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*

# 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

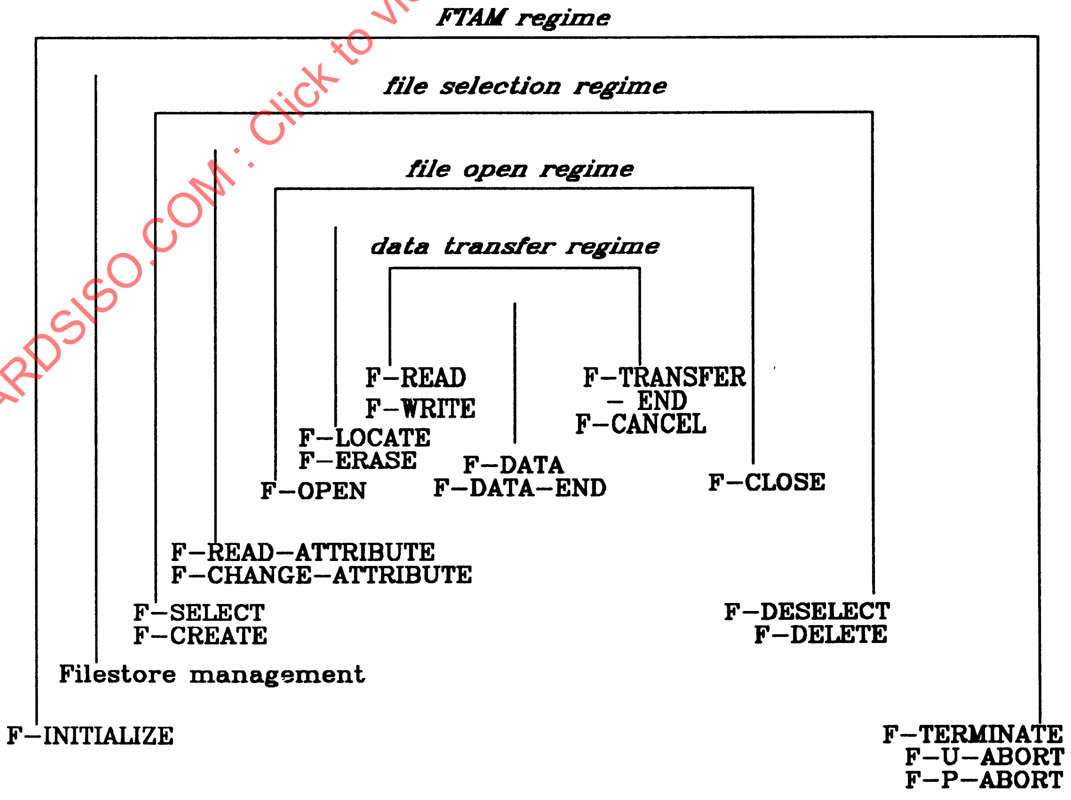


**Figure 2 - File service regimes and related primitivees**

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

### 16.1 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<br>Transfer number | F-DATA-END action result |
| F-WRITE | No | Initiator | Bulk data transfer specification<br>Transfer number | F-CANCEL |
| F-DATA | No | Sender | Data value | F-CANCEL |
| F-DATA-END | No | Sender | Action result<br>Diagnostic | F-CANCEL |
| F-TRANSFER-END | Yes | Initiator | Action result<br>Shared ASE information<br>Diagnostic<br>Request type<br>Transfer number<br>[Last transfer end read response]<br>[Last transfer end write response] | action result |
| F-CANCEL | Yes | Either | Action result<br>Shared ASE information<br>Diagnostic<br>Request type<br>Transfer number<br>[Last transfer end read request]<br>[Last transfer end read response]<br>[Last transfer end write request]<br>[Last transfer end write response] | none |
| [F-CHECK] | Yes | Either | [Checkpoint identifier<br>Transfer number] | F-CANCEL |
| [F-RESTART] | Yes | Either | [Checkpoint identifier<br>Request type<br>Transfer number<br>Last transfer end read request<br>Last transfer end read response<br>Last transfer end write request<br>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 |

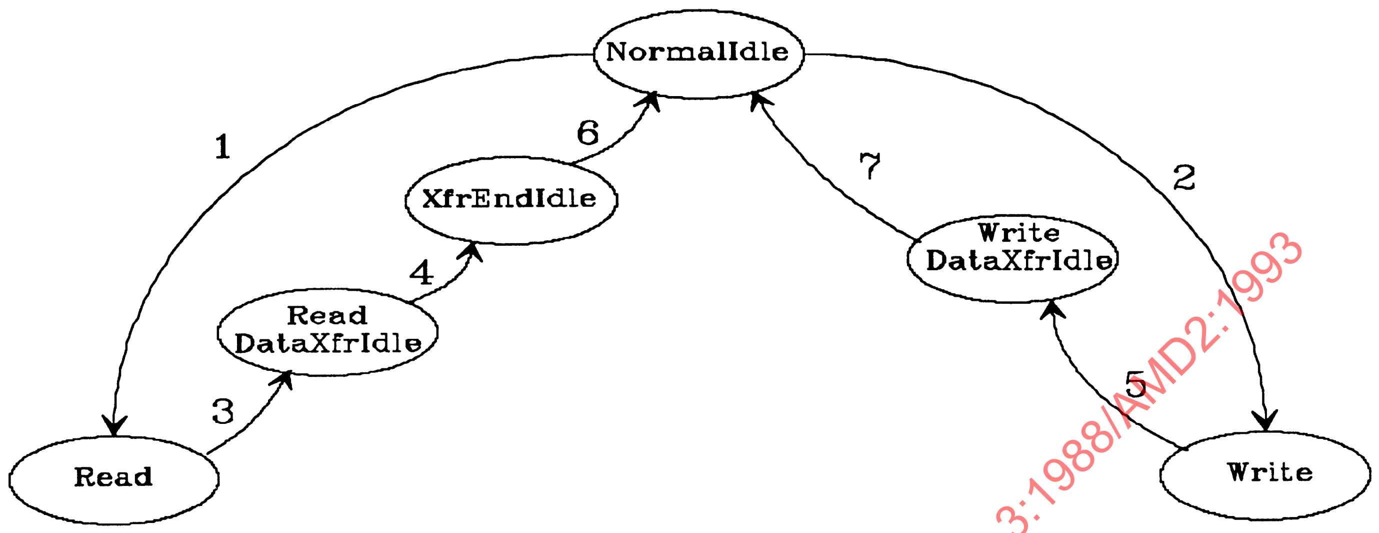
# Annex E
# State transition diagrams

**(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.



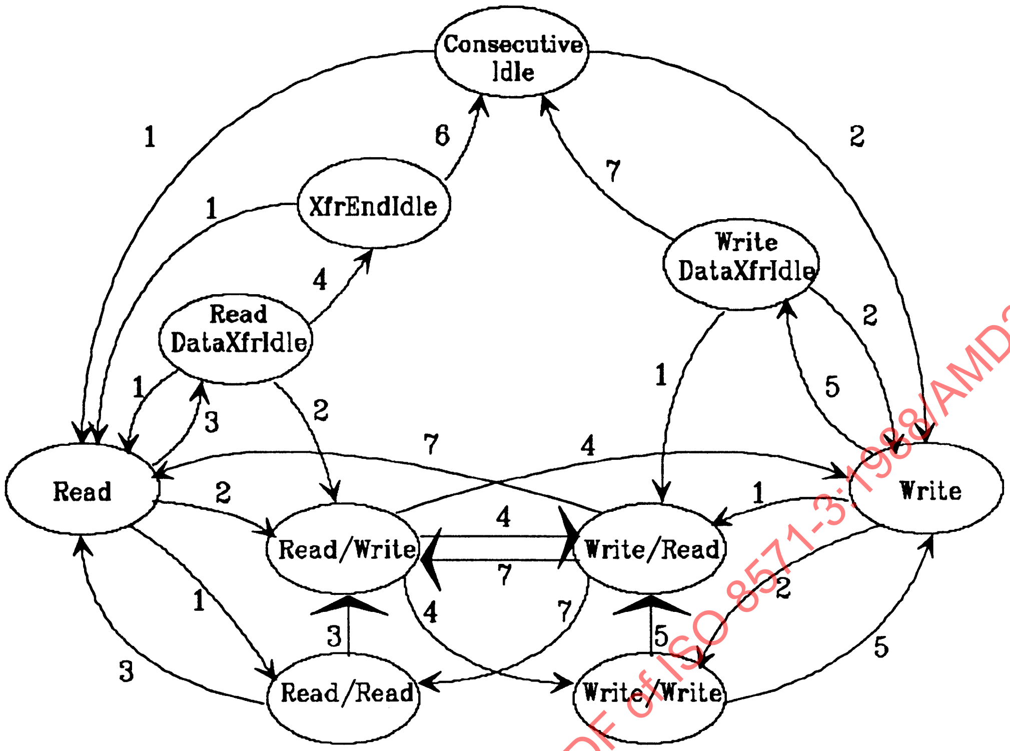
**Initiator**

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**



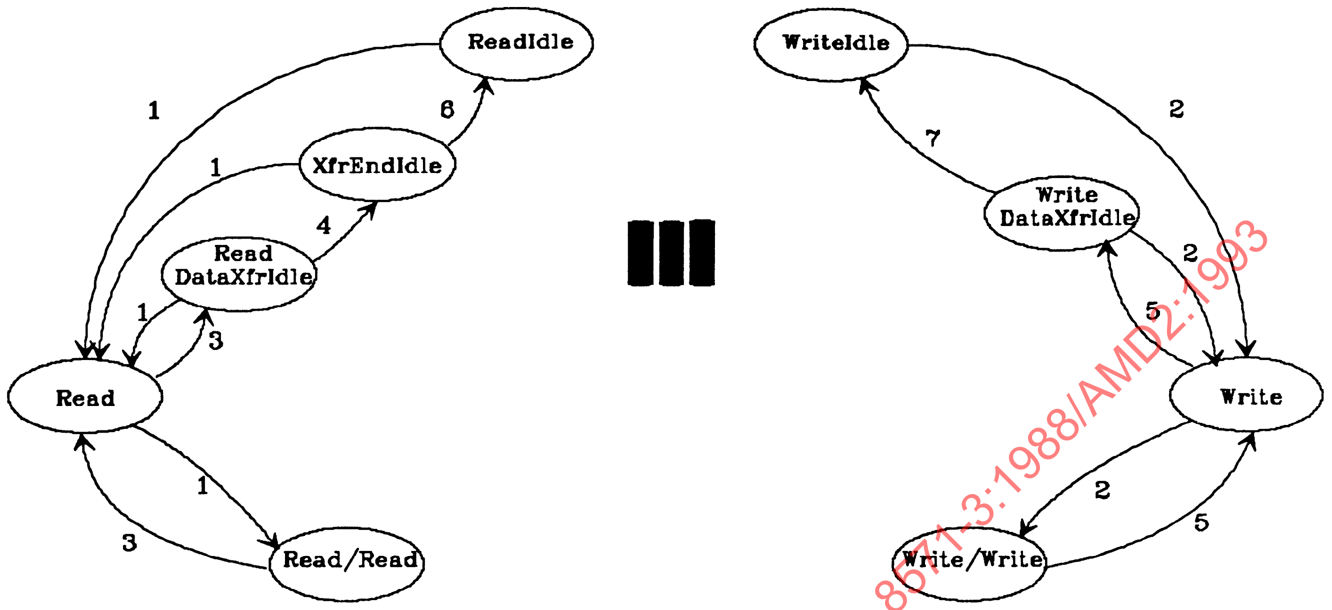
**Initiator**

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)

endproc (* ReadRead *)
```

**Figure 23 (continued) - Responder service definition**