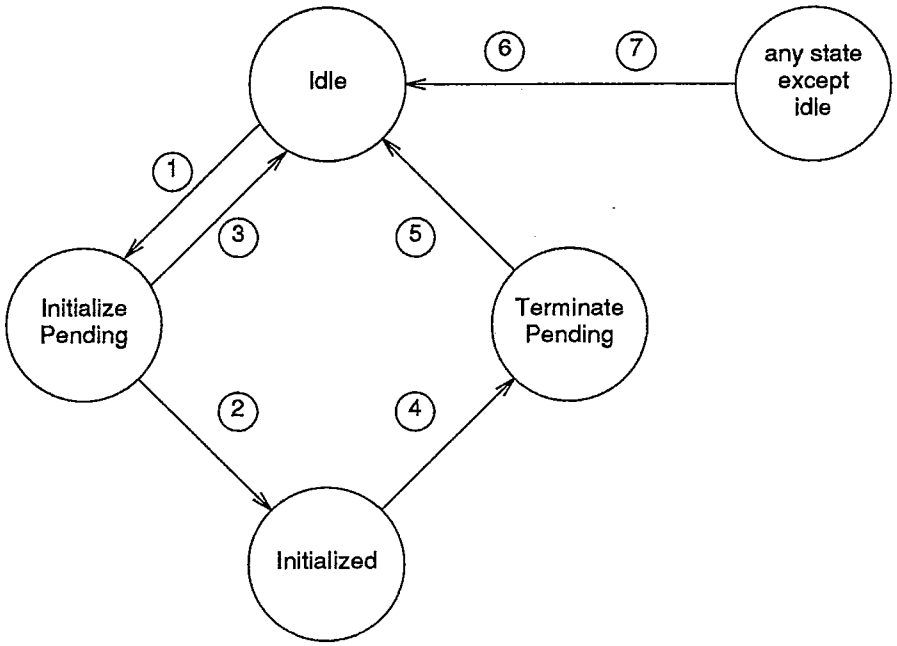


Figure 12 — State Transition Diagram for Association Establishment (Responder)

Key — Transitions

1 F-INITIALIZE indication
2 F-INITIALIZE response (positive)
3 F-INITIALIZE response (negative)
4 F-TERMINATE indication
5 F-TERMINATE response
6 F-U-ABORT request or indication
7 F-P-ABORT indication

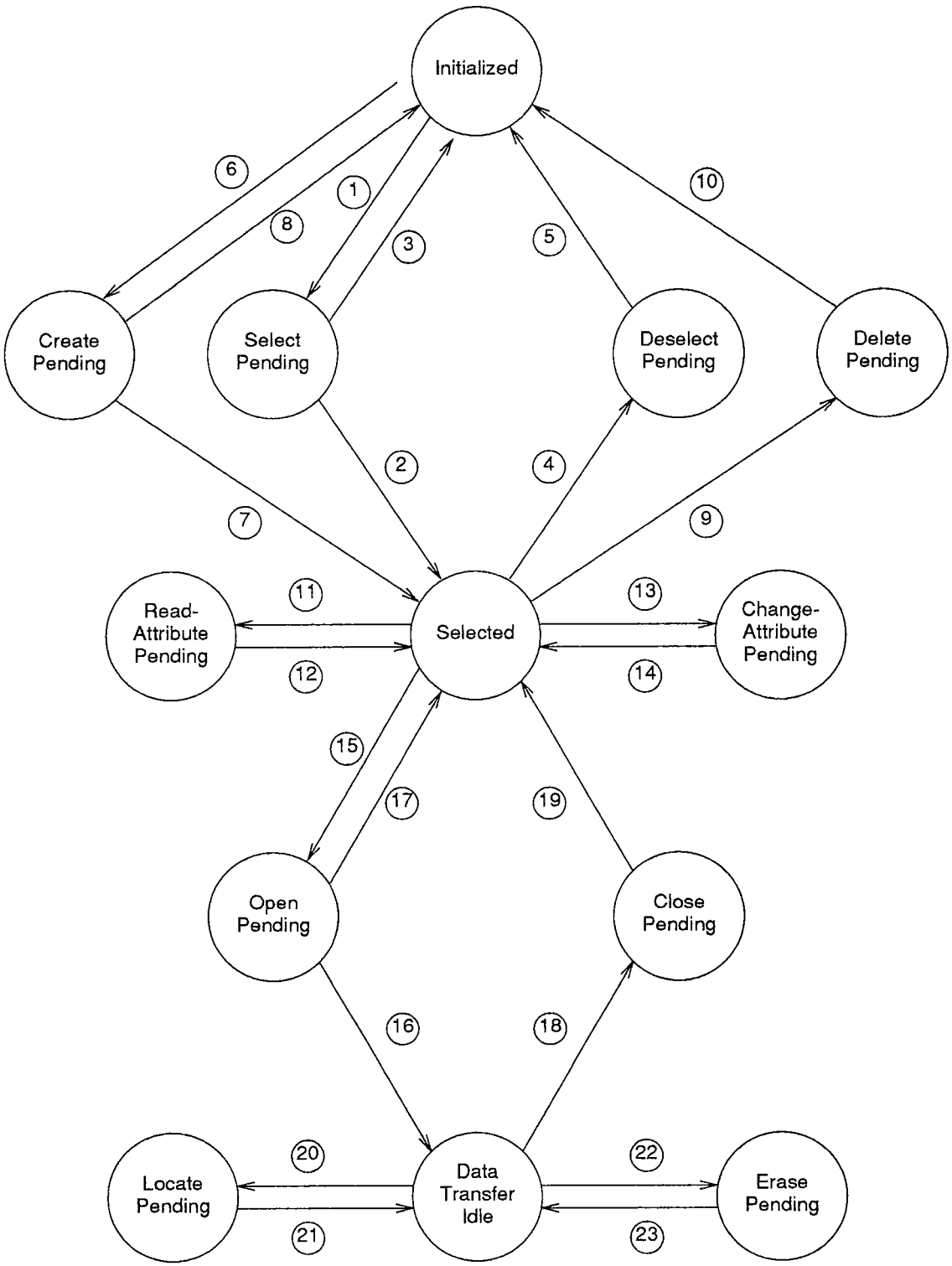


Figure 13 — State Transition Diagram of the File Regime Establishment Service (Initiator)

1 F-SELECT request
2 F-SELECT confirm (positive)
3 F-SELECT confirm (negative)
4 F-DESELECT request
5 F-DESELECT confirm
6 F-CREATE request
7 F-CREATE confirm (positive)
8 F-CREATE confirm (negative)
9 F-DELETE request
10 F-DELETE confirm
11 F-READ-ATTRIB request
12 F-READ-ATTRIB confirm
13 F-CHANGE-ATTRIB request
14 F-CHANGE-ATTRIB confirm
15 F-OPEN request
16 F-OPEN confirm (positive)
17 F-OPEN confirm (negative)
18 F-CLOSE request
19 F-CLOSE confirm
20 F-LOCATE request
21 F-LOCATE confirm
22 F-ERASE request
23 F-ERASE confirm

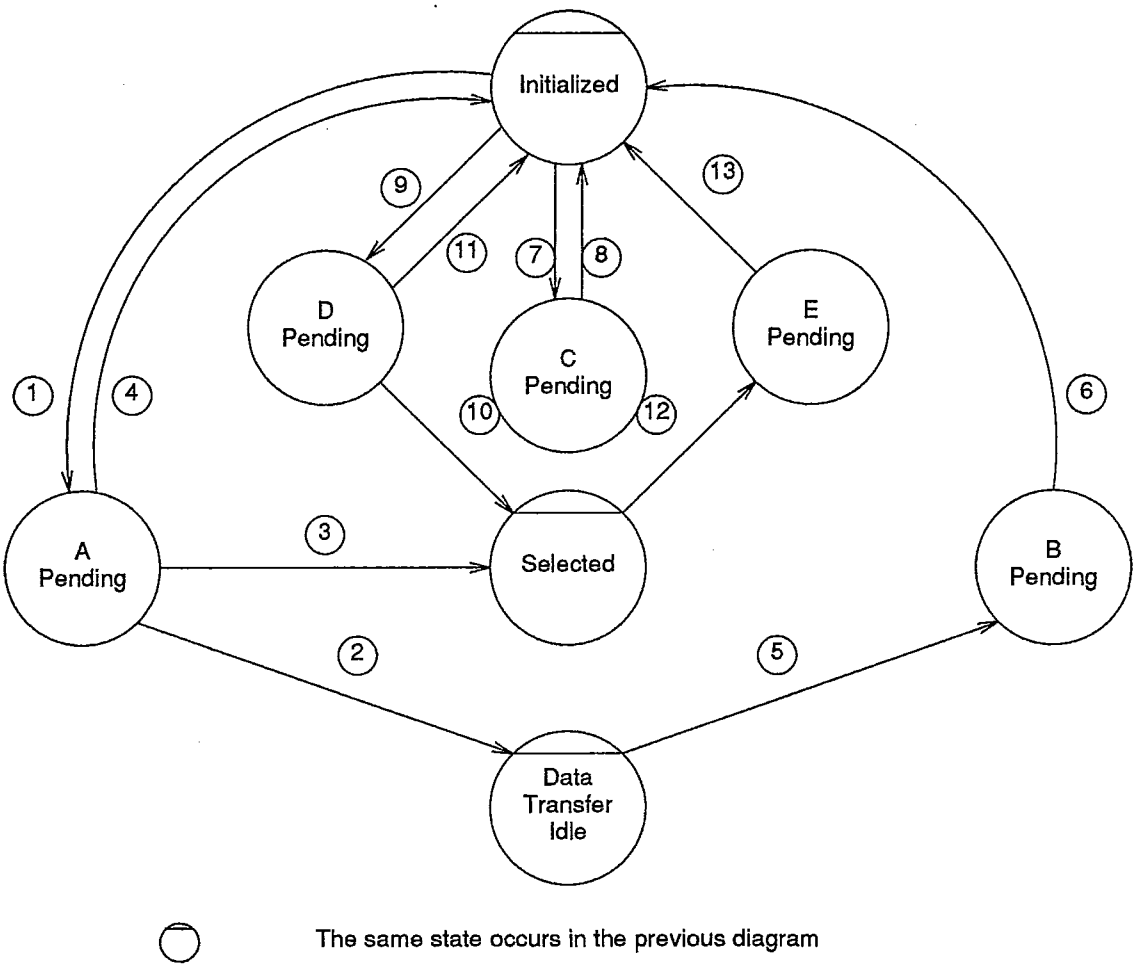


Figure 14 — State Transition Diagram for Grouped Sequences (Initiator)

Key — Transitions

1	"A" request
2	"A" confirm (positive)
3	"A" confirm (open negative)
4	"A" confirm (select negative)
5	"B" request
6	"B" confirm
7	"C" request
8	"C" confirm
9	"D" request
10	"D" confirm (positive)
11	"D" confirm (select negative)
12	"E" request
13	"E" confirm

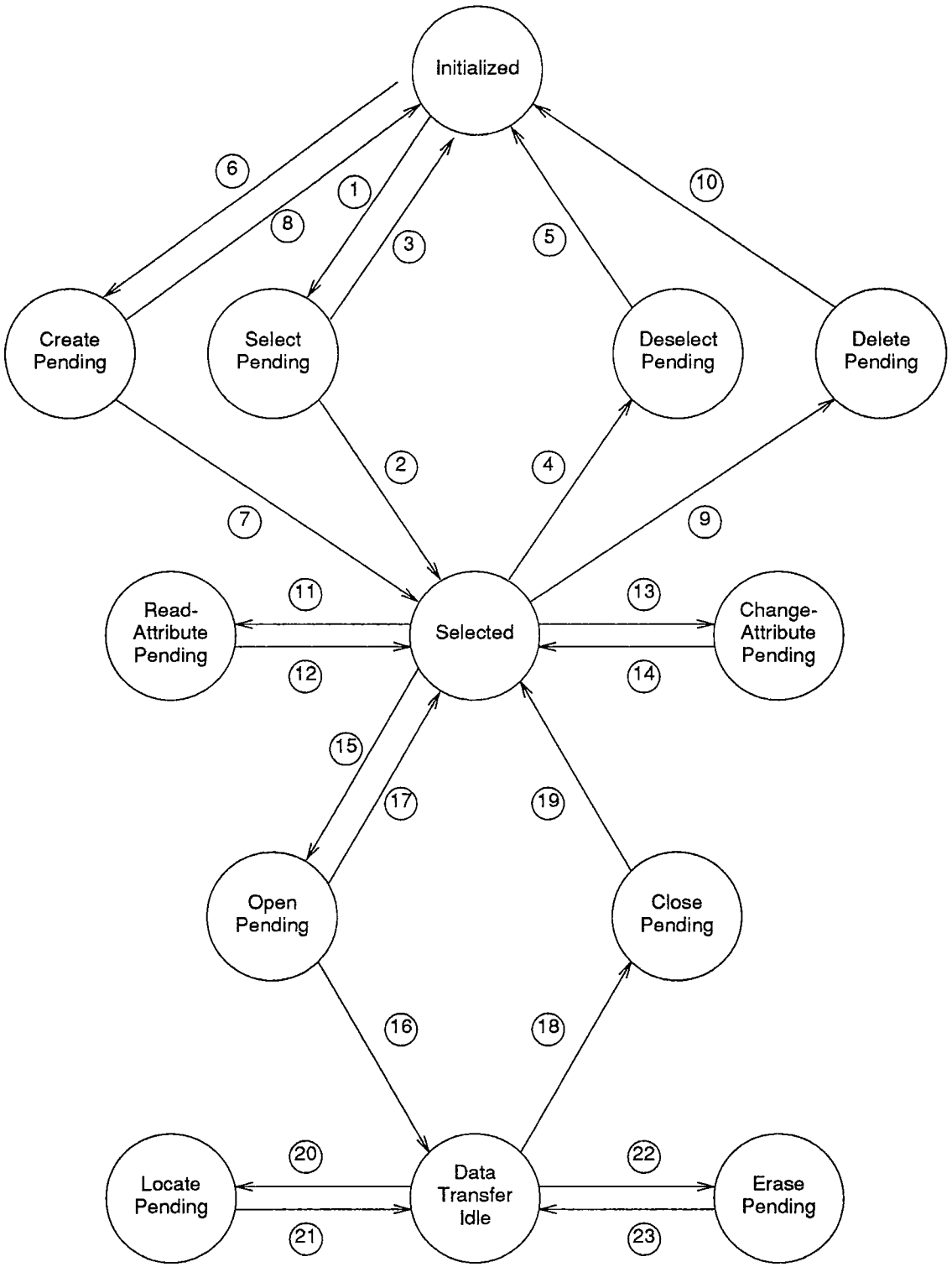


Figure 15 — State Transition Diagram of the File Regime Establishment Service (Responder)

1 F-SELECT indication
2 F-SELECT response (positive)
3 F-SELECT response (negative)
4 F-DESELECT indication
5 F-DESELECT response
6 F-CREATE indication
7 F-CREATE response (positive)
8 F-CREATE response (negative)
9 F-DELETE indication
10 F-DELETE response
11 F-READ-ATTRIB indication
12 F-READ-ATTRIB response
13 F-CHANGE-ATTRIB indication
14 F-CHANGE-ATTRIB response
15 F-OPEN indication
16 F-OPEN response (positive)
17 F-OPEN response (negative)
18 F-CLOSE indication
19 F-CLOSE response
20 F-LOCATE indication
21 F-LOCATE response
22 F-ERASE indication
23 F-ERASE response

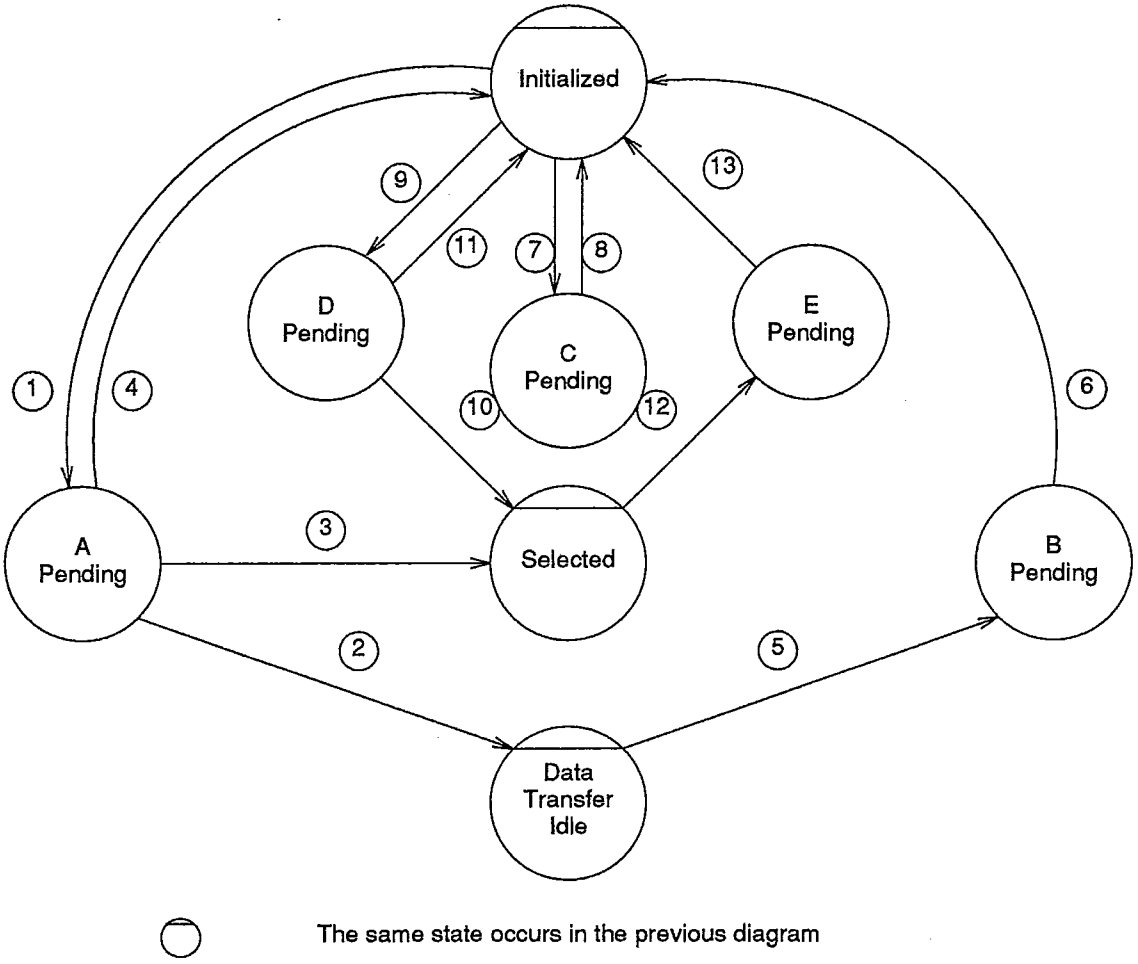


Figure 16 — State Transition Diagram for Grouped Sequences (Responder)

Key — Transitions

1	"A" indication
2	"A" response (positive)
3	"A" response (open negative)
4	"A" response (select negative)
5	"B" indication
6	"B" response
7	"C" indication
8	"C" response
9	"D" indication
10	"D" response (positive)
11	"D" response (select negative)
12	"E" indication
13	"E" response

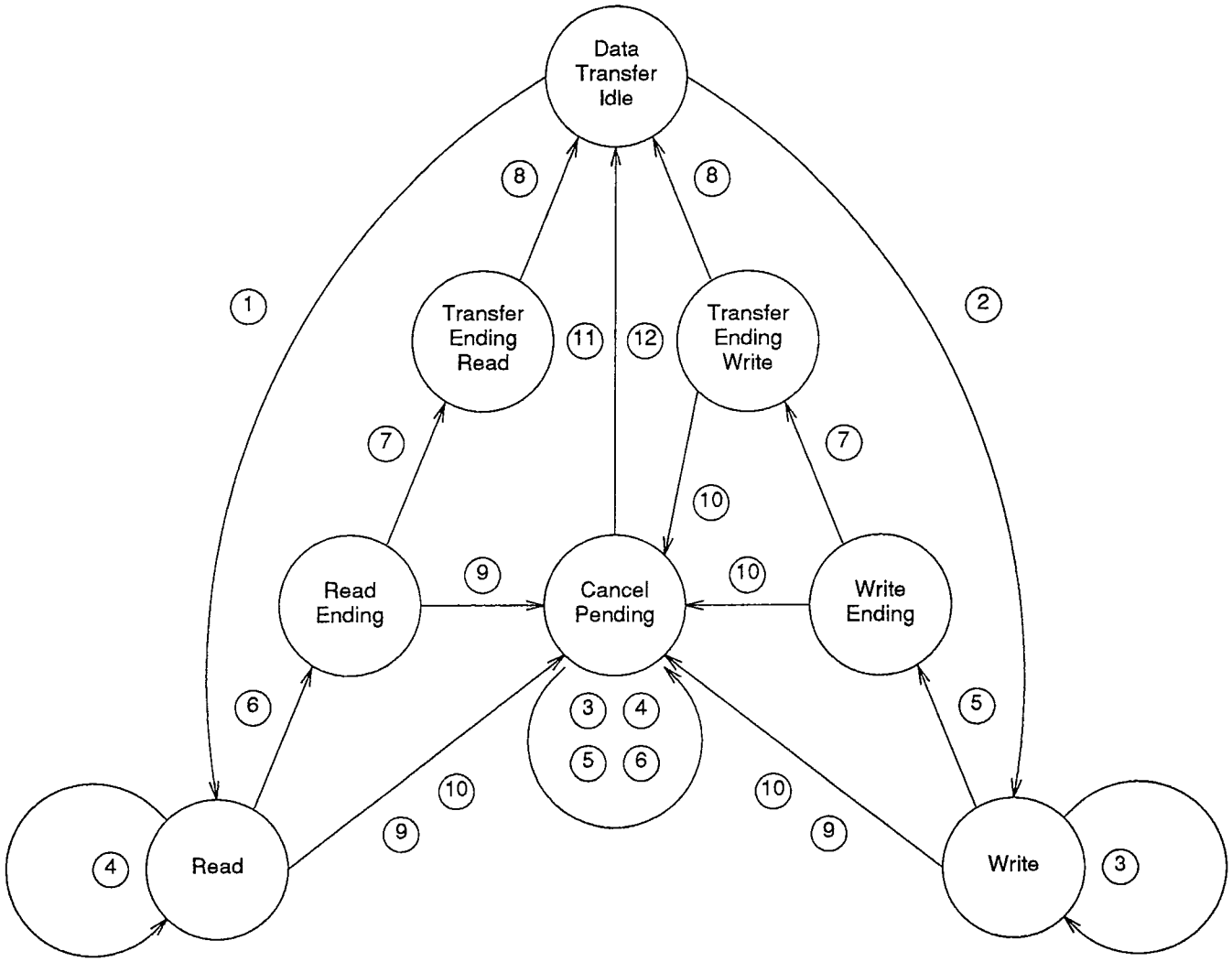


Figure 17 — State Transition Diagram for the Bulk Data Transfer Service (Initiator)

Key — Transitions

1	F-READ request
2	F-WRITE request
3	F-DATA request
4	F-DATA indication
5	F-DATA-END request
6	F-DATA-END indication
7	F-TRANSFER-END request
8	F-TRANSFER-END confirm
9	F-CANCEL request
10	F-CANCEL indication
11	F-CANCEL response
12	F-CANCEL confirm

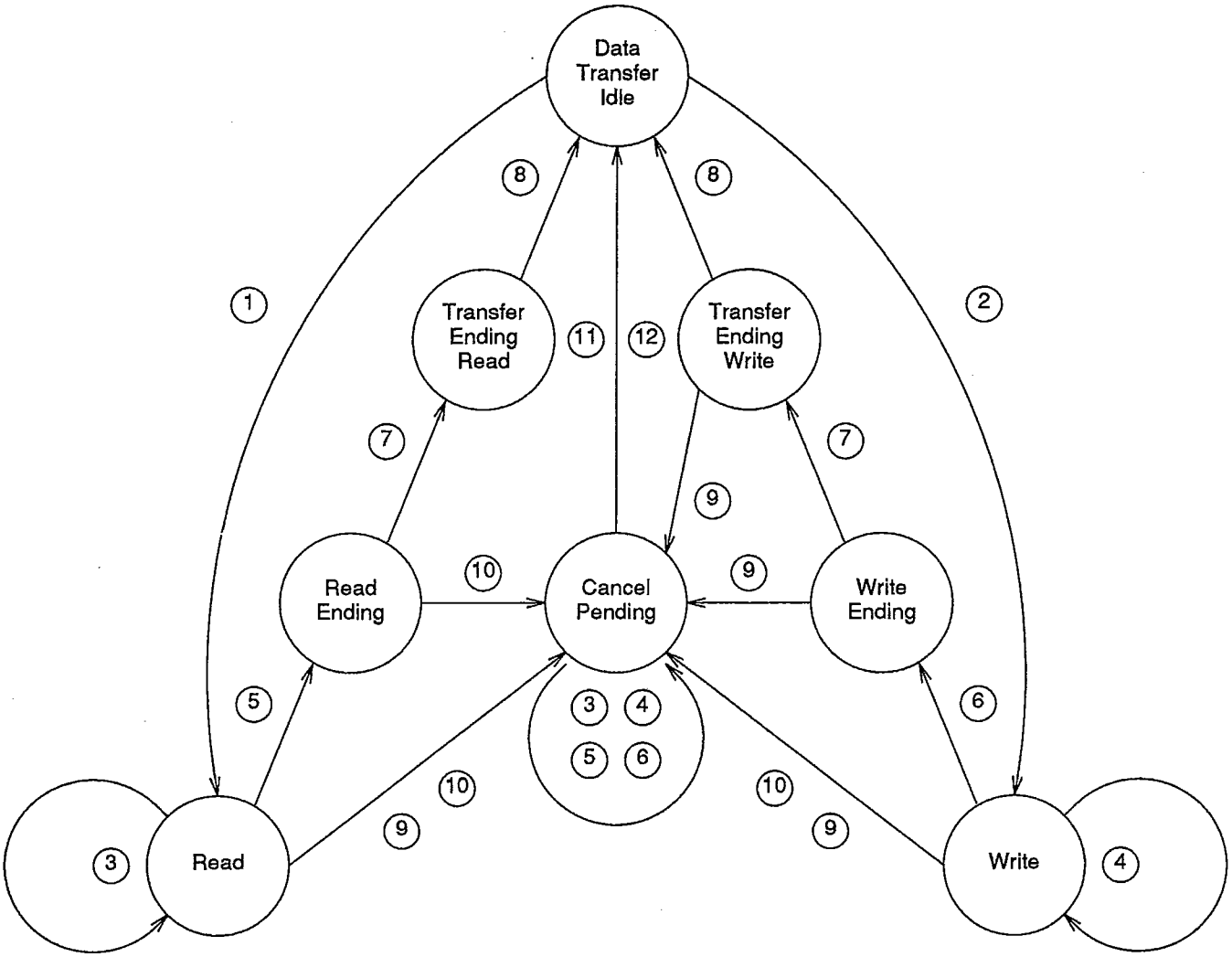


Figure 18 — State Transition Diagram for the Bulk Data Transfer Service (Responder)

Key — Transitions

1	F-READ indication
2	F-WRITE indication
3	F-DATA request
4	F-DATA indication
5	F-DATA-END request
6	F-DATA-END indication
7	F-TRANSFER-END indication
8	F-TRANSFER-END response
9	F-CANCEL request
10	F-CANCEL indication
11	F-CANCEL response
12	F-CANCEL confirm

UDC 681.3 : 621.39

Descriptors : data processing, information interchange, network interconnection, open systems interconnection, files, communication procedure, management.

Price based on 67 pages