Protocole d'Echanges pour un Système Interbancaire de Télécompensation

July 1989

CONTENTS

ANNEXE 1 COMPRESSION

ANNEXE 2 STORE AND FORWARD OPERATION

ANNEXE 3 USE OF THE SECURITY MECHANISMS

ANNEXE 4 ERROR DIAGNOSTICS

ANNEXE 5 SUMMARY OF THE PROTOCOL UNITS AND THEIR PARAMETERS

CHAPTER 1

INTRODUCTION

1 INTRODUCTION

1.1 Scope

The object of these specifications is to define the **PeSIT file transfer protocol.**

PeSIT was originally conceived for use within the French Interbank Electronic payment clearance system hence its name (**P**rotocol for data **E**xchange within the French **S**ystem for **I**nterbank **T**ele-clearance).

This protocol is used in particular to connect the Bank Processing Centres (CTB) owned by the members of the SIT network to the SIT gateways.

However the use of PeSIT is not limited to these connections and may indeed be used in the most varied environnements.

With the intention of enabling the diverse implementations of PeSIT to inter-operate, several PeSIT user profiles have been defined which correspond with the different distinct application domaines of PeSIT.

Four profiles (SIT, Non-SIT, Secure Non-SIT and ETEBAC5) are described in these specifications. Others may be defined if the need becomes felt though the proliferation of profiles is not encouraged.

1.2 Presentation

1.2.1 Outline

The **first chapter** presents the ***Introduction*** to this document. Having presented the outline, a second paragraph details the modifications introduced by the version E[1] compared to the version D of 15 november 1987.

The **second chapter** ***Architecture and functional description***, compares PeSIT with the ISO/OSI reference model, introduces the virtual file concept and summarizes the different functions provided by PeSIT. This chapter terminates with an introduction to the concepts of the functional unit and the profile.

The **third chapter** ***PeSIT service description*** commences with a presentation of the concept of service, lists the different service phases in PeSIT, followed by the different services which make up the PeSIT service. The functional units are defined as a collection of services and the concept of profile is further detailed. The paragraphs **Description of the service primitives** and **Description of the parameters** presents the service primitives one by one with their parameters, except for those directly related to the Security functional unit. The paragraph **Profile descriptions** details the characteristics of each profile. The paragraph **PeSIT security service** covers all aspects of the service linked with the Security functional unit (both the primitives and the parameters). The last paragraph **Examples of primitive sequences** presents the characteristic automates of the PeSIT service and gives some examples of primitive sequencing.

The **fourth chapter** ***Description of the PeSIT protocol*** describes the correspondance between the service primitives and the protocol units. The paragraph **Use of the communication system service** presents the different types of interface for PeSIT.F, PeSIT.F', PeSIT.F" and PeSIT.F". This is followed by the procedures corresponding with each protocol unit (FPDU). The succeding paragraphs give the rules for concatenating FPDUs and the use of the protocol time-outs. The paragraph **Structure and coding of PeSIT protocol units** presents the parameter coding conventions, lists the parameters and for each one schematizes its format for each of the four profiles. Following this the contents of each parameter is detailed for each profile. The last paragraph provides the automate state tables for the PeSIT protocol.

Annexe A defines the different compression modes allowed and the method used to negotiate the compression mode between two partners.
Annexe B defines how to use PeSIT in a Store and Forward mode.
Annexe C details the use of the security mechanisms (ETEBAC5 and Secure Non-SIT profiles).
Annexe D contains the list of the error diagnostics.
Annexe E summarizes the protocol units and their parameters.

[1] References in this document to previous versions of PeSIT are to the French versions (the version E is the first version translated into English).

1.2.2 Modifications introduced by the Version E

The concept of functional units and profiles has been added in the version E to clarify the presentation of the document. The increase in the number of parameters whose format is dependant on the profile has led to presenting each parameter on a separate page allowing its format and significance to be detailed for each profile. This modification means that the presentation of contents of the FPDUs has also been changed : for each profile the FPDUs are represented as a collection of parameters whose graphical representation defines their use (required, optional with or without a default value, conditional). However the contents of the parameters is not shown in the description of the FPDUs.

In detail the modifications are as follows :

SIT Profile :

The PeSIT protocol described in this document using the SIT profile is identical to the version D of 15 november 1987. All the special features of PeSIT SIT have been assembled in the paragraph 3.8.1 and can be found in the description of the parameters and the protocol units for the SIT profile. Some details which were absent from the version D of PeSIT, existing only in internal documents of the SIT project, have been included in the paragraph 3.8.1 when they were considered useful for programming a PeSIT product destined to be connected to the SIT network.

Non-SIT Profile :

The Non-SIT profile of the PeSIT protocol has seen the following modifications compared to the version D of 15 november 1987 :

the **password** parameter (PI 5) has been enlarged from 2 to 16 bytes and may be modified dynamically,

the **access type** parameter (PI 22) can have the value 2 : mixed access type,

the **use of the CRC** parameter (PI 1) has been added in the FPDU.CONNECT to allow PeSIT to be used with a PAD,

the file type and file name parameters (PI 11 and PI 12) contained in the FPDU.SELECT and FPDU.ACK(SELECT) may be different thus allowing **generic selection**,

the possibility of using PeSIT in a **Store and Forward** mode is described and two parameters have been added : initial caller identification and final server identification (PI 61 and PI 62),

the length of the **caller and server identification** parameters (PI 3 and PI 4) have been increased from 16 to 24 bytes,

the **transfer identifier** parameter becomes optional in the FPDU.ACK(CREATE),

two parameters have been added **key length** (PI 38) and **key offset** (PI 39) to allow the use of indexed file formats,

the **aléa** and **remainder parameters** (PI 24 and PI 35) have been deleted,

the length of the **free text** parameter (PI 99) has been increased to 254 bytes and becomes optional in the FPDU.ACK(CREATE),

the **file identifier** parameter (PGI 9) has been added to the FPDU.ACK(SELECT),

a **datagram** service has been introduced which uses some new FPDUs (FPDU.MSG, FPDU.MSGDM, FPDU.MSGMM, FPDU.MSGFM and FPDU.ACK(MSG)) and a new parameter **datagram**,

the use of **padding** in the vertical compression algorithm is clarified,

the behaviour of a file receiver during the transmission of an FPDU.TRANS.END or an FPDU.ACK(TRANS.END) is clarified,

the **version number** (PI 6) should be used to differentiate between versions of PeSIT Non-SIT in conformity with the version D and versions corresponding with the version E.

Securise Non-SIT Profile :

This is a **new profile**. It requires the use of a special set of parameters which have been introduced in the version E (parameters PI 71 to PI 83).

ETEBAC 5 Profile :

This is a **new profile**. It has been defined in collaboration with the ETEBAC 5 transport ans security work groups of the French **CFONB** (French comittee for Bank Organisation and Standardisation) to satisfy the file transfer requirements in conformity with the ETEBAC 5 standard (Data communication exchange protocol between Banks and their Customers). ETEBAC 5 is described in a document produced by the CFONB and available from the **GSIT**.

CHAPTER 2

ARCHITECTURE and FUNCTIONAL DESCRIPTION

2 ARCHITECTURE and FUNCTIONAL DESCRIPTION

2.1 Reference model

Two important basic principles appear in the PeSIT protocol :

* A stable interface with the applications.

* A unique protocol regardless of the different communications layers used.

The dispersed evolution of processing centers has led to the definition of four versions of PeSIT, which differ uniquely by their interface with the lower communication layers used.

- PeSIT-F relies on the standardised ISO Session layer for use with a packet switching network (X-25).

- PeSIT-F' relies on the standardised ISO Network layer (CCITT X-25 recommendation).

- PeSIT-F" relies on the NETEX layer developped with the use of Hyperchannel.

- PeSIT-F"' relies on the standardised ISO session layer for use with a local area network in conformity with the ISO 8802-3 specification.

The diagram on the following page describes the logical architectural environnement of PeSIT.

An application prepares and processes the transfered files.

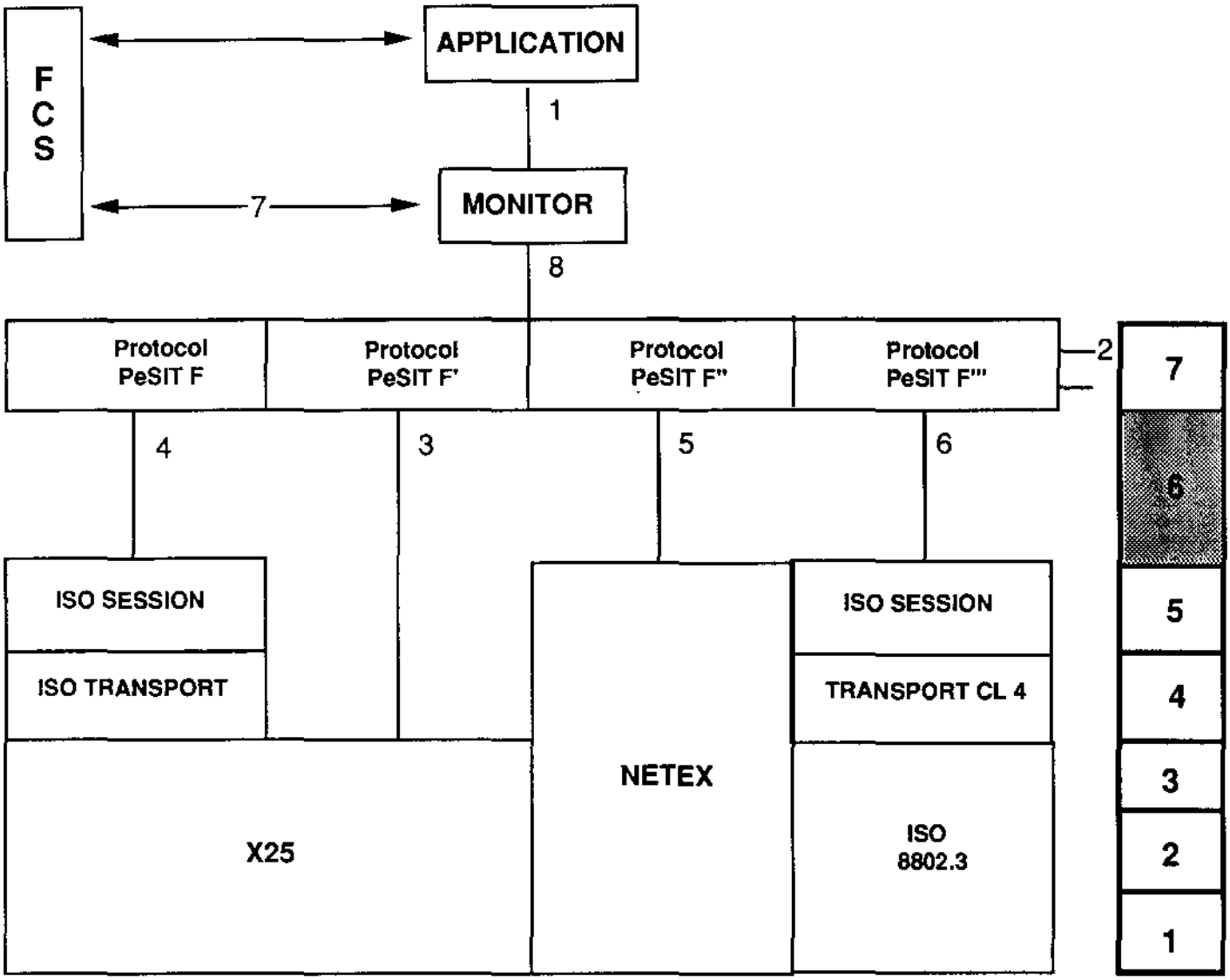
- A transfer monitor orders and controls the transfers.

- A protocol machine (PeSIT) executes the intended dialogue.

Each element uses, directly or indirectly, the file control system.

Only the PeSIT protocol machine is described in this document.

PeSIT FUNCTIONAL ARCHITECTURE



The diagram depicts :

1 Program interface between the application and the M.SIT monitor. This stable interface is essential for the long term durability of the application software.

2 The PeSIT protocol which is detailed in the second part of this document.

3 The group of primitives between the ISO layer 3 and the protocol PeSIT-F' (section 4.3.2 of this document).

4 The group of primitives between the ISO layer 5 and the protocol PeSIT-F (section 4.3.1 of this document).

5 The group of primitives between the NETEX layer and the protocol PeSIT-F" (section 4.3.3 of this document).

6 The group of primitives between the ISO session layer and the protocol PeSIT-F'" (section 4.3.4 of this document).

7 Accesses between the file transfer monitor (M.SIT) and the file control system (OPEN FILE, READ/WRITE, CLOSE FILE primitives).

8 Access primitives to the PeSIT service.

2.2 Virtual file model

Existing file systems vary considerably in their implementations of file format and file storage. It is thus necessary to create a common model of the file for use by any protocol working in a heterogeneous network environnement. This model is called the "virtual file". A virtual file enables the inner workings of a file storage system specific to a particular operating system to be made transparent to the protocol by means of conversion functions which map the local file description into a standardized file description and vice-versa.

Although the object of a file transfer protocol is to transfer real files, the model and the protocol are limited to the processing of virtual files. The correspondance between real files and virtual files is considered to be dependant on the local installation and as such it is not described in this document.

2.3 Functional description of the PeSIT Service and protocol

2.3.1 PeSIT service functions

PeSIT provides the following functions :

* writing of a distant file
* reading a distant file
* checkpointing during a transfer
* resuming a interrupted transfer from a negotiated restart point
* resynchronising during a transfer
* suspension of a transfer
* securitize a transfer
* data compression
* error control
* datagram service

2.3.1.1 Write file

This function allows a user of the PeSIT service to transfer the contents of a file to another user of the PeSIT service. Prior to transfering the file the sender must establish a logical link with his partner. The user who establishes the connection is called the Caller, and his correspondant is called the Server. In this example the file data transfer is between the Caller/Sender and the Server/Receiver.

2.3.1.2 Read file

This function allows a user of the PeSIT service to request that another user of the PeSIT service transfer the contents of a file to him. Prior to transfering the file the receiving user must establish a logical link with his partner. The user who establishes the connection is called the Caller, and his correspondant is called the Server. In this example the file data transfer is between the Server/Sender and the Caller/Receiver.

2.3.1.3 Checkpointing

This function allows the sender to set milestones, called checkpoints, which are numbered sequentially, during the transfer. The receiver can acknowledge these checkpoints, which signifies that he has received and saved the data correctly up to that point. This mechanism permits a transfer which is interrupted to be restarted at a position corresponding with one of the acknowledged checkpoints or from the beginning of the file.

2.3.1.4 Transfer recovery

This function allows a calling user to recover an interrupted transfer. This recovery is possible for a read or a write transfer, but can only be requested by the caller. The data receiver determines the checkpoint from which the transfer can be recovered..

2.3.1.5 Restart during a transfer

This function allows a user, following a problem during a file transfer, to request his partner to restart the transfer from a previous checkpoint. The difference between the recovery and the restart services is that the recovery occurs after an interrupted transfer (with subsequent closure and de-selection of the file) wheras the restart occurs during the transfer and the file is still open and selected.

2.3.1.6 Transfer suspension

This function allows a user to interrupt a transfer (which implies the closure and de-selection of the associated file) so as to re-use the existing logical connection for another transfer of a higher priority. The suspended transfer will be recovered later by the recovery procedure.

2.3.1.7 Transfer security

This function allows users of PeSIT to implement the following security mechanisms :

* reciprocal authentication of the partners
* confidentiality of the transmitted data
* integrity of the transmitted data
* reciprocal non repudiation

2.3.1.8 Data compression

This function allows users to implement the data compression mechanisms of PeSIT thus reducing the quantity of data transmitted.

2.3.1.9 Error control

By using a polynomial error detection algorithm applied to each PeSIT protocol message, this function allows the detection of messages which have been deteriorated by an unreliable transmission medium.

2.3.1.10 Datagram transfer

This function allows a user of the PeSIT service to transfer a quantite of unstructured information to another user of the PeSIT service. The transfer service information overhead is kept to the minimum necessary to identify the transfer, and thus with a minimum of protocol exchanges. The synchronisation, compression, restart, resynchronisation and suspension services are therefore not available for this type of transfer.
However certain security functions may still be used :

* integrity of the transmitted data
* reciprocal non repudiation

2.3.2 Functional unit concept

The implemention of all the functions described above is not always necessary to satisfy the needs of a PeSIT service user. Thus certain functions can be omitted in certain implementations of the protocol. To standardize the functions which can be omitted the functions are organized into functional units:

The functional units are :

Kernel : the kernel contains all the services necessary for the establishment and the termination of a logical link between two users of the PeSIT service (PeSIT connection).

Write : this functional unit contains all the services necessary to write a distant file.

Read : this functional unit contains all the services necessary to read a distant file.

Checkpointing : the checkpointing functional unit contains the services which allow checkpoints to be set during a transfer.

Restarting : the restarting functional unit contains the services which allow the restarting during a transfer.

Suspension : this functional unit contains all the services which allow the suspension of a transfer. I.e. which allow the interruption of a transfer so as to reuse the logical connection for a transfer of a higher priority.

Datagram : the datagram functional unit contains all the services necessary to provide the datagram service.

Error control : the error control functional unit does not contain any specific services but implies that the error detection mechanism be used by the protocol for all the messages sent or received. The user indicates that this functional unit should be activated by means of a parameter in the service primitive which requests the connection with the distant user.

Security : the security functional unit does not contain any specific services but implies that the security mechanisms (in particular the algorithms) have been implemented in the protocol. It also requires parameters to be exchanged between the user and the PeSIT service provider during the different service phases.

2.3.3 Profile concept

The use of the PeSIT protocol by a group of users or for a particular application can call for a choice of a certain number of functional units to be implemented, of parameters to be used and of their values. These choices constitute a usage profile for the protocol.

Actually four PeSIT usage profiles have been defined :

* the SIT profile
* the Non-SIT profile
* the Secure Non-SIT profile
* the ETEBAC 5 profile

They are differentiated by :

. the functional units used,

. the optional parameters used,

. the limit values authorised for certain parameters,

. the addressing conventions (such as caller and server identification) and file naming conventions (types and names of files).

CHAPTER 3

PeSIT SERVICE DESCRIPTION

3 PeSIT SERVICE DESCRIPTION

3.1 INTRODUCTION TO THE SERVICE

3.1.1 Scope

The service model describes the interactions between the user service entities (the applications) and the service provider entity (the protocol layer). Information is transfered between the service user and the service provider by means of **service primitives** which may contain parameters.

3.1.2 Partner roles

During a PeSIT session[1] between two users of PeSIT, the dialogue is always asymmetric, i.e. the two partners fulfil different and complementary rôles. The session originator, named the **CALLER**, is the partner who determines the work to be done during the session and is in the closest contact with the user whose requests are serviced. The other PeSIT user, named the **SERVER**, executes the work proposed by the caller and supplies a report on the work carried out.

3.2 REGIMES OF THE FILE SERVICE

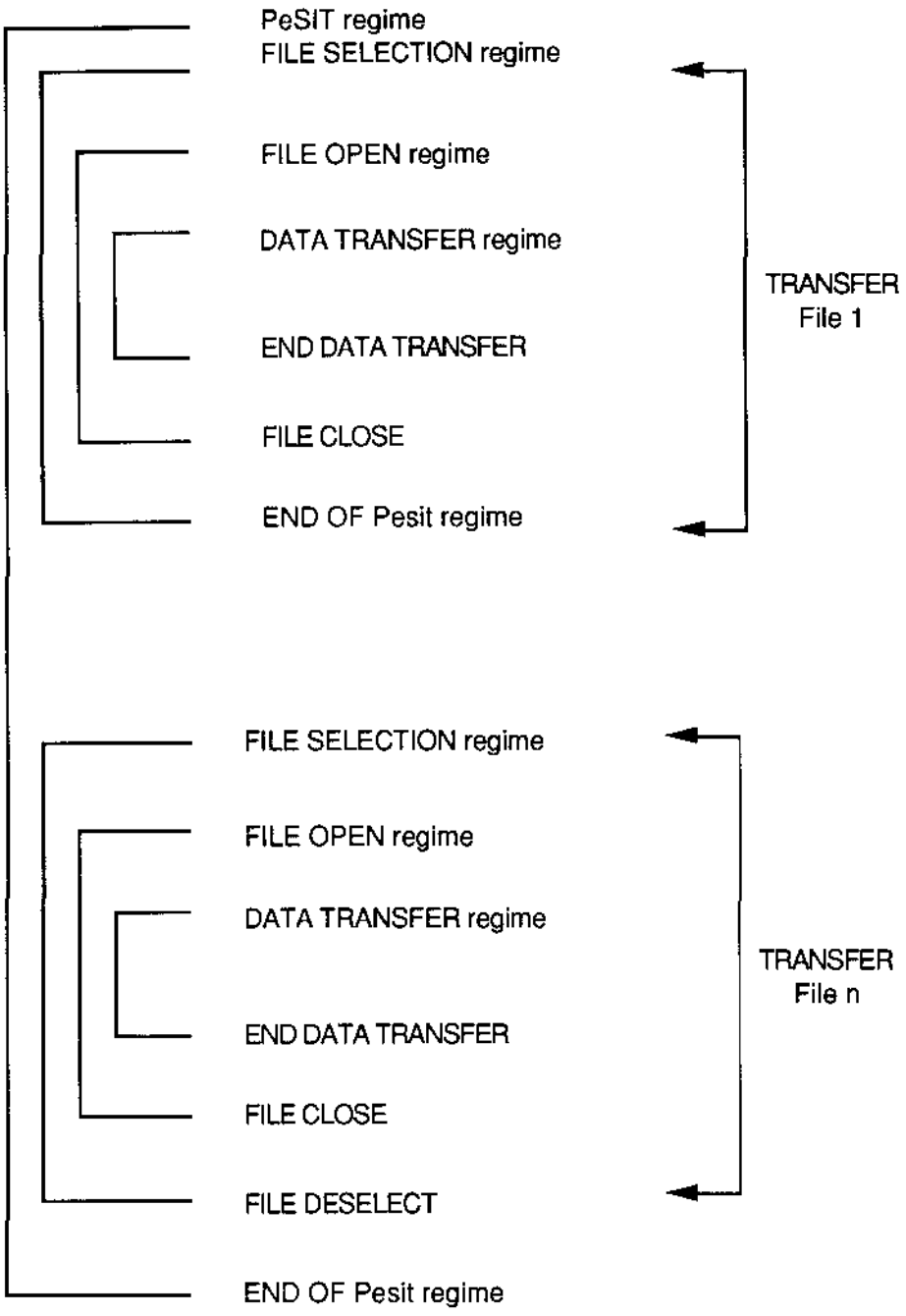
The PeSIT protocol only allows one file to be transfered at a time during a particular session. Several files can be transfered in parallel if several sessions are open.

The work carried out during a PeSIT session can be structured dynamically into a series of regimes within each other which must be opened in a hierarchical order and closed in the reverse order. If the session is interrupted or discontinued prematurely by a user, all the regimes which are still active are considered to have been implicitly closed.

These regimes fit together as shown in the following schema : a regime defines a step in the transfer process, within which a certain set of services may be used while others are prohibited. To exit a regime, the regimes nested within it must first have been closed.

[1] A session is defined as "a logical link established between two distant PeSIT protocols by means of a communication system"

SERVICE REGIMES



The description of the different regimes is as follows, in hierarchical order starting from the outermost layer :

PeSIT regime

Exists between the opening and the closing of the session; this regime creates the logical link between two conversing users using a PeSIT session.

FILE SELECTION Regime

A file is reserved for the data transfer (between the SELECTION/CREATION and the DESELECTION of the file). A PeSIT regime may contain zero or more FILE SELECTION regimes.

FILE OPEN Regime

The current file is ready for data transfer (between the opening and the closing of the file).

DATA TRANSFER Regime

The file data is transmitted (between the beginning and the end of the transfer). In a data transfer regime, the PeSIT users adopt the rôles of **SENDER** and **RECEIVER.**

3.3 SERVICES OF THE PeSIT FILE SERVICE

The different services which make up the PeSIT service, classified by regime, are as follows :

PeSIT Regime

PeSIT regime establishment service

PeSIT regime termination service

PeSIT regime user abort service

PeSIT regime provider abort service

FILE SELECTION Regime

File creation service

File selection service

File deselection service

Datagram service

FILE OPEN Regime

File open service

File close service

DATA TRANSFER Regime

Write bulk data file service

Read bulk data file service

Data unit transfer service

Checkpointing service

End of data transfer service

End of transfer service

Cancel data transfer service

Restarting transfer service

3.4 FUNCTIONAL UNITS

The different services are organised into functional units. The functional units and the services that they require are as follows :

Kernel

The services associated with this functional unit are :

PeSIT regime establishment service

PeSIT regime termination service

PeSIT regime user abort service

PeSIT regime provider abort service

Write

The services associated with this functional unit are :

File creation service

File deselection service

File open service

File close service

Write bulk data file service

Data unit transfer service

End of data transfer service

End of transfer service

Read

The services associated with this functional unit are :

File selection service

File deselection service

File open service

File close service

Read bulk data file service

Data unit transfer service

End of data transfer service

End of transfer service

Checkpointing

The service associated with this functional unit is :

Checkpointing service

Restarting

The service associated with this functional unit is :

Restarting transfer service

Suspension

The service associated with this functional unit is :

Cancel data transfer service

Datagram

The service associated with this functional unit is :

datagram service

Security

This functional unit does not require any particular services.

Error control

This functional unit does not require any particular services.

3.5 DEFINITION OF A PROFILE

A profile is differentiated by :

the functional units used,

the optional parameters used by the service primitives and therefore by the protocol elements,

the limit values authorised for certain parameters,

addressing conventions (such as caller and server identification) and file naming conventions (types and names of files).

The functional units necessary for each profile are described in paragraph 3.8 which also indicates the particular choice of parameters for each profile.

The format of each parameter and of each protocol element of the PeSIT protocol is detailed in chapter 4, profile by profile.

However for the description of the PeSIT service primitives given in paragraphs 3.6 and 3.7 we have preferred not to distinguish the different profiles, mentioning only which parameters are optional. Obviously parameters that are optional in a general description may become either mandatory or prohibited in a particular profile ; the detail is given in chapter 4.

To enable the Security functional unit, whose transfers require a global understanding and whose implementation is specific to certain usages, to be isolated from the rest of the description we have volontarily omitted all the parameters related to this functional unit in the general description of the service primitives (paragraph 3.6) and of the parameters (paragraph 3.7).

The description of the service primitives and their parameters used by the Security functional unit (specific to the Secure Non-SIT and ETEBAC5 profiles) are in paragraph 3.9 : PeSIT SECURITY SERVICE.

3.6 DESCRIPTION OF THE SERVICE PRIMITIVES

3.6.1 Correspondance between service and primitives

The following table shows the correspondance between the services and the primitives.

Service primitive	Confirmation	Originator	Service name
F.CONNECT	YES	Caller	PeSIT regime establishment
F.RELEASE	YES	Caller	PeSIT regime termination
F.ABORT	NO	Caller/Server	User/provider abort
F.CREATE	YES	Caller	File creation
F.SELECT	YES	Caller	File selection
F.DESELECT	YES	Caller	File deselection
F.OPEN	YES	Caller	File open
F.CLOSE	YES	Caller	File close
F.WRITE	YES	Caller	Write bulk data
F.READ	YES	Caller	Read bulk data
F.DATA	NO	Sender	Data unit transfer
F.DATA-END	NO	Sender	End of data transfer
F.TRANSFER-END	YES	Caller	End of transfer
F.CANCEL	YES	Caller/Server	Cancel data transfer
F.CHECK	YES	Sender	Checkpointing
F.RESTART	YES	Sender/Receiver	Restarting transfer
F.MESSAGE	YES	Caller	Datagram

3.6.2 Conventions

The SERVICE concept is an abstract notion which defines the interactions between the PeSIT protocol layer and the user at either end of the logical link. It is defined by a number of service PRIMITIVES.

Four types of primitives exist :

the request primitive : a user invokes the corresponding service,

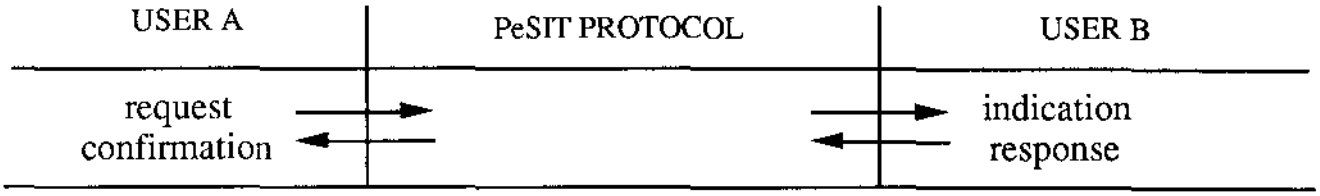
the indication primitive : a user is informed, by the service provider, that a service request has been received,

the response primitive : a user responds to a corresponding service indication,

the confirmation primitive : a user is informed by the service provider, that a response has been received.

Each elementary service is made up of a combination of primitives as the figures below illustrates :

SERVICE WITH CONFIRMATION



SERVICE WITHOUT CONFIRMATION

Each service element is defined by :

- its function,
- the associated parameter table.

For each parameter, a "+" in the corresponding column indicates that the parameter may be used in the corresponding primitive.

The parameters are described in paragraph 3.7.

In the parameter table, a number in brackets, next to a parameter, e.g. "(§i)" indicates the sub-paragraph of paragraph 3.7 which describes the parameter.

3.6.3 Description of the primitives

a) F.CONNECT Service

*** Function**

This service element allows a logical link to be set up between two PeSIT service users. The user who initiates the F.CONNECT request primitive becomes the caller and the user who receives the F.CONNECT indication primitive becomes the server. The caller is responsable for the connection until it is cleared-down. If the connection cannot be established then a diagnostic code informs the caller of the reason.

During the connection establishment phase certain options (use of functional units) are negotiated. This negotiation covers :

- the access type "read", "write" or "read/write" (Read and Write functional units),

- the checkpoint option (Checkpointing functional unit),

- the restarting (whether the Restarting data transfer functional unit will be used),

- use of a polynomial error detector (Error Control functional unit).

The Caller indicates the functional units that he can support in the F.CONNECT request and the server replies with the intersection between these functions and his own capacities in the F.CONNECT response. The version of the protocol to be used is also negotiated at this stage.

*** Parameter table**

PARAMETER		F.CONNECT request/indication	F.CONNECT response/confirmation
(§a)	CRC Usage	+	
(§b)	Diagnostics		+
(§c)	Caller identification	+	
(§c)	Server identification	+	
(§d)	Access control	+	+
(§e)	Version number	+	+
(§f)	Option : checkpoint	+	+
(§q)	Access type	+	
(§r)	Restarting	+	+
(§t)	Protocol monitoring time-out (optional)	+	
(§w)	Diagnostic complements (optional)		+
(§ab)	Free text (optional)	+	+

b) F.RELEASE Service

* **Function**

This service element allows a logical link between two PeSIT service users to be cleared down. Only the calling user may initiate the F.RELEASE request primitive during the connection as long as a file is not selected.

* **Parameter table**

PARAMETER	F.RELEASE request/indication	F.RELEASE response/confirmation
(§b) Diagnostics	+	
(§w) Diagnostic complements (optional)	+	
(§ab) Free text (optional)	+	+

c) F.ABORT Service

* **Function**

This service element allows a logical link between two PeSIT service users to be brutally and unconditional cleared down while abandonning any current activity.

Once an F.ABORT service element has been initiated or received any transfer regime which is open should be closed.

Either the caller or the server may initiate the F.ABORT request primitive at any time during a connection.

The F.ABORT service is not confirmed.

* **Parameter table**

PARAMETER	F.ABORT request/indication
(§b) Diagnostics	+
(§w) Diagnostic complements (optional)	+

d) F.CREATE Service

* Function

This service element allows a new file to be created and to select it to receive a file to be transfered (write transfer).

Only the calling user may initiate the F.CREATE request primitive and only during the connection regime, as long as the access type selected during the connection was "write" or "read/write".

The server user upon receiving an F.CREATE indication creates the file and selects it prior to responding with an F.CREATE response primitive. If the create fails the F.CREATE response primitive is negative and the diagnostic parameter indicates the reason for the failure.

Maximum size of a data unit

During the file selection regime the maximum size of a data unit is negotiated and the resulting value is passed on to the communication layer :

- the caller proposes the size that he wishes to use in the F.CREATE request primitive,

- the server replies in the F.CREATE response primitive with the maximum size accepted, which should be less than or equal to the requested size.

Within the limit of this maximum size it is possible to concatenate several FPDUs into a single data unit passed on to the communication layer. The concatenation rules are given in paragraph 4.5.

It should be noted that if the Segmentation functional unit is not used then the maximum data unit size must be greater than or equal to the sum of the maximum article size of the file to be transfered and the FPDU header (six bytes).

Transfer identifier

The transfer identifier parameter in the F.CREATE request primitive should be set to a non zero value by the caller if the transfer is new (not recovered). The server may provide a different (non zero) transfer identifier in the F.CREATE response primitive. The use of the transfer identifier parameter in the F.CREATE response primitive is not fixed. If the transfer is recovered then the caller indicates the same transfer identifier in the F.CREATE request primitive as he originally used in the first attempt.

Transfer recovery

The recovery facility is intended to avoid resuming a transfer which was suspended or cancelled prior to the end of the deselection regime, from the beginning. The recovery may occur during the same connection or at a later moment. In either case the file whose transfer is to be recovered may be selected and reopened with the same parameters as in the original transfer. The recovered transfer parameter, in the F.CREATE service, indicates if the transfer is new or a recovered transfer. For a write transfer, the point of recovery is negotiated by the F.WRITE service.

* **Parameter table**

PARAMETER	F.CREATE request/indication	F.CREATE response/confirmation
(§b) Diagnostics		+
(§g) File identifier	+	+
(§h) Transfer identifier	+	+
(§j) Recovered transfer	+	
(§k) Data coding	+	
(§l) Transfer priority	+	
(§s) Maximum size of a data unit	+	+
(§w) Diagnostic complements (optional)		+
(§x) File attributes	+	
(§y) Customer identifier (optional)	+	
(§y) Bank identifier (optional)	+	
(§z) File access control (optional)	+	
(§aa) Server Date and time (optional)		+
(§ab) Free text (optional)	+	

e) F.SELECT Service

* **Function**

This service element allows an existing distant file to be selected for reading (read transfer).

Only the calling user may initiate the F.SELECT request primitive and only during the connection regime, as long as the access type selected during the connection was "read" or "read/write".

Transfer recovery

In the same way as a write transfer, a read transfer may be recovered. The recovery facility is intended to avoid resuming a transfer which was suspended or interrupted prior to the end of the deselection regime from the beginning. The recovery may occur during the same connection or at a later moment. In either case the file whose transfer is to be recovered may be selected and reopened with the same parameters as in the original transfer. The recovered transfer parameter, in the F.SELECT service, indicates if the transfer is new or a recovered transfer. For a read transfer, the point of recovery is negotiated by the F.READ service.

File identifier

In the F.SELECT request primitive, the file name parameter may be either the real file name or a wild card file description which allows the server to search for a generic file name which satisfies various parameters (e.g. version number, creation date). Even the file type parameter, in the F.SELECT request primitive, may indicate either a precise file type or a generic file type. When generic parameters are used the server should indicate in the F.SELECT response primitive either that no file satisfied the selection criteria supplied by the caller or the complete name of the file selected.

It should be noted that for generic requests, each transfer concerns only one unambiguously identified file (for the partners and dates provided) identified by the file type and name parameters given in the F.SELECT response primitive. Consequently, the file type and name parameters will be different in the request primitive and in the response primitive.

Transfer identifier

The transfer identifier parameter in the F.SELECT request primitive should be set to zero by the caller if the transfer is new (not recovered). The server will provide a (non zero) transfer identifier in the F.SELECT response primitive. If the transfer is recovered then the caller indicates the same transfer identifier in the F.SELECT request primitive as was originally provided by the server during the first attempt.

* **Parameter table**

PARAMETER	F.SELECT request/indication	F.SELECT response/confirmation
(§b) Diagnostics		+
(§g) File identifier	+	+
(§h) Transfer identifier	+	+
(§i) Requested attributes	+	
(§j) Recovered transfer	+	
(§k) Data coding		+
(§l) Transfer priority	+	
(§s) Maximum size of a data unit	+	+
(§w) Diagnostic complements (optional)		+
(§x) File attributes		+
(§y) Customer identifier (optional)	+	
(§y) Bank identifier (optional)	+	
(§z) File access control (optional)	+	
(§aa) Server Date and time (optional)		+
(§ab) Free text (optional)	+	

f) F.OPEN Service

* **Function**

This service element allows a file to be open.

Only the calling user may initiate the F.OPEN request primitive after selecting the file. The server user upon receiving an F.OPEN indication primitive should open the file which was previously selected prior to responding with an F.OPEN response primitive.

The data presentation is negotiated during this phase. The possible options are :

* compression

The caller indicates in the F.OPEN request primitive whether he wishes to use data compression and the algorithm to be used for the current transfer. The server replies indicating in the F.OPEN response primitive whether he can provide the compression facility and the algorithm requested. The details of the compression algorithms and the rules of negotiation of these algorithms are given in Annexe A.

* **Parameter table**

PARAMETER	F.OPEN request/indication	F.OPN response/confirmation
(§b) Diagnostics		+
(§p) Compression	+	+
(§w) Diagnostic complements (optional)		+

g) F.CLOSE Service

* **Function**

This service element allows a file which was previously open to be closed. Once this service element has been initiated or received, no other service element may be initiated prior to receiving the F.CLOSE response. A request to close a file may not be refused.

Only the calling user may initiate the F.CLOSE request primitive. Upon receiving the F.CLOSE indication primitive, the server user should stop any actions under way and close the file prior to responding with an F.CLOSE response primitive.

* **Parameter table**

PARAMETER	F.CLOSE request/indication	F.CLOSE response/confirmation
(§b) Diagnostics	+	+
(§w) Diagnostic complements (optional)	+	+

h) F.DESELECT Service

* **Function**

This service element frees up the association between the caller and the selected file. A deselect may not be refused.

Only the calling user may initiate the F.DESELECT request primitive upon a previously selected file. Upon receiving the F.DESELECT indication primitive, the server user should free up the current file prior to responding with an F.DESELECT response primitive. Following a deselect, the file is preserved and may be re-selected.

* **Parameter table**

PARAMETER	F.DESELECT request/indication	F.DESELECT response/confirmation
(§b) Diagnostics	+	+
(§w) Diagnostic complements (optional)	+	+

i) F.READ Service

* **Function**

This service element allows a read data transfer to be initiated on a file from a particular point (start of the file or a recovery point). The F.READ service may only be initiated by the calling user following a select and file open regime. F.READ implies that the file data will be transfered from the server user to the calling user.

Recovery point negotiation

During this phase the recovery point is negotiated if the F.SELECT service requests that this transfer be recovered. The recovery point is determined by the file receiver who knows how much data has been correctly received. The recovery point parameter is in the F.READ request primitive. It should be either zero (recovery from the beginning of the file) or greater than or equal to the last checkpoint acknowledged by the calling receiver. The data sender therefore need keep only the context related to the checkpoints which have not been acknowledged. The server sender may indicate in the F.READ response primitive (diagnostic parameter) that he is unable to recover the transfer from the recovery point requested by the caller.

* **Parameter table**

PARAMETER	F.READ request/indication	F.READ response/confirmation
(§b) Diagnostics		+
(§m) Restart point	+	
(§w) Diagnostic complements (optional)		+

j) F.WRITE Service

* **Function**

This service element allows a write data transfer to be initiated on a file from a particular point (start of the file or a recovery point). The F.WRITE service may only be initiated by the calling user following a select and file creation regime. F.WRITE implies that the file data will be transfered from the calling user to the server user.

Recovery point negotiation

During this phase the recovery point is negotiated if the F.CREATE service requests that this transfer be recovered. The recovery point is determined by the file receiver who knows how much data has been correctly received. The recovery point parameter is in the F.WRITE response primitive. It should be either zero (recovery from the beginning of the file) or greater than or equal to the last checkpoint acknowledged by the server receiver. The data sender therefore need keep only the context related to the checkpoints which have not been acknowledged.

* **Parameter table**

PARAMETER	F.WRITE request/indication	F.WRITE response/confirmation
(§b) Diagnostics		+
(§m) Recovery point		+
(§w) Diagnostic complements (optional)		+

k) F.DATA Service

* **Function**

This service element allows a file data article to be transfered from the sender to the receiver.

Only the sender user (who may be the caller or the server depending on the initialization of F.READ or F.WRITE) may initialize the F.DATA resquest primitive.

F.DATA is not acknowledged.

* **Parameter table**

PARAMETER	F.DATA request/indication
(§ac) File article	+

l) F.DATA.END Service

* **Function**

This service element flags the end of data transfer. It is initiated only when all the file data has been transfered.

Only the sender user may initialise the F.DATA.END request primitive.

F.DATA.END is not acknowledged.

* **Parameter table**

PARAMETER	F.DATA.END request/indication
(§b) Diagnostics	+
(§w) Diagnostic complements (optional)	+

m) F.TRANSFER.END Service

* **Function**

This service element flags the end of the data transfer regime.

Only the calling user may initialise the F.TRANSFER.END request primitive.

Note

When the **caller** is **sender**, this request follows the F.DATA.END primitive, and the response, returned by the server, is an implicit acknowledgement of all the checkpoints. Specifically this primitive indicates that all the data have been received and written to file by the server.

For a **caller receiver**, the request is sent after the F.DATA.END has been received and constitutes an acknowledgement of all the checkpoints. Specifically this primitive indicates that all the data have been received and written to file by the caller.

* **Parameter table**

PARAMETER	F.TRANSFER.END request/indication	F.TRANSFER.ENDA response/confirmation
(§b) Diagnostics		+
(§u) Number of data bytes (optional)	+	+
(§v) Number of articles (optional)	+	+
(§w) Diagnostic complements (optional)		+

n) F.CANCEL Service

* **Function**

This service element allows a data transmission to be interrupted during the transfer regime.

Either the caller user or the server user may initiate the F.CANCEL request primitive.

* **Parameter table**

PARAMETER	F.CANCEL request/indication	F.CANCEL response/confirmation
(§b) Diagnostics	+	
(§n) End of transfer code	+	
(§w) Diagnostic complements (optional)	+	

o) F.CHECK Service

* **Function**

This service element allows checkpoints to be set on the data transferred. Only the sender user may initiate the F.CHECK request primitive during the data transfer regime. It can only be used if the Checkpointing functional unit was accepted during the connection regime.

During the connection negotiations the maximum number of bytes which may be transmitted between two F.CHECK services is determined as well as the rules for confirmation of the service.

The F.CHECK service is not acknowledged if, during the negotiations on the use of the Checkpointing functional unit, the window was set to zero. Otherwise the window determines the number of F.CHECK service primitives which may be issued successively without waiting for the reception of a confirmation primitive.

Each F.CHECK primitive need not be confirmed individually as the confirmation of a checkpoint implicitly confirms all the previous non-confirmed checkpoints.

Prior to sending an F.CHECK response primitive the receiver must flush write all the data received to file.

* **Parameter table**

PARAMETER	F.CHECK request/indication	F.CHECK response/confirmation
(§o) Checkpoint number	+	+

p) F.RESTART Service

* **Function**

This service element allows a transfer to be restarted from a previous checkpoint. It can only be used if the Checkpointing and Restarting functional units have been selected during the connection regime. Either the caller or the sender may initiate the F.RESTART request primitive during the data transfer regime. Following acceptance by the partner of the restart point, the data transfer continues from this point without leaving the data transfer regime.

Restart point negotiation

The restart may be either from the beginning of the file (restart point 0) or from any of the checkpoints prior or equal to the last acknowledged checkpoint. The restart point to be used is determined by negotiation between the F.RESTART request and response primitives. It is always the receiver who determines the restart point.
The receiver who requests a restart should indicate the requested restart point in the F.RESTART request primitive (above or equal to the last confirmed checkpoint) and the sender may either accept to continue from this point (F.RESTART response primitive with the same restart point as requested in the F.RESTART request primitive) or impose a restart from the beginning of the file (F.RESTART response primitive with a zero restart point).

The sender who requests a restart should indicate the requested restart point in the F.RESTART request primitive (above or equal to the last received checkpoint confirmation) and the receiver may either accept to continue from this point or from a later point (F.RESTART response primitive with a restart point greater than or equal to the restart point requested in the F.RESTART request primitive) or impose a restart from the beginning of the file (F.RESTART response primitive with a zero restart point).

Note

The restart point is conveyed by the recovery point parameter.

* **Parameter table**

PARAMETER	F.RESTART request/indication	F.RESTART response/confirmation
(§b) Diagnostics	+	
(§m) Recovery point	+	+
(§w) Diagnostic complements (optional)	+	

q) F.MESSAGE Service

* **Function**

This service element allows a user of the PeSIT service to send a quantity of unstructured information to another user of the PeSIT service. Only the calling user may initiate the F.MESSAGE request primitive during the connection regime.

The F.MESSAGE service is acknowledged.

* **Parameter table**

PARAMETER	F.MESSAGE request/indication	F.MESSAGE response/confirmation
(§b) Diagnostics		+
(§g) File identifier	+	
(§h) Transfer identifier	+	+
(§i) Requested attributes	+	
(§k) Data coding	+	+
(§w) Diagnostic complements (optional)		+
(§x) File attributes	+	
(§y) Customer identifier (optional)	+	
(§y) Bank identifier (optional)	+	
(§ad) Datagram (optional)	+	

3.7 DESCRIPTION OF THE PARAMETERS

a) CRC Usage

This parameter indicates if a polynomial error detection checksum (CRC) is added to each FPDU to check the validity of these messages (see §4.3.2). This parameter is mandatory for use of the protocol with a PAD.

b) Diagnostics

This parameter indicates the gravity and the type of error encountered. It is made up of two fields :

- error type which indicates the gravity of the error,
- diagnostic code which gives the detail of the type of error.

The list of the diagnostics is given in Annexe D and shows within which service primitives they may occur.

c) Caller and server identification

This parameter specifies the name of the caller and server users.

d) Access control

The access key allows reciprocal identification of the caller and the server (password). If a user wishes to modify his password, the new one is put in the access control parameter after the password which is to be replaced.

e) Version number

Version number of the software. If an incompatibility exists between the version indicated in the F.CONNECT request primitive and the version supported by the server then the server may refuse the connection. The server may also suggest a previous version of the protocol in the F.CONNECT response primitive in which case the caller may either close down the logical link or accept the version suggested by the server.
Version number 1 corresponds with the protocol described in the version D of the PeSIT technical specifications dated 15 November 1987.
Version number 2 corresponds with the protocol described in the current version E of the PeSIT technical specifications dated 14 July 1989.

f) Option : checkpointing

This parameter allows the Checkpointing functional unit to be negotiated.

- interval between two checkpoints :

* 0 in this field indicates : no checkpoints,

* 65535 in this field indicates : undefined interval

* any other value indicates the maximum number of bytes of the file (expressed in kilo-bytes, 1 kilo-byte = 1024 bytes) that the sender may transmit between two consecutive checkpoints. This value includes only the data fields of the FPDU.DTF (not counting the header) and excluding the article length fields of multi-article FPDUs. However if compression is used then the data is counted after compression and includes the compression headers (if present).

- window :

* 0 in this field indicates : no acknowledgement of checkpoints required,

* a non zero value defines the size of the window for acknowledging checkpoints. The window defines the greatest difference allowed between :

- the number of the last checkpoint transmitted,

- and, the number of the last checkpoint acknowledged.

Once the window is full, data transmission is suspended until a checkpoint confirmation is received. When the window is larger than one, each checkpoint need not be explicitly acknowledged, as a confirmation implicitly acknowledges all the previous checkpoints.

Negotiation of the checkpoint option

The negotiation takes place during the connection regime. The caller proposes the values that he wishes to use, and the server replies with the negotiated values using the following rules :

- if both parties wish to use checkpoints then the option is selected. Otherwise the option is refused.

- if the option is selected, the server may decide on values for the checkpointing interval and the window which are less than or equal to those suggested by the caller.

g) File identifier

This parameter is exchanged between the caller and the server during the file selection phase and makes up the file identification. A non-ambiguous file identification may require other parameters depending on the profile used : see §3.8.

The file identifier is composed of :

- caller identification : optional parameter, if absent the previous known value is taken from either the connection regime or the last selection regime in which it was given. If present, its value may be different from the value given during the connection regime or by a preceding selection regime in which case the new value is adopted until the connection is cleared-down or a different value is given in a succeding file selection regime.

- server identification : optional parameter, its use is governed by the same rules as the caller identification.

- file type : this parameter defines the file class type : its use is determined by each particular profile.

- file name : this parameter allows the correct identification of a file within the file type.

For a read transfer the file type and file name parameters may be of a generic form in the F.SELECT request primitive specifying a group of files. In this case the file type and file name parameters returned in the F.SELECT response primitive will be different and will describe a single file out of the generic group description. If the server does not find any files corresponding with the generic description then the F.SELECT response primitive will be negative.

h) Transfer identifier

This parameter has a numeric value which allows the transfer to be identified. For a recovered transfer the transfer identifier should be identical to that used during the initial transfer.

For a write operation the transfer identifier is chosen by the caller (non zero value in the F.CREATE request primitive).

For a read operation the transfer identifier is chosen by the server. The caller should set a zero transfer identifier (except for a recovery) in the F.SELECT request primitive. The server will determine the value of the transfer identifier (non-zero) in the F.SELECT response primitive. For a recovered read operation the caller puts the same transfer identifier as was chosen by the server during the initial transfer in the transfer identifier field and the server should response with the same value.

i) Requested attributes

Indicates the file attributes which should be included in the response to a caller. They are made up of any combination (even nul) of the following categories : logical, physical and historical.

j) Recovered transfer

This parameter indicates that the transfer is a retry of a previous unfinished transfer. The recovery is always initiated by the caller however it is the receiver who determines from which recovery point the transfer will be continued.

k) Data coding

This parameter indicates the type of coding used for the data in the file to be transfered. The possible values are : "binary" (transparent transmission), "ASCII" or "EBCDIC".

l) Transfer priority

This parameter determines the relative priority given to a transfer by the caller.

m) Recovery point

This is the checkpoint number which allows a recovery or a restart to take place from a particular point in a file. The value is determined by the receiver, in an F.WRITE response primitive for a write operation and in an F.READ request primitive for a read operation. A zero value is used to indicate the beginning of the file.

n) End of transfer code

This code gives the reason for terminating data transfer in an F.CANCEL primitive. The possible values are :

- Error
- Suspension
- Cancellation by the server
- Cancellation by the caller

o) Checkpoint number

A numeric value which identifies a checkpoint un-ambiguously. It is incremented by one at each successive F.CHECK primitive, starting from 1 for the first checkpoint. The maximum value is 999 999.

p) Compression

This parameter allows the use of compression during the transmission of file data to be negotiated during the open regime. The possible types of compression are :

- horizontal compression,
- vertical compression,
- both.

The details of the negotiation mechanisms and the compression algorithms are given in Annexe A.

q) Access type

Indicates the type of access allowed during the transfer viz: "Read, Write or Read/Write".

- **Read** : when the caller is sender.

- **Write** : when the server is sender.

- **Read/Write** : used when both send and receive transfers may occur in the same connection.

r) Restarting

This parameter is used to negotiate the Restarting functional unit during the connection regime (use of the F.RESTART service).

s) Maximum size of a data element

This parameter specifies the maximum number of bytes that may be transported in a data unit (NSDU, SSDU, ...). Its value is negotiated during the file selection regime. The caller proposes a maximum value and the server replies with a value which is less than or equal to this value. The values selected for the maximum data element size and maximum article size determine the use of the segmentation, concatenation and multi-article FPDU mechanisms.

t) Protocol monitoring time-out

This parameter allows the value of the protocol monitoring time-out to be used for a connection to be negotiated during the connection regime.

u) Number of data bytes

This parameter gives the total number of bytes (less the length fields for multi-article FPDUs but including compression string headers) which were transmitted or received during the transfer regime of a file. It is used by the F.TRANSFER.END service primitives as a check value.

v) Number of articles

This parameter gives the number of articles transmitted or received for a file during the file transfer regime. It is used by the F.TRANSFER.END service primitives as a check value.

w) Diagnostic complements

This parameter contains complementary information following a refusal diagnostic (explanation of a format error, call back time, backup number, etc.).

x) File attributes

This parameter contains the characteristic parameters of a file. There are three type of file attributes : logical, physical and historical.

*** logical attributes**

These are the characteristics which allow access to the file :

- **article format** :
specifies the format of the articles in the file. The permitted values are : fixed or variable.

- **article length** :
specifies the length in bytes of an article in the file. It is the exact length for a fixed format file and a maximum length for a variable format file.

- **file organisation** :
describes the organisation of the data within the file and thus the access method to be used for the file transfer. The possible values are : sequential, relative or indexed.

- **digital signature usage** :
determines whether the file is covered by a SIT MAC.

- **SIT MAC** :
present for files transmitted by a SIT station towards a Bank Processing Center.

- **file label** :
may be used to associate a symbolic name with a file.

- **key length** :
contains the key length in bytes for an indexed file format.

- **key offset** :
contains the offset of the key relative to the beginning of an article for an indexed file format.

*** physical attributes**

These are the physical characteristics of the file :

- **storage reservation unit** :
defines the unit used when reserving space for a file. The units possible are : kilo-bytes or articles.

note :
Units will be used in the following way :

. **kilo-bytes** for files with variable length articles,

. **articles** or **kilo-bytes** for files with fixed length articles.

- **maximum reserved space** :
defines the maximum size that the file may not exceed.

*** historical attributes**

These parameters characterise the past history of the file

- **date and time of creation** :

- **date and time of last access** :
the date when the last transfer was completed whether normally or following an interruption.

y) Customer and bank identifiers

This parameter contains the identification of the client or the bank for whom the transfer was performed.

z) File access control

Access key allowing the client to be identified by the bank. This password is exchanged during the selection regime and is thus associated with the file. If a user wishes to modify his password, the new one is put in the file access control parameter after the password which is to be replaced.

aa) Server date and time

This parameter contains the date and time as known by the server when the file was selected.

ab) Free text

This parameter allows a message (string of ASCII characters) to be passed from one service user to another during the execution of one of the transfer regimes.

ac) File article

This parameter contains the data of a file article. The correspondance between an article in the virtual file and the record in the real file is the responsability of the local installation.

ad) Datagram

This parameter allows a message to be passed from one service user to another using the specific datagram service.

3.8 PROFILE DESCRIPTIONS

3.8.1 SIT profile

The SIT profile is specified by :

* the functional units :

Kernel
Write
Checkpointing

* the limit values of certain parameters :

Option - checkpointing : the interval between two checkpoints should be greater than or equal to 4 kilo-bytes, the window less than or equal to 16.

Maximum size of a data element : must be greater than or equal to 800 bytes.

* a specific address system :

the caller and server identifiers are the installation references made up of :
. 1 byte which indicates the installation type (symbolic value :
1 for CTE,
2 for CTR,
3 for IE,
4 for IR).
. 2 bytes which indicate the installation number within the type (numeric value).

The CTE and CTR installation types are contained in a Bank Processing Center, the IE and IR installation types are contained in a SIT station.

It should be noted that the concepts of sending installation (IE and CTE) or receiving installation (IR and CTR) should be considered relative to the flow of banking operations (e.g. an outbound deposit may be sent from a CTE to a CTR, via transfers from the CTE to the IE by PeSIT, from the IE to the IR on the primary network, and from the IR to the CTR by PeSIT) rather than in the sense of sending or receiving a file by PeSIT. Consequently any installation (CTE, CTR, IE or IR) may be, at any particular moment in time, in relation to PeSIT either a caller/sender or a server/receiver. There are thus four flow types possible :

CTE to IE
IE to CTE
CTR to IR
IR to CTR

* a file naming convention : the "File name" parameter is a string of 5 ASCII numeric characters. The parameters used by a SIT station to describe non-ambiguously a SIT file are :
- the sending or receiving installation number (caller or server identification)
- the file type
- the file name
- the file creation date

The file type characterises the sort of file to be transfered : outbound deposits, presentation reports, day end summaries, etc.. The list of the file types used by the SIT Inter Bank Clearing and SIT Stock Exchange networks, classified by flow type (IE to CTE, IR to CTR, etc.) may be found in the "Functional Analysis of ASIT" and "CTB Commands and Reports" SESA 70296 LP 01 210.

The file creation date is the specific SIT date which may not correspond with the current system date.

* The three levels of priority defined by PeSIT (0, 1 and 2) are used in the transfers between a SIT station and the CTBs for each data flow. The choice of priority for a particular transfer is determined by the applications using PeSIT.

* The SIT station limits the number of incoming transfers to three for each installation, regardless of their priorities.
The SIT station limits also the total number of incoming transfers for each priority. These limits are set by the GSIT.

* The files transfered between a SIT station and a CTB may be either fixed or variable format and the maximum size of an article is 4044 bytes (which implies a maximum data element size of 4050 bytes). The maximum size of an article may not be null.

* The choice of the data coding (ASCII or EBCDIC) for the file contents is decided as an installation parameter when a user defines its SIT connection characteristics. The SIT station does not care for the Data coding parameter (PI 16). According to the installation concerned, the station sends or expects to receive files appropriatly coded.

* Data compression is not implemented at the file transfer level.

* The SIT MAC is an encrypted time-stamp associated with the file. It is only sent for transfers originating from the station to the CTB. Thus :

For transfers from the CTB to the station : the parameter "Digital signature usage" has a value of 0 (or is absent) and the parameter "SIT MAC" is absent. For transfers from the station to the CTB : the parameter "Digital signature usage" has the value 1 and the parameter "SIT MAC" contains the MAC. The deciphering of the MAC and its validation are carried out by the application.

3.8.2 Non-SIT Profile

The Non-SIT profile is specified by :
* the obligatory functional units :

Kernel
Write
Checkpointing

* the optional functional units

Read
Restarting
Suspension
Datagram
Error control

* At the protocol level, the options of using multi-article FPDUs and of FPDU segmentation (use of FPDU.DTFDA, FPDU.DTFMA and FPDU.DTFFA) are allowed. It should be noted that the use of these options is not negotiated dynamically by the protocol and so should be determined between the partners in advance.

* addressing : the caller and server identifiers are ASCII strings of 1 to 24 characters chosen by the PeSIT service users.

* the implementation by file transfer monitors using the PeSIT Non-SIT profile of the "Store and Forward" functions (file re-route) is possible. This procedure is detailed in Annexe B.

* file naming : the parameter "file name" is a string of one to sixty-four ASCII characters. The file naming conventions depend on the PeSIT service users.

* the parameter "file type" should have the value 0 unless a specific meaning has been decided upon between two file transfer monitors.

* data compression may be implemented at the file transfer level. The horizontal and vertical compression algorithms are given in Annexe A.

* the pre-connection phase :

The connection phase of PeSIT is preceded by a pre-connection phase, independant of the PeSIT protocol. This extra phase allows the file transfer monitors to know which file transfer protocol is being used as soon as the lower communication layers are ready and also to identify the caller.

Two messages have been defined for this function :

Message 1 : is composed of 24 bytes :

* the first 8 bytes : protocol used (PESIT : 5 characters left justified, followed by 3 blanks)

* the 8 following bytes : identifier (1 to 8 characters left justified and blank padded)

* the last 8 bytes : password

Message 2 : acknowledgement : 4 bytes :

ACK0 or NAK0

Both these messages are coded in EBCDIC. They are not part of the PeSIT protocol elements.

Notice :

When using PeSIT.F', the pre-connection mesages are sent in the first data packets exchanged immediatly after the virtual circuit set up. This pre-connection phase must, while using PeSIT.F', be considered as mandatory.

When using PeSIT.F or PeSIT.F", no pre-connection phase is defined. Nevertheless a 24 byte message similar to message 1 defined above, may be used :

* in PeSIT.F" in the O-DATA field of the CONNECT primitive

* in PeSIT.F in the user reference field of the S-CONNECT primitive

3.8.3 Secure Non-SIT Profile

The secure Non-SIT profile is identical to the Non-SIT profile except that the use of the Security functional unit is obligatory in this profile.

It should be noted that the Security functional unit which is common to the Secure Non-SIT profile and the ETEBAC5 profile uses different encryption algorithms in the two cases and so does not offer exactly the same security functions.

The Secure Non-SIT profile only requires the use of the DES (Data Encryption Standard) encryption/MAC computation algorithm.

In this profile the Security functional unit provides :

* reciprocal authentication

* confidential data transmission

* integrity of the transmitted data

Annexe C : *Use of Security mechanisms* describes how to implement the security mechanisms for the Secure Non-SIT and ETEBAC5 profiles.

Note :

The use of encryption devices for the transmission of coded data across public data networks is covered by numerous laws which differ between countries and which users of the Secure Non-SIT and ETEBAC5 profiles should take into account prior to selecting these profiles.

3.8.4 ETEBAC5 profile

The ETEBAC5 profile is characterised by :

* the obligatory functional units :

Kernel
Write
Read
Checkpointing

* the optional functional units :

Restarting
Suspension
Security

* The description of the use of the PeSIT protocol by the ETEBAC5 transport layer is described in the document : "*Computer Information Exchanges between Banks and their Customers - Standard ETEBAC5 - Version 1.1*".

* In the protocol chapter of the current document only the format of the parameters and the protocol elements used in the ETEBAC5 profile are described.

* Annexe C : *Use of Security mechanisms* describes how to implement the security mechanisms for the Secure Non-SIT and ETEBAC5 profiles.

3.9 PeSIT SECURITY SERVICE

3.9.1 Functions provided

The security functions required for file transfer are :

* reciprocal authentication of the partners

* confidential data transmission (file contents)

* integrity of the transmitted data (file contents)

* reciprocal non-repudiation

The PeSITservice and protocol describe the parameters needed to implement the security mechanisms and the manner they are exchanged. However the total implementation (algorithms, key management, certificate management) are not part of the protocol. Annexe C of this document describes how to use the security mechanisms where they have an effect on the protocol.

The security parameters defined in PeSIT are intended to allow the use of different algorithms to provide the functions listed below. In the two profiles which use the Security functional unit, it was decided to use RSA and DES for the ETEBAC5 profile. The Secure Non-SIT profile has a reduced function mode where only DES is required. It should be noted that in this case the reciprocal non-repudiation function is not available.

The fact that the security is parameterised in the protocol allows considerable independance between the different functions. In the Secure Non-SIT profile each function may be used completely independantly of the others. In the ETEBAC5 profile, the reciprocal non-repudiation requires the integrity function.

The implementation of the security functions may be negotiated for each file transfer. However it is up to the system designers, depending on their security requirements, to decide if the security functions may really be re-negotiated for each transfer, or whether several consecutive transfer within the same connection should use the same security parameters.

3.9.2 Description of the primitives

The PeSIT service primitives are described in paragraph 3.6.3 except for the parameters specific to the Security functional unit. The present paragraph is intended to complete their description by the aspects related to the implementation of the Security functional unit.

The rôle of each primitive in the Security functional unit is described in this paragraph as well as the parameters supplementary to those described in paragraph 3.6.3.

a) F.CREATE Service

* Function

This service element allows the caller to instruct the server which security functions will be used for the following transfer. The caller indicates which of the following functions will be used :

* reciprocal authentication
* integrity
* confidentiality
* digital signature

It should be noted that a digital signature requires the MAC to have been calculated beforehand.

The server idicates in a positive or negative F.CREATE response primitive if he accepts the security functions requested by the caller.

This service element also allows the exchange of certificates and the authentication elements which make up the reciprocal authentication of the partners (the complete reciprocal authentication process uses the F.OPEN service as well).

The detail of the management of the certificates is given in Annexe C.

If several transfers take place in the same connection, the reciprocal authentication need not be repeated for each transfer. In the same way the certificates used to transport the keys and the digital signatures need not be repeated for each transfer within a connection (as long as the partners concerned, identified by the customer and bank identifier parameters, do not change). However the indication of the intention to use the encryption, integrity and digital signature functions as well as the encryption and MAC elements must be transmitted prior to each transfer.

* Parameter table

PARAMETER	F.CREATE request/indication	F.CREATE response/confirmation
(§a) Authentication type (optional)	+	
(§b) Authentication elements (optional)	+	+
(§c) MAC computation type (optional)	+	
(§e) Encryption type (optional)	+	
(§g) Digital signature type (optional)	+	
(§j) Certificate (optional)	+	+
(§m) Second certificate (optional)		+

b) F.SELECT Service

* **Function**

The function of this service element within the Security functional unit is identical to the F.CREATE service, the choice of the security functions to be implemented for a transfer is always the responsability of the Caller whether the file is being sent or received.

* **Parameter table**

PARAMETER	F.SELECT request/indication	F.SELECT response/confirmation
(§a) Authentication type (optional)	+	
(§b) Authentication elements (optional)	+	+
(§c) MAC computation type (optional)	+	
(§e) Encryption type (optional)	+	
(§g) Digital signature type (optional)	+	
(§j) Certificate (optional)	+	+
(§m) Second certificate (optional)		+

c) F.OPEN Service

* **Function**

The function of this service element is to allow the exchange of the third reciprocal authentication element for a write transfer (or the second and third exchanges for a read transfer). The MAC computation and encryption elements as well as the certificates that these exchanges may require may also be exchanged at this time.

The detail of the management of the certificates is given in Annexe C.

* **Parameter table**

PARAMETER	F.CHECK request/indication	F.CHECK response/confirmation
(§d) MAC (optional)	+	

d) F.CHECK Service

* **Function**

This service element allows partial MACs to be transmitted if this option has been selected. The partial MACs are the results of the intermediary MAC computations carried out on the entire file (plus certain identification parameters of the file).

* **Parameter table**

PARAMETER	F.OPEN request/indication	F.OPEN response/confirmation
(§b) Authentication elements (optional)	+	
(§d) MAC computation elements (optional)	+	+
(§f) Encryption elements (optional)	+	+
(§j) Certificate (optional)	+	
(§m) Second certificate (optional)	+	

e) F.DATA.END Service

* **Function**

This service element allows the MAC, the digital signature and a possible second digital signature to be transfered.

* **Parameter table**

PARAMETER	F.DATA.END request/indication
(§h) MAC (optional)	+
(§i) Digital signature (optional)	+
(§l) Second digital signature (optional)	+

f) F.TRANSFER.END Service

* **Function**

This service element allows the acknowledgement of the digital signature to be transfered.

* **Parameter table**

PARAMETER	F.TRANSFER.END request/indication	F.TRANSFER.END response/confirmation
(§k) Acknowledgment of the digital signature (optional)	+	+

g) F.MESSAGE Service

* **Function**

This service element contains a message which may be protected by the security mechanisms.

The functions which may be implemented with this service are :

* integrity
* digital signature

The MAC or the digital signature are applied to the message contained in the F.MESSAGE primitive.

The mechanisms and the parameters required to implement these functions for the service are identical to those used for a file transfer.

* **Parameter table**

PARAMETER	F.MESSAGE request/indication	F.MESSAGE response/confirmation
(§c) MAC computation type (optional)	+	
(§d) MAC computation elements (optional)	+	
(§g) Digital signature type (optional)	+	
(§h) MAC (optional)	+	
(§i) Digital signature (optional)	+	+
(§j) Certificate (optional)	+	+
(§k) Acknowledgment of the digital signature (optional)	+	+

3.9.3 Parameter description

a)Authentication type

This parameter indicates whether an authentication procedure will be used and which mechanisms will used within it.

- authentication (yes/no)
- used algorithm
- operating mode

b)Authentication elements

This parameter contains the elements needed for the authentication of the partners. Depending on the algorithm and the operating mode determined by the "authentication" parameter, these authentication elements will contain either random numbers in plain or encrypted and/or encrypted keys.

c)MAC computation type

This parameter indicates if the file is to be transfered with a MAC. If so the parameter indicates which MAC computation algorithm will be used and its operating mode. The operating mode indicates if there is only one MAC applied to the entire file or if, as well as the global MAC, there will be partial MACs transmitted with each checkpoint (MAC calculated only on the data transmitted since the previous MAC was transmitted). It is also explicited if the MAC computation elements are transmitted , in plain or encrypted form, in the "MAC computation element" parameter. In order to calculate the MAC the file identifiers (PI 11, PI 12, PI 51, PI 61 and PI 62) are used : the contents of these PI will be treated by the algorithm as if they had been concatenated to the beginning of the file.

- MAC computation (yes/no)
- used algorithm
- operating mode
- transfer of MAC computation elements

d)MAC computation elements

This parameter contains the elements (key, initialisation vector) required to initialise the chosen algorithm for computation of the MAC. This parameter is absent if the elements are transfered by another means than the protocol. They may be protected by encryption if this is indicated in the "transfer of Mac computation elements" field of the "MAC computation type" parameter.

e)Encryption type

This parameter indicates if the file is transmitted in encrypted form, and which algorithm is used to encrypt it.
This parameter also indicates if the encryption elements are to be transmitted in the "encryption elements" parameter and the encryption method to be used (these elements may transit by another means than the protocol).

- encrypted data (yes/no)
- used algorithm
- operating mode
- transfer of encryption element

f) Encryption elements

This parameter contains the elements (key, initialisation vector) used to initialise the chosen algorithm for encryption of the file. This parameter may be absent if one wishes to transfer them by another means than the protocol. They may be protected by encryption if this is indicated in the "transfer of encryption element" field of the "Encryption type" parameter.

g)Digital signature type

This parameter indicates if the file is to be transfered with a digital signature. Since the digital signing mode used requires the MAC to be encrypted, this parameter has no sense unless the MAC computation option has been selected in the "MAC computation type" parameter. The algorithm indicated in the "used algorithm" field is the algorithm used to transform the MAC into a digital signature by encryption. Actually the only algorithm used is RSA. The field "double digital signature" indicates that a second signature is present. In which case the second signature will be another encrypted form of the MAC using a different secret key (whose corresponding public key is provided in the parameter "second certificate").

- signature type
- used algorithm
- operating mode
- double digital signature

h)MAC

This parameter contains the result of the MAC computation . It may be a partial MAC (an intermediate result of the computation of the global MAC calculated using the data transmitted since the previous partial MAC was sent), or the global MAC. In order to calculate the MAC the file identifiers (PI 11, PI 12, PI 51, PI 61 and PI 62) are used : the contents of these PI will be treated by the algorithm as if they had been concatenated in front of the first article in the file.

i) Digital signature

This parameter contains the digital signature of the file which is an RSA encryption using the sender's secret key of the concatenation of a MAC calculated using the PI 11, PI 12, PI 51, PI 61 and PI 62 and the MAC of the file (which already includes the above PI). These two MACs are calculated independantly but using the same MAC computation elements.

j)Certificate

This parameter contains an entity's certificate. This certificate is made up of the entity identifier, the serial number of the certificate storage device (serial number of a smart card), the entity's public RSA key, the number of the security manager's RSA key used to sign the certificate and the digital signature of the certificate (RSA encryption under the security manager's secret key of a shadow - resulting from hash coding the preceding fields of the certificate).

k)Acknowledgment of the Digital signature

This parameter contains a protected acknowledgment of the digital signature. The acknowledgment is an RSA encryption under its sender's secret key of the concatenation of the MAC received in the digital signature, the date and time of the acknowledgment and a control field describing the security controls made.

l) Second Digital signature

This parameter contains the second digital signature of a file produced by an identical computation as that used for the "Digital signature" parameter but using an RSA encryption under another of the sender's secret keys whose associated public key is contained in the "second certificate" parameter.

m)Second certificate

This parameter contains a second entity certificate needed when several certificates are required for the same exchange (authentication and digital signature or multiple digital signatures). The structure of this parameter is identical to the "certificate" parameter.

3.10 EXAMPLES OF PRIMITIVE SEQUENCES

3.10.1 Normal sequence

The normal logical progression of services is illustrated by the following state diagrams (figures 1, 2, 3, 4 and 5). The tables which follow show the restrictions applicable to the sequencing of primitive events for the caller and the server. These tables also show the valid primitive events.

FIGURE 1 : SIMPLIFIED STATE DIAGRAM

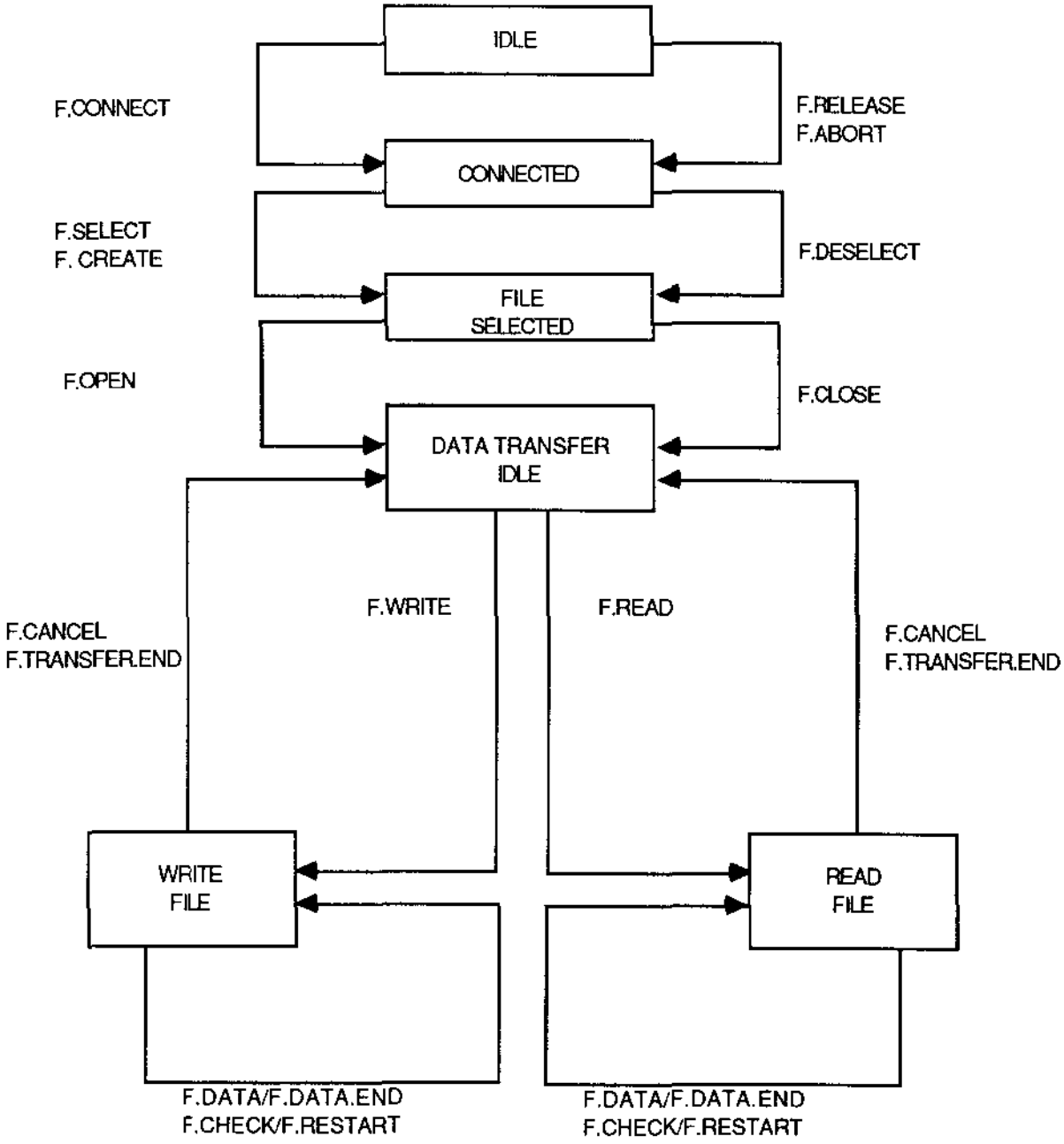
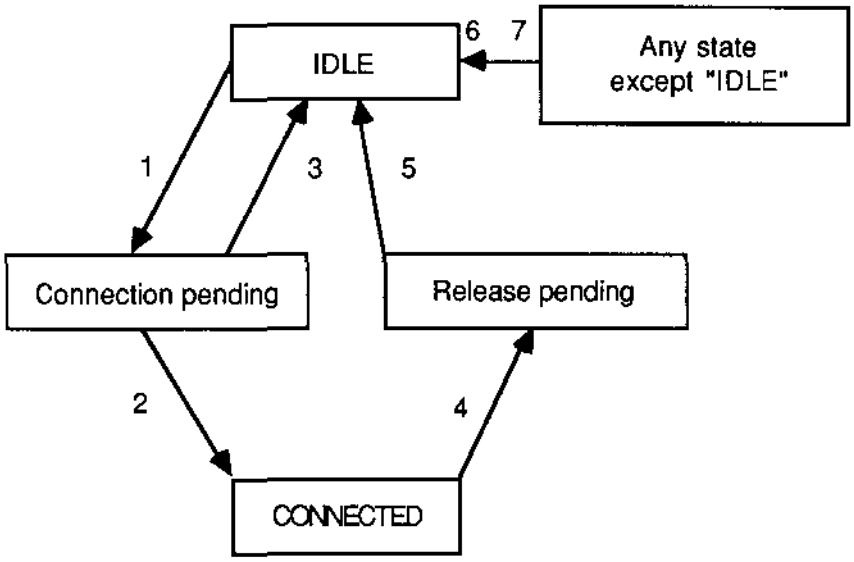


FIGURE 2 : STATE DIAGRAM
CONNECTION PHASE



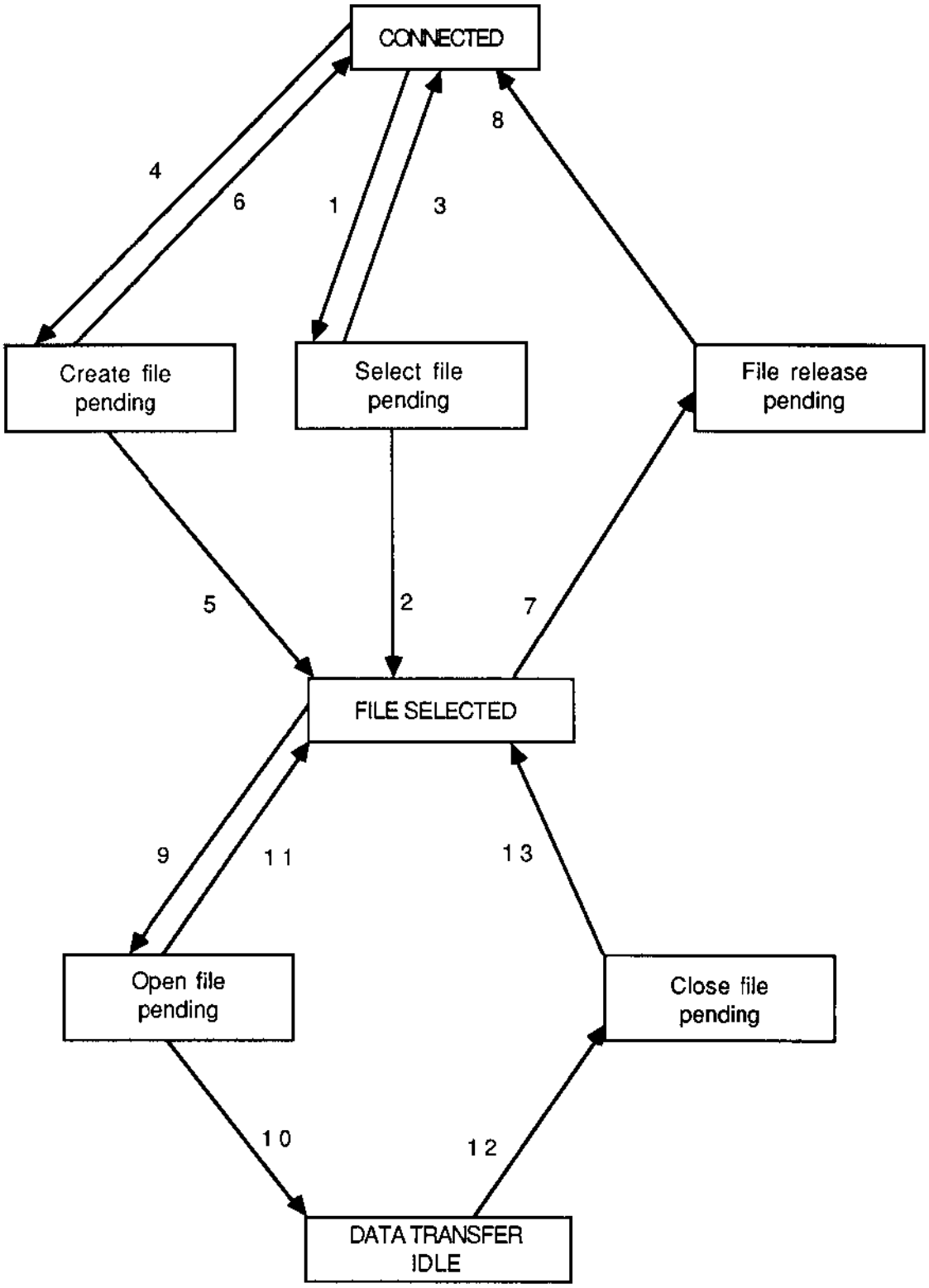
Caller transitions

1 F.CONNECT request
2 F.CONNECT confirmation (positive)
3 F.CONNECT confirmation (negative)
4 F.RELEASE request
5 F.RELEASE confirmation
6 F.ABORT request
7 F.ABORT indication

Server transitions

1 F.CONNECT indication
2 F.CONNECT response (positive)
3 F.CONNECT response (negative)
4 F.RELEASE indication
5 F.RELEASE response
6 F.ABORT request
7 F.ABORT indication

FIGURE 3 : STATE DIAGRAM
SELECTION AND FILE OPENING PHASES



TRANSITIONS (figure 3)	CALLER	SERVER
1	F.SELECT.request	F.SELECT.indication
2	F.SELECT.confirmation (+Ve)	F.SELECT.response (+Ve)
3	F.SELECT.confirmation (-Ve)	F.SELECT.response (-Ve)
4	F.CREATE.request	F.CREATE.indication
5	F.CREATE.confirmation (+Ve)	F.CREATE.response (+Ve)
6	F.CREATE.confirmation (-Ve)	F.CREATE.response (-Ve)
7	F.DESELECT.request	F.DESELECT.indication
8	F.DESELECT.confirmation	F.DESELECT.response
9	F.OPEN.request	F.OPEN.indication
10	F.OPEN.confirmation (+Ve)	F.OPEN.response (+Ve)
11	F.OPEN.confirmation (-Ve)	F.OPEN.response (-Ve)
12	F.CLOSE.request	F.CLOSE.indication
13	F.CLOSE.confirmation	F.CLOSE.response

Note : + Ve : positive
- Ve : negative

FIGURE 4 : CALLER STATE DIAGRAM
DATA TRANSFER PHASE

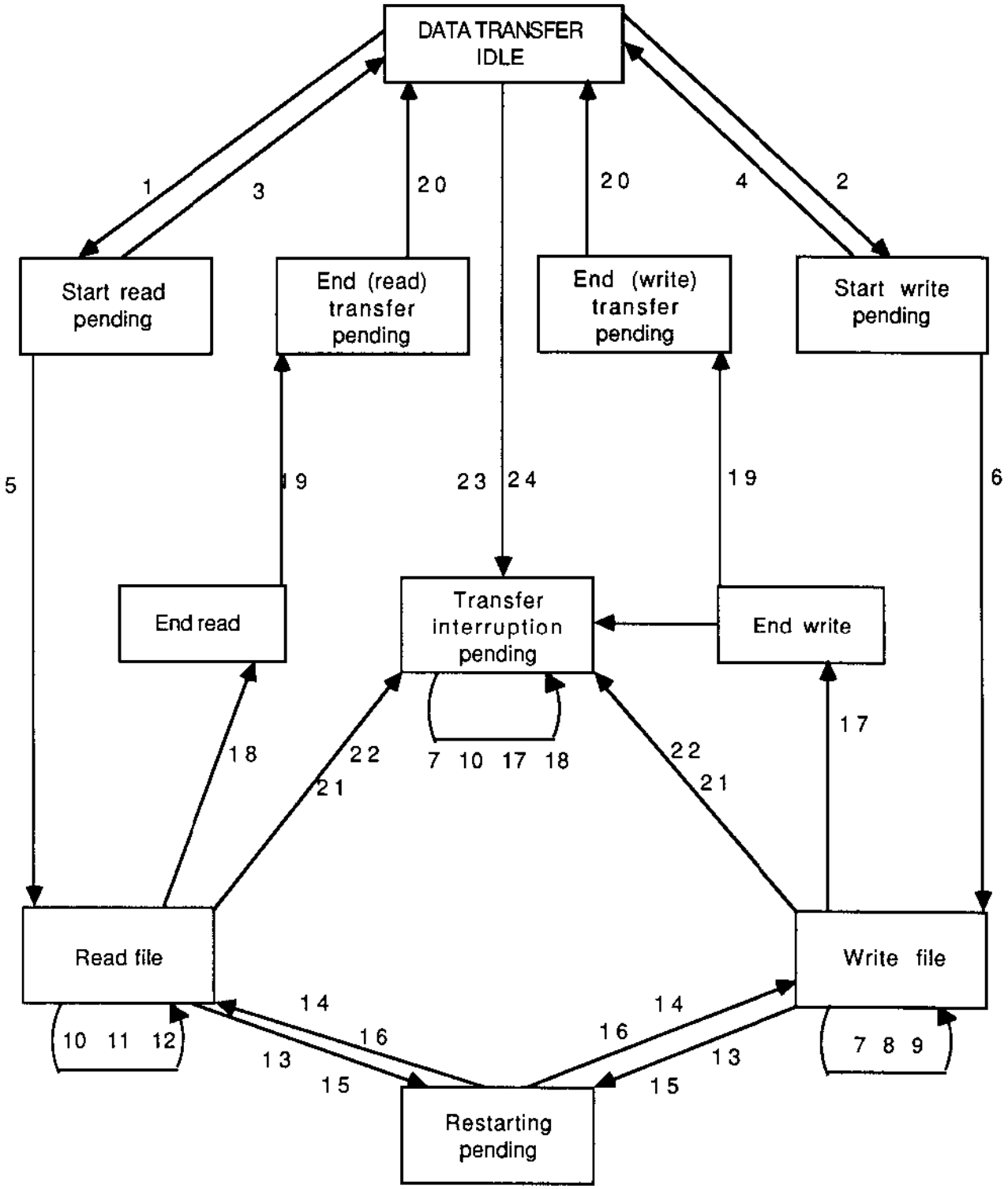
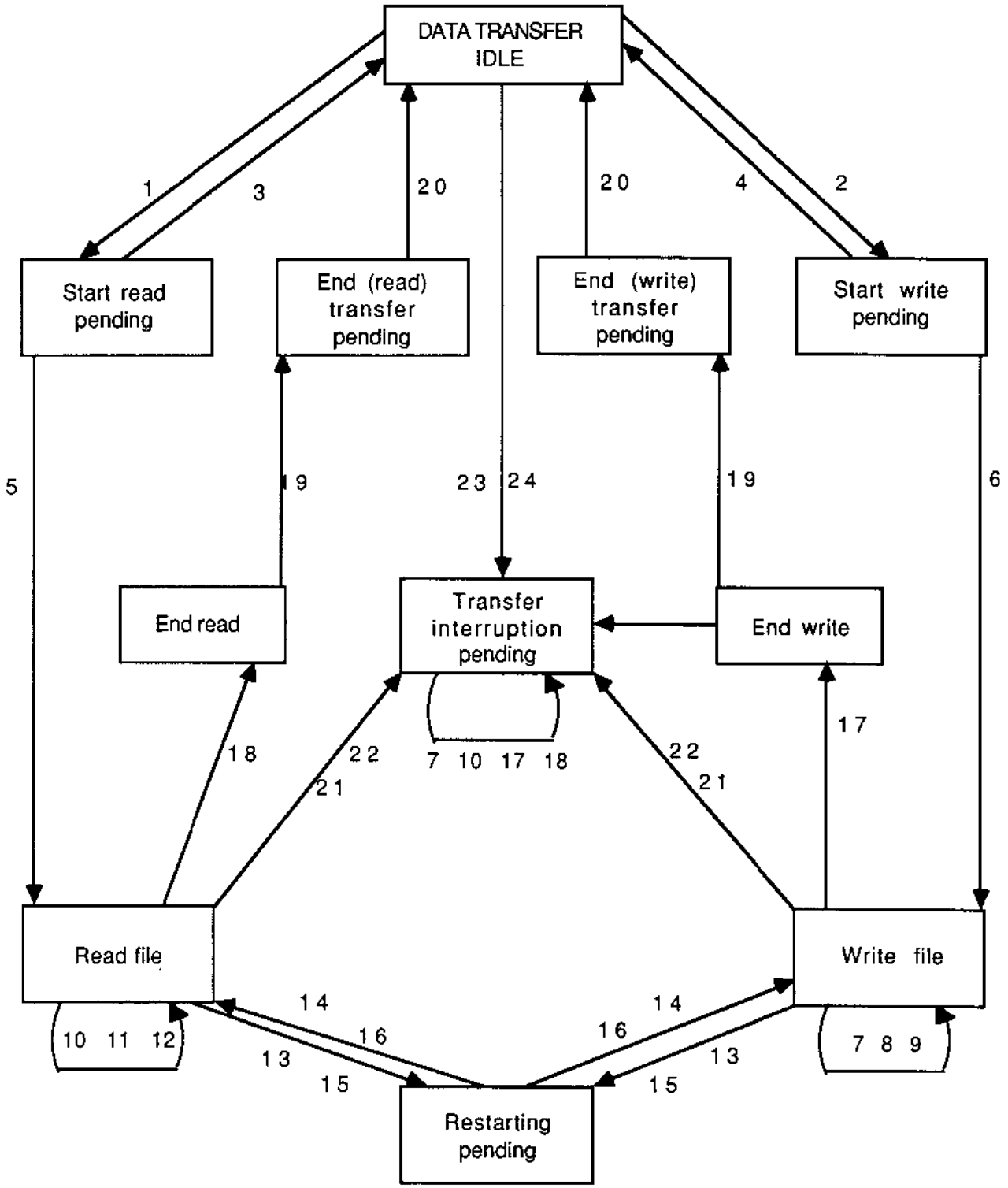


FIGURE 5 : SERVER STATE DIAGRAM
DATA TRANSFER PHASE



CALLER TRANSITIONS (figure 4)	SERVER TRANSITIONS (figure 5)
1 F.READ,D	F.READ,I
2 F.WRITE,D	F.WRITE,I
3 F.READ,C(-Ve)	F.READ,R (-Ve)
4 F.WRITE,C(-Ve)	F.WRITE,R (-Ve)
5 F.READ,C(+Ve)	F.READ,R (+Ve)
6 F.WRITE,C (+Ve)	F.WRITE,R (+Ve)
7 F.DATA,D	F.DATA,I
8 F.CHECK,D	F.CHECK,I
9 F.CHECK,C	F.CHECK,R
10 F.DATA,I	F.DATA,D
11 F.CHECK,I	F.CHECK,D
12 F.CHECK,R	F.CHECK,C
13 F.RESTART,D	F.RESTART,I
14 R.RESTART,C	F.RESTART,R
15 F.RESTART,I	F.RESTART,D
16 F.RESTART,R	F.RESTART,C
17 F.DATA.END,D	F.DATA.END,I
18 F.DATA.END,I	F.DATA.END,D
19 F.TRANSFER,END,D	F.TRANSFER.END,I
20 F.TRANSFER,END,C	F.TRANSFER.END,R
21 F.CANCEL,D	F.CANCEL,D
22 F.CANCEL,I	F.CANCEL,I
23 F.CANCEL,R	F.CANCEL,R
24 F.CANCEL,C	F.CANCEL,C

TABLE 1 : VALID CALLER SERVICE PRIMITIVE SEQUENCES

PRECEDED BY \ FOLLOWED BY	F.CONNECT.D	F.SELECT.D	F.OPEN.D	F.WRITE.D	F.DATA.D	F.DATA.END.D	F.TRANSFER.END.D	F.CLOSE.D	F.DESELECT.D	F.CANCEL.D	F.CANCEL.R	F.CREATE.D	F.CHECK.D	F.CHECK.R	F.RESTART.D	F.RESTART.R	F.READ.D	F.RELEASE.D	F.ABORT.D
F.CONNECT,D																			O
F.CONNECT,C		*										*						*	*
F.SELECT,D																			O
F.SELECT,C(+)			*						*										*
F.SELECT,C(-)		*										*						*	*
F.CREATE,D																			O
F.CREATE,C(+)			*						*										*
F.CREATE,C(-)		*										*						*	*
F.OPEN,D																			O
F.OPEN,C(+)				*				*									*		*
F.OPEN,C(-)			*						*										*
F.WRITE,D																			O
F.WRITE,C(+)					*	*				*					*				*
F.WRITE,C(-)				*				*											*
F.DATA,D					*	*				*			*		*				*
F.DATA.END,D							*								*				*
F.TRANSFER.END,D																			O
F.TRANSFER.END,C								*											*
F.READ,D																			O
F.READ,C(+)										*					*				*
F.READ,C(-)								*									*		*
F.DATA,I										*				*	*				*

Key : D request - I indication - R response - C confirmation - * possible - O possible (but confirmation of the preceeding primitive is awaited).

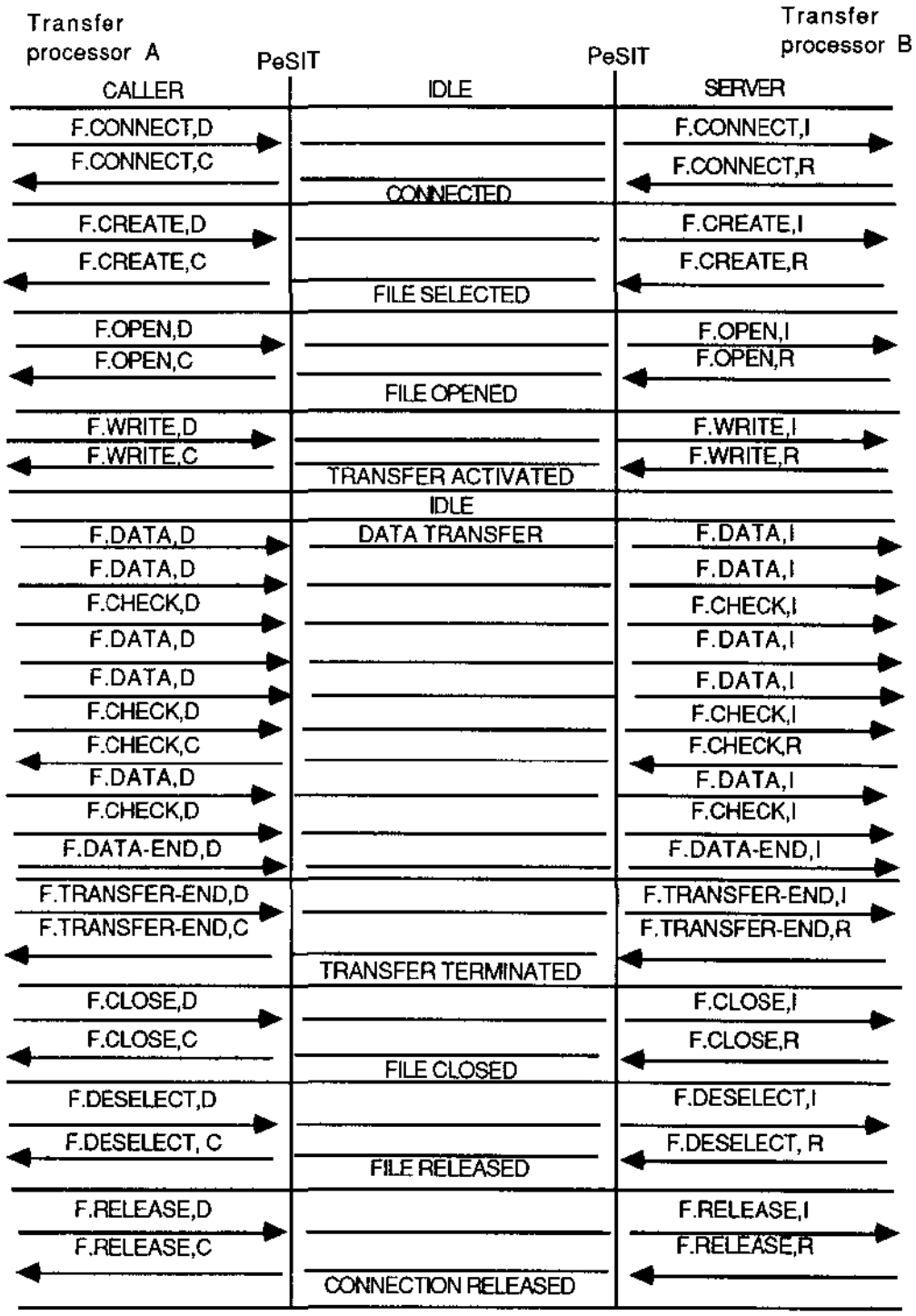
PRECEDED BY \ FOLLOWED BY	F.CONNECT.D	F.SELECT.D	F.OPEN.D	F.WRITE.D	F.DATA.D	F.DATA.END.D	F.TRANSFER.END.D	F.CLOSE.D	F.DESELECT.D	F.CANCEL.D	F.CANCEL.R.	F.CREATE.D	F.CHECK.D	F.CHECK.R	F.RESTART.D	F.RESTART.R	F.READ.D	F.RELEASE.D	F.ABORT.D
F.DATA.END,I														*	*				*
F.CANCEL,D																			O
F.CANCEL,I											*								*
F.CANCEL,R								*											*
F.CANCEL,C								*											*
F.CLOSE,D																			O
F.CLOSE, C			*						*										*
F.DESELECT,D																			O
F.DESELECT,C		*										*						*	*
F.RELEASE,D																			O
F.RELEASE,C	*																		
F.ABORT,D	*																		
F.ABORT,I	*																		
F.CHECK,D					*	*				*					*				*
F.CHECK,I										*				*	*				*
F.CHECK,R										*				*	*				*
F.CHECK,C					*	*				*					*				*
F.RESTART,D										O									O
F.RESTART,I										*						*			*
F.RESTART,R					*	*				*			*		*				*
F.RESTART,C					*	*				*			*		*				*

TABLE 2 : VALID SERVER SERVICE PRIMITIVE SEQUENCES

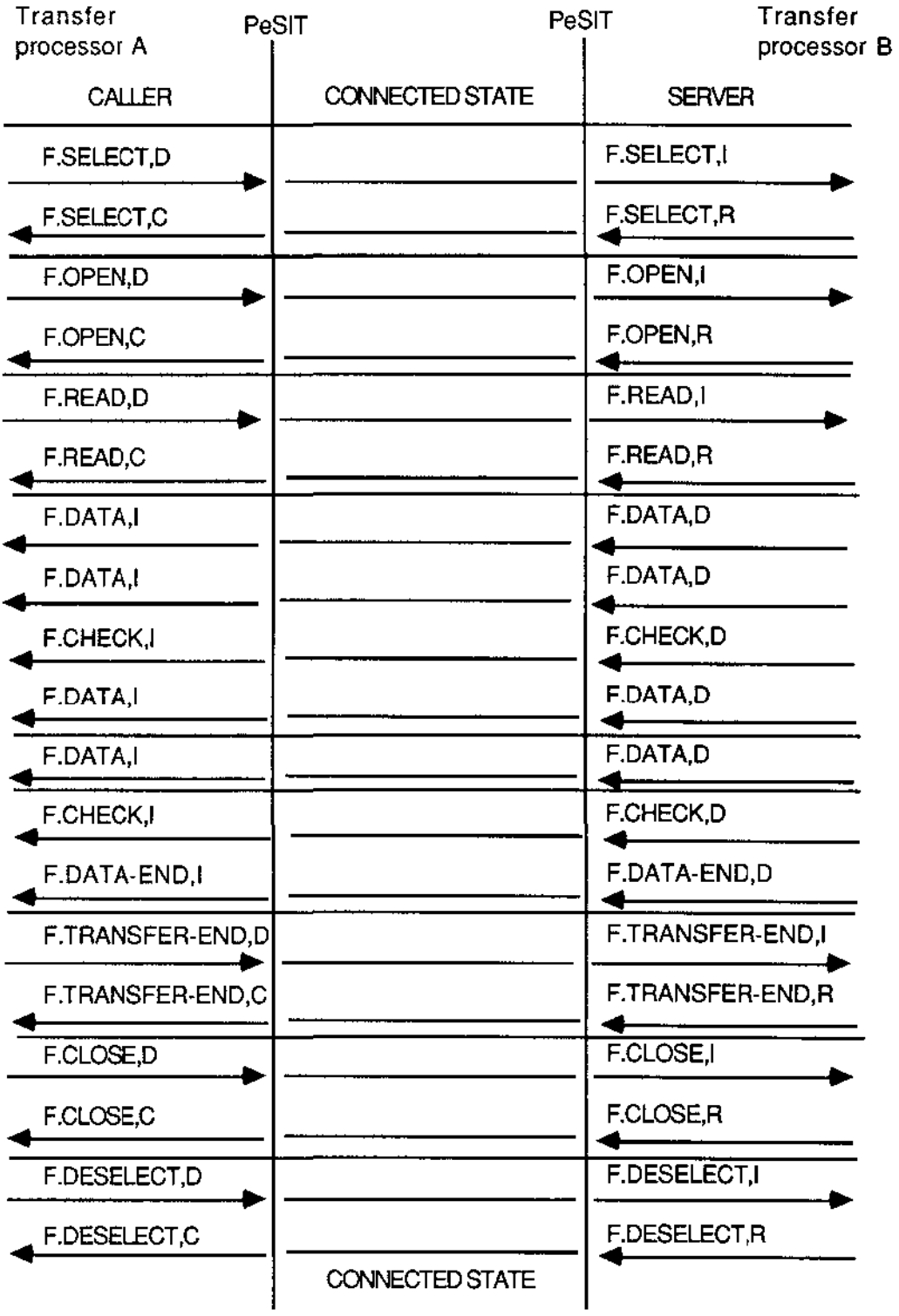
FOLLOWED BY / PRECEDED BY	F.CONNECT.R	F.SELECT.R	F.OPEN.R	F.WRITE.R	F.TRANSFER.END.R	F.CLOSE.R	F.DESELECT.R	F.CANCEL.D	F.CANCEL.R	F.CREATE.R	F.READ.R	F.DATA.D	F.DATA.END.D	F.CHECK.D	F.CHECK.R	F.RESTART.D	F.RESTART.R	F.RELEASE.R	F.ABORT.D
F.RELEASE,I																		*	*
F.RELEASE,R																			
F.ABORT,D																			
F.ABORT,I																			
F.CHECK,D								*				*	*			*			*
F.CHECK, I								*							*	*			*
F.CHECK,R								*							*	*			*
F.CHECK,C								*				*	*	*		*			*
F.RESTART,D								○											○
F.RESTART,I								*									*		*
F.RESTART,R								*				*	*	*		*			*
F.RESTART,C								*				*	*	*		*			*
F.CONNECT,I	*																		
F.CONNECT,R																			*
F.SELECT,I		*																	*
F.SELECT,R(+)																			*
F.SELECT,R(-)																			*
F.CREATE,I										*									*
F.CREATE,R(+)																			*
F.CREATE,R (-)																			*
F.OPEN,I			*																*
F.OPEN,R(+)																			*
F.OPEN,R(-)																			*
F.WRITE,I				*															*
F.WRITE,R(+)								*								*			*
F.WRITE,R(-)																			*
F.DATA.I								*							*	*			*
F.DATA.END,I															*	*			*

FOLLOWED BY / PRECEDED BY	F.CONNECT.R	F.SELECT.R	F.OPEN.R	F.WRITE.R	F.TRANSFER.END.R	F.CLOSE.R	F.DESELECT.R	F.CANCEL.D	F.CANCEL.R	F.CREATE.R	F.READ.R	F.DATA.D	F.DATA.END.D	F.CHECK.D	F.CHECK.R	F.RESTART.D	F.RESTART.R	F.RELEASE.R	F.ABORT.D
F.TRANSFER-END,I					*														*
F.TRANSFER-END,R																			*
F.READ,I											*								*
F.READ,R(+)								*				*	*	*		*			*
F.READ,R(-)																			*
F.DATA,D								*				*	*	*		*			*
F.DATA-END,D																*			*
F.CANCEL,D																			*
F.CANCEL,I									*										*
F.CANCEL,R																			*
F.CANCEL,C																			*
F.CLOSE,I						*													*
F.CLOSE,R																			*
F.DESELECT,I							*												*
F.DESELECT,R																			*

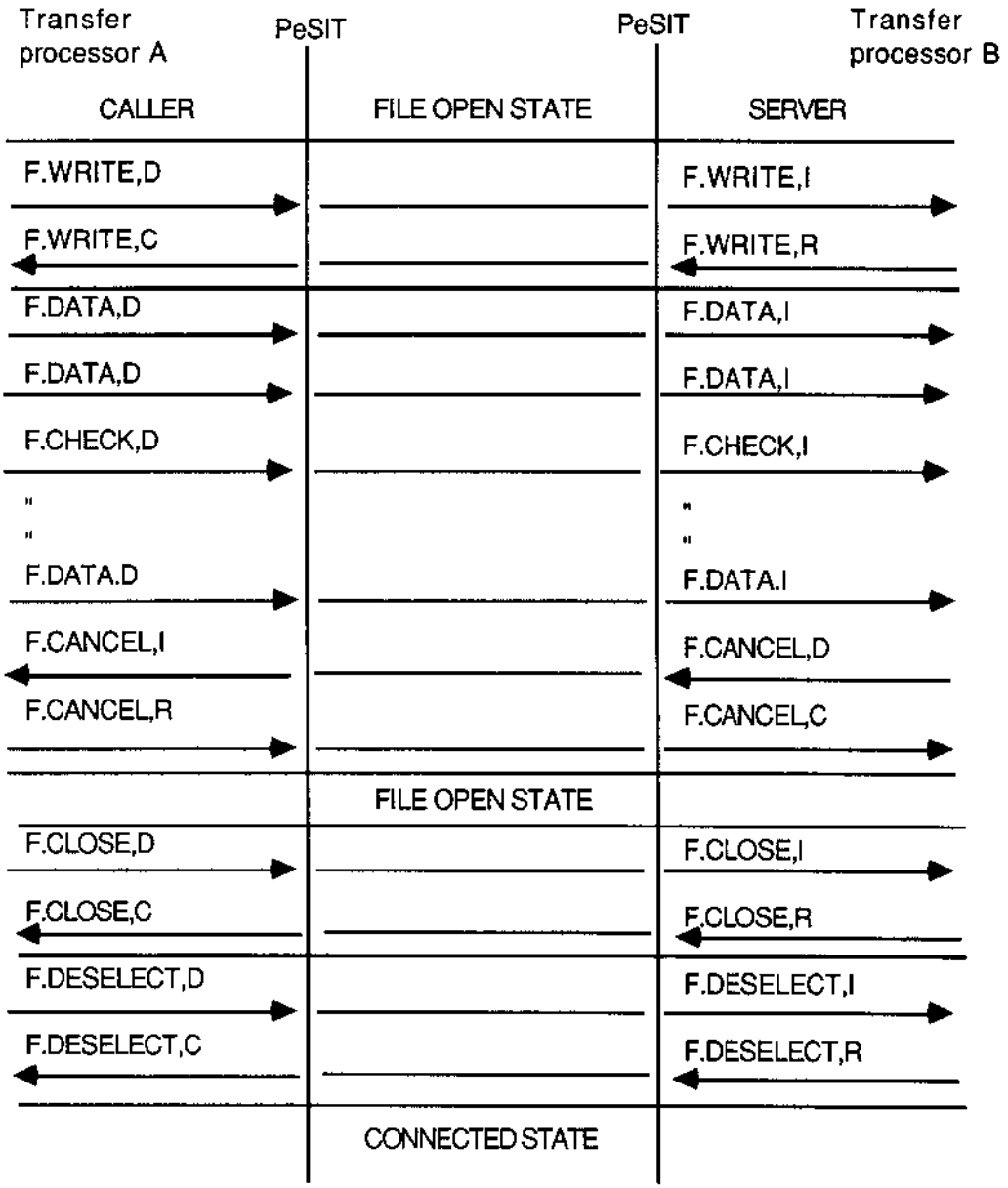
3.10.2 Normal sequence for a write transfer



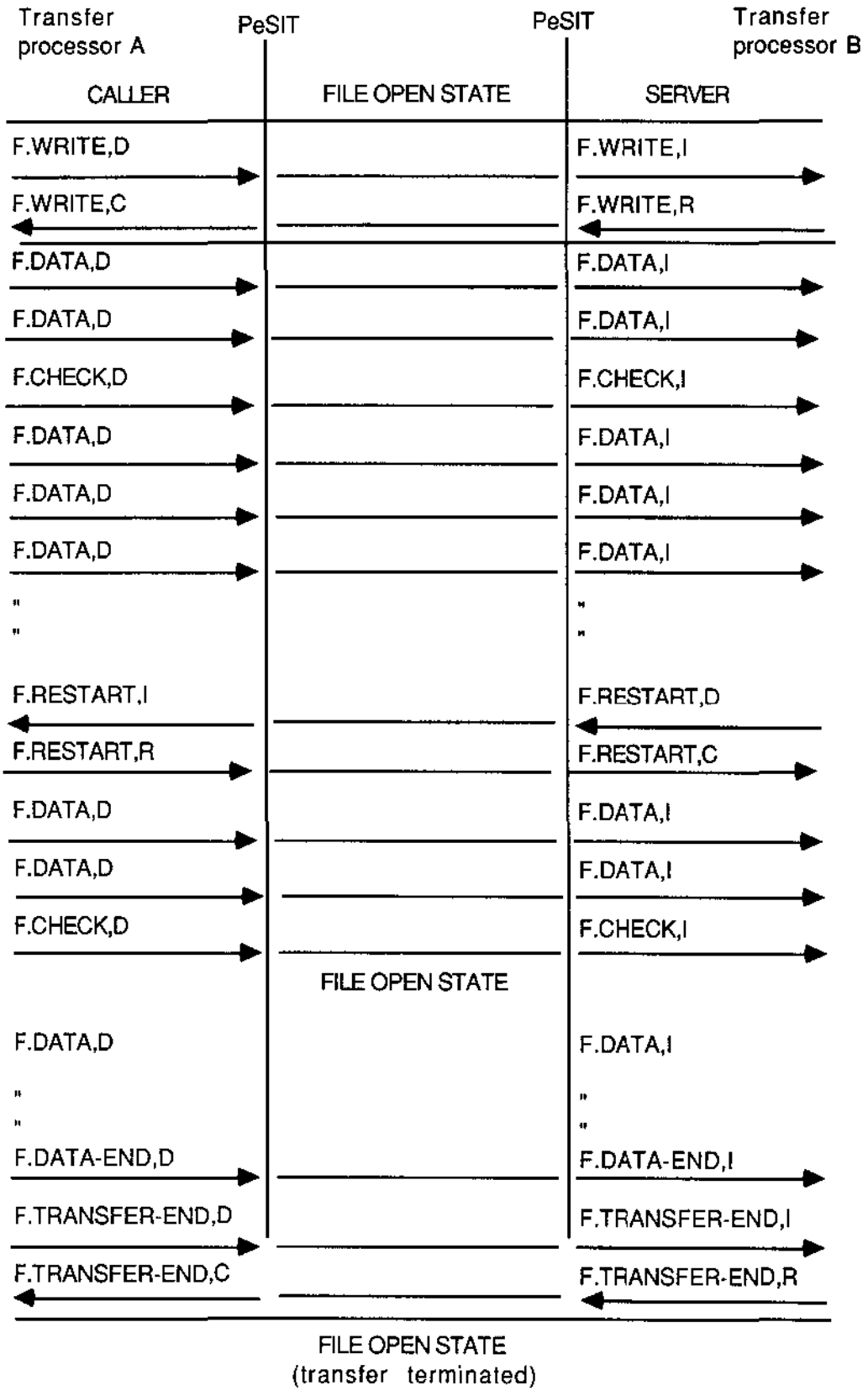
3.10.3 Normal sequence for a read transfer



3.10.4 Sequence with interruption of the file transfer



3.10.5 Sequence with restarting



CHAPTER 4
DESCRIPTION OF THE PeSIT PROTOCOL

4. DESCRIPTION OF THE PeSIT PROTOCOL

4.1 INTRODUCTION

The PeSIT protocol is conceptualised as an abstract machine in which messages (FPDU : File transfer Protocol Data Unit) are exchanged between two corresponding PeSIT units : the caller and the server. These messages contain a protocol specific header, a variable zone containing some PeSIT protocol management information (i.e. the parameters) and the file data. The variable zone and the file data may be absent from certain messages.

The complete description of the protocol is based on the following elements :

- the specification of the message transfer procedures between two PeSIT units,

- the specification and the coding of the protocol data units (FPDU).

These procedures are defined in terms of :

- interactions between corresponding PeSIT units, in terms of FPDUs exchanged,

- interactions between a PeSIT unit and the PeSIT service user on the same system, in terms of PeSIT service primitive exchanged,

- interactions between a PeSIT unit and the "Communication system" service provider, in terms of "Communication system" service primitives exchanged.

The description of the protocol being largely identical for PeSIT.F, PeSIT.F', PeSIT.F" and PeSIT.F"', the differences are indicated whenever necessary.

4.2 SERVICE AND PROTOCOL CORRESPONDANCE

The PeSIT protocol layer communicates with the user by means of primitives which were defined in the previous chapter (PeSIT service). The primitives cause or are the result of FPDU messages exchanged between two corresponding PeSIT units over a "Communication system" connection.

The following table gives a list of the FPDUs and their correspondance with the PeSIT service primitives. The meaning of the abreviations used is :

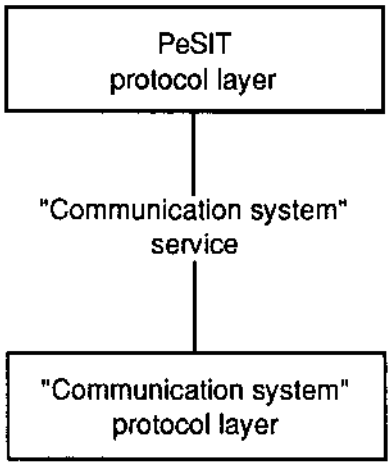
- DI : request/indication,

- RC : response/confirmation.

SERVICE	PRIMITIVES	RELATED FPDU	DEFINITION
PeSIT regime establishment	F. CONNECT, DI	FPDU.CONNECT	Regime establish. request
	F. CONNECT, RC positive	FPDU.ACONNECT	Regime establish. confirm
	F. CONNECT, RC negative	FPDU. RCONNECT	Regime establish. refusal
File selection and deselection	F.CREATE, DI	FPDU.CREATE	File creation
	F.CREATE, RC	FDPU.ACK(CREATE)	Creation confirm
	F.SELECT, DI	FPDU.SELECT	File selection
	F.SELECT, RC	FDPU.ACK(SELECT)	Selection confirmation
	F.DESELECT, DI	FPDU.DESELECT	File deselection
	F.DESELECT, RC	FDPU.ACK(DESELECT)	Deselection confirmation
	F.MESSAGE, DI	FPDU.MSG	Datagram
		FPDU.MSGDM	Segmented datagram : begin
		FPDU.MSGMM	Segmented datagram : current
		FPDU.MSGFM	Segmented datagram : end
	F.MESSAGE, RC	FPDU.ACK (MSG)	Datagram confirmation
File open and close	F.OPEN,DI	FPDU.ORF	File open
	F.OPEN, RC	FPDU.ACK(ORF)	Open confirmation
	F.CLOSE,DI	FPDU.CRF	File close
	F. CLOSE, RC	FPDU.ACK(CRF)	Close confirmation
File transfer begin and end	F.WRITE,DI	FPDU.WRITE	File write
	F.WRITE, RC	FPDU.ACK (WRITE)	Write confirmation
	F.READ,DI	FPDU.READ	File read
	F.READ, RC	FPDU.ACK (READ)	Read confirmation
	F. TRANSFER.END, DI	FPDU.TRANS.END	Transfer end
	F.TRANSFER.END, RC	FPDU.ACK(TRANS.END)	Transfer end confirmation
Bulk data transfer	F. DATA,DI	FPDU.DTF	File data
		FPDU.DTFDA	Segmented data : begin
		FPDU.DTFMA	Segmented data : current
		FPDU.DTFFA	Segmented data : end
	F.DATA-END, DI	FPDU.DTF.END	Data end
	F.CHECK, DI	FDPU.SYN	Checkpoint
	F.CHECK, RC	FDPU.ACK(SYN)	Checkpoint confirmation
	F.RESTART, DI	FPDU. RESYN	Restart
	F.RESTART, RC	FPDU.ACK (RESYN)	Restart confirmation
Transfer interruption	F. CANCEL, DI	FPDU.IDT	Transfert interrupt
	F. CANCEL, RC	FPDU.ACK (IDT)	Interrupt confirmation
Regime termination	F.RELEASE, DI	FPDU.RELEASE	Regime termination
	F. RELEASE, RC	FPDU.RELCONF	Termination confirmation
	F.ABORT, DI	FPDU.ABORT	Abrupt termination

4.3 USE OF THE "COMMUNICATION SYSTEM" SERVICE

This paragraph defines the way in which the "Communication system" service primitives are used by PeSIT.



The "Communication system" can be one of four types :

- ISO Session layer, used with a packet switching network,
- network layer (X25 or an alternative type),
- NETEX layer (when Hyperchannel is used),
- ISO session layer used with an ISO 8802-3 local area network.

4.3.1 Use of the Session service by PeSIT.F

The PeSIT.F file transfer protocol uses the ISO Session layer directly.

Usually the ISO Session layer is made up of a Kernel functional unit, which all implementations should offer, and eleven other functional units which may or may not be provided by a particular Session entity and whose usage is negotiated between two Session entities during the connection set-up phase.

For the PeSIT.F file transfer protocol the following functional units are required :

- kernel,

- half-duplex transmission (with the associated data token),

- typed data transfer.

The following table details which of the Session services are used.

SERVICE	FUNCTION	FUNCTIONAL UNIT
S-CONNECT	Session connection request	Kernel
S-ACCEPT	Session connection acceptance	Kernel
S-RELEASE	End of Session	Kernel
S-REFUSE	Session connection refusal	Kernel
S-U-ABORT	User break	Kernel
S-P-ABORT	Supplier break	Kernel
S-DATA	Normal data transfer	Kernel
S-TOKEN-GIVE	(Data) Token release	Half-duplex
S-TYPED-DATA	Typed data transfer	Typed data transfer

The PeSIT.F messages (FPDU) are transmitted using either the S-DATA Session service element, or as user data within the user service elements used.

The following table shows the Session service elements used to transmit the PeSIT.F messages (FPDU).

a) Association of a Session connection with a PeSIT.F connection.

A PeSIT connection is associated with a session connection set up for this purpose. The session connection is set up via the S-CONNECT primitive which transports the FPDU.CONNECT in the user data field. This primitive is always sent by the caller. During this phase the PeSIT protocol options (use of F.CHECK and F.RESTART) and the session functional units to be used whithin the session are negotiated.

Connection acceptance is provided by the S-ACCEPT primitive, which is always sent by the server.

If the connection cannot be set up, the refusal is indicated by the S-REFUSE primitive. The S-REFUSE primitive does not contain any user data so the FPDU.RCONNECT is inserted in the cause field which may contain apart from the cause code, up to 512 bytes of user data.

b) End of session connection

The session may terminate in one of three ways :

- normal termination which may only be invoked by the caller (connection limitation). This is provided by the S-RELEASE service.

- abnormal termination invoked by either of the PeSIT.F units. This is provided by the S-U-ABORT service. The user data field contains the FPDU.ABORT.

- spontaneous disconnection by the session protocol. This is provided by the S-P-ABORT service.

c) Normal data transmission

Nearly all the PeSIT.F protocol units (FPDU) are transported by the S-DATA service.

The dialogue is of the alternate bi-directional type (semi-duplex) the right to transmit being transfered, in either direction, by the S-TOKEN-GIVE service.

d) Checkpointing/restart

When the checkpointing option is chosen, the checkpoints are set by the emission of an FPDU.SYN, in the user data field of the S-DATA service. The acknowledgement of the checkpoint is made by the emission of an FPDU.ACK(SYN), in the information field of an S-TYPED-DATA service element.

If the restart option has been upheld, a restart request is made by sending an FPDU.RESYN, in the user data field of an S-DATA service element. The confirmation of the restart is made by sending an FPDU.ACK(RESYN), in the user data field of an S-DATA service element.

In case the PeSIT unit does not hold a data token, the emission of the FPDU.RESYN is provided by use of the S-TYPED-DATA service element.

e) Transmission of typed data

The S-TYPED-DATA service is used by PeSIT units who do not hold a data token to :

- interrupt the data transfer : emission of an FPDU.IDT (F.CANCEL service element),

- confirm checkpoints : emission of FPDU.ACK(SYN),

- request the restart of a transfer : emission of an FPDU.RESYN.

f) Restrictions imposed by the tokens on the use of the Session service

The rules concerning the use of typed data and the transfer of data tokens are defined in the preceding table. Collision cases are resolved by applying the following rules :

1 - the rules of paragraph 4.8.3,

2 - a unit which holds a data token and is waiting for an FPDU.ACK(IDT) or an FPDU.ACK(RESYN), should release it to the other unit with whom it is corresponding, by a token transfer without emission of an FPDU,

The last rule is provided for times when a unit has sent an FPDU.IDT or an FPDU.RESYN and then receives an FPDU with token, which should be ignored in application of the priority rules.

The following pictograms illustrate this rule and its use in a certain number of typical cases.

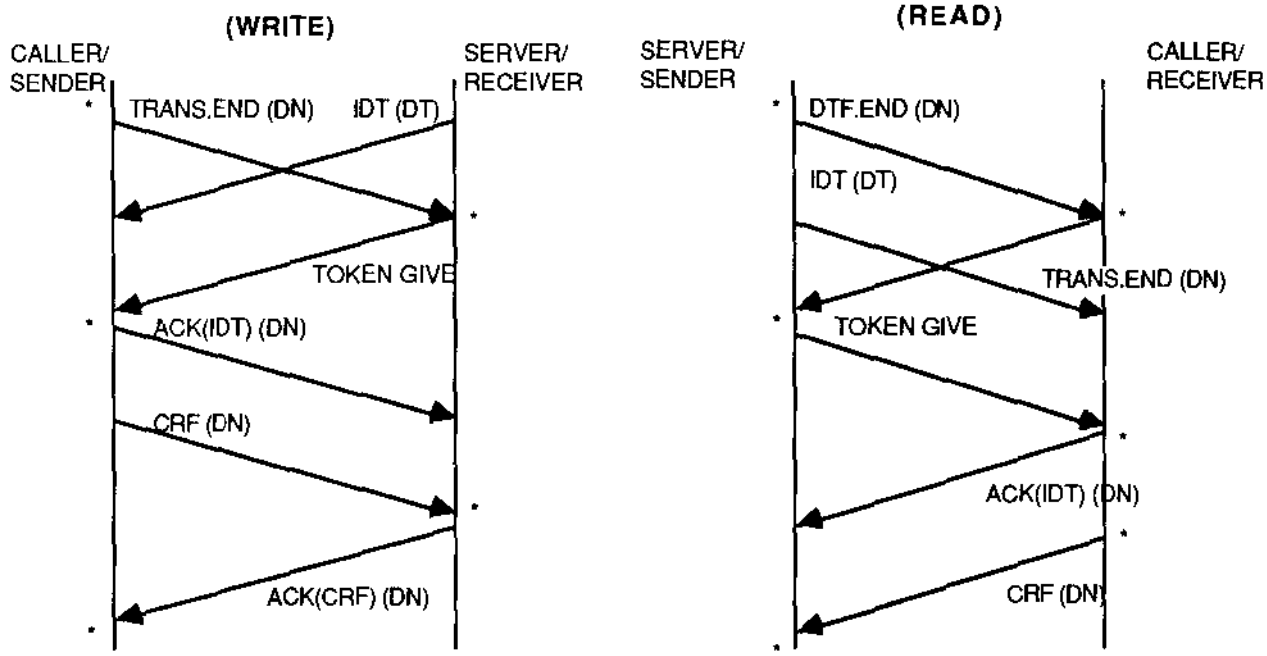
Conventions :

- ND : normal data

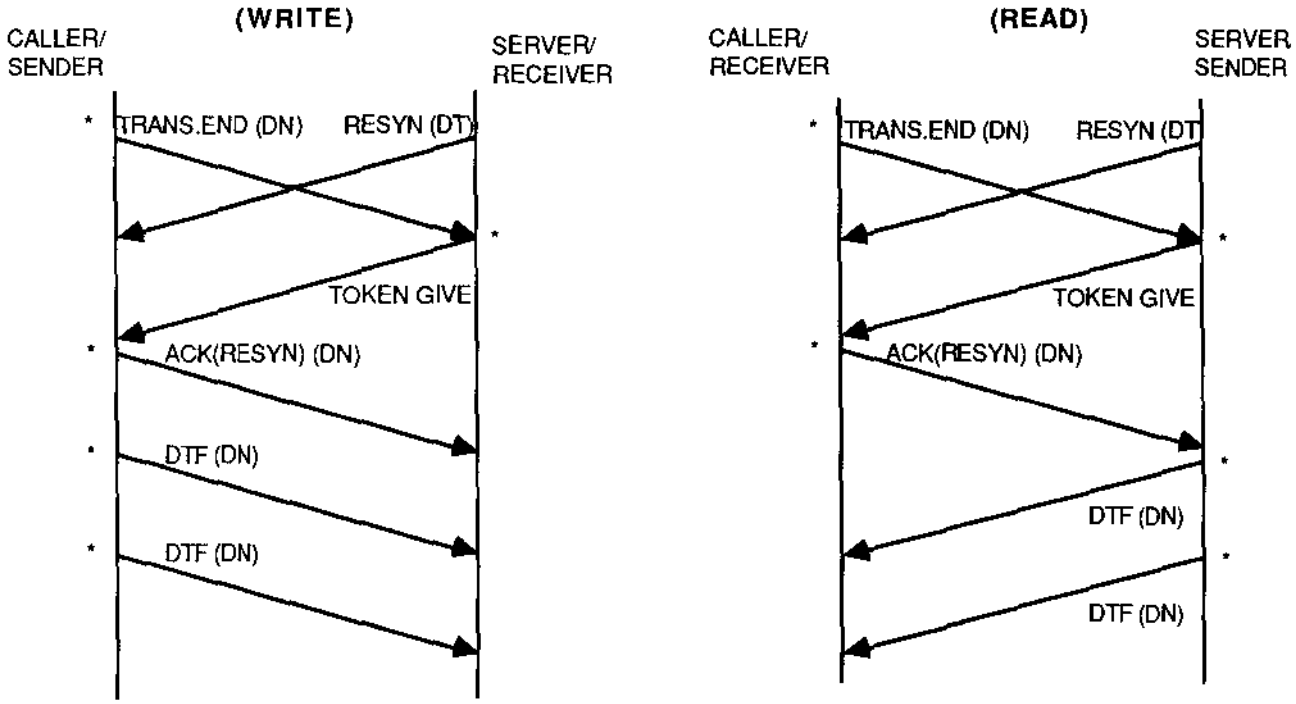
- TD : typed data

- * : token.

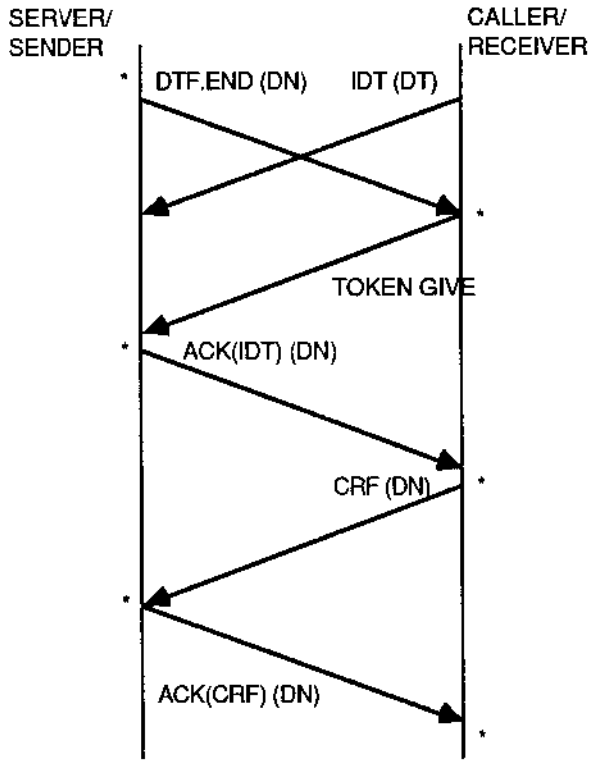
COLLISION TRANSFER.END/IDT



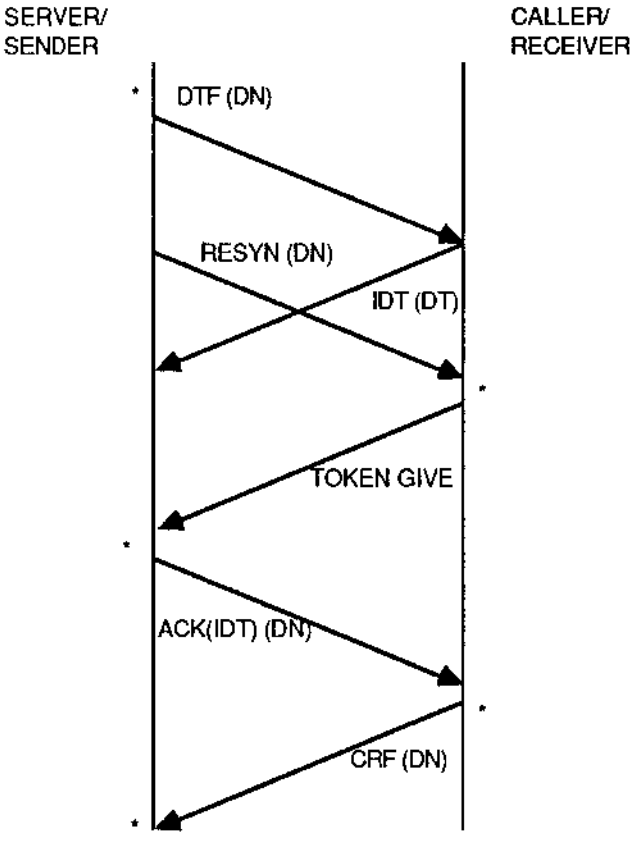
COLLISION TRANSFER.END/RESYN



COLLISION DTF.END IDT (READ)



COLLISION RESYN/IDT (READ)



4.3.2 Use of the Network service by PeSIT.F'

The PeSIT.F' protocol defined in this document relies on a level 3 network service.

Three different cases may be identified :

- use of a synchronous link in X25 packet mode,

- use of a telephone link in X32 mode,

- use of an asynchronous link (possibly via the PSTN) to access an X25 network via a PAD (Packet Assembler-Disassembler).

4.3.2.1 Use of a synchronous X.25 link

It is presumed that the network connection has been set up previously and that it is maintained throughout the PeSIT connection.

All the PeSIT.F' protocol units (FPDU) are transmitted using the N-DATA service.

The service indications provided by the network layer are interpreted in the following way :

- N-RESET-IND : emission of an FPDU-ABORT,

- N-DISC : emission of an F-ABORT-INDICATION,

- N-EXPEDITED-DATA : (interrupt packet) ignored by PeSIT.

4.3.2.2 Use of a dial-up X.32 link

The interface provided for a telephone link used in X32 mode is identical to that provided for a synchronous link used in X25 mode. The reactions of PeSIT.F' are therefore identical to those described above.

4.3.2.3 Use of an asynchronous link (PAD)

Two specific problems must be resolved when using an asynchronous link :

- protection against transmission errors on the terminal link,

- non transparent data transmission by the PAD.

a) protection against transmission errors on the terminal link

To forsee the possibility of transmission errors on the terminal link, an error control mechanism, using a polynomial calculation (CRC), may be used in PeSIT.F'.

The caller indicates in the FPDU.CONNECT that he will be using a CRC. All the succeding FPDUs (including the FPDU.CONNECT) will then be completed by a 16 bit CRC. The CRC is calculated on all the bytes of the FPDU including both the header and the parameters. The two bytes of the CRC are not included in the length field of the FPDU header.

The CRC calculation algorithm is the same as in the ISO Class 4 Transport protocol.

The receiver of an FPDU should check the validity of the CRC. If an FPDU with an incorrect CRC is detected, the receiver replies with either a FPDU.RESYNC (diagnostic code : transmission error) during the data transfer phase or an FPDU.ABORT (diagnostic code 310 : network incident) during any phase other than the data transfer phase.

In addition to this protection, the use of the "number of data bytes" (PI 27) and "number of articles" (PI 28) parameters in the FPDU.TRANS.END and FPDU.ACK(TRANS.END) eliminates the possibility of losing one or more FPDU.DTF.

b) PAD transparency

To eliminate the problem of the PAD being non transparent to certain control characters the transparent PAD profile is selected (Transpac profile 14).

This profile may be selected either prior to setting up the virtual circuit by use of a local command from the asynchronous DTE to the PAD, or after establishment of the virtual circuit by a transparent profile selection message sent by the synchronous X25 DTE to the PAD. In either case the selection of the PAD profile must be made prior to the FPDU.CONNECT being sent by the caller.

By choosing the Transpac profile 14 the asynchronous DTE still has to ability to return to command mode by sending a BREAK signal to the PAD.

4.3.3 Use of the Netex service by PeSIT.F"

The PeSIT.F" file transfer protocol relies on the session type interface provided by NETEX.

Within this interface PeSIT.F" uses the following primitives :

PRIMITIVE	FUNCTION
OFFER	Accept incoming calls
CONNECT	Connection request
CONFIRM	Accept connection
DISCONNECT	Connection break
CLOSE	Connection shut down
READ	Receive data
WRITE	Transmit data

The OFFER primitive is used by a PeSIT.F" unit which is prepared to act as a server to notify NETEX that it will accept incoming calls.

All the NETEX interface primitives authorise data emission to another correspondant. The maximum length transmitted at a time is negotiated during the connection set up phase. It is presumed to be always sufficient to transport the PeSIT.F" messages (FPDU).

The PeSIT.F" messages (FPDU) are therefore transmitted using either the WRITE primitive (for emission) and the READ primitive (for reception) or as user data within other primitives.

The following table lists the primitives used to transmit the PeSIT.F" messages (FPDU).

PeSIT.F" MESSAGES	TRANSPORTED BY	COMMENTS
FPDU.CONNECT	CONNECT	
FPDU.ACONNECT	CONFIRM	Provided that the PeSIT.F"server has sent an OFFER beforehand
FPDU.RCONNECT	DISCONNECT	
FPDU.RELEASE	CLOSE	
FPDU.RELCONF	CLOSE	
FPDU.ABORT	DISCONNECT	
FPDU.CREATE	READ/WRITE	
FPDU.ACK(CREATE)	READ/WRITE	
FPDU.SELECT	READ/WRITE	
FPDU.ACK(SELECT)	READ/WRITE	
FPDU.DESELECT	READ/WRITE	
FPDU.ACK(DESELECT)	READ/WRITE	
FPDU.MSG	READ/WRITE	
FPDU.MSGDM	READ/WRITE	
FPDU.MSGMM	READ/WRITE	
FPDU.MSGFM	READ/WRITE	
FPDU.ACK(MSG)	READ/WRITE	
FPDU.ORF	READ/WRITE	
FPDU.ACK(ORF)	READ/WRITE	
FPDU.CRF	READ/WRITE	
FPDU.ACK (CRF)	READ/WRITE	
FPDU.READ	READ/WRITE	
FPDU.ACK(READ)	READ/WRITE	
FPDU.WRITE	READ/WRITE	
FPDU.ACK(WRITE)	READ/WRITE	
FPDU.TRANSFER.END	READ/WRITE	
FPDU.ACK(TRANSFER.END)	READ/WRITE	
FPDU.DTF	READ/WRITE	
FPDU.DTFDA	READ/WRITE	
FPDU.DTFMA	READ/WRITE	
FPDU.DTFFA	READ/WRITE	
FPDU.DTF.END	READ/WRITE	
FPDU.SYN	READ/WRITE	
FPDU.ACK(SYN)	READ/WRITE	
FPDU.RESYN	READ/WRITE	
FPDU.ACK(RESYN)	READ/WRITE	
FPDU.IDT	READ/WRITE	
FPDU.ACK(IDT)	READ/WRITE	

a) Association of a NETEX connection with a PeSIT.F" connection.

A PeSIT connection is associated with a NETEX connection set up for this purpose. A PeSIT.F" which accepts to act as a server must use the OFFER primitive to indicate to NETEX that it will accept incoming calls. The session connection is set up via the CONNECT primitive which transports the FPDU.CONNECT. This primitive is always sent by the caller.

Connection acceptance is provided by sending an FPDU.ACONNECT in a CONFIRM primitive.

If the connection cannot be set up, the refusal is indicated by an FPDU.RCONNECT which is sent in a DISCONNECT primitive.

b) End of session connection

The session may terminate in one of three ways :

- normal termination invoked by the caller who sends an FPDU.RELEASE in a CLOSE primitive. The server replies with an FPDU.RELCONF in a CLOSE primitive.

- abnormal termination invoked by either of the PeSIT.F" units by sending an FPDU.ABORT in a DISCONNECT primitive.

- spontaneous disconnection by the NETEX layer. This is provided by a DISCONNECT primitive returned after a READ.

c) Normal data transmission

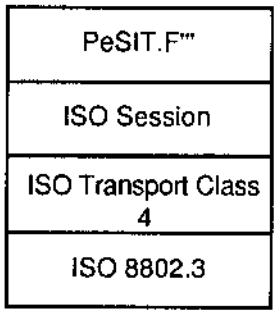
All the PeSIT.F" protocol units (FPDU) are sent and received by the READ and WRITE primitives. The length of data which may be transmitted by a READ or WRITE is fixed by the two units during the connection phase.

4.3.4 Use of the Session service on a local area network by PeSIT.F"'

The PeSIT.F"' file transfer protocol allows a local area network which complies to ISO 8802.3 (identical to IEEE 802.3) to be used as the "Communication system".

Since such a local area network does not provide a sufficiently reliable service, a Class 4 ISO Transport layer must be used above it (this class allows error detection and restarts upon errors). In order to provide as close a service as PeSIT.F , it has been decided to use an ISO Session layer above the Transport layer.

This defines the following architecture for PeSIT.F"' :



identical to PeSIT.F

Note : a local area network which conforms to ISO 8802.3 (or IEEE 802.3) is very similar to an Ethernet local area network (defined by Intel, Xerox and DEC). The only difference is in the frame header, the length field in the ISO 8802.3 standard corresponds with the type field of the Ethernet definition.

4.4 PROTOCOL UNIT (FPDU) SPECIFIC PROCEDURES

This chapter defines the valid sequences of PeSIT protocol elements.

Each FPDU message is described in the following manner :

- contents of the FPDU,

- transmission procedure for the FPDU,

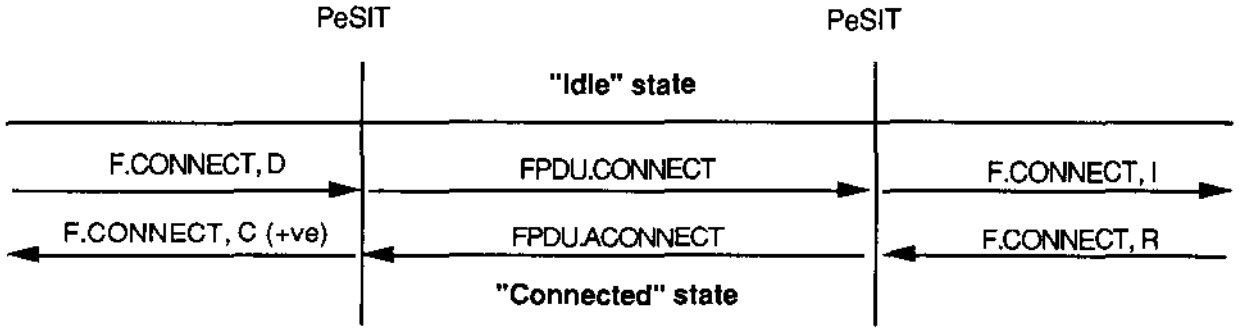
- receiving procedure for the FPDU.

An exchange diagram is provided for each phase.

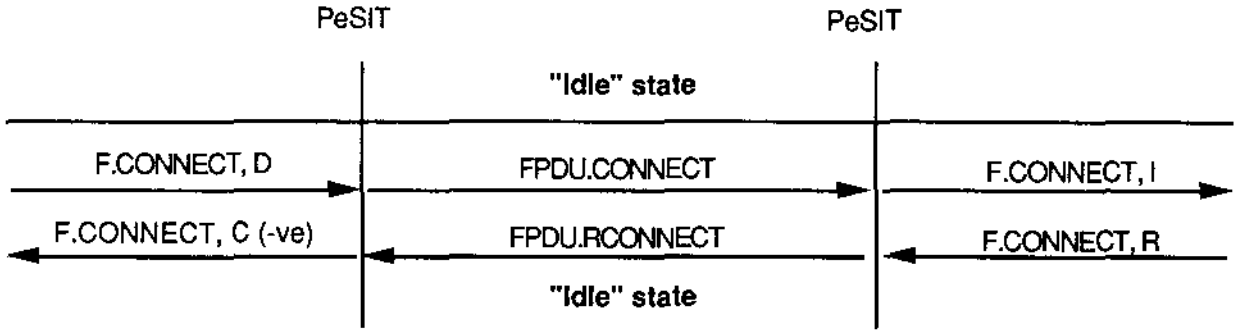
4.4.1 FPDU.CONNECT

The FPDU.CONNECT is sent by the PeSIT caller in the "Idle" state to set up a PeSIT connection, over a "Communication system" connection which has already been set up for the same PeSIT caller.

- Call acceptance



- Call refusal



a) Contents of the FPDU.CONNECT

The FPDU.CONNECT contains all the parameters of the F.CONNECT request primitive, plus :

- ID.SRC : FPDU originator's connection identification allocated by the calling PeSIT. Any non zero value.

- ID.DST : FPDU receiver's connection identification. Equal to zero in the FPDU.CONNECT.

b) Sending the FPDU.CONNECT

An F.CONNECT request primitive entails the allocation of a "Communication system" connection by PeSIT to the PeSIT connection. To set up this connection :

- PeSIT.F sends the FPDU.CONNECT as user data in the session connection request primitive S-CONNECT. PeSIT.F then passes into the "connection pending" state.

- PeSIT.F' sends an explicit FPDU.CONNECT in the normal data flow by an N-DATA primitive, after seting up the network connection. PeSIT.F' then passes into the "connection pending" state.

c) Receiving an FPDU.CONNECT

The reception of an FPDU.CONNECT validated by PeSIT in the "Idle" state causes the user defined by the "server identification" parameter of the FPDU.CONNECT to be notified via an F.CONNECT indication primitive.

The PeSIT server waits for an F.CONNECT response primitive from the PeSIT service user requested whilst in the "connection pending" state.

4.4.2 FPDU.ACONNECT

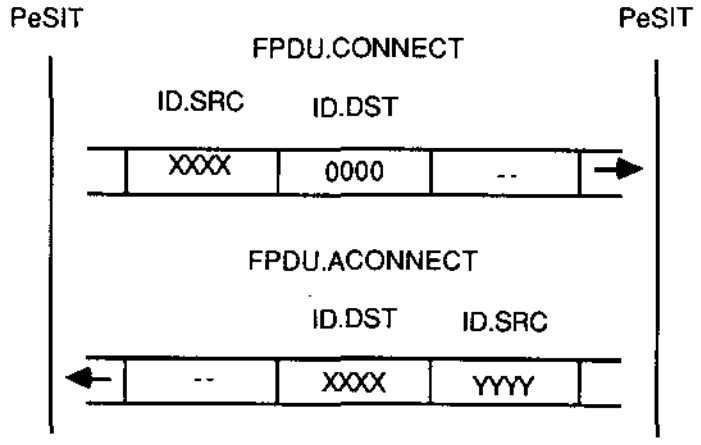
A PeSIT unit which receives an FPDU.CONNECT may accept the connection request by sending an FPDU.ACONNECT in reply to the requesting PeSIT caller, via the same "Communication system" connection.

a) Contents of the FPDU.ACONNECT

The FPDU.ACONNECT contains all the parameters of the F.CONNECT response primitive, plus :

- ID.SRC : FPDU originator's connection identification allocated by the PeSIT server in the FPDU.ACONNECT. Any non zero value.

- ID.DST : FPDU receiver's connection identification. Equal to the ID.SRC parameter received in the FPDU.CONNECT.



b) Sending the FPDU.ACONNECT

The F.CONNECT (positive) response primitive provokes the emission of an FPDU.ACONNECT in the user data field of an S-ACCEPT session primitive for PeSIT.F or in the normal data stream.

The connection is now established and the PeSIT server unit attains the "CONNECTED" state and may receive any of the service requests or FPDUs authorised by the server procedures.

c) Receiving an FPDU.ACONNECT

The reception of an FPDU.ACONNECT validated by the calling PeSIT whilst in the "connection pending" state causes an F.CONNECT confirmation primitive to be notified to the calling user. Since the connection has been successfully established the calling PeSIT unit attains the "CONNECTED" state and may receive any service requests or FPDUs authorised by the caller procedure.

4.4.3 FPDU.RCONNECT

The FPDU.RCONNECT is used by the called PeSIT unit to refuse an attempt to establish a PeSIT connection.

a) Contents of the FPDU.RCONNECT

The FPDU.RCONNECT contains all the parameters of the F.CONNECT response primitive, plus :

- ID.SRC : FPDU originator's connection identification allocated by the PeSIT server. Value equals zero since the connection is refused.

- ID.DST : FPDU receiver's connection identification. Equal to the ID.SRC parameter received in the FPDU.CONNECT.

b) Sending the FPDU.RCONNECT

The F.CONNECT (refusal) response primitive provokes the emission of an FPDU.RCONNECT in the user data field of an S-REFUSE session primitive for a PeSIT.F server or in the normal data stream by N-DATA for a PeSIT.F' server. No PeSIT connection is established.

c) Receiving an FPDU.RCONNECT

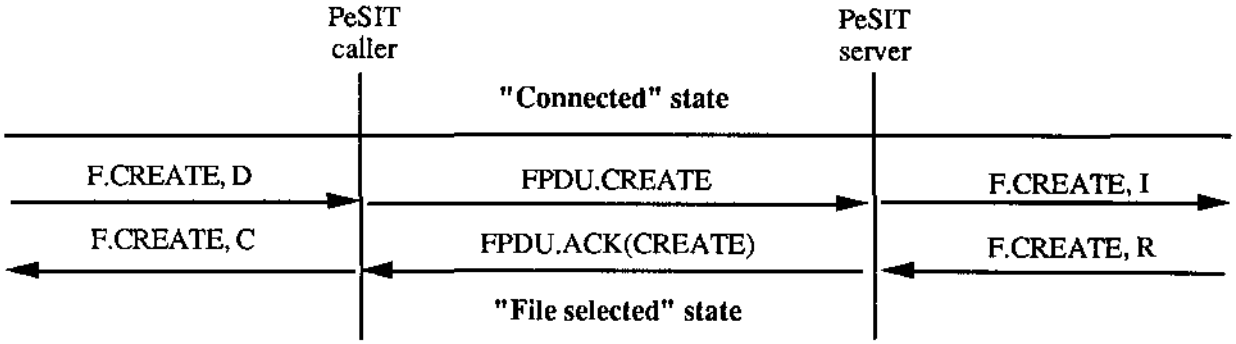
The reception of an FPDU.RCONNECT validated by the calling PeSIT whilst in the "connection pending" state causes an F.CONNECT (refusal) confirmation primitive to be notified to the calling user and :

- the PeSIT caller returns to the "Idle" state,

- the PeSIT.F caller requests a network service disconnection (N-DISCONNECT) and returns to the "Idle" state.

4.4.4 FPDU.CREATE

The FPDU.CREATE is always sent by the PeSIT caller, once in the "connected" state, when it wishes to create a file upon the corresponding PeSIT server machine in order to carry out a write file transfer. The file is completely identified by the "file identifier" parameter.



a) Contents of the FPDU.CREATE

The FPDU.CREATE contains all the parameters of the F.CREATE primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.CREATE

An F.CREATE request primitive provokes the emission by the PeSIT unit of an FPDU.CREATE in the normal data stream of the "communication system". The calling PeSIT unit attains the "File creation pending" state.

c) Receiving an FPDU.CREATE

The reception of an FPDU.CREATE validated by the PeSIT server in the "connected" state provokes a notification by an F.CREATE indication primitive to be sent to the server user. The PeSIT unit attains the "file creation pending" state.

4.4.5 FPDU.ACK(CREATE)

The FPDU.ACK(CREATE) is sent by the PeSIT server whilst in the "file creation pending" state to indicate the acceptance or refusal of a file creation for the transfer. The "diagnostic" parameter in the FPDU indicates the exact reason for a refusal if the file creation was not possible.

a) Contents of the FPDU.ACK(CREATE)

The FPDU.ACK(CREATE) contains all the parameters of the F.CREATE response primitive, plus :

ID.DST : FPDU receiver's connection identification.

c) Receiving an FPDU.SELECT

The reception of an FPDU.SELECT validated by the PeSIT server in the "connected" state provokes a notification by an F.SELECT indication primitive to be sent to the server user. The PeSIT unit attains the "file selection pending" state.

4.4.7 FPDU.ACK(SELECT)

The FPDU.ACK(SELECT) is sent by the PeSIT server whilst in the "file creation pending" state to indicate the acceptance or refusal of a file creation for the transfer. The "diagnostic" parameter in the FPDU indicates the exact reason for a refusal if the file creation was not possible.

a) Contents of the FPDU.ACK(SELECT)

The FPDU.ACK(SELECT) contains all the parameters of the F.SELECT response primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.ACK(SELECT)

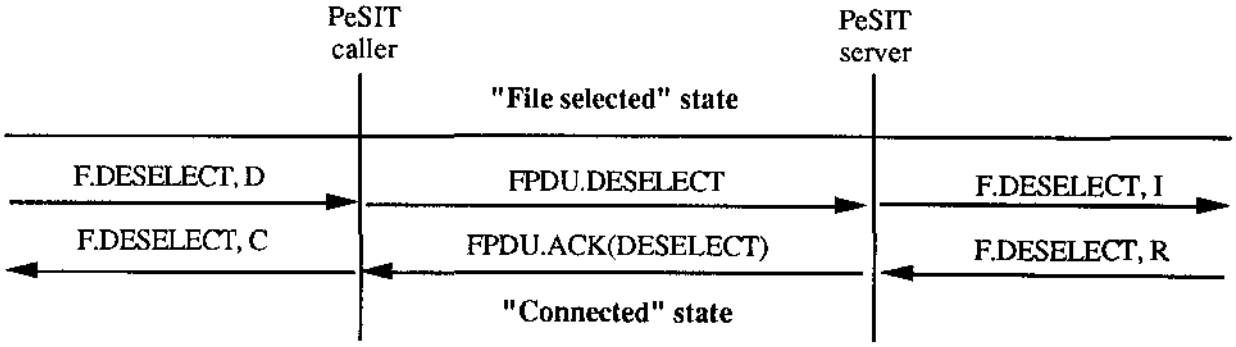
An F.SELECT response primitive provokes the emission by the PeSIT server whilst in the "file selection pending" state of an FPDU.ACK(SELECT) in the normal data stream of the "communication system". If the "diagnostic" parameter of the F.SELECT response primitive indicates "success", the PeSIT server attains the "file selected" state. Otherwise, the PeSIT server returns to the "CONNECTED" state.

c) Receiving an FPDU.ACK(SELECT)

The reception of an FPDU.ACK(SELECT) validated by the PeSIT caller in the "file selection pending" state provokes a notification by an F.SELECT confirmation primitive to be sent to the caller user. If the "diagnostic" parameter indicates "success", the PeSIT caller attains the "file selected" state, otherwise it returns to the "CONNECTED" state.

4.4.8 FPDU.DESELECT

The FPDU.DESELECT is always sent by the PeSIT caller whilst in the "file selected" state to request the release of a file which was previously selected for the transfer.



a) Contents of the FPDU.DESELECT

The FPDU.DESELECT contains all the parameters of the F.DESELECT request primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.DESELECT

An F.DESELECT request primitive provokes the emission by the PeSIT caller whilst in the "file selected" state of an FPDU.DESELECT in the normal data stream of the "communication system". The PeSIT unit attains the "File release pending" state.

c) Receiving an FPDU.DESELECT

The reception of an FPDU.DESELECT validated by the PeSIT server in the "file selected" state provokes a notification by an F.DESELECT indication primitive to be sent to the server user. The PeSIT unit attains the "file release pending" state.

4.4.9 FPDU.ACK(DESELECT)

The FPDU.ACK(DESELECT) is always sent by the PeSIT server whilst in the "file release pending" state to indicate the result of the execution of the file release request. The "diagnostic" parameter of the FPDU indicates the exact reason in case of a failure.

a) Contents of the FPDU.ACK(DESELECT)

The FPDU.ACK(DESELECT) contains all the parameters of the F.DESELECT response primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.ACK(DESELECT)

An F.DESELECT response primitive provokes the emission by the PeSIT server whilst in the "file selection pending" state of an FPDU.ACK(DESELECT) in the normal data stream of the "communication system". The PeSIT server returns to the "connected" state.

c) Receiving an FPDU.ACK(DESELECT)

The reception of an FPDU.ACK(DESELECT) validated by the PeSIT caller in the "file release pending" state provokes a notification by an F.DESELECT confirmation primitive to be sent to the caller user. and the PeSIT returns to the "connected" state.

4.4.10 FPDU.ORF

The FPDU.ORF is always sent by the PeSIT caller whilst in the "file selected" state to request that the distant file be opened.

a) Contents of the FPDU.ORF

The FPDU.ORF contains all the parameters of the F.ORF resquest primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.ORF

An F.OPEN request primitive provokes the emission by the PeSIT caller whilst in the "file selected" state of an FPDU.ORF in the normal data stream of the "communication system". The PeSIT unit attains the "File open pending" state.

c) Receiving an FPDU.ORF

The reception of an FPDU.ORF validated by the PeSIT server in the "file selected" state provokes a notification by an F.OPEN indication primitive to be sent to the server user. The PeSIT unit attains the "file open pending" state.

4.4.11 FPDU.ACK(ORF)

The FPDU.ACK(ORF) is always sent by the PeSIT server whilst in the "file open pending" state to indicate the result of the file open request. The "diagnostic" parameter of the FPDU indicates the exact reason in case of a failure.

a) Contents of the FPDU.ACK(ORF)

The FPDU.ACK(ORF) contains all the parameters of the F.OPEN response primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.ACK(ORF)

An F.OPEN response primitive provokes the emission by the PeSIT server of an FPDU.ACK(ORF) in the normal data stream of the "communication system". The PeSIT server attains the state:

- of "data transfer - idle" if the "diagnostic" indicates "success",

- of "file selected" in all other cases.

c) Receiving an FPDU.ACK(ORF)

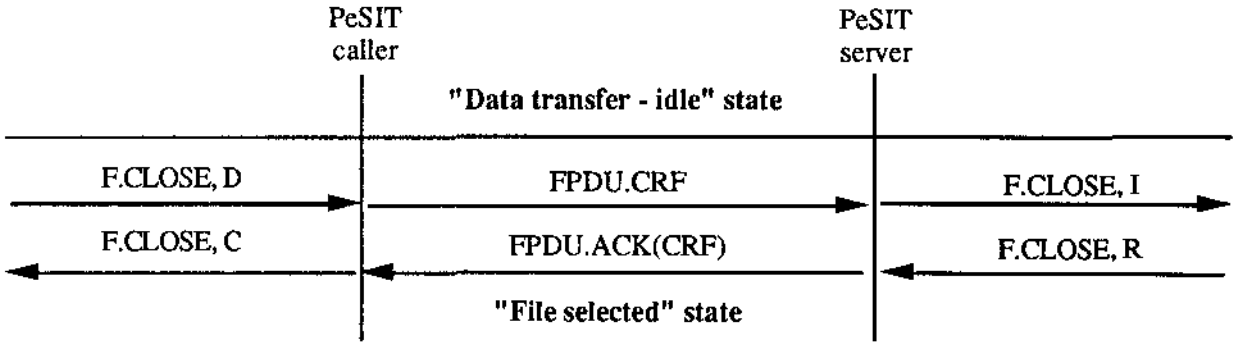
The reception of an FPDU.ACK(ORF) validated by the PeSIT caller in the "file open pending" state provokes a notification by an F.OPEN confirmation primitive to be sent to the caller user. and the PeSIT attains the state :

- of "data transfer - idle" if the "diagnostic" indicates "success",

- of "file selected" in all other cases.

4.4.12 FPDU.CRF

The FPDU.CRF is always sent by the PeSIT caller whilst in the "data transfer - idle" state to request file closure.



a) Contents of the FPDU.CRF

The FPDU.CRF contains all the parameters of the F.CLOSE request primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.CRF

An F.CLOSE request primitive provokes the emission by the PeSIT caller whilst in the "data transfer - idle" state of an FPDU.CRF in the normal data stream of the "communication system". The PeSIT unit attains the "File close pending" state.

c) Receiving an FPDU.CRF

The reception of an FPDU.CRF validated by the PeSIT server in the "data transfer - idle" state provokes a notification by an F.CLOSE indication primitive to be sent to the server user. The PeSIT unit attains the "file close pending" state.

4.4.13 FPDU.ACK(CRF)

The FPDU.ACK(CRF) is always sent by the PeSIT server whilst in the "file close pending" state to indicate the result of the file close request. The "diagnostic" parameter of the FPDU indicates the exact reason in case of a failure.

a) Contents of the FPDU.ACK(CRF)

The FPDU.ACK(CRF) contains all the parameters of the F.CLOSE response primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.ACK(CRF)

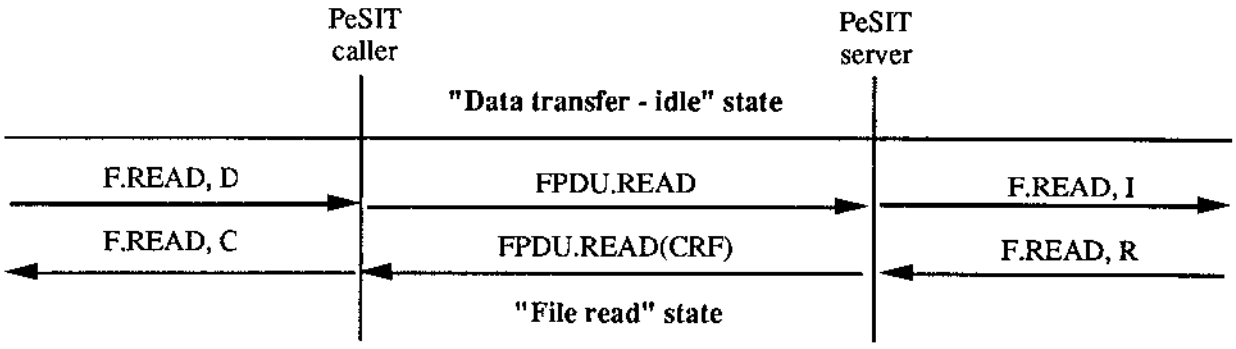
An F.CLOSE response primitive provokes the emission by the PeSIT server of an FPDU.ACK(CRF) in the normal data stream of the "communication system". The PeSIT server returns to the "file selected" state.

c) Receiving an FPDU.ACK(CRF)

The reception of an FPDU.ACK(CRF) validated by the PeSIT caller in the "file close pending" state provokes a notification by an F.CLOSE confirmation primitive to be sent to the caller user and the PeSIT returns to the "file selected" state.

4.4.14 FPDU.READ

The FPDU.READ is always sent by the PeSIT caller whilst in the "data transfer - idle" state to request the beginning of a read data transfer. It is during this phase that the recovery point is negotiated for a recovered transfer.



a) Contents of the FPDU.READ

The FPDU.READ contains all the parameters of the F.READ request primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.READ

An F.READ request primitive provokes the emission by the PeSIT caller whilst in the "data transfer - idle" state of an FPDU.READ in the normal data stream of the "communication system". The PeSIT unit attains the "File read pending" state.

c) Receiving an FPDU.READ

The reception of an FPDU.READ validated by the PeSIT server in the "data transfer - idle" state provokes a notification by an F.READ indication primitive to be sent to the server user. The PeSIT unit attains the "file read pending" state.

4.4.15 FPDU.ACK(READ)

The FPDU.ACK(READ) is always sent by the PeSIT server whilst in the "file read pending" state to indicate the result of the file read request. The "diagnostic" parameter of the FPDU indicates the exact reason in case of a failure.

a) Contents of the FPDU.ACK(READ)

The FPDU.ACK(READ) contains all the parameters of the F.READ response primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.ACK(READ)

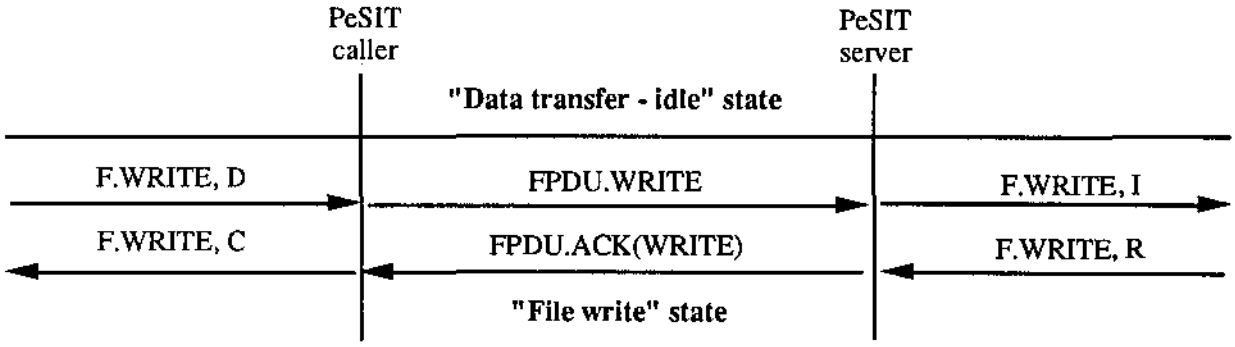
An F.READ response primitive provokes the emission by the PeSIT server of an FPDU.ACK(READ) in the normal data stream of the "communication system". The PeSIT server attains the "read file" state if the "diagnostic" code indicates "success", otherwise it returns to the "data transfer - idle" state.

c) Receiving an FPDU.ACK(READ)

The reception of an FPDU.ACK(READ) validated by the PeSIT caller in the "file read pending" state provokes a notification by an F.READ confirmation primitive to be sent to the caller user. and the PeSIT attains the "read file" state if the "diagnostic" code indicates "success", otherwise it returns to the "data transfer - idle" state.

4.4.16 FPDU.WRITE

The FPDU.WRITE is always sent by the PeSIT caller whilst in the "data transfer - idle" state to request the beginning of a write data transfer. It is during this phase that the recovery point is negotiated for a recovered transfer.



a) Contents of the FPDU.WRITE

The FPDU.WRITE contains all the parameters of the F.WRITE request primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.WRITE

An F.WRITE request primitive provokes the emission by the PeSIT caller whilst in the "data transfer - idle" state of an FPDU.WRITE in the normal data stream of the "communication system". The PeSIT unit attains the "File write pending" state.

c) Receiving an FPDU.WRITE

The reception of an FPDU.WRITE validated by the PeSIT server in the "data transfer - idle" state provokes a notification by an F.WRITE indication primitive to be sent to the server user. The PeSIT unit attains the "file write pending" state.

4.4.17 FPDU.ACK(WRITE)

The FPDU.ACK(WRITE) is always sent by the PeSIT server whilst in the "file write pending" state to indicate the result of the file write request. The "diagnostic" parameter of the FPDU indicates the exact reason in case of a failure.

a) Contents of the FPDU.ACK(WRITE)

The FPDU.ACK(WRITE) contains all the parameters of the F.WRITE response primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.ACK(WRITE)

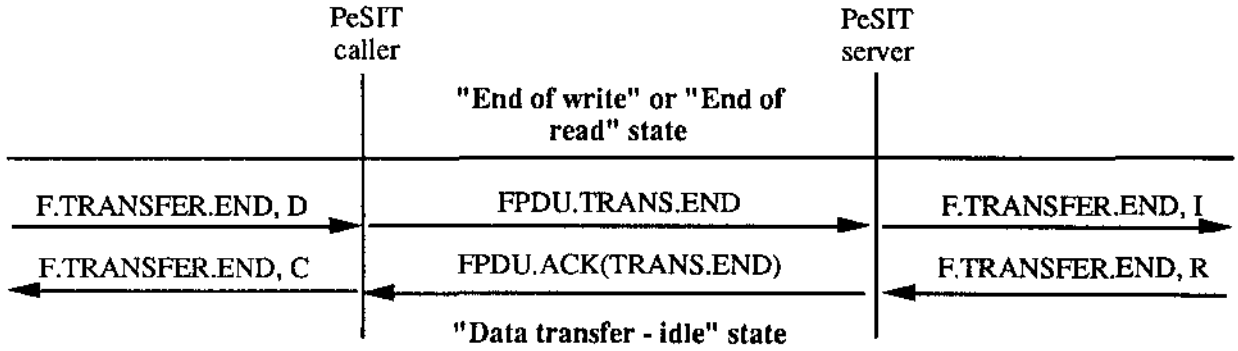
An F.WRITE response primitive provokes the emission by the PeSIT server of an FPDU.ACK(WRITE) in the normal data stream of the "communication system". The PeSIT server attains the "write file" state if the "diagnostic" code indicates "success", otherwise it returns to the "data transfer - idle" state.

c) Receiving an FPDU.ACK(WRITE)

The reception of an FPDU.ACK(WRITE) validated by the PeSIT caller in the "file write pending" state provokes a notification by an F.WRITE confirmation primitive to be sent to the caller user and the PeSIT attains the "write file" state if the "diagnostic" code indicates "success", otherwise it returns to the "data transfer - idle" state.

4.4.18 FPDU.TRANS.END

The FPDU.TRANS.END is always sent by the PeSIT caller whilst in the "end of read" or "end of write" state to request the end of a file data transfer.



a) Contents of the FPDU.TRANS.END

The FPDU.TRANS.END contains all the parameters of the F.TRANS.END request primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.TRANS.END

An F.TRANSFER.END request primitive provokes the emission by the PeSIT caller whilst in the "end of read" or "end of write" state of an FPDU.TRANS.END in the normal data stream of the "communication system". The PeSIT unit attains the "end of write transfer pending" or "end of read transfer pending" state.

c) Receiving an FPDU.TRANS.END

The reception of an FPDU.TRANS.END validated by the PeSIT server in the "end of read" or "end of write" state provokes a notification by an F.TRANSFER.END indication primitive to be sent to the server user. The PeSIT unit attains the "end of write transfer pending" or "end of read transfer pending"state.

4.4.19 FPDU.ACK(TRANS.END)

The FPDU.ACK(TRANS.END) is always sent by the PeSIT server whilst in the "end of write transfer pending" or "end of read transfer pending" state to indicate the result of the end of file transfer request. The "diagnostic" parameter of the FPDU indicates the exact reason in case of a failure.

a) Contents of the FPDU.ACK(TRANS.END)

The FPDU.ACK(TRANS.END) contains all the parameters of the F.TRANSFER.END response primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.ACK(TRANS.END)

An F.TRANSFER.END response primitive provokes the emission by the PeSIT server of an FPDU.ACK(TRANS.END) in the normal data stream of the "communication system". The PeSIT server returns to the "data transfer - idle" state .

c) Receiving an FPDU.ACK(TRANS.END)

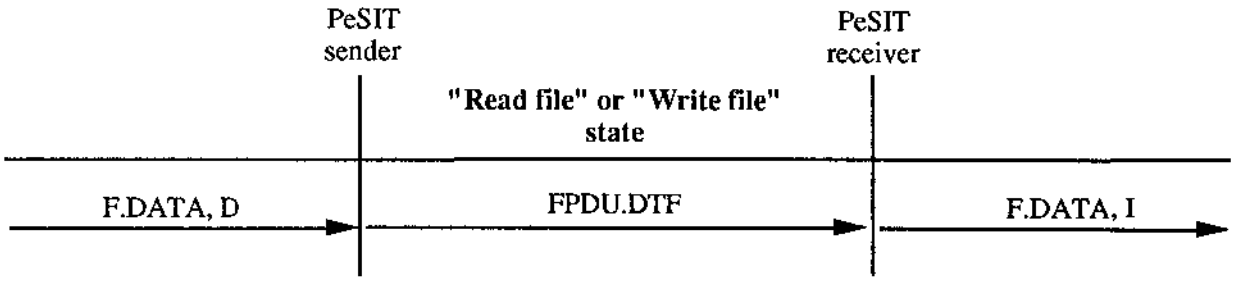
The reception of an FPDU.ACK(TRANS.END) validated by the PeSIT caller in the "end of write transfer pending" or "end of read transfer pending" state provokes a notification by an F.TRANSFER.END confirmation primitive to be sent to the caller user and the PeSIT returns to the "data transfer - idle" state.

4.4.20 FPDU.DTF, FPDU.DTFDA, FPDU.DTFMA, FPDU.DTFFA

The FPDU.DTF, FPDU.DTF.DA, FPDU.DTF.MA, FPDU.DTF.FA are sent by the PeSIT sender whilst in the "write file" or "read file" state to transfer the file data.

4.4.20.1 FPDU.DTF single article

The FPDU.DTF transport one and only one article from the file.



a) Contents of the FPDU.DTF

The FPDU.DTF contains all the parameters of the F.DATA request primitive (file article), plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.DTF

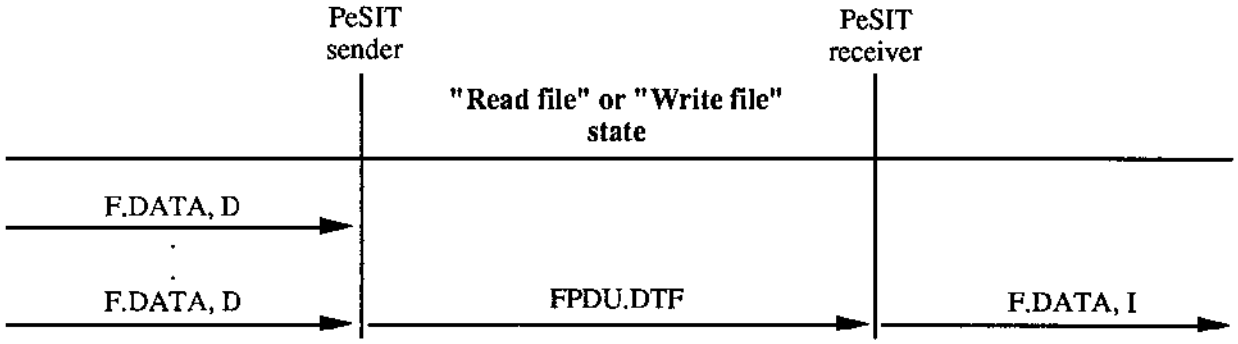
An F.DATA request primitive provokes the emission by the PeSIT sender whilst in the "read file" or "write file" state of an FPDU.DTF in the normal data stream of the "communication system". The PeSIT unit remains in the same state.

c) Receiving an FPDU.DTF

The reception of an FPDU.DTF validated by the PeSIT receiver in the "read file" or "write file" state provokes a notification by an F.DATA indication primitive to be sent to the receiver user. The PeSIT unit remains in the same state.

4.4.20.2 FPDU.DTF multi article

The FPDU.DTF transports several articles from the file.



a) Contents of the FPDU.DTF multi article

The FPDU.DTF multi article contains all the parameters of several F.DATA request primitives (several file articles), plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.DTF multi article

An F.DATA request primitive provokes the concatenation of the article in the FPDU.DTF by the PeSIT sender whilst in the "read file" or "write file" state, and may provoke the emission of the FPDU.DTF in the normal data stream of the "communication system". The number of articles and the criteria which determine when the FPDU is sent are left up to the choice of the protocol designer. The PeSIT unit remains in the same state.

c) Receiving an FPDU.DTF multi article

The reception of an FPDU.DTF validated by the PeSIT receiver in the "read file" or "write file" state provokes a notification by an F.DATA indication primitive to be sent to the receiver user. The PeSIT unit remains in the same state.

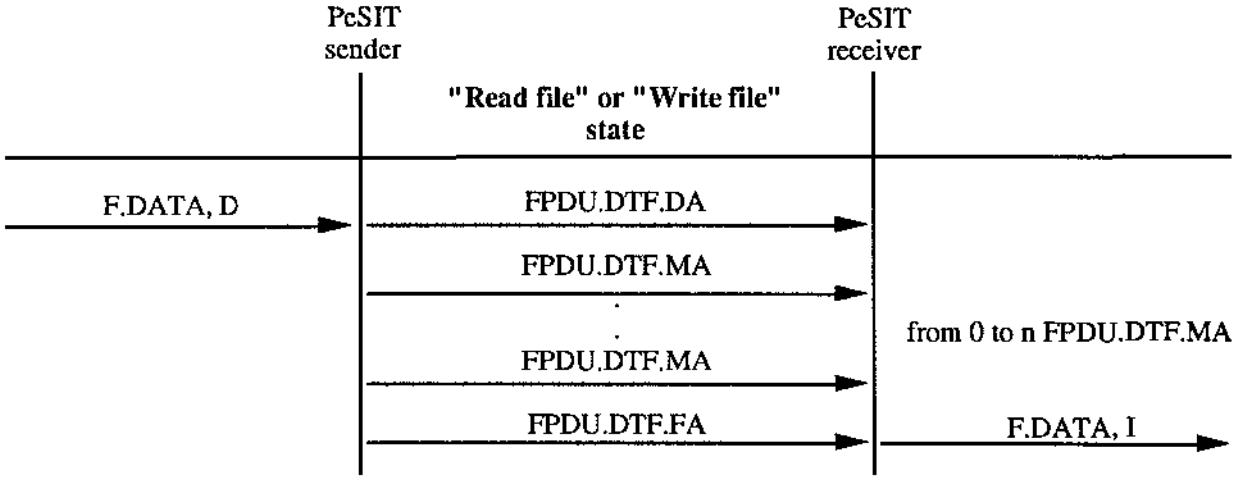
4.4.20.3 Segmentation of articles

Once the size of the article to be transported is greater than the maximum size supported by an FPDU ("maximum size of a data element" less the FPDU header size), the article can be segmented and transported in several FPDUs :

- an FPDU.DTF.DA (beginning of the article),

- zero or more FPDU.DTF.MA (middle of an article),

- an FPDU.DTF.FA (end of article).



a) Contents of the FPDU.DTF.DA, FPDU.DTF.MA, FPDU.DTF.FA

The FPDU.DTF.DA, FPDU.DTF.MA, FPDU.DTF.FA each contain a fraction of the contents of the F.DATA request primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.FPDU.DTF.DA, FPDU.DTF.MA, FPDU.DTF.FA

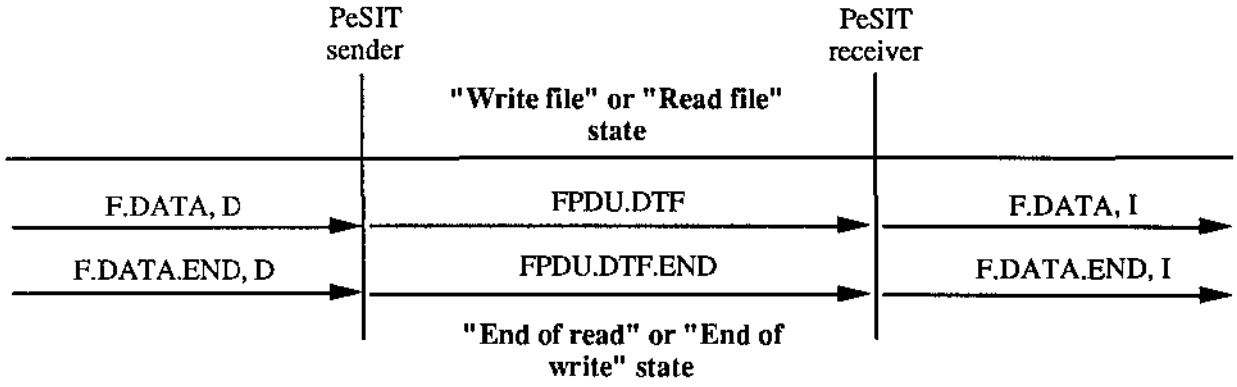
An F.DATA request primitive provokes the emission by the PeSIT sender whilst in the "read file" or "write file" state of an FPDU.DTF.DA, a zero or greater number of FPDU.DTF.MA, and an FPDU.DTF.FA in the normal data stream of the "communication system". The PeSIT unit remains in the same state.

c) Receiving an FPDU.DTF.FA

The reception of an FPDU.DTF validated by the PeSIT receiver in the "read file" or "write file" state provokes a notification by an F.DATA indication primitive to be sent to the receiver user. The PeSIT unit remains in the same state.

4.4.21 FPDU.DTF.END

The FPDU.DTF.END is always sent by the PeSIT sender whilst in the "write file" or "read file" state to indicate the end of file data transfer. The "diagnostic" parameter of the FPDU indicates the exact reason for the end of transfer.



a) Contents of the FPDU.DTF.END

The FPDU.DTF.END contains all the parameters of the F.DATA.END request primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.DTF.END

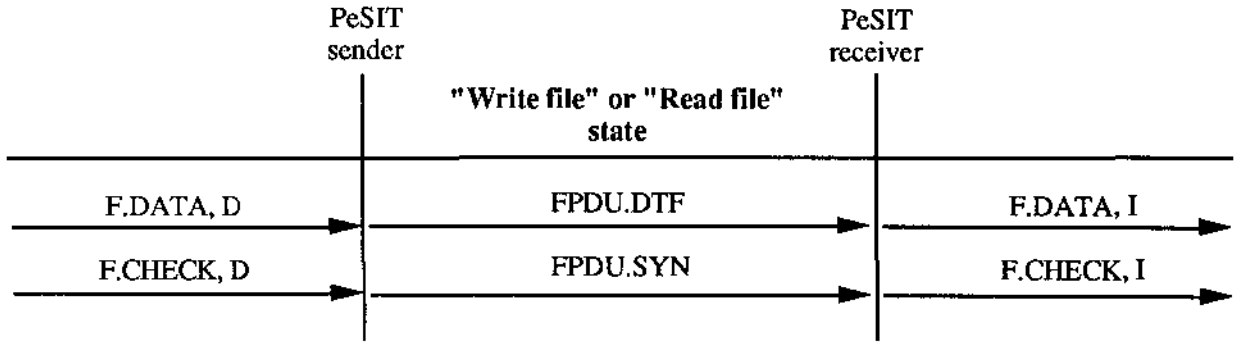
An F.DATA.END request primitive provokes the emission by the PeSIT sender whilst in the "read file" or "write file" state of an FPDU.DTF.END in the normal data stream of the "communication system". The PeSIT unit attains the "end of read" or "end of write" state.

c) Receiving an FPDU.DTF.END

The reception of an FPDU.DTF.END validated by the PeSIT receiver in the "read file" or "write file" state provokes a notification by an F.DATA.END indication primitive to be sent to the receiver user. The PeSIT unit attains the "end of read" or "end of write" state.

4.4.22 FPDU.SYN

The FPDU.SYN is always sent by the PeSIT sender whilst in the "write file" or "read file" state to request the setting of checkpoints.



a) Contents of the FPDU.SYN

The FPDU.SYN contains all the parameters of the F.CHECK request primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.SYN

An F.CHECK request primitive provokes the emission by the PeSIT sender of an FPDU.SYN in the normal data stream of the "communication system". The PeSIT unit remains in the same state.

c) Receiving an FPDU.SYN

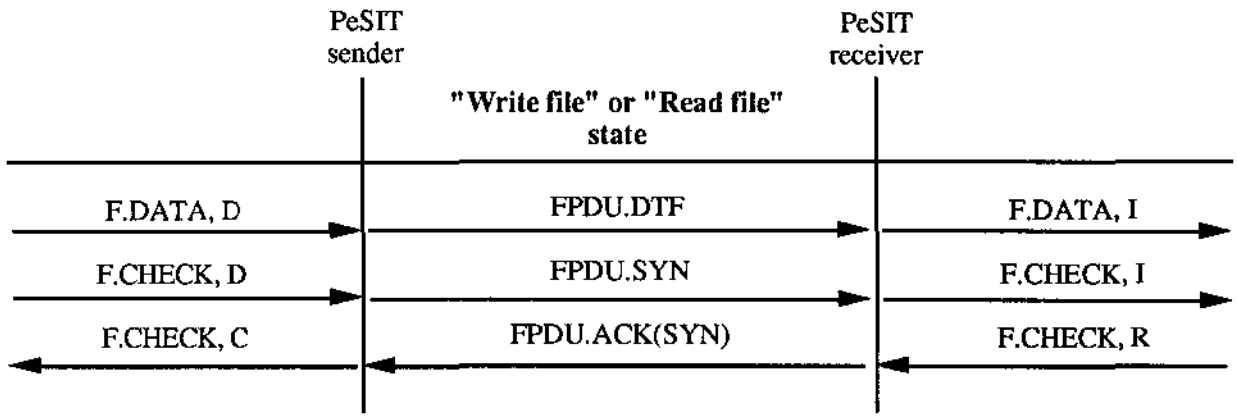
The reception of an FPDU.SYN validated by the PeSIT receiver in the "read file" or "write file" state provokes a notification by an F.CHECK indication primitive to be sent to the receiver user. The PeSIT unit remains in the same state.

d) Note

The FPDU.SYN should be sent between articles from the file. This implies that if article segmentation is used then the FPDU.SYN should only be placed after an FPDU.DTF.FA and prior to an FPDU.DTF.DA.

4.4.23 FPDU.ACK(SYN)

The FPDU.ACK.SYN is always sent by the PeSIT receiver whilst in the "write file" or "read file" state to acknowledge the checkpoints set previously.



a) Contents of the FPDU.ACK(SYN)

The FPDU.ACK(SYN) contains all the parameters of the F.CHECK response primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.ACK(SYN)

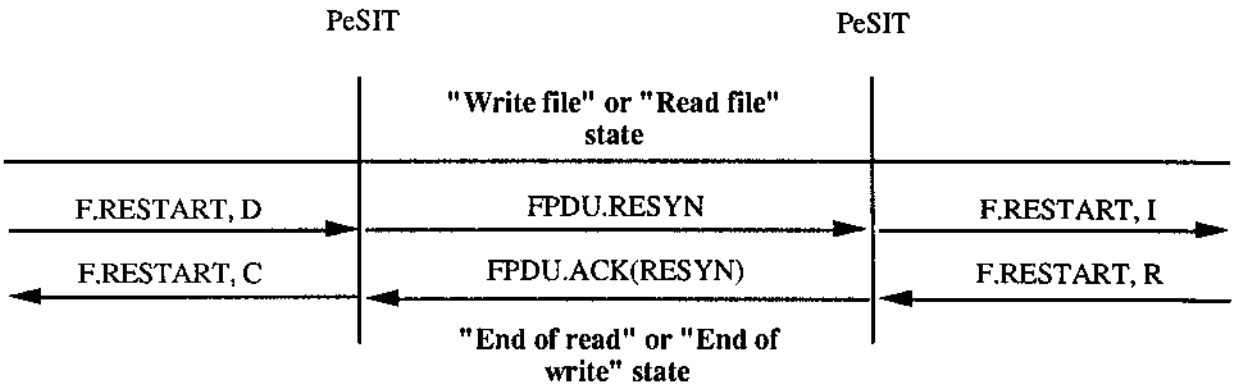
An F.CHECK response primitive provokes the emission by the PeSIT receiver of an FPDU.ACK(SYN) in the normal data stream of the "communication system". The PeSIT unit remains in the same state. For PeSIT.F, the FPDU.ACK(SYN) is sent in the typed data stream of the session layer (S-TYPED-DATA).

c) Receiving an FPDU.ACK(SYN)

The reception of an FPDU.SYN validated by the PeSIT sender in the "read file" or "write file" state provokes a notification by an F.CHECK confirmation primitive to be sent to the sender user. The PeSIT unit remains in the same state.

4.4.24 FPDU.RESYN

The FPDU.RESYN is sent by the PeSIT sender or receiver whilst in the "write file" or "read file" state to request that a transfer be restarted from a previously set checkpoint.



a) Contents of the FPDU.RESYN

The FPDU.RESYN contains all the parameters of the F.RESTART request primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.RESYN

An F.RESTART response primitive provokes the emission by the PeSIT sender or receiver of an FPDU.RESYN in the normal data stream of the "communication system". The PeSIT unit attains the "restart pending" state.

c) Receiving an FPDU.RESYN

The reception of an FPDU.RESYN validated by the PeSIT sender or receiver in the "read file" or "write file" state provokes a notification by an F.RESTART indication primitive to be sent to the user. The PeSIT unit attains the "restart pending" state.

4.4.25 FPDU.ACK(RESYN)

The FPDU.ACK.RESYN is sent by the PeSIT sender or receiver whilst in the "restart pending" state to acknowledge the restart.

a) Contents of the FPDU.ACK(RESYN)

The FPDU.ACK(RESYN) contains all the parameters of the F.RESTART request primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.ACK(RESYN)

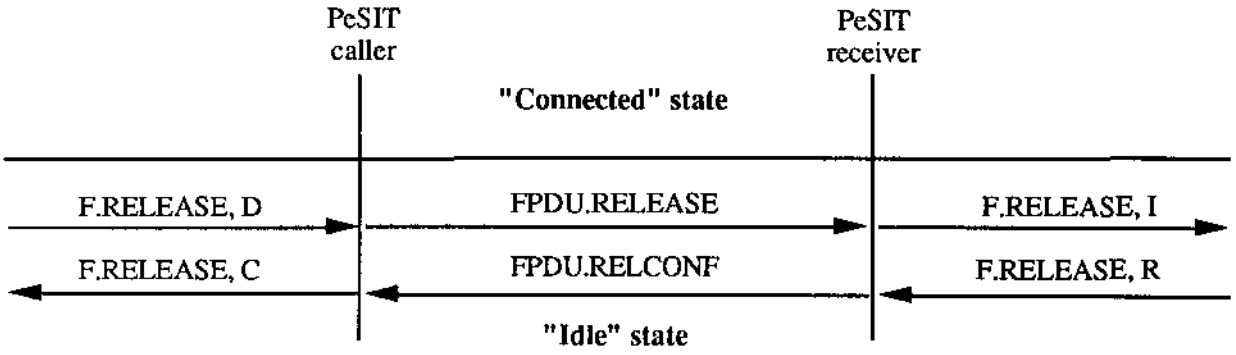
An F.RESTART response primitive provokes the emission by PeSIT of an FPDU.ACK(RESYN) in the normal data stream of the "communication system". The PeSIT unit returns to the "read file" or "write file" state.

c) Receiving an FPDU.ACK(RESYN)

The reception of an FPDU.ACK(RESYN) validated by PeSIT in the "restart pending" state provokes a notification by an F.RESTART confirmation primitive to be sent to the user. The PeSIT unit returns to the "read file" or "write file" state.

4.4.26 FPDU.RELEASE

In the same way as the FPDU.CONNECT, the FPDU.RELEASE is always sent by the PeSIT caller whilst in the "connected" state to request connection shut-down.



a) Contents of the FPDU.RELEASE

The FPDU.RELEASE contains all the parameters of the F.RELEASE request primitive, plus :

ID.DST : FPDU receiver's connection identification.

ID.SRC : FPDU sender's connection identification.

b) Sending the FPDU.RELEASE

An F.RELEASE request primitive provokes the emission by the PeSIT caller of an FPDU.RELEASE in :

- the user field of the session end request primitive S-RELEASE, by PeSIT.F,

- the normal data stream of the N-DATA network service by PeSIT.F'.

The PeSIT unit attains the "liberation pending" state.

c) Receiving an FPDU.RELEASE

The reception of an FPDU.RELEASE validated by PeSIT in the "connected" state provokes a notification by an F.RELEASE indication primitive to be sent to the user. The PeSIT unit attains the "liberation pending" state.

4.4.27 FPDU.RELCONF

The FPDU.RELCONF is always sent by the PeSIT server whilst in the "liberation pending" state to acknowledge release of the connection.

a) Contents of the FPDU.RELCONF

The FPDU.RELCONF contains all the parameters of the F.RELEASE request primitive, plus :

ID.DST : FPDU receiver's connection identification.

ID.SRC : FPDU sender's connection identification.

b) Sending the FPDU.RELCONF

An F.RELEASE request primitive provokes the emission by the PeSIT server of an FPDU.RELCONF in :

- the user field of the session end response primitive S-RELEASE by PeSIT.F,
- the normal data stream of the N-DATA network service by PeSIT.F', and the arming of the protocol monitoring time-out Tr (See 4.5.2).

The PeSIT unit returns to the "idle" state.

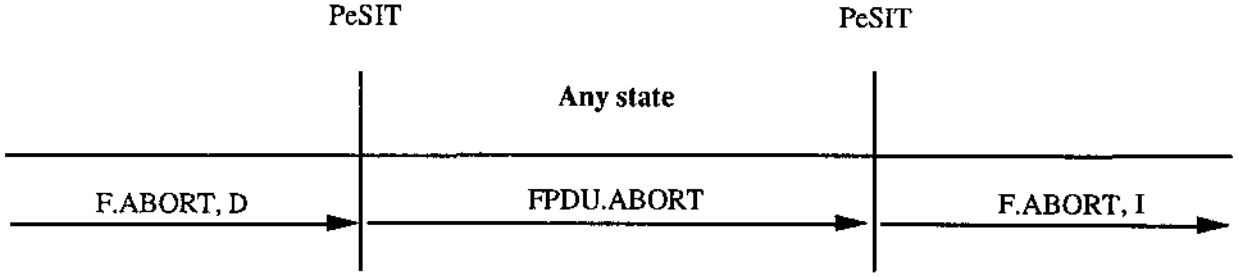
c) Receiving an FPDU.RELCONF

The reception of an FPDU.RELCONF validated by the PeSIT caller in the "liberation pending" state provokes a notification by an F.RELEASE indication primitive to be sent to the calling user. The PeSIT unit returns to the "idle" state as well as requesting network service shut-down with the N-DISCONNECT primitive by PeSIT.F'.

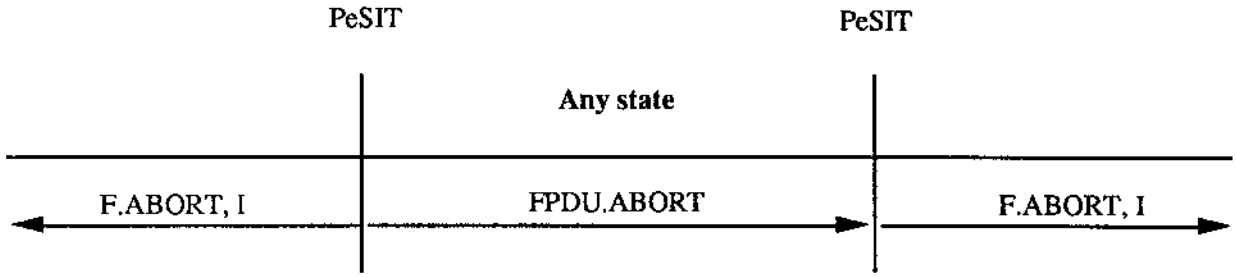
4.4.28 FPDU.ABORT

The FPDU.ABORT may be sent by a PeSIT caller or server at any time during the dialogue to indicate the abrupt termination of a connection. The "diagnostic" parameter provides the reason for terminating.

- Abrupt termination requested by the user (caller/server).



- Abrupt termination requested by PeSIT or the "communication system".



a) Contents of the FPDU.ABORT

The FPDU.ABORT contains all the parameters of the F.ABORT primitive, plus :

ID.DST : FPDU receiver's connection identification.

ID.SRC : FPDU sender's connection identification.

b) Sending the FPDU.ABORT

- The reception by PeSIT of an F.ABORT request primitive,
- or PeSIT detecting a class 3 error (see Annexe D),
- or an N-RESET indication being received by PeSIT.F' from the network layer,

provoke the emission by PeSIT of an FPDU.ABORT in :

- the user field of the session service primitive S-ABORT by PeSIT.F and a return to the "idle" state,
- the normal data stream of the N-DATA network service by PeSIT.F', followed by arming of the protocol monitoring time-out Tr (See 4.6).

The PeSIT user is systematicaly notified by an F.ABORT indication primitive.

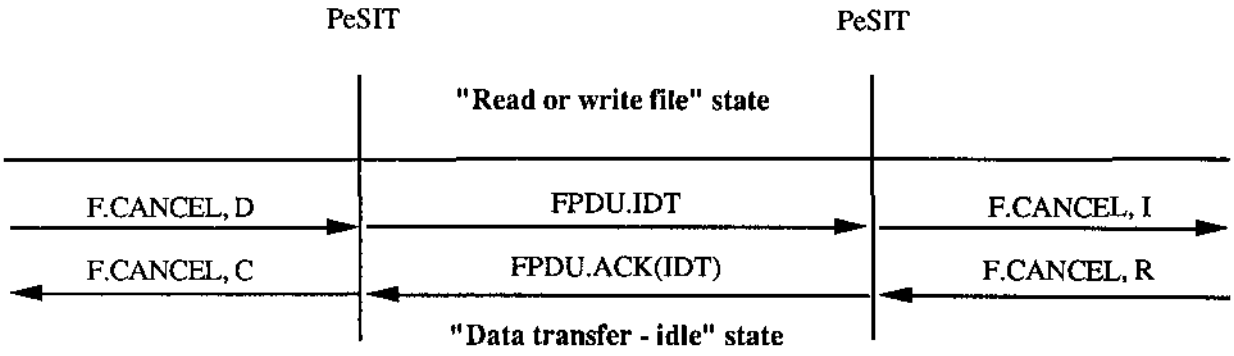
c) Receiving an FPDU.ABORT

The reception of an FPDU.ABORT by PeSIT causes the user to be notified by an F.ABORT indication primitive and :

- for PeSIT to return to the "idle" state,
- and network service shut-down to be requested with the N-DISCONNECT primitive by PeSIT.F'.

4.4.29 FPDU.IDT

The FPDU.IDT is sent by the PeSIT caller or server whilst in the "write file" or "read file" state to request the interruption of a transfer. The end of transfer code indicates why the transfer was interrupted : "cancelled", "suspended" or "error".



a) Contents of the FPDU.IDT

The FPDU.IDT contains all the parameters of the F.CANCEL primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.IDT

An F.CANCEL request primitive provokes the emission by the PeSIT sender or receiver of an FPDU.IDT to request interruption of the current transfer. The PeSIT unit attains the "transfer interruption pending" state.

Typed data is used by the PeSIT.F session service.

c) Receiving an FPDU.IDT

The reception of an FPDU.IDT validated by PeSIT in the "read file" or "write file" state provokes a notification by an F.CANCEL indication primitive to be sent to the user. The PeSIT unit attains the "transfer interruption pending" state.

4.4.30 FPDU.ACK(IDT)

The FPDU.ACK(IDT) is sent by the PeSIT sender or receiver whilst in the "transfer interruption pending" state to acknowledge the transfer interrupt request.

a) Contents of the FPDU.ACK(IDT)

The FPDU.ACK(IDT) contains all the parameters of the F.CANCEL response primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.ACK(IDT)

An F.CANCEL response primitive provokes the emission by PeSIT of an FPDU.ACK(IDT). The PeSIT unit returns to the "data transfer - idle" state.

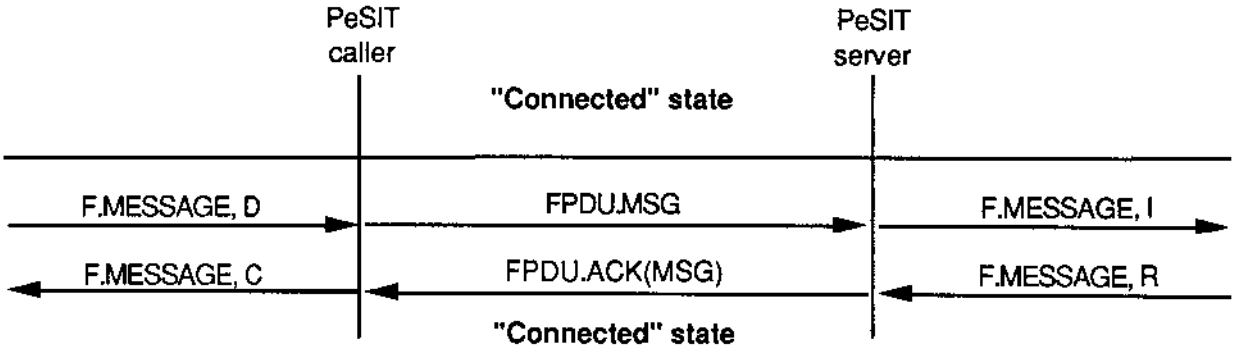
Typed data is used by the PeSIT.F session service.

c) Receiving an FPDU.ACK(IDT)

The reception of an FPDU.ACK(IDT) validated by PeSIT in the "transfer interrupt pending" state provokes a notification by an F.CANCEL indication primitive to be sent to the user. The PeSIT unit returns to the "data transfer - idle" state.

4.4.31 FPDU.MSG, FPDU.MSGDM, FPDU.MSGMM, FPDU.MSGFM

The FPDU.MSG is always sent by the PeSIT caller whilst in the "connected" state to request transmission of a datagram to another PeSIT server unit without entering the file transfer mode.



4.4.31.1 FPDU.MSG

a) Contents of the FPDU.MSG

The FPDU.MSG contains all the parameters of the F.MESSAGE request primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.MSG

An F.MESSAGE request primitive provokes the emission by the PeSIT caller of an FPDU.MSG in the normal data stream of the "communication system". The PeSIT calling unit attains the "file release pending" state.

c) Receiving an FPDU.MSG

The reception of an FPDU.MSG validated by the PeSIT server in the "connected" state provokes a notification by an F.MESSAGE indication primitive to be sent to the user. The PeSIT unit attains the "file release pending" state.

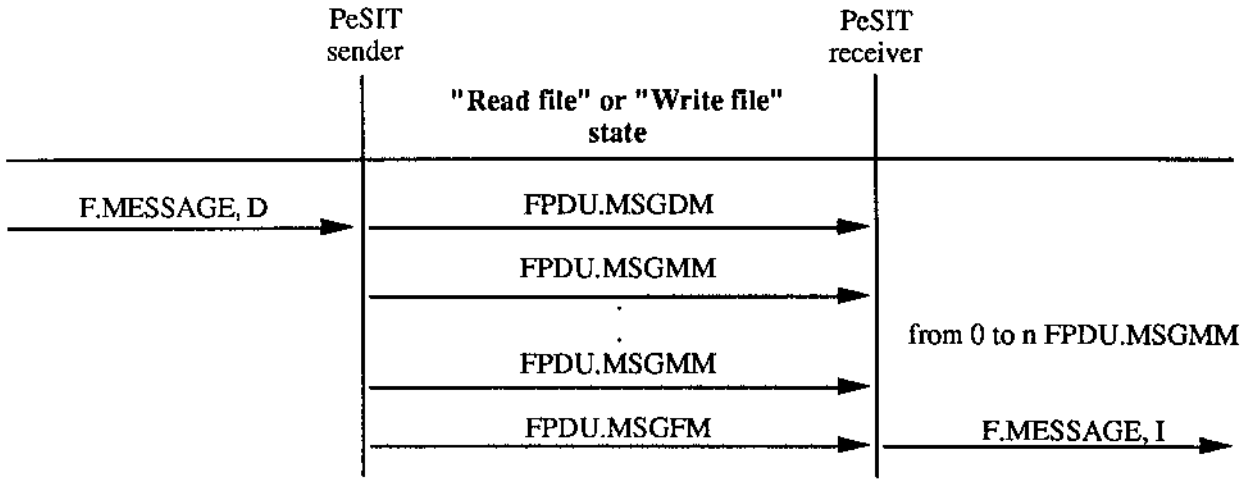
4.4.31.2 Segmentation of Datagrams

Once the size of the message to be transported is greater than the maximum size supported by an FPDU ("maximum size of a data element" less the FPDU header size), the message can be segmented and transported in several FPDUs :

- an FPDU.MSGDM (beginning of the message),

- zero or more FPDU.MSGMM (middle of an message),

- an FPDU.MSGFM (end of message).



a) Contents of the FPDU.MSGDM, FPDU.MSGMM, FPDU.MSGFM

The FPDU.MSGDM, FPDU.MSGMM, FPDU.MSGFM each contain a fraction of the contents of the F.MESSAGE request primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.FPDU.MSGDM, FPDU.MSGMM, FPDU.MSGFM

An F.MESSAGE request primitive provokes the emission by the PeSIT caller whilst in the "connected" state of an FPDU.MSGDM, a zero or greater number of FPDU.MSGMM, and an FPDU.MSGFM in the normal data stream of the "communication system". The PeSIT unit remains in the same state.

c) Receiving an FPDU.MSGFM

The reception of an FPDU.MSGFM validated by the PeSIT server in the "connected" state provokes a notification by an F.MESSAGE indication primitive to be sent to the server user. The PeSIT unit attains the "file release pending" state.

4.4.32 FPDU.ACK(MSG)

The FPDU.ACK(MSG) is always sent by the PeSIT server whilst in the "file release pending" state to indicate the outcome of execution of the message delivery request. The "diagnostic" parameter of the FPDU indicates the exact reason in case of a failure. The FPDU.ACK(MSG) may itself contain a message to be returned to the PeSIT calling station.

a) Contents of the FPDU.ACK(MSG)

The FPDU.ACK(MSG) contains all the parameters of the F.MESSAGE response primitive, plus :

ID.DST : FPDU receiver's connection identification.

b) Sending the FPDU.ACK(MSG)

An F.MESSAGE response primitive provokes the emission by PeSIT of an FPDU.ACK(MSG) in the normal data stream of the "communication system". The PeSIT server unit returns to the "connected" state.

c) Receiving an FPDU.ACK(MSG)

The reception of an FPDU.ACK(MSG) validated by the PeSIT caller in the "connected" state provokes a notification by an F.MESSAGE confirmation primitive to be sent to the calling user. The PeSIT calling unit returns to the "connected" state.

4.5 CONCATENATION OF FPDUS

Some FPDUs may be concatenated into the basic data unit of the lower protocol layer (NSDU, SSDU, ...), whithin the maximum size limit negotiated during the creation or selection phase of the file.

This rule may be applied to the following FPDUs :

- FPDU.DTF
- FPDU.DTF.DA
- FPDU.DTF.MA
- FPDU.DTF.FA
- FPDU.DTF.END
- FPDU.SYN

4.6 PESIT PROTOCOL TIME-OUTS

To monitor the PeSIT protocol several time-outs exist.

1) Protocol monitoring time-out : Tp

One of the partners stops sending messages while the other is awaiting something. This anomaly can be detected localy by a time-out function called "protocol monitoring time-out" : Tp. The only solution in this case is to terminate the connection with a diagnostic code which indicates that the monitoring time-out has expired.

For the server, the protocol monitoring time-out Tp, is used to monitor the activity of the oppposit PeSIT unit :

- waiting for an FPDU in the following list : ORF, READ, WRITE, DTF, DTF.DA, DTF.MA, DTF.FA, DTF.END, SYN, RESYN, TRANS.END, CRF or DESELECT.

For the caller, the protocol monitoring time-out Tp is used to monitor the replies of the opposit PeSIT unit to the above FPDUs, as well as the reply to an FPDU CONNECT or RELCONF.

The length of this time-out as intended by the caller may be transmitted to the server in the FPDU CONNECT (optional parameter with default value of 30 seconds).

2) Network disconnect time-out : Tr (used by PeSIT.F')

*** Normal disconnect case**

The PeSIT.F' server arms the timer Tr and awaits the network server disconnect indication primitive "N-DISCONNECT,I". If the network service disconnect indication primitive is received then the timer Tr is disabled and the PeSIT server returns to the "idle" state. If the timer Tr expires prior to reception of this disconnect indication, the PeSIT server requests network service disconnection using the N-DISCONNECT request primitive and returns to the "idle" state.

*** F.ABORT service usage**

If the F.ABORT service is invoked, the PeSIT unit which sent the FPDU.ABORT arms the timer Tr and awaits the network server disconnect indication primitive "N-DISCONNECT,I". If the timer Tr expires prior to reception of this disconnect indication, the PeSIT unit which sent the FPDU.ABORT requests network service disconnection using the N-DISCONNECT request primitive and returns to the "idle" state. If the network service disconnect indication primitive is received then the timer Tr is disabled and the PeSIT unit returns to the "idle" state.

The time-out delay of the timer Tr (Network disconnect time-out) is related to the service quality and depends on the local system implementation.

This timer should have a value of approximately 30 seconds.

3) FPDU-CREATE, SELECT or RELEASE (request) time-out : Td

This timer is used to allow a network connection to be used for several transfers. Between two transfers, some time may go by without the connection being released.

This timer should have a value of several minutes (greater than 5 minutes).

This timer is armed only by the server.

4) Connection set-up time-out : Tc

The timer Tc is armed by the server whilst a PeSIT.F' network connection has been set-up to monitor the emission of an FPDU.CONNECT by the caller.

This timer should have a value of approximately 30 seconds.

4.7 STRUCTURE AND CODING OF PESIT PROTOCOL UNITS (FPDU)

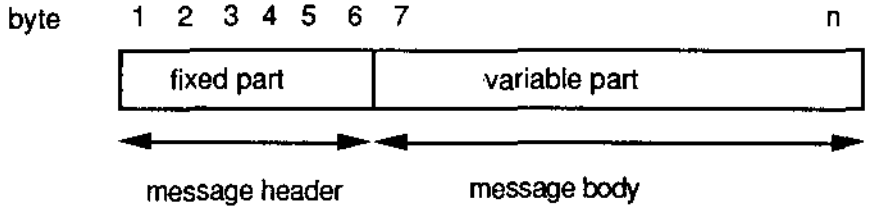
4.7.1 Structure of a protocol element

Note : in this paragraph, the word message is used as "protocol data unit".

Each PeSIT message is made up of two parts :

- a six byte message header,

- a variable length message body.



a) message header

The message header has the following structure :

Bytes 1 and 2
Total length of the message (header + body, in bytes).

Byte 3
Phase indicator :
40h : protocol element for the connection phase
00h : FPDU.DTF, FPDU.DTF.DA, FPDU.DTF.MA, FPDU.DTF.FA
C0h : other FPDUs

Byte 4
message type

Byte 5
Receiver's connection identification

Byte 6

- FPDU used during the connection phase :	Sender's connection identification (ID.SRC)
- FPDU.DTF single article :	0
- FPDU.DTF multi article :	number of articles N (N>1)
- Other FPDUs :	0

In the following table, X denotes the callers connection identification and Y denotes the server connection identification. X and Y are arbitrary non-zero numerical values, determined at connection time by the caller and the server.

The coding Y/X indicates that the value to be used is Y if the FPDU is sent by the PeSIT caller or X if the FPDU is sent by the PeSIT server.

PHASE	MESSAGE	Byte 3	Byte 4	Byte 5	Byte 6
Connection	FPDU.CONNECT	40h	20	0	X
	FPDU.ACONNECT	40h	21	X	Y
	FPDU.RCONNECT	40h	22	X	0
	FPDU.RELEASE	40h	23	Y	X
	FPDU.RELCONF	40h	24	X	Y
	FPDU.ABORT	40h	25	Y/X (1)	X/Y
File selection and release	FPDU.CREATE	C0h	11	X	0
	FPDU.ACK(CREATE)	C0h	30	X	0
	FPDU.SELECT	C0h	12	Y	0
	FPDU.ACK(SELECT)	C0h	31	X	0
	FPDU.DESELECT	C0h	13	Y	0
	FPDU.MSG	C0h	16	Y	0
	FPDU.MSGDM	C0h	17	Y	0
	FPDU.MSGMM	C0h	18	Y	0
	FPDU.MSGFM	C0h	19	Y	0
	FPDU.ACK(MSG)	C0h	3B	X	0
File opening and closing	FPDU.ORF	C0h	14	Y	0
	FPDU.ACK(ORF)	C0h	33	X	0
	FPDU.CRF	C0h	15	Y	0
	FPDU.ACK(CRF)	C0h	34	X	0
Beginning and end of transfer	FPDU.READ	C0h	01	Y	0
	FPDU.ACK(READ)	C0h	35	X	0
	FPDU.WRITE	C0h	02	Y	0
	FPDU.ACK(WRITE)	C0h	36	X	0
	FPDU.TRANS.END	C0h	08	Y	0
	FPDU.ACK(TRANS.END)	C0h	37	X	0
Data transfer	FPDU.DTF	0	00	Y/X	0 (single-article) N (multi-articles)
	FPDU.DTFDA	0	41	Y/X	0
	FPDU.DTFMA	0	40	Y/X	0
	FPDU.DTFFA	0	42	Y/X	0
	FPDU.DTF.END	C0h	04	Y/X	0
	FPDU.SYN	C0h	03	Y/X	0
	FPDU.ACK(SYN)	C0h	38	Y/X	0
	FPDU.RESYN	C0h	05	Y/X	0
	FPDU.ACK(RESYN)	C0h	39	Y/X	0
Transfer interruption	FPDU.IDT	C0h	06	Y/X	0
	FPDU.ACK(IDT)	C0h	3A	Y/X	0

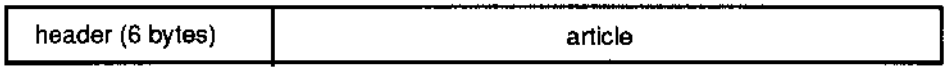
(1) If an FPDU.ABORT is sent before the ID.DST is known, then the ID.DST field should be set to 0.

b) message body

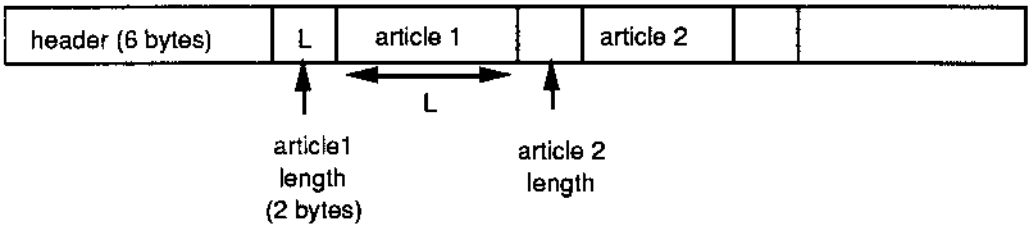
. FPDU.DTF, FPDU.DTF.DA, FPDU.DTF.MA, FPDU.DTF.FA

For these FPDUs, this field contains the file data.

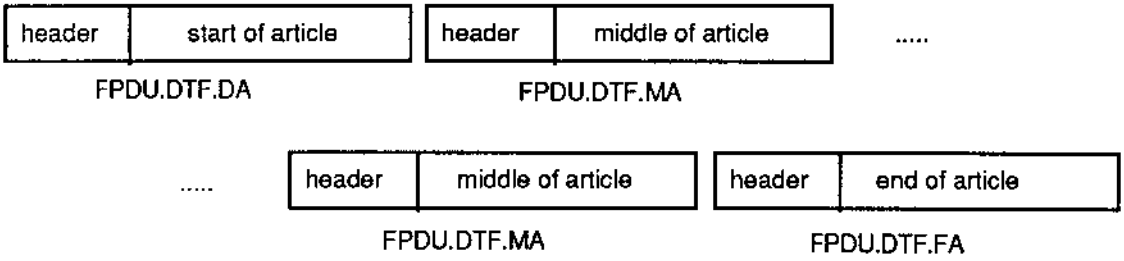
- In a single article FPDU.DTF (such as in the SIT), this field contains one complete article :



- In a multi-article FPDU.DTF, this field contains several articles :



- An article which is longer than the maximum size of the SSDU-DATA is segmented into several FPDU comprised of :
an FPDU.DTF.DA (start of article), zero or more FPDU.DTF.MA (middle article) and an FPDU.DTF.FA (end of article) :



. OTHER FPDUs

This field contains the message parameters, each identified by a PI (parameter identifier) which are assembled together into parameter groups, identified by a PGI (parameter group identifier). The PGI and PI blocks should be ordered into a list sorted into increasing value of the PGI and PI codes.

The method of representing the parameters is described hereafter.

4.7.2 Coding of the parameters

4.7.2.1 Coding conventions

a) PGI blocks

The PGI blocks contain, in the following order :

a) the PG field which identifies this group of parameters,

b) the LI field which indicates the length of the associated parameter field,

c) the parameter field which consists of :

- either a single parameter value (see note),
- or one or more PI blocks.

NOTE :
A PGI block which contains a single parameter is structurally equivalent to a PI block.

b) PI blocks

The PI blocks consist of, in the following order :

a) the PI field which identifies the parameter,

b) the LI field which indicates the length of the associated parameter field,

c) the parameter field which contains the parameter value.

c) PI and PGI field identifier

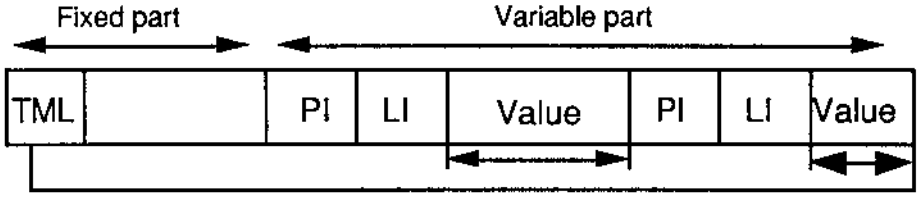
The PI and PGI fields each include a byte which contains respectively the PI or the PGI code. The PI and PGI codes are expressed as decimal numbers as listed in §4.7.3 and which should be coded as binary numbers.

d) Length indicater field LI

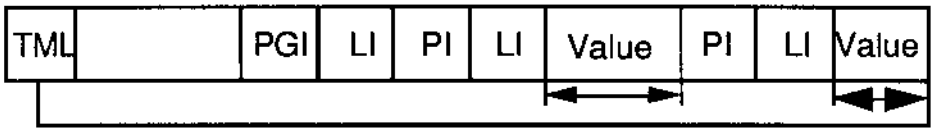
The value of the LI field is expressed as a binary number which represents the length, in bytes, of the associated parameter field (see note). A zero value is rejected by the protocol.

The length of the LI field is variable. For parameter field length comprised between 1 and 254 the length field is one byte long.
So as to indicate parameter field lengths in excess of 254, but less less 65536 bytes, the LI field is extended to three bytes long. The first field is coded as 0xFF and the second and third bytes should contain the length of the associated parameter field, the most significant byte is the first of these two bytes.

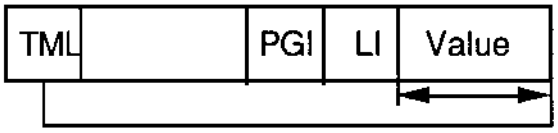
NOTE :
The value of the LI field does not include the length of the LI field itself.



TML* : 14 bytes, with 2 parameters, of 3 and 1 byte length.



TML* : 17 bytes, with 1 PGI containing 2 PI, of 2 and 3 byte length.



TML* : 9 bytes, with 1 PGI and no PI, total parameter length = 1.

* Total message length = TML

e) Representation of parameter values

To be able to describe the parameters completely, we must first define the following value types which are used to code the different parameters in the coding tables :

- C : String of characters. Unfixed length (within the limits defined); the characters are coded using ASCII 7 bit code. Right hand spaces are non-significant, completely space filled value are illegal.

- N : Numerical. Unsigned binary integer.

- S : Symbolic. As N, an unsigned binary integer except that the values have a particular significance. Length 8 bits.

- M : Bit mask. Byte or word in which each bit has a particular significance. Length 8 or 16 bits. All undefined bits should have a zero value. For bits corresponding to particular options, a bit set to the value "1" indicates that the option is requested, and the value "0" the opposite.

- D : Date and Time. These are coded according to the ISO 2014 and ISO 3307 standards : YYMMDD and hhmmss, in which YY = year, MM = month, DD = day, hh = hour, mm = minutes and ss = seconds. It is represented as a character string 12 bytes long.

- A : Agregate (mixed). Composed of two or more of the above value types (e.g. 2 zones - numerical and symbolic).

f) optimising the value field

It is recommended to use the minimum length possible for variable length character strings expressing values.

The character strings should not contain unnecessary spaces unless the field is of a fixed length.

For the numerical and symbolic data, all unused leftmost bits with a zero value should be eliminated. Nevertheless, the minimum length of a string is one byte.

g) omitting parameters

Certain parameters may be omitted. The different types are :

- obligatory parameters whose absence from an FPDU would produce a protocol error.

- optional parameters where a implicit value is defined. This value is underligned in the parameter description (§4.7.3). If the parameter is omitted it will be considered to have this value.

- optional parameters without implicit values, which may or may not be present in an FPDU.

h) Repeating parameters

Unless specifically indicated in the service description, any particular parameter should be present once and once only within a given message. In an illegal repetition is made, only the first instance is kept, and an error is notified.

4.7.2.2 List of the PGI and PI codes

The different PI codes used are :

PI Code	Parameter description
1	CRC usage
2	Diagnostics
3	Caller identification
4	Server identification
5	Access control
6	Version number
7	Option : checkpointing
11	File type
12	File name
13	Transfer identifier
14	Requested attributes
15	Recovered transfer
16	Data coding
17	Transfer priority
18	Recovery point
19	End of transfer code
20	Checkpoint number
21	Compression
22	Access type
23	Restarting
25	Maximum size of a data element
26	Protocol monitoring time-out
27	Number of data bytes
28	Number of articles
29	Diagnostic complements
31	Article format
32	Article length
33	File attributes
34	Use of the signature
36	SIT MAC
37	File Label
38	Key length
39	Key offset
41	Storage reservation unit
42	Maximum reserved space
51	Date and time of creation
52	Date and time of last access
61	Customer identifier
62	Bank identifier
63	File access control
64	Server date and time

71	Authentication type
72	Authentication elements
73	MAC computation type
74	MAC computation elements
75	Encryption type
76	Encryption elements
77	Digital signature type
78	MAC
79	Digital signature
80	Certificate
81	Acknowledgement of Digital signature
82	Second digital signature
83	Second certificate
91	Datagram
92	Free text

The different PGI codes used are :

TITLE	PGI CODE	PI
File identifier	9	3 4 11 12
Logical attributes	30	31 32 33 34 36 37 38 39
Physical attributes	40	41 42
Historical attributes	50	51 52

4.7.3 Parameter descriptions

The following pages give the detailed coding of each parameter, profile by profile. For each parameter the following information is given :

- the name of the parameter,

- the coding type (C, N, S, M, D, A),

- the maximum length of the zone in bytes,

- the coding values allowed (a default value, if it exists, is underligned),

- the "optional" or "optional with default value" indication as defined in § 4.7.2.1g.

The parameter may also be :

- "conditional" : the parameter is mandatory if the condition is met and should be omitted if the condition is not met,

- "conditional with default value" : if the condition is met and the parameter is absent then the default value will be used,

If none of these indications is given then the parameter is mandatory.

Note : when the format of a parameter is not described specifically for the Secure Non-SIT profile, it is identical to the Non-SIT profile.

CRC USAGE 1

Type : S **Length : 1** optional parameter with default value

SIT

Prohibited

NON-SIT

0 = CRC not used
1 = CRC used

Use of the CRC is mandatory when PeSIT.F' is used with an asynchronous access point (PAD).

ETEBAC5

Idem NON-SIT

DIAGNOSTICS 2

Type : A **Length : 3**

SIT

Diagnostics (see ANNEXE D)
Byte 1 : error type
Bytes 2-3 : diagnostic code

NON-SIT

Idem SIT

ETEBAC5

Idem SIT

CALLER IDENTIFICATION 3

Optional parameter except in the FPDU.CONNECT

SIT

Type : A Length : 3

Byte 1 (S) : application type
value = 1 : CTE
value = 2 : CTR
value = 3 : IE
value = 4 : IR

Bytes 2 and 3 (N) : Application number (assigned by the Control Center)

NON-SIT

Type : C Length : 24

Caller Identification (cf annexe B : mode store and forward, for more information)

ETEBAC5

Type : C Length : 24

Caller Identification :

Bytes 1 to 3 : identifier type (C)
value = ZZZ : mutually agreed
value = 005 : CFONB standard

Bytes 4 to 24 : identifier (C)

if standard identifier:
. corporate identifier
. service and individual identifier

SERVER IDENTIFICATION 4

Optional parameter except in the FPDU.CONNECT

SIT

Type : A Length : 3

Byte 1 (S) : application type
value = 1 : CTE
value = 2 : CTR
value = 3 : IE
value = 4 : IR

Bytes 2 and 3 (N) : Application number (assigned by the Control Center)

NON-SIT

Type : C Length : 24

Server Identification (cf annexe B : mode store and forward, for more information)

ETEBAC5

Type : C Length : 24

Server Identification :

Bytes 1 to 3 : identifier type (C)
value = ZZZ : mutually agreed
value = 005 : CFONB standard
value = ZBF : Banque de France code (bank + branch)

Bytes 4 to 24 : identifier (C)

if standard identifier:
. corporate identifier
. service and individual identifier

ACCESS CONTROL 5

SIT

Type : C Length : 2

Not used

NON-SIT

Type : C Length : 16

optional parameter.
Bytes 1 to 8 : current password
Bytes 9 to 16 : new password
(if bytes 9 to 16 are absent the password is not altered)

ETEBAC5

Idem NON-SIT.

VERSION NUMBER 6

Type : N **Length : 1**

SIT

Version number of PeSIT = 1

NON-SIT

Version number of PeSIT

value 1 : version D of 15 novembre 1987 (November 15th 1987)

value 2 : version E of 14 juillet 1989 (July 14th 1989)

ETEBAC5

Idem NON-SIT

OPTION : CHECKPOINTING 7

Type : A **Length : 3** optional parameter with default value

---

SIT

Bytes 1 and 2 : interval between two checkpoints expressed in Kbytes (N)

Special Values :
0 = no checkpoints
FFFF (hexadecimal) = unlimited interval
The smallest interval takes precedance over a larger or unlimited interval.

Byte 3 : window (N) (if bytes 1 and 2 are different from zero)
Special values :
0 = checkpoints are not acknowledged.

The interval between two checkpoints should be greater than or equal to 4 Kbytes. The window should be less than or equal to 16.

NON-SIT

Bytes 1 and 2 : interval between two checkpoints expressed in Kbytes (N)

Special Values :
0 = no checkpoints
FFFF (hexadecimal) = unlimited interval
The smallest interval takes precedance over a larger or unlimited interval.

Byte 3 : window (N) (if bytes 1 and 2 are different from zero)
Special values :
0 = checkpoints are not acknowledged.

ETEBAC5

Idem NON-SIT

FILE TYPE 11

SIT

Type : N Length : 2

The list of all file types has been given exhaustively in the network specification documents of the SIT.

NON-SIT

Type : N Length : 2

Value = 0 : no specific action required from the file transfer monitor
Other values : used to specify a particular action required between two file transfer monitors :

in an FPDU.MSG :

value FFFF (hexadecimal) : outgoing message
value FFFE (hexadecimal) : return message
other values : file reception acknowledgment

ETEBAC5

Type : C Length : 8

Byte 1: coding type (C)
value = 0 : mutually agreed
value = 1 : CFONB standard

Byte 2: syntax type (C)
value = 0 : CFONB records
value = 1 : EDIFACT messages

Bytes 3 to 7 : application nature (C) (cf. Annexe A2 of "Data exchange between Banks and their Corporate Customers")

Byte 8 : transfer nature (C)
value = 0 : data file
value = 1 : data file requesting an ETEBAC 5 execution order
value = 2 : execution order
value = 3 : execution report
value = 4 : bank confirmation
value = 5 : customer acknowledgment

FILE NAME 12

SIT

Type : C Length : 5

Numerical ASCII string.

NON-SIT

Type : C Length : 76

Alphanumerical ASCII string.

ETEBAC5

Type : C Length : 14

Byte 1 : identifier type (C)
value = 0 : mutually agreed
value = 1 : CFONB standard

IDENTIFIER :
if standard identifier :
Byte 2 : reference type (C)
value = 0 : reference by its name
value = 1 : request latest version
value = 2 : request non-transmitted versions
value = 3 : request all available versions

format 1 (type = 0)
Bytes 3 to 14 : file reference (C)

format 2 (type = 1, 2, 3)
Bytes 3 to 8 : start date : YYMMDD (D)
Bytes 9 to 14 : end date : YYMMDD (D)

TRANSFER IDENTIFIER 13

Type : N Length : 3

SIT

Numerical value chosen by the caller

NON-SIT

In the FPDU.CREATE
Non zero numerical value chosen by the caller.
If the transfer has been recovered, the transfer identifier should be identical to that used for the previous transfer try.

In the FPDU.ACK(CREATE) : optional parameter
Non zero numerical value chosen by the server.

In the FPDU.SELECT
Value = 0 for a new transfer.
For a recovered transfer, the value was determined by the server during the previous transfer try.

In the FPDU.ACK(SELECT)
Non zero numerical value chosen by the server.
(For a recovered transfer, identical value to that provided in the FPDU.SELECT).

ETEBAC5

Idem NON-SIT

REQUESTED ATTRIBUTES 14

Type : M **Length : 1** optional parameter with default value

SIT

Not used

NON-SIT

The appropriate bit is set to 1 if the corresponding attributes are required :
b1 : logical attributes
b2 : physical attributes
b3 : historical attributes
b4-b8 : must be set to 0 and otherwise ignored.
default value = no attributes

In the FPDU.MSG :
value = 0 : no message expected in the FPDU.ACK(MSG)
value = 1 : message expected in the FPDU.ACK(MSG)

ETEBAC5

Idem NON-SIT

RECOVERED TRANSFER 15

Type : S **Length : 1** optional parameter with default value

SIT

0 : new transfer
1 : recovered transfer

NON-SIT

Idem SIT

ETEBAC5

Idem SIT

DATA CODING 16

Type : S **Length : 1** optional parameter with default value

---

SIT

Not used

NON-SIT

Type : S **Length : 1**

0 = ASCII
1 = EBCDIC
2 = binary
> 2 : reserved for future use

ETEBAC5

Idem NON-SIT

TRANSFER PRIORITY 17

Type : S **Length : 1**

SIT

0 = priority 0 (urgent)
1 = priority 1 (semi-urgent)
2 = priority 2 (least urgent)

NON-SIT

Idem SIT

ETEBAC5

Idem SIT

RECOVERY POINT 18

Type : N **Length : 3**

SIT

Recovery point (0 = beginning of the file)

NON-SIT

Idem SIT

ETEBAC5

Idem SIT

END OF TRANSFER CODE 19

Type : S **Length : 1**

SIT

4 = error (a restart should follow)
8 = suspension
12 = annulation (by server)
16 = annulation (by caller)

NON-SIT

Idem SIT

ETEBAC5

Idem SIT

CHECKPOINT NUMBER 20

Type : N **Length : 3**

SIT

Checkpoint number (0 = beginning of the file)

NON-SIT

Idem SIT

ETEBAC5

Idem SIT

COMPRESSION 21

Type : A **Length : 2** optional parameter with default value

---

SIT

Not used

NON-SIT

Byte 1 : data compression
0 = no
1 = yes

Byte 2 : type of compression
1 = horizontal compression
2 = vertical compression
3 = combination of horizontal and vertical compression

ETEBAC5

Idem NON-SIT

ACCESS TYPE 22

Type : S Length : 1

SIT

0 = write

NON-SIT

0 = write
1 = read
2 = mixed (read or write)

ETEBAC5

Idem NON-SIT

RESTARTING 23

Type : S **Length : 1** optional parameter with default value

SIT

Not used

NON-SIT

0 = F.RESTART not autorised
1 = F.RESTART autorised

ETEBAC5

Idem NON-SIT

MAXIMUM SIZE OF A DATA ELEMENT 25

Type : N **Length : 2**

SIT

Maximum size of a data element measured in bytes. This length should be greater than 800 bytes. A data element of 4050 bytes should be able to be supported.

NON-SIT

Maximum size of a data element measured in bytes.

ETEBAC5

Idem NON-SIT

PROTOCOL MONITORING TIME-OUT 26

Type : N **Length : 2** optional parameter with default value

SIT

Prohibited

NON-SIT

Not used

ETEBAC5

Protocol monitoring time-out (in seconds)
30 = 30 seconds

NUMBER OF DATA BYTES 27

Type : N **Length : 8** Optional parameter

SIT

Prohibited

NON-SIT

Number of data bytes (excluding the length field for multi-article FPDUs), and including the header bytes of the compression string.
Mandatory parameter for PAD accesses.

ETEBAC5

Idem NON-SIT

NUMBER OF ARTICLES 28

Type : N **Length : 4** optional parameter

SIT

Prohibited

NON-SIT

Number of articles
Mandatory parameter for PAD accesses.

ETEBAC5

Idem NON-SIT

DIAGNOSTIC COMPLEMENTS 29

Type : A **Length : 254** optional parameter

SIT

Prohibited

NON-SIT

Optional parameter of un-specified format

ETEBAC5

Format 1 (diagnostic = 310)
byte 1 : cause X25 (N)
byte 2 : diagnostic X25 (N)

Format 2 (diagnostic = 318)
byte 1 : number of incorrect PIs (N)
byte 2 : PI code (N)
byte 3 : error code (S)
value = 1 : PI absent
value = 2 : syntax error
value = 3 : unsupported value
value = 4 : value outside limit
bytes 4 to 23 : description

Format 3 (diagnostic = 321)
bytes 1 to 15 : backup number (N)

Format 4 (diagnostic = 322)
bytes 1 to 6 : call back waiting time : HHMMSS (D)

ARTICLE FORMAT 31

Type : M **Length : 1** optional parameter with default value

---

SIT

value = 0 : fixed
value = 0x80 (hexadecimal) : variable

NON-SIT

Idem SIT

ETEBAC5

Idem SIT

ARTICLE LENGTH 32

Type : N **Length : 2**

SIT

Length of an article (in bytes)

NON-SIT

Idem SIT

ETEBAC5

Idem SIT

FILE ATTRIBUTES 33

Type : S **Length : 1** optional parameter with default value

---

SIT

0 = sequential
1 = relative
2 = indexed

NON-SIT

Idem SIT

ETEBAC5

Idem SIT

USE OF THE SIGNATURE 34

Type : N **Length : 2** optional parameter with default value

SIT

0 = no signature
1 = file signed by the SIT
For transfers from the CTB to the station this parameter is absent or has a value = 0
For transfers from the station to the CTB, value = 1

NON-SIT

Not used

ETEBAC5

Not used

SIT MAC

36

Type : N **Length : 64** conditional parameter

SIT

Present if the PI 34 value = 1

NON-SIT

Prohibited

ETEBAC5

Prohibited

FILE LABEL

37

Type : C **Length : 80** optional parameter

SIT

File label

NON-SIT

File label

ETEBAC5

File label

KEY LENGTH 38

Type : N **Length : 2** conditional parameter

SIT

Prohibited

NON-SIT

Parameter present if the file is indexed (PI 33 = 2).

ETEBAC5

Idem NON-SIT

KEY OFFSET 39

Type : N **Length : 2** conditional parameter with default value

SIT

Prohibited

NON-SIT

Offset in bytes of the key in the article.
Parameter present if the file is indexed (PI 33 = 2)
default value = 0

ETEBAC5

Idem NON-SIT

STORAGE RESERVATION UNIT 41

Type : S **Length : 1** optional parameter with default value

---

SIT

Storage reservation unit :
0 = Kbytes
1 = articles

The storage reservation unit should be expressed in kilo-bytes if the file is variable format (PI 31 = 0x80, hexadecimal)

NON-SIT

Idem SIT

ETEBAC5

Idem SIT

MAXIMUM RESERVED SPACE 42

Type : N **Length : 4**

SIT

Maximum reserved disk space

NON-SIT

Idem SIT

ETEBAC5

Idem SIT

DATE AND TIME OF CREATION 51

Type : D **Length : 12**

SIT

Date and time of creation (the dates and times are those defined by the SIT, may differ from legal date and time)
bytes 1 to 6 : date (YYMMDD)
bytes 7 to 12 : time (HHMMSS)

NON-SIT

Date and time of creation

ETEBAC5

Idem NON-SIT

DATE AND TIME OF LAST ACCESS 52

Type : D **Length : 12** optional parameter

SIT

Date and time of the last access
bytes 1 to 6 : date (YYMMDD)
bytes 7 to 12 : time (HHMMSS)

NON-SIT

Idem SIT

ETEBAC5

Not used

CUSTOMER IDENTIFIER 61

Type : C **Length : 24**

SIT

Prohibited

NON-SIT

optional parameter :
Identifier of the initial sender (cf annexe B : mode store and forward, for more information)

ETEBAC5

mandatory parameter :

Customer identification :

Bytes 1 to 3 : identifier type (C)
value = ZZZ : mutually agreed
value = 005 : CFONB standard

Bytes 4 to 24 : identifier (C)

if standard identifier
. corporate identifier
. service and individual identifier

BANK IDENTIFIER 62

Type : C **Length : 24**

SIT

Prohibited

NON-SIT

optional parameter :
Identification of the final receiver : (cf annexe B : mode store and forward, for more information)

ETEBAC5

mandatory parameter :

Bank identification :

Bytes 1 to 3 : identifier type (C)
value = ZZZ : mutually agreed
value = 005 : CFONB standard
value = ZBF : BdF code (bank + branch)

Bytes 4 to 24 : identifier (C)

if standard identifier
. corporate identifier
. service and individual identifier

FILE ACCESS CONTROL 63

Type : C **Length : 16** optional parameter

SIT

Prohibited

NON-SIT

Not used

ETEBAC5

Bytes 1 to 8 : current customer password
Bytes 9 to 16 : new password

SERVER DATE AND TIME 64

Type : D **Length : 12**

SIT

Prohibited

NON-SIT

Not used

ETEBAC5

Bytes 1 to 6 : date (YYMMDD)
Bytes 7 to 12 : time (HHMMSS)

AUTHENTICATION TYPE 71

Type : A **Length : 3** optional parameter with default value

SIT

Prohibited

NON-SIT

Prohibited

Secure NON-SIT

Byte 1 : presence of authentication (S)
0 = no
1 = yes

Byte 2 : used algorithm (S)
0 = RSA
1 = DES

Byte 3 : procedure (S)
0 = certificate exchange
1 = three-way authentication
2 = three-way authentication using only DES

ETEBAC5

Byte 1 : presence of authentication (S)
0 = no
1 = yes

Byte 2 : used algorithm (S)
0 = RSA

Byte 3 : procedure (S)
0 = certificate exchange
1 = three-way authentication

AUTHENTICATION ELEMENTS 72

Type : N **Length : n** conditional parameter

SIT

Prohibited

NON-SIT

Prohibited

Secure NON-SIT

Parameter present if PI 71, byte 1 = 1 and PI 71, byte 3 not equal to 0.

if PI 71, byte 3 = 1: cf : ETEBAC 5 profile

if PI 71, byte 3 = 2

in CREATE/SELECT :	KEKNAME1 name of the KEK1 encryption key : 8 bytes KAUTH1 authentication key encrypted under KEK1 : 8 bytes RN1 DES encrypted under KAUTH1 : 8 bytes
in ACK(CREATE/SELECT) :	KEKNAME2 name of the KEK2 encryption key : 8 bytes KAUTH2 authentication key encrypted under KEK2 : 8 bytes RN1* DES encrypted under KAUTH2 : 8 bytes RN2 DES encrypted under KAUTH2 : 8 bytes
in ORF :	RN2* DES encrypted under KAUTH2 : 8 bytes

* : cf annexe C : use of security mechanisms

ETEBAC5

Parameter present if PI 71, byte 1 = 1 and PI 71, byte 3 = 1.

in CREATE/SELECT :	RN1 : 8 bytes
in ACK(CREATE/SELECT) :	RN1 RSA encrypted under sender's secret key : 64 bytes RN2 : 8 bytes
in ORF :	RN2 RSA encrypted under sender's secret key : 64 bytes

MAC COMPUTATION TYPE 73

Type : A **Length:4** optional parameter with default value

SIT

Prohibited

NON-SIT

Prohibited

Secure NON-SIT

Byte 1: presence of MAC computation (S)
0 = no
1 = yes

Byte 2 : used algorithm (S)
1 = DES

Byte 3 : procedure (S)
1 = transmission of partial MACs, article based computation
2 = transmission of final MAC only, article based computation
3 = transmission of partial MACs, computed on complete file
4 = transmission of final MAC only, computed on complete file

Byte 4 : transfer of MAC computation elements (S)
0 = no transfer
1 = plaintext transfer
2 = RSA encrypted transfer
3 = DES encrypted transfer

ETEBAC5

Byte 1: presence of MAC computation (S)
0 = no
1 = yes

Byte 2 : used algorithm (S)
1 = DES

Byte 3 : procedure (S)
1 = transmission of partial MACs, article based computation
2 = transmission of final MAC only, article based computation
3 = transmission of partial MACs, computed on complete file
4 = transmission of final MAC only, computed on complete file

Byte 4 : transfer of MAC computation elements (S)
0 = no transfer
1 = plaintext transfer
2 = RSA encrypted transfer

MAC COMPUTATION ELEMENTS 74

Type : N **Length : n** conditional parameter

SIT

Prohibited

NON-SIT

Prohibited

Secure NON-SIT

MAC computation elements.

Parameter present if PI 73, byte 1 = 1 and PI 73, byte 4 different to 0.

if PI 73, byte 4 = 1 : MAC computation key K2 : 8 bytes
initialisation vector IV2

if PI 73, byte 4 = 2 : MAC computation key K2 + initialisation vector IV2, encrypted under the addressee's RSA public key : 64 bytes

if PI 73, byte 4 = 3 : KEKNAME name of the encryption key of key KEK : 8 bytes
MAC computation key K2 DES encrypted under KEK : 8 bytes
initialisation vector IV2 DES encrypted under KEK

ETEBAC5

MAC computation elements.

Parameter present if PI 73, byte 1 = 1 and PI 73, byte 4 different to 0.

if PI 73, byte 4 = 1 : MAC computation key K2 : 8 bytes
initialisation vector IV2

if PI 73, byte 4 = 2 : MAC computation key K2 + initialisation vector IV2, encrypted under the addressee's RSA public key : 64 bytes

ENCRYPTION TYPE 75

Type : A **Length : 4** optional parameter with default value

---

SIT

Prohibited

NON-SIT

Prohibited

Secure NON-SIT

Byte 1: presence of encryption (S)
0 = no
1 = yes

Byte 2 : used algorithm (S)
0 = RSA
1 = DES
2 = GOC
>= 3 : other

Byte 3 : procedure (S)
1 = article based computation
2 = computation on complete file

Byte 4 : transfer of encryption elements (S)
0 = no transfer
1 = RSA encrypted transfer
2 = DES encrypted transfer

ETEBAC5

Byte 1: presence of encryption (S)
0 = no
1 = yes

Byte 2 : used algorithm (S)
1 = DES

Byte 3 : procedure (S)
1 = article based computation
2 = computation on complete file

Byte 4 : transfer of encryption elements (S)
0 = no transfer
1 = RSA encrypted transfer

ENCRYPTION ELEMENTS 76

Type : N **Length : n** conditional parameter

SIT

Prohibited

NON-SIT

Prohibited

Secure NON-SIT

Encryption elements.

Parameter present if PI 75, byte 1 = 1 and PI 75, byte 4 different of 0.

if PI 75, byte 4 = 1 : encryption key (K1) + initialisation vector (IV1) encrypted under the addressee's RSA public key : 64 bytes

if PI 75, byte 4 = 2 : KEKNAME name of the encryption key of key KEK : 8 bytes
encryption key (K1) DES encrypted under KEK : 8 bytes
initialisation vector (IV1) DES encrypted under K1 : 8 bytes
initialisation vector (IV1)* DES encrypted under K1 : 8 bytes

ETEBAC5

Encryption elements.

Parameter present if PI 75, byte 1 = 1 and PI 75, byte 4 different to 0.

if PI 75, byte 4 = 1 : encryption key (K1) + initialisation vector (IV1) encrypted under the addressee's RSA public key : 64 bytes

DIGITAL SIGNATURE TYPE 77

Type : A **Length : 4** optional parameter with default value

---

SIT

Prohibited

NON-SIT

Prohibited

Secure NON-SIT

Byte 1: presence of digital signature (S)
0 = no
1 = yes

Byte 2 : used algorithm (S)
0 = RSA

Byte 3 : procedure (S)
1 = ETEBAC 5 digital signature
2 = MAC digital signature

Byte 4 : double signature (S)
1 = single signature
2 = double signature

ETEBAC5

Byte 1: presence of digital signature (S)
0 = no
1 = yes

Byte 2 : used algorithm (S)
0 = RSA

Byte 3 : procedure (S)
1 = ETEBAC 5 digital signature

Byte 4 : double signature (S)
1 = single signature
2 = double signature

MAC 78

Type : N **Length : 4** conditional parameter

SIT

Prohibited

NON-SIT

Prohibited

Secure NON-SIT

MAC.

The MAC is the result of applying the DES encryption algorithm to the file data and to the parameters which make up the file FID : PI 11, PI 12, PI 51, PI 61, PI 62.

Parameter present in the FPDU.SYN if the PI 73 byte 1 = 1 and PI 73 byte 3 = 1.
Parameter present in the FPDU.DTF.END if PI 73 byte 1 = 1.

ETEBAC5

MAC.

The MAC is the result of applying the DES encryption algorithm to the file data and to the parameters which make up the file FID : PI 11, PI 12, PI 51, PI 61, PI 62.

Parameter present in the FPDU.SYN if the PI 73 byte 1 = 1 and PI 73 byte 3 = 1.

DIGITAL SIGNATURE 79

Type : N **Length : 64** conditional parameter

SIT

Prohibited

NON-SIT

Prohibited

Secure NON-SIT

Digital signature.

The signature is the result of an RSA encryption under the sender's secret key of the MAC related to the file (file data and FID) and of the MAC related to the FID only : parameters PI 11, PI 12, PI 51, PI 61, PI 62.

Parameter present in the FPDU.DTF.END if PI 73 byte 1 = 1 and PI 77 byte 1 = 1.

ETEBAC5

Digital signature.

The signature is the result of an RSA encryption under the sender's secret key of the MAC related to the file (file data and FID) and of the MAC related to the FID only : parameters PI 11, PI 12, PI 51, PI 61, PI 62.

Parameter present in the FPDU.DTF.END if PI 73 byte 1 = 1 and PI 77 byte 1 = 1.

CERTIFICATE 80

Type : N **Length : 168** conditional parameter

SIT

Prohibited

NON-SIT

Prohibited

Secure NON-SIT

Certificate.

The nature of the certificate used in the secure Non-SIT PeSIT profile is left up to the choice of the user.

ETEBAC5

Certificate.

Byte 1 (C) : certificate type
0 = authentication
1 = digital signature
2 = test

Bytes 2 to 25 (C) : owner identification IDi

Bytes 26 to 37 (C) : device type and serial number NSi

Bytes 38 to 103 (N) owner's public key (modulus + exponent) CPi

Byte 104 (C) : Certification Authority RSA key pair reference

Bytes 105 to 168 (N) : certificate digital signature

ACKNOWLEDGEMENT OF DIGITAL SIGNATURE 81

Type : N **Length : 64** conditional parameter

SIT

Prohibited

NON-SIT

Prohibited

Secure NON-SIT

Acknowledgement of the digital signature.

The use and the nature of the acknowledgement of the digital signature in the secure Non-SIT PeSIT profile is left up to the choice of the user.

ETEBAC5

Acknowledgement of the digital signature.

The acknowledgement of the digital signature is the result of an RSA encryption under the sender's secret key of : the received MACs + date and time + ACK/NAK.

The field acknowledement contains two bytes :

byte 1 : check of security elements (C)
0 = accepted
1 = rejected
2 = not carried out

byte 2 : acknowledgement value (C)
X = meaningless
0 = file transmitted for processing (if protocol integrated security)

SECOND DIGITAL SIGNATURE 82

Type : N **Length : 64** conditional parameter

SIT

Prohibited

NON-SIT

Prohibited

Secure NON-SIT

Not used.

ETEBAC5

Second digital signature.

Parameter present in the FPDU.DTF.END if PI 73 byte 1 = 1, PI 77 byte 1 = 1, PI 77 byte 4 = 2.

Signature of same format as the PI 79 but obtained with a different secret key (cf annexe C : Use of security mechanisms).

SECOND CERTIFICATE 83

Type : N **Length : 168** conditional parameter

SIT

Prohibited

NON-SIT

Prohibited

Secure NON-SIT

Not used

ETEBAC5

Second certificate.

Certificate of the same format as the PI 80 but relative to a different key pair (cf annexe C : Use of security mechanisms).

DATAGRAM 91

Type : N **Length : 4096** optional parameter

SIT

Not used

NON-SIT/Secure NON-SIT

Datagram

ETEBAC5

Not used

FREE TEXT 99

Type : **Length : 254** optional parameter

SIT

Not used

NON-SIT

Free text

No check is made by the protocol on the coding, the structure or the semantics of the contents of the free text.

ETEBAC5

Free text

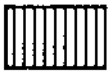
The ETEBAC 5 standard imposes that the contents of the free text should be of character type (ASCII coding).

4.7.4 Protocol element structure

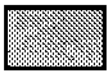
The following pages give the stucture of each message of the protocol for each profile.

The representation provides a list of the PI and PGI for each message and whether the PI is optional or conditional.

Optional PI



Optional PI with a default value



Conditional PI

EXAMPLE :

profile : SIT

PGI code :	9							
PI code :		3	4	11	12	13	15	17	25

KEY :

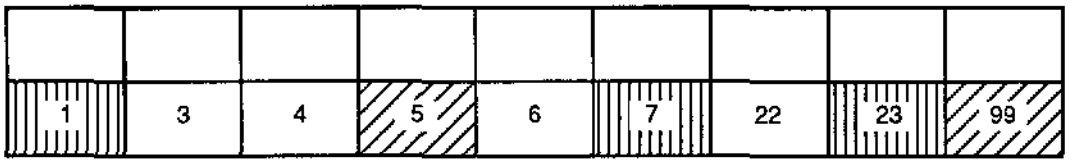
(W) : PI present during a write transfer only
(R) : PI present during a read transfer only

1) MESSAGE TYPE 20 = FPDU.CONNECT

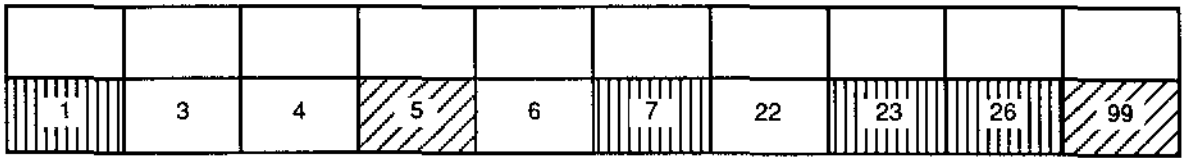
SIT

The FPDU.CONNECT sent by the station does not contain a PI 5. It is accepted that the FPDU.CONNECT sent by the CTB contain a PI 5, in which case it is ignored by the station.

NON-SIT/Secure NON-SIT

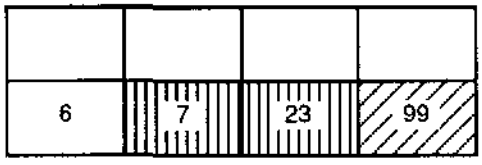


ETEBAC5



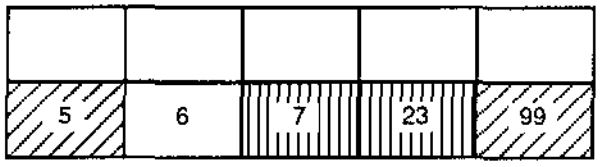
2) MESSAGE TYPE 21 = FPDU.ACONNECT

SIT



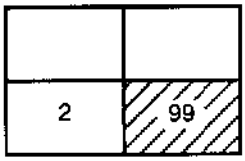
The FPDU.ACONNECT sent by the station does not contain a PI 5. It is accepted that the FPDU.ACONNECT sent by the CTB contain a PI 5, in which case it is ignored by the station.

NON-SIT/Secure NON-SIT/ETEBAC 5

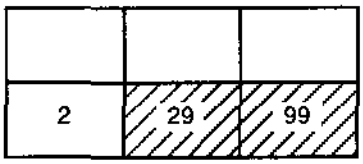


3) MESSAGE TYPE 22 = FPDU.RCONNECT

SIT/NON-SIT/Secure NON-SIT

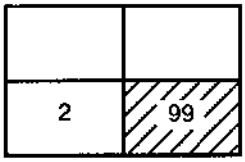


ETEBAC5

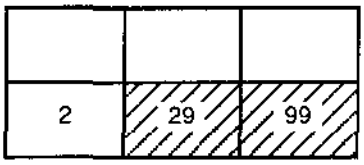


4) MESSAGE TYPE 23 = FPDU.RELEASE

SIT/NON-SIT/Secure NON-SIT

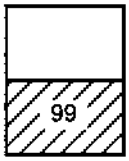


ETEBAC5



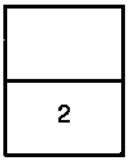
5) MESSAGE TYPE 24 = FPDU.RELCONF

SIT/NON-SIT/Secure NON-SIT/ETEBAC 5



6) MESSAGE TYPE 25 = FPDU.ABORT

SIT/NON-SIT/Secure NON-SIT



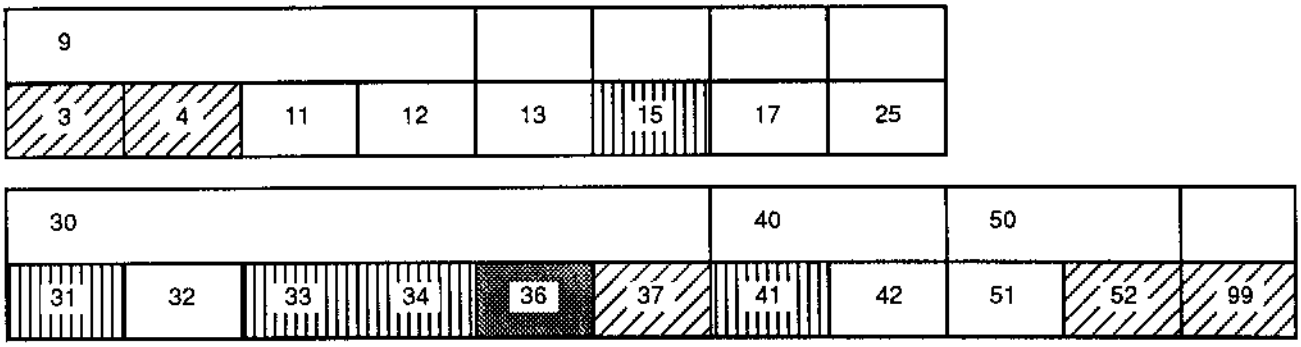
ETEBAC 5

Note :

The code PI and the length are not present in the FPDU-ABORT, because use of the S-ABORT service in PeSITonly allows 9 bytes in the user data field (cf : §4.3.1).

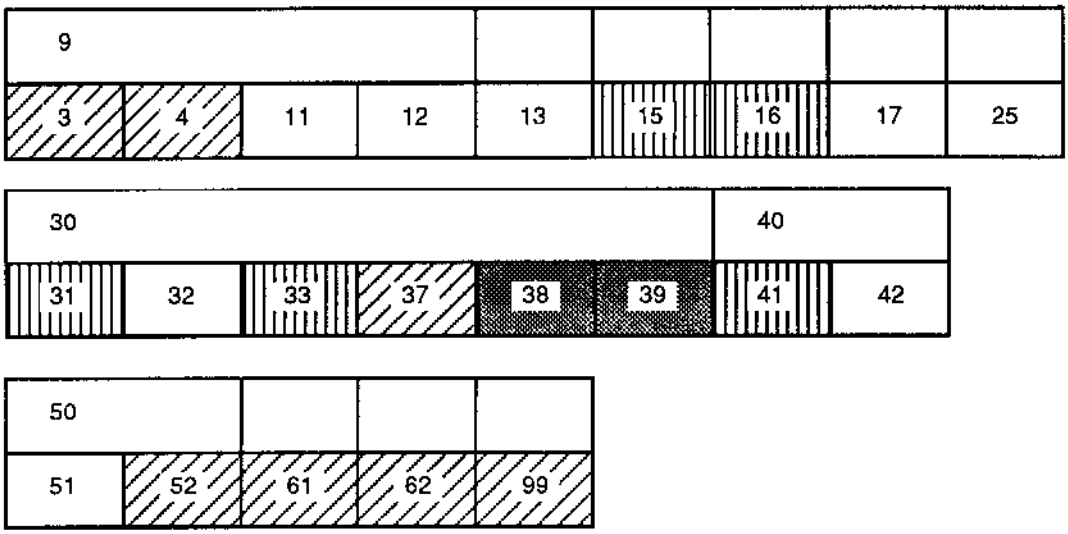
7) MESSAGE TYPE 11 = FPDU.CREATE

SIT

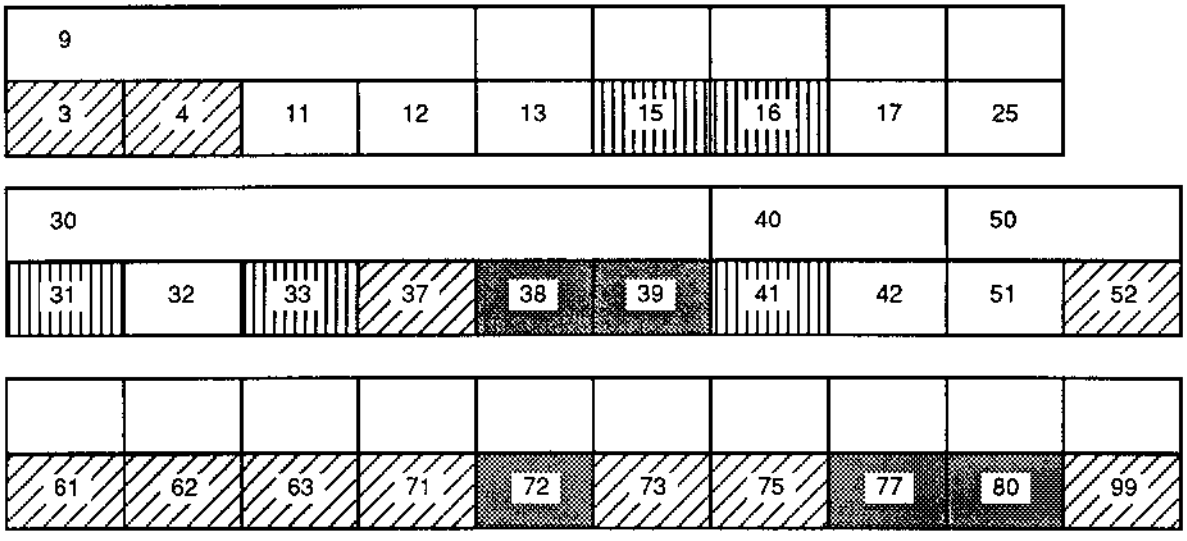


The FPDU.CREATE sent by the station does not contain a PI 16. It is accepted that the FPDU.CREATE sent by the CTB contain a PI 16, in which case it is ignored by the station.

NON-SIT

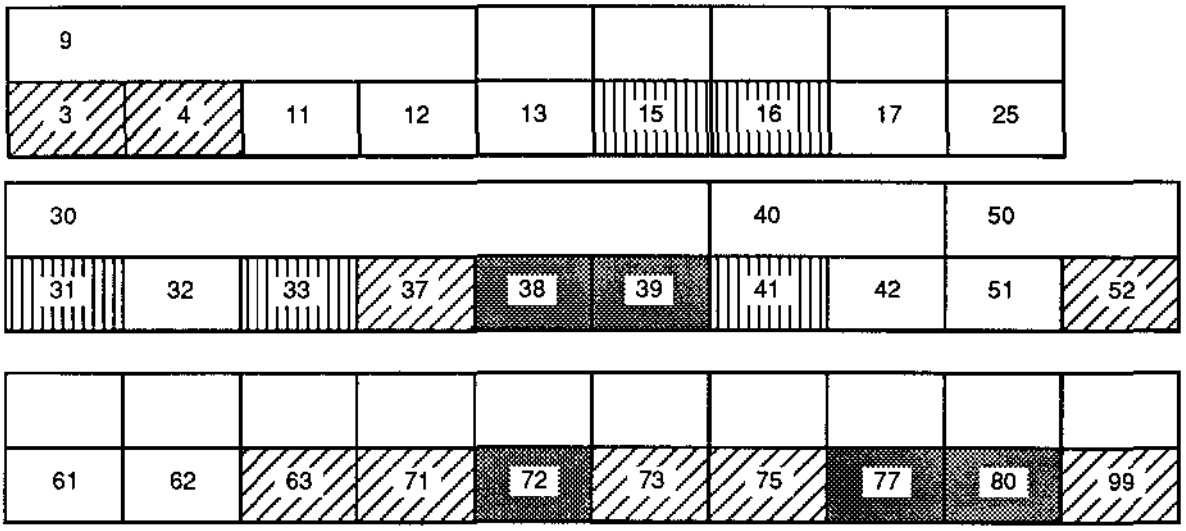


Secure NON-SIT



7) MESSAGE TYPE 11 = FPDU.CREATE (following)

ETEBAC5



8) MESSAGE TYPE 30 = FPDU.ACK (CREATE)

SIT

2	25	99

NON-SIT

2	13	25	99

Secure NON-SIT

2	13	25	72	80	83	99

ETEBAC 5

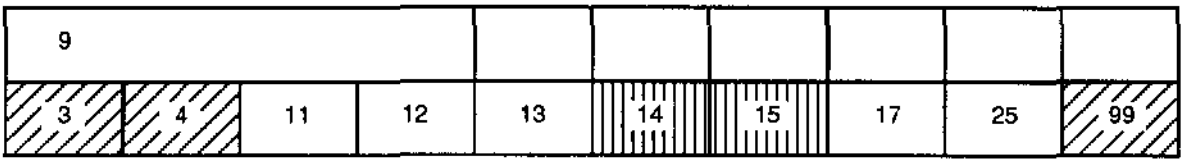
2	13	25	29	64	72	80	83	99

9) MESSAGE TYPE 12 = FPDU.SELECT

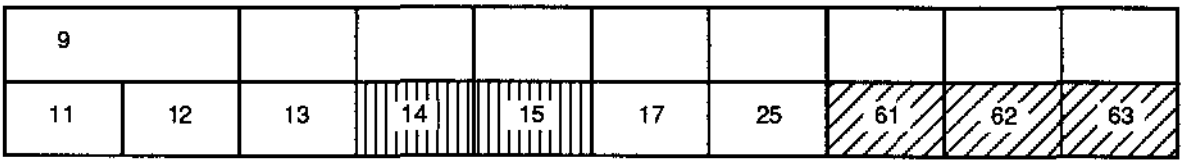
SIT

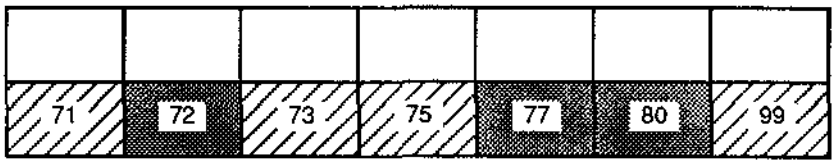
This FPDU is not used in the SIT profile

NON-SIT

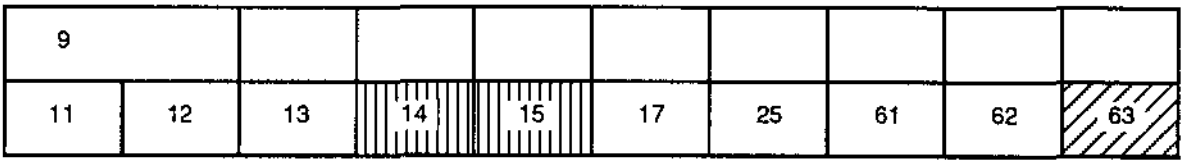


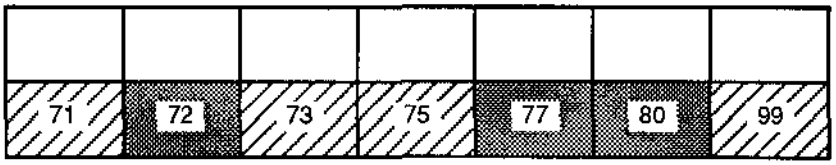
Secure NON-SIT





ETEBAC 5



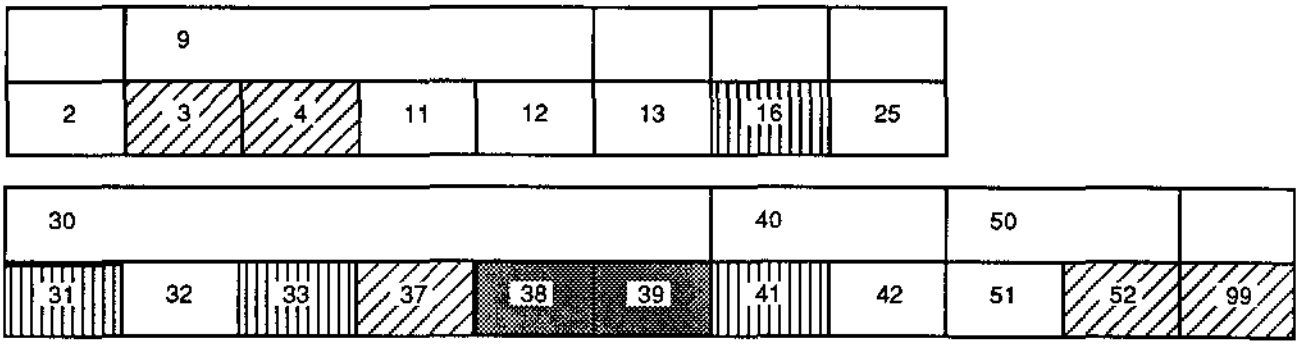


10) MESSAGE TYPE 31 = FPDU.ACK (SELECT)

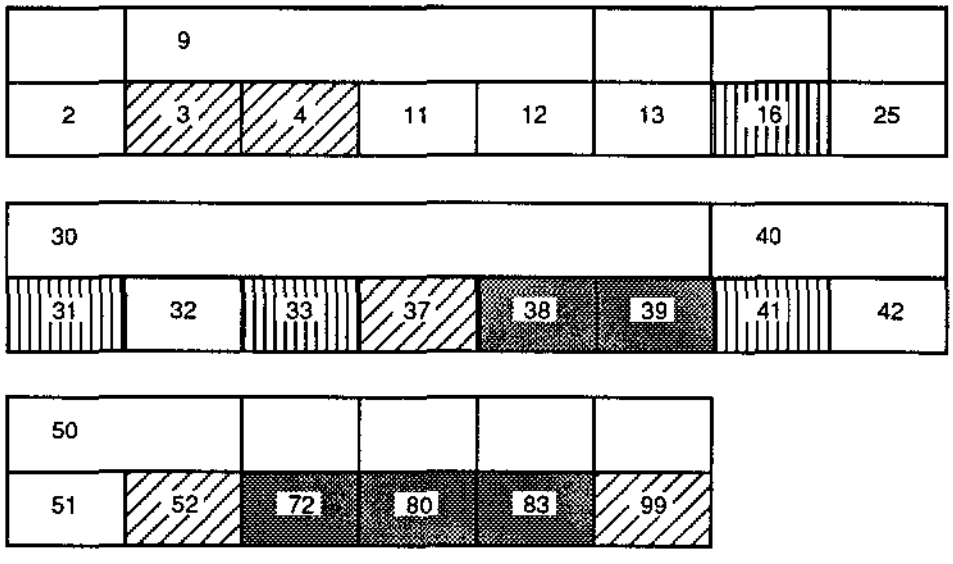
SIT

This FPDU is not used by the SIT profile.

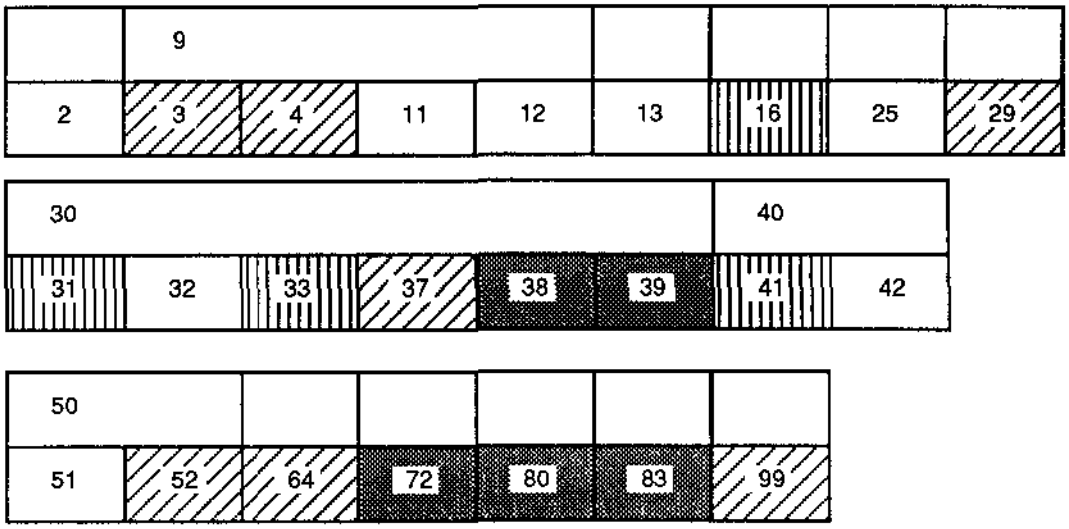
NON-SIT



Secure NON-SIT

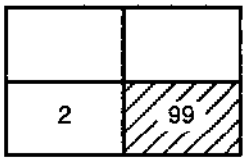


ETEBAC 5

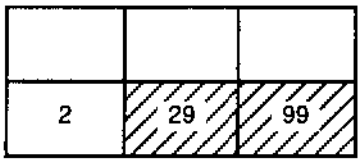


11) MESSAGE TYPE 13 = FPDU.DESELECT

SIT/NON-SIT/Secure NON-SIT

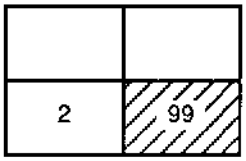


ETEBAC5

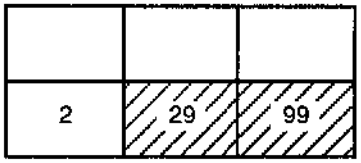


12) MESSAGE TYPE 32 = FPDU.ACK(DESELECT)

SIT/NON-SIT/Secure NON-SIT



ETEBAC5



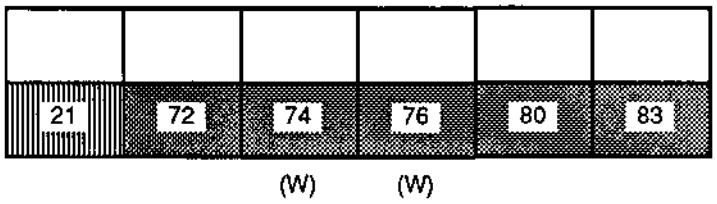
13) MESSAGE TYPE 14 = FPDU.ORF

SIT

The FPDU.ORF sent by the station does not contain any parameter. It is accepted that the FPDU.ORF sent by the CTB contains a PI 21, in which case it is ignored by the station.

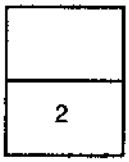
NON-SIT

Secure NON-SIT/ETEBAC 5



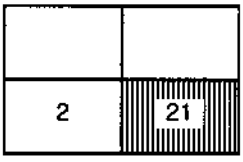
14) MESSAGE TYPE 33 = FPDU.ACK(ORF)

SIT



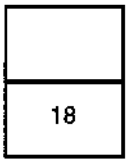
The FPDU.ACK(ORF) sent by the station does not contain a parameter PI 21. It is accepted that the FPDU.ACK(ORF) sent by the CTB contains a PI 21, in which case it is ignored by the station.

NON-SIT



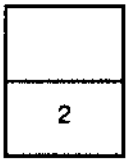
17) MESSAGE TYPE 01 = FPDU.READ

SIT/NON-SIT/Secure NON-SIT/ETEBAC 5

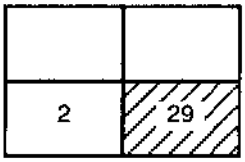


18) MESSAGE TYPE 35 =FPDU.ACK(READ)

SIT/NON-SIT/Secure NON-SIT



ETEBAC 5



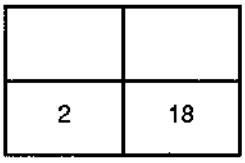
19) MESSAGE TYPE 02 = FPDU.WRITE

SIT/NON-SIT/Secure NON-SIT/ETEBAC 5

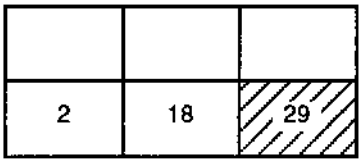
does not contain any parameters

20) MESSAGE TYPE 36 = FPDU.ACK(WRITE)

SIT/NON-SIT/Secure NON-SIT



ETEBAC 5

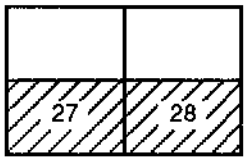


21) MESSAGE TYPE 08 = FPDU.TRANS.END

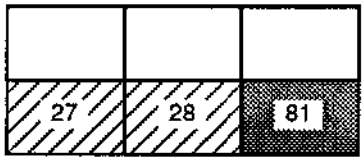
SIT

does not contain any parameters

NON-SIT



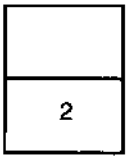
Secure NON-SIT/ETEBAC 5



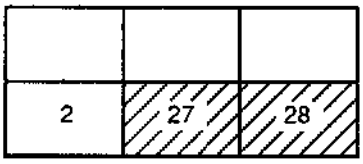
(R)

22) MESSAGE TYPE 37 = FPDU.ACK(TRANS.END)

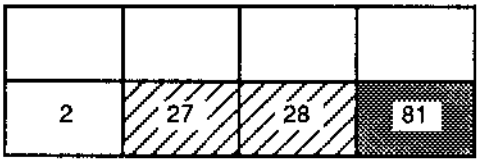
SIT



NON-SIT

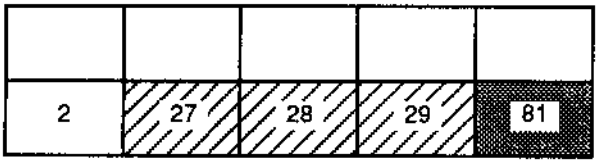


Secure NON-SIT



(W)

ETEBAC 5



23) MESSAGE TYPE 00 = FPDU.DTF

SIT/NON-SIT/Secure NON-SIT/ETEBAC5

The FPDU.DTF contains the file data but does not contain any parameters.

Note : The FPDU.DTF cannot be empty.

24) MESSAGE TYPE 41 = FPDU.DTFDA
MESSAGE TYPE 40 = FPDU.DTFMA
MESSAGE TYPE 42 = FPDU.DTFFA

SIT

These FPDU are not used in the SIT

NON-SIT/Secure NON-SIT/ETEBAC 5

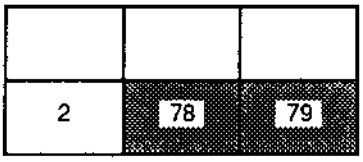
The FPDU.DTFDA, FPDU.DTFMA, FPDU.DTFFA contain part of an article of the file but do not contain any parameters.

Note : the FPDU.DTFDA, FPDU.DTFMA, FPDU.DTFFA cannot be empty.

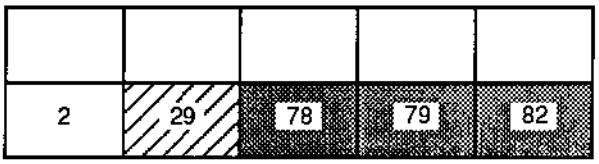
25) MESSAGE TYPE 04 = FPDU.DTF.END

SIT/NON-SIT

Secure NON-SIT



ETEBAC 5



26) MESSAGE TYPE 03 = FPDU.SYN

SIT/NON-SIT

20

Secure NON-SIT/ETEBAC 5

20	78

27) MESSAGE TYPE 38 = FPDU.ACK(SYN)

SIT/NON-SIT/Secure NON-SIT/
ETEBAC5

20

28) MESSAGE TYPE 05 = FPDU.RESYN

SIT

This FPDU is not used in the SIT.

NON-SIT/Secure NON-SIT

2	18

ETEBAC 5

2	18	29

29) MESSAGE TYPE 39 = FPDU.ACK(RESYN)

SIT

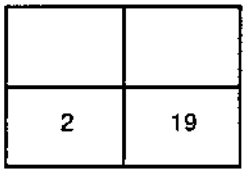
This FPDU is not used in the SIT.

NON-SIT/Secure NON-SIT/ETEBAC5

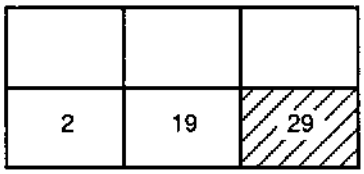
18

30) MESSAGE TYPE 06 = FPDU.IDT

SIT/NON-SIT/Secure NON-SIT



ETEBAC 5



31) MESSAGE TYPE 3A = FPDU.ACK(IDT)

SIT, NON-SIT, Secure NON-SIT,ETEBAC5

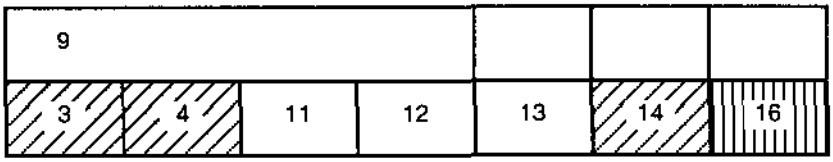
does not contain any parameters.

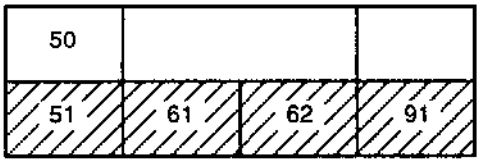
32) MESSAGE TYPE 16 = FPDU.MSG

SIT

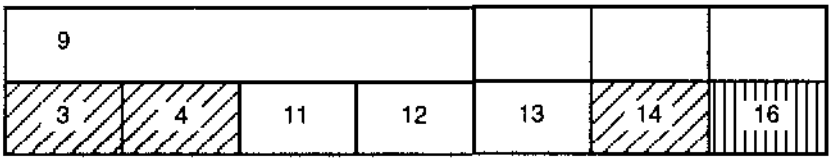
This FPDU is not used in the SIT.

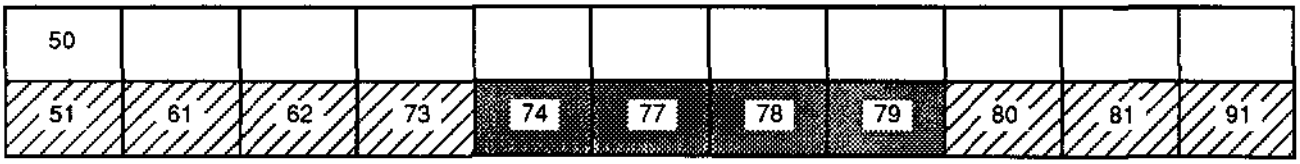
NON-SIT





Secure NON-SIT





ETEBAC 5

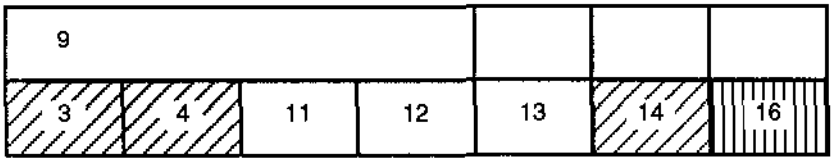
This FPDU is not used.

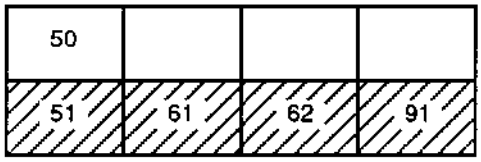
33) MESSAGE TYPE 17 = FPDU.MSGDM

SIT

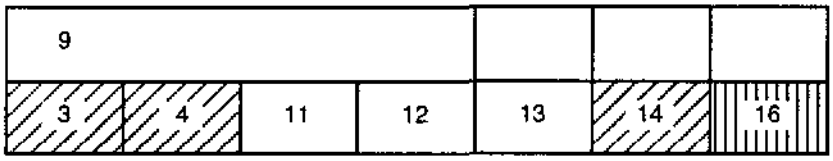
This FPDU is not used in the SIT.

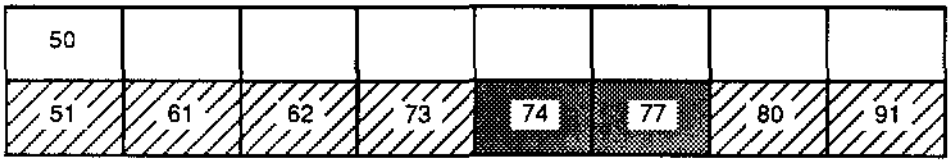
NON-SIT





Secure NON-SIT





ETEBAC 5

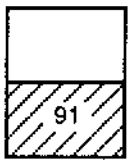
This FPDU is not used.

34) MESSAGE TYPE 18 = FPDU.MSGMM

SIT

This FPDU is not used in the SIT.

NON-SIT/Secure NON-SIT



ETEBAC 5

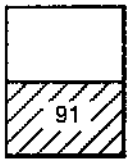
This FPDU is not used.

35) MESSAGE TYPE 19 = FPDU.MSGFM

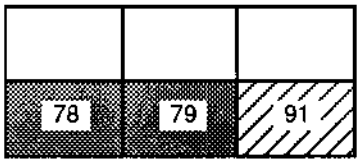
SIT

This FPDU is not used in the SIT.

NON-SIT



Secure NON-SIT



ETEBAC 5

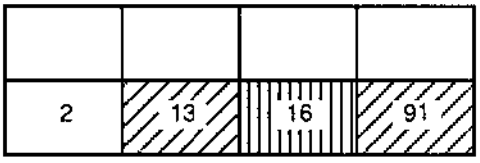
This FPDU is not used.

36) MESSAGE TYPE 3B = FPDU.ACK(MSG)

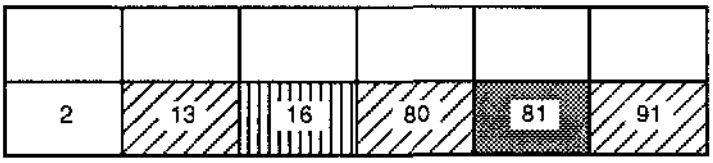
SIT

This FPDU is not used in the SIT.

NON-SIT



Secure NON-SIT



ETEBAC 5

This FPDU is not used.

4.8 PeSIT PROTOCOL STATE MACHINE TABLES

4.8.1 Formal description elements

This chapter provides a formal description of the protocol in terms of a complete finite state machine with the states, events and associated actions.

This description is equally applicable to PeSIT.F, PeSIT.F', PeSIT.F" and PeSIT.F'".

The state tables described in this chapter indicate, for each state possible during a PeSIT connection, the events which may occur in the protocol, the actions to be carried out and the resulting state.

Before describing the state tables, the abreviations used will be defined.

4.8.1.1 States

ABREVIATION	NAME AND DESCRIPTION
CN...	PeSIT establishment regime
CN01	IDLE, not connected
CN02A	Connection pending reception of an FPDU.ACONNECTor RCONNECT
CN02B	Connection pending reception of an F.CONNECT, R primitive
CN03	CONNECTED
CN04B	Release pending reception of an FPDU.RELCONF
CN04B	Release pending reception of an F.RELEASE, R primitive
SF...	SELECT FILE REGIME
SF01A	File create pending reception of an FPDU.RELCONF
SF01B	File create pending reception of an F.CREATE, R primitive
SF02A	File seclect pending reception of an FPDU.ACK(SELECT)
SF02B	File seclect pending reception of an F.SELECT, R primitive
SF03	FILE SELECTED
SF04A	File release pending reception of an FPDU.ACK(DESELECT)
SF04B	File release pending reception of an F. DESELECT, R primitive
OF...	FILE OPEN REGIME
OF01A	Open file pending reception of an FPDU.ACK(ORF)
OF01B	Open file pending reception of an F.OPEN, R primitive
OF02	DATA TRANSFER - IDLE
OF03A	File close pending reception of an FPDU.ACK(CRF)
OF03B	File close pending reception of an F.CLOSE, R primitive

ABREVIATION	NAME AND DESCRIPTION
TDL...	BULK DATA TRANSFER - REGIME
TDL01A	Read pending reception of an FPDU.ACK(READ) (Caller)
TDL01B	Read pending reception of an F.READ,R primitive (Server)
TDL02A	Data reception (Caller)
TDL02B	Data transmission (Server)
TDL03	Restarting pending reception of an FPDU.ACK(RESYN)
TDL04	Restarting pending reception of an F.RESTART,R primitive
TDL05	Transfer interruption pending reception of an FPDU.ACK(IDT)
TDL06	Transfer interruption pending reception of an F.CANCEL, R primitive
TDL07	End read
TDL08A	End read transfer pending reception of an FPDU.ACK(TRANS.END) (Caller)
TDL08B	En read transfer pending reception of an F.TRANSFER.END,R primitive (Server)
TDE...	BULK DATA TRANSFER - REGIME
TDE01A	Write pending reception of an FPDU.ACK(WRITE) (Caller)
TDE01B	Write pending reception of an F.WRITE, R primitive (Server)
TDE02A	Data transmission (Caller)
TDE02B	Data reception (Server)
TDE03	Restarting pending reception of an FPDU.ACK(RESYN)
TDE04	Restarting pending reception of an F.RESTART,R primitive
TDE05	Transfer interruption pending reception of an FPDU.ACK(IDT)
TED06	Transfer Interruption pending reception of an F.CANCEL, R primitive
TDE07	End write
TDE08A	End write transfer pending reception of an FPDU.ACK(TRANS.END) (Caller)
TDE08B	End write transfer pending reception of an F.TRANSFER.END, R primitive (Server)

Note :
The F.XXXX,R abbreviation indicates a response primitive.

All states ending in the letter "A" concern the PeSIT caller and all those ending in the letter "B" concern the server. The other states are common to both units.

4.8.1.2 Events

In order to describe the complete finite state machine of a PeSIT unit, the incoming events are :

- either the reception of a service primitive from the local user (request or response),

- or the reception of an FPDU from the opposite PeSIT,

- or the detection of an error (expiration of a monitoring time-out or a protocol error).

The main outgoing events are comprised of the emission of :

- a service primitive towards the local user (indication or confirmation),

- or a message intended for the opposite PeSIT unit.

The following abreviations are used to define the events :

XXX : FPDU.XXX

A (XXX) : FPDU.ACK replying to an FPDU.XXX

YYY(D) : request primitive for the YYY service

YYY(I) : indication primitive for the YYY service

YYY(R) : response primitive for the YYY service

YYY(C) : confirmation primitive for the YYY service.

The allowed values for XXX and YYY are given in 4.2.

4.8.1.3 Conditions

The state transitions and the actions are sometimes conditional. The abreviations related to these conditions are defined as follows :

ABREVIATION	NAME AND DESCRIPTION
+	Positive ACK (OK or warning)
-	Negative ACK (correctable or blocking)
ab	Premature transfer termination (1)
ck	Checkpointing option negotiated
dr	The caller is the receiver
rel	Recovery requested
cft	end of transfer code = cancel or suspend
cc>0	There are unacknowledged checkpoints

(1) "ab" is set by an F.CANCEL when the "end of transfer code" = suspend or cancel. It is reset by an F.DESELECT. It is used to prevent F.READ (or WRITE) or F.OPEN from looping if a transfer is terminated prematurely.

The following symbol - condition equivalence is used :

- & AND.

4.8.1.4 Actions

The actions may be conditional or unconditional. Any particular action consists in :

- sending an outgoing event, indicated by its abbreviated name,

- or executing a specific action,

- or imposing a condition.

When several actions are specified, they must be executed in the order shown.

a) Implicit actions

A certain number of actions are not specified in the state tables, because they are implicit. These actions are :

- when there is an empty intersection in the table (the corresponding event is invalid), the connection is prematurely terminated (ABORT) with an appropriate diagnostic code,

- for each FPDU received, a systematic check is made (validation of the parameters used). If an error is detected, the recovery procedure is invoked,

- if a premature termination request is received, the following state is always "idle" regardless of the current state (except CN01), the ABORT is notified to the local user or the opposite PeSIT unit and all the parameters are re-initialised.

b) Specific actions

REC	The following message was received from the "communication system"
SAB	Set premature termination flag
RESAB	Reset premature termination flag
SCK	Set checkpoint negotiated flag
RESCK	Reset checkpoint negotiated flag
SDR	Set caller = receiver flag
RESDR	Reset caller = receiver flag
SREL	Set recovery flag
RESREL	Reset recovery flag.

4.8.2 Conventions

To simplify the presentation, each state table is a part of the general table. The left column indicates the events and the uppermost line the different states.

Each intersection in the table between an event and a state contains :

general condition	(optional)
(cond :) NEXT STATE ...	(mandatory)
(cond :) ACTION ...	(optional)

in which :

- the general condition is a condition which is valid for the entire intersection (expressed as "COND : cond"),

- (cond :) is an optional condition which applies to the following line : the conditions are always written in lower case letters,

- NEXT STATE : is the next state once all the indicated actions have been executed. The states are always written in upper case letters,

- ACTION entails sending an outgoing event, imposing a condition or executing a specific action. If several actions are specified, the must be executed in that order. The actions are always written in upper case letters.

The following special characters are also used :

":" end of condition code,

";" separator between list items.

An intersection in the table is considered to be invalid if no indication is given or if the general condition indicated by "COND" is not met.

All invalid intersections between states and "incoming service primitive" events (request or response) are considered to be local errors and as such are not standardised in this document.

When the general state table was broken down into smaller sub-sections, all the lines and all the columns of the sub-tables which contained empty intersections were removed. Thus any intersection which does not occur in the state tables is implicitly invalid.

Intersections marked "null" are valid event/state intersections were no action is executed and the finite state macine remains in the same state.

4.8.3 Collision rules

In the data transfer phase, collision cases may occur. The following rules should be applied in these cases :

1. Priority order of the FPDUs

   1 - ABORT
   2 - IDT
   3 - RESYN
   4 - TRANS.END
   5 - ACK.SYN
   6 - DTF.END
   7 - DTF

2. If a collision occurs between two FPDUs of the same type, then the caller is given priority over the server.

4.8.4 State tables

There are two sets of state tables which are very depending on the roles assumed by the PeSIT units : caller or server. For the data transfer state machine, four separate tables have been defined :

- caller-sender, caller-receiver, server-receiver, server-sender.

TABLE	PROTOCOL PHASE	ROLE
1	regime establishment	caller
2	regime establishment	server
3	file select	caller
4	file select	server
5	file open	caller
6	file open	server
7	data transfer	caller-sender (write)
8	data transfer	server-receiver (write)
9	data transfer	caller-receiver (read)
10	data transfer	server-sender (read)

TABLE 1 : regime establishment (caller)

	CN01	CN02A	CN03	CN04A
F.CONNECT(D)	CN02A CONNECT			
A.CONNECT		CN03 F.CONNECT(C) ck : SCK		
R.CONNECT		CN01 F.CONNECT(C)		
F.RELEASE(D)			CN04A RELEASE	
RELCONF				CN01 F.RELEASE(C) ck : RESCK
F.ABORT(D)		CN01 ABORT	CN01 ABORT	CN01 ABORT
ABORT		CN01 F.ABORT(I)	CN01 F.ABORT(I)	CN01 F.ABORT(I)

TABLE 2 : regime establishment (server)

	CN01	CN02B	CN03	CN04B
CONNECT(R)	CN02B F.CONNECT(I)			
F.CONNECT(R)		+ : CN03 - : CN01 + : A.CONNECT ck : SCK - : R.CONNECT		
RELEASE			CN04B F.RELEASE(I)	
F.RELEASE(R)				CN01 RELCONF ck : RESCK
ABORT		CN01 F.ABORT(I)	CN01 F.ABORT(I)	CN01 F.ABORT(I)
F.ABORT(D)		CN01 ABORT	CN01 ABORT	CN01 ABORT

TABLE 3 : File selection phase (caller)

	CN03	SF01A	SF02A	SF03	SF04A
F.SELECT(D)	SF02A SELECT				
A.(SELECT)			+ : SF03 - : CN03 F.SELECT(C) ; SDR rel : SREL		
F.CREATE(D)	SF01A CREATE				
A.(CREATE)		+ : SF03 - : CN03 F.CREATE(C) rel : SREL			
F.DESELECT(D)				SF04A DESELECT	
A(DESELECT)					CN03 F.DESELECT(C) ab : RESAB dr : RESDR rel : RESREL

TABLE 4 : File selection phase (server)

	CN03	SF01B	SF02B	SF03	SF04B
SELECT	SF02B F.SELECT(I)				
F.SELECT(R)			+ : SF03 - : CN03 A(SELECT) rel : SREL		
CREATE	SF01B F.CREATE(I)				
F.CREATE(R)		+ : SF03 - : CN03 A(CREATE) rel : SREL			
DESELECT				SF04B F.DESELECT(I)	
F.DESELECT(R)					CN03 A(DESELECT) ab : RESAB

TABLE 5 : File opening phase (caller)

	SF03	OF01A	OF02	OF03A
F.OPEN (D)	COND : NOT ab OF01A CRF			
A(ORF)		+ : OF03 - : SF03 F.OPEN (C)		
F.CLOSE (D)			OF03A CRF	
A(CRF)				SF03 F.CLOSE (C)

TABLE 6 : File opening phase (server)

	SF03	OF01B	OF02	OF03B
CRF	COND : NOT ab OF01B F.OPEN (I)			
F.OPEN (R)		+ : OF02 - : SF03 A(ORF)		
CRF			OF03B F.CLOSE (I)	
F.CLOSE (R)				SF03 A (CRF)

TABLE 7 : Data transfer phase Caller-Sender

	OF02	TDE01A	TDE02A	TDE03	TDE04	TDE05	TDE06	TDE07	TDE08A
F.WRITE(D)	COND.NCT ab TDE01A WRITE								
A(WRITE)		+:TDE02A -:OF02 F.WRITE(C)							
F.CANCEL(D)			TDE05 IDT	TDE05 IDT	TDE05 IDT		TDE05 IDT	TDE05 IDT	TDE05 IDT
A(IDT)						OF02 F.CANCEL(C) SAB			
IDT			TDE06 F.CANCEL(I)	TDE06 F.CANCEL(I)	TDE06 F.CANCEL(I)	NULL	NULL	TDE06 F.CANCEL(I)	TDE06 F.CANCEL(I)
F.CANCEL(R)							OF02 A(IDT) SAB		
F.DATA.END(D)			TDE07 DTF.END crt : SAB		NULL		NULL		
F.TRANSFER.END(D)					NULL			TDE08A TRANS.END	
A(TRANS.END)				NULL		NULL			OF02 F.TRANSFER. END(C)
F.DATA(D)			TDE02A DTF		NULL		NULL		
F.CHECK(D)			COND:ck&cc <256 TDE02A SYN		NULL		NULL		
A(SYN)			COND:cc>0 TDE02A F.CHECK(C)	NULL		NULL		TDE07 F.CHECK(C)	NULL
F.RESTART(D)			COND:ck TDE03 RESYN		TDE03 RESYN		NULL		
A(RESYN)				TDE02A F.RESTART(C)		NULL			
RESYN			COND:ck TDE04 F.RESTART(I)	NULL		NULL		CONDck TDE04 F.RESTART(I)	CONDck TDE04 F.RESTAR(I)
F.RESTART(R)					TDE02A A(RESYN)		NULL		

TABLE 8 : Data transfer phase Server-Receiver

	OF02	TDE01B	TDE02B	TDE03	TDE04	TDE05	TDE06	TDE07	TDE08B
WRITE	COND.NOT ab TDE01B F.WRITE(I)								
F.WRITE(R)		+ : TDE02B - : OF02 A(WRITE) + : REC							
F.CANCEL(D)			TDE05 IDT	TDE05 IDT	TDE05 IDT		NULL	TDE05 IDT	TDE05 IDT
A(IDT)						OF02 F.CANCEL(C) SAB			
IDT			TDE06 F.CANCEL(I)	TDE06 F.CANCEL(I)	TDE06 F.CANCEL(I)	TDE06 F.CANCEL(I)		TDE06 F.CANCEL(I)	TDE06 F.CANCEL(I)
F.CANCEL(R)							OF02 A(IDT) SAB		
DTF.END			TDE07 F.DATA.END(I) ctt : SAB	NULL		NULL			
TRANS.END				NULL		NULL		TDE08B F.TRANSER. END(I)	
F.TRANSFER.END(R)							NULL		OF02 A(TRANS.END)
F.CHECK(R)			TDE02B A(SYN) REC		NULL		NULL	TDE07 (ASYN)	
DTF			TDE02B F.DATA(I) REC	NULL		NULL			
SYN			COND:ck&cc <256 F.CHECK(I) REC	NULL		NULL			
F.RESTART(D)			COND:ck TDE03 RESYN		NULL		NULL	COND: ck TDE03 RESYN	COND : ck TDE03 RESYN
A(RESYN)				TDE02B F.RESTART(C) REC		NULL			
RESYN			COND : ck TDE04 F.RESTART(I)	COND : ck TDE04 F.RESTART(I)		NULL			
F.RESTART(R)					TDE02B A(RESYN) REC		NULL		

TABLE 9 : Data transfer phase Caller-Receiver

	OF02	TDL01A	TDL02A	TDL03	TDL04	TDL05	TDL06	TDL07	TDL08A
F.READ(D)	COND : NOT ab TDL01A READ								
A(READ)		+ : TDL02A - : OF02 F.READ(C) + : REC							
F.CANCEL(D)			TDL05 IDT	TDL05 IDT	TDL05 IDT		TDL05 IDT	TDL05 IDT	TDL05 IDT
A(IDT)						OF02 F.CANCEL(C) SAB			
IDT			TDL06 F.CANCEL(I)	TDL06 F.CANCEL(I)	TDL06 F.CANCEL(I)	NULL		TDL06 F.CANCEL(I)	TDL06 F.CANCEL(I)
F.CANCEL(R)							OF02 A(IDT) SAB		
DTF.END			TDL07 F.DATA.END(I) cft : SAB	NULL		NULL			
F.TRANSFER(D)					NULL		NULL	TDL08A TRANS.END	
A(TRANS.END)				NULL		NULL			OF02 F.TRANSFER. END(C)
DTF			TDL02A F.DATA(I)	NULL		NULL			
F.CHECK(R)			COND : ck&cc <256 A(SYN)		NULL		NULL	NULL	
SYN			COND : cc>0 TDL02A F.CHECK(I)	NULL		NULL			
F.RESTART(D)			COND : ck TDL03 RESYN		COND : ck TDL03 RESYN		NULL	COND : ck TDL03 RESYN	COND : ck TDL03 RESYN
A(RESYN)				TDL02A F.RESTART(C)		NULL			
RESYN			COND : ck TDL04 F.RESTART(I)	NULL		NULL			
F.RESTART(R)					TDL02A A(RESYN)				

TABLE 10 : Data transfer phase Server-Sender

	OF02	TDL01B	TDL02B	TDL03	TDL04	TDL05	TDL06	TDL07	TDL08B
READ	COND : NOT ab TDL01B F.READ(I)								
F.READ(R)		+ : TDL02B - OF02 A(READ)							
F.CANCEL(D)			TDL05 IDT	TDL05 IDT	TDL05 IDT		NULL	TDL05 IDT	TDL05 IDT
A(IDT)						OF02 F.CANCEL(C) SAB			
IDT			TDL06 F.CANCEL(I)	TDL06 F.CANCEL(I)	TDL06 F.CANCEL(I)	TDL06 F.CANCEL(I)		TDL06 F.CANCEL(I)	TDL06 F.CANCEL(I)
F.CANCEL(R)							OF02 A(IDT) SAB		
F.DATA.END(D)			TDL07 DTF.END clt : SAB		NULL		NULL		
TRANS.END						NULL		TDL08B F.TRANS.END(I)	
F.TRANSFER.END(R)							NULL		OF02 A(TRANS.END)
F.DATA(D)			TDL02B DTF REC		NULL		NULL		
A(SYN)			COND : ck&cc <256 F.CHECK(C)	NULL		NULL		NULL	
F.CHECK(D)			TDL02B SYN REC		NULL		NULL		
F.RESTART(D)			COND : ck TDL03 RESYN		NULL		NULL		
A(RESYN)				TDL02B F.RESTART(C) REC		NULL			
RESYN			COND : ck TDL04 F.RESTART(I)	COND : ck TDL04 F.RESTART(I)		NULL		COND : ck TDL04 F.RESTART(I)	
F.RESTART(R)					TDL02B A(RESYN) REC		NULL		

ANNEXES

ANNEXE 1 : COMPRESSION

1. Negotiation of compression type
2. Definition of compression types
3. Interval between checkpoints when compression is used

1. NEGOTIATION OF COMPRESSION TYPE

The PeSIT protocol allows data compression to be requested in the FPDU.ORF. To do so the PI 21, which contains two bytes, is used :

- The first byte has two possible values :

  . 0 no compression,

  . 1 compression suggested.

- The second byte has the following meaning :

  . 1 horizontal compression,

  . 2 vertical compression,

  . 3 both horizontal and vertical compression.

The FPDU.ACK(ORF) contains the same PI21. The allowable values in this reply are :

- Byte 1 = 0 compression is refused,

- Byte 1 = 1 compression is accepted. In this case and provided that the second byte of the PI 21 in the FPDU.ORF was 3, then the second byte of the reply has the following meaning :

  . 1 horizontal compression only,

  . 2 vertical compression only,

  . 3 horizontal and vertical compression accepted.

2. DEFINITION OF COMPRESSION TYPES

- **Horizontal compression**

The horizontal compression is applied to identical consecutive characters. If this compression type was accepted at file opening time then each article in the file is broken up into strings. Each string is preceeded by a string header byte whose significance is as follows :

bit 7,	bit 6,	bit 5,	bit 4,	bit 3,	bit 2,	bit 1,	bit 0

- bit 7 :

  . 0 the string is not compressed,
  . 1 the string is compressed.

- bit 6

. 0 horizontal compression,
. 1 vertical compression (not used in this context).

Note :

Once horizontal compression has been negotiated the only permissible values of bits 7 and 6 are 00 or 10 (01 and 11 are prohibited).

- bits 0 to 5 :

Length of the string (1 to 63 bytes), excluding the header byte.

If the string is compressed the single character folowing the header byte is the component character.

Example :

The string 01 02 02 02 02 02 02 03 becomes 01 01 86 02 01 03.

- **Vertical compression**

This compression type is applied by comparing consecutive articles.

The first article is never compressed, but should nevertheless be preceeded by the string header byte defined below.

From the second article onwards, each article is compared with the previous article so as to identify identical character sequences.

Note :

In order to simplify the recovery mechanism the first article following each checkpoint is never compressed vertically.

Each article in the file is broken up into strings preceeded by a string header byte whose significance is as follows :

bit 7,	bit 6,	bit 5,	bit 4,	bit 3,	bit 2,	bit 1,	bit 0

- bit 7 :

. 0 the string is not compressed,
. 1 the string is compressed.

- bit 6

. 0 horizontal compression (not used in this context),
. 1 vertical compression.

Note :

Once vertical compression has been negotiated the only permissible values of bits 7 and 6 are 00 or 11 (01 and 10 are prohibited).

- bits 0 to 5 :

  Length of the string (1 to 63 bytes), excluding the header byte.

  Example :

  . 1st article 01 02 03 02 03 05 06 07
  . 2nd article 05 06 03 02 03 05 08 09

  These two articles become :

  . 1st article 08 01 02 03 02 03 05 06 07
  . 2nd article 02 05 06 C4 02 08 09

  The string 03 02 03 05 in the second article has been compressed.

  Note :

  If two consecutive articles do not have the same length then the hexadecimal character 0x40 is used to pad the shorter string before compression.

- **Combined horizontal and vertical compression**

  During the same transfer it is possible to use both horizontal and vertical compresion at the same time. In which case each article is broken up into sub-strings and each sub-string may be compressed either horizontally or vertically. The choice of which compression type is applied to a particular sub-string is left up to the protocol designer.

  If vertical compression has been chosen for a sub-string and the previous article to which the vertical compression algorithm refers was itself compressed horizontally, then the vertical compression algorithm will refer to the version of the article before it was compressed.

  Example :

  . 1st article 01 01 01 01 02 03
  . 2nd article 01 01 01 01 02 04

  These two articles become :

  . 1st article 84 01 02 02 03
  . 2nd article C5 01 04

3. INTERVAL BETWEEN CHECKPOINTS WHEN COMPRESSION IS USED

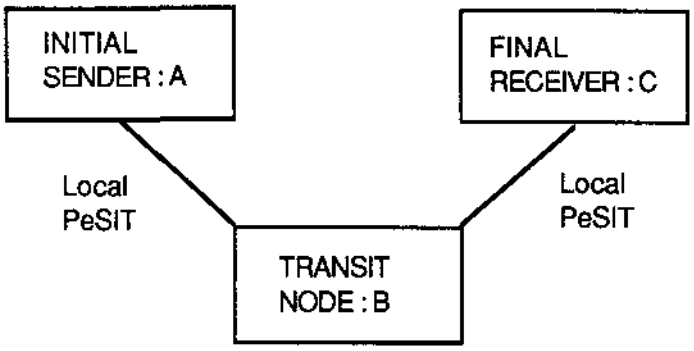
The interval between checkpoints, when compression is being used, is calculated using the compressed data, such as they were actually transmitted.

The string header bytes should be included in the character count for calculating the interval between checkpoints.

ANNEXE 2 : STORE and FORWARD OPERATION

1. INTRODUCTION

The term *store and forward* is used to describe a mechanism which allows files to be routed from one machine to another. In this mode a file transmitted by machine A and intended for machine C to which there is no direct connection, will transit by a third (or more) machine(s) B which having received the file from A will be able to retransmit it to C.



STORE and FORWARD

PeSIT is a local protocol - i.e. the parameters are only significant during a connection between two corresponding PeSIT units linked by a virtual circuit (PeSIT.F'), an ISO session connection (PeSIT.F), or a NETEX connection (PeSIT.F") - so to authorise a *store and forward* mode requires a definition of how the PeSIT addressing scheme may allow a file to be routed and to determine which of the parameters should be transfered with the file during the successive connections between the different intermediary nodes in a network.

As a case study we shall only define here the case of a write file transfer and we shall consider the following points :

* **addressing** : for any given transfer we must identify the partners who will execute this stage of the transfer (caller and server for a PeSIT connection) and the partners for whom the transfer is being executed (initial sender and final receiver).

* **transfer identifiers** : we must define non-ambiguous transfer identifiers within the *store and forward* domain.

* **transfer acknowledgement** : we must provide the initial sender with an indication that the file has effectively reached the final receiver (end to end acknowledgement).

2. INCIDENCE ON THE PROTOCOL

1. For the FPDU.CONNECT

The Connection phase sets up a relation between two partners linked by a direct connection : the parameters in the FPDU.CONNECT and the FPDU.ACONNECT are strictly local.

PI 3 and PI 4 : identify the partners between whom the connection is set up : a string of 1 to 24 bytes.

2. For the FPDU.CREATE

As of the selection phase it is possible to distinguish the local partners from the end-to-end partners. The PI 61 and PI 62 (Customer identifier and Bank identifier from ETEBAC5) are used to identify the initial sender and the final receiver. These parameters will allow the intermediate monitors to reroute the file to the next machine in the chain. The PI 3 and PI 4 (optional) in the FPDU.CREATE are used to provide are more precise address.

PI 3 and PI 4 : 24 bytes :

- bytes 1 to 8 : application name
- bytes 9 to 16 : user name
- bytes 17 to 24 : unreserved

PI 61 and PI 62 (optional) : define the end-to-end partners for whom this transfer is being carried out : initial sender and final receiver : 24 bytes.

The PI 13 (transfer identifier) helps to provide an unambiguous identification of the transfer : the same PI value will be propagated in each successive transfer for the same file. This applies only to the PI 13 in the FPDU.CREATE, the optional PI 13 in the FPDU.ACK(CREATE) that the server returns is "for information only" and will be specific to each transfer.

The file identification will be made up of the following parameters :

PI 11 (file type),
PI 12 (file name),
PI 13 (transfer identifier),
PI 61 (initial sender identification),
PI 62 (final receiver identification).

3. FPDU.MSG

The FPDU.MSG will be used to transport the end-to-end transfer acknowledgement. The contents of this FPDU.MSG will be :

PGI 9 :

- PI 3 : caller's identification (optional) : identical to the PI 4 which was in the FPDU.CREATE used for the file transfer being acknowldeged by this FPDU.MSG*,

- PI 4 : server's identification (optional) : identical to the PI 3 which was in the FPDU.CREATE used for the file transfer being acknowledged by this FPDU.MSG*,

- PI 11 : file type : PI 11 of the file transfer being acknowledged,

- PI 12 : file name : PI 12 of the file transfer being acknowledged,

- PI 13 : transfer identifier : PI 13 used end-to-end for the file transfer which is being acknowledged,

- PI 14 : requested attributes : unused,

- PI 16 : Data coding (optional) : may allow decoding of the contents of the PI 91, if present,

PGI 50 : Historical attributes :

- PI 51 : PI 51 of the file transfer being acknowledged,

- PI 61 : Initial sender identification : identical to the PI 62 of the file transfer which is being acknowledged,

- PI 62 : Final receiver identification : identical to the PI 61 of the file transfer which is being acknowledging,

- PI 91 : text (optional).

* if this PI makes up part of the file identification.

ANNEXE 3 : USE OF THE SECURITY MECHANISMS

1. Introduction
2. Implementation of the algorithms
3. Use of certificates

1. INTRODUCTION

The use of security mechanisms within file transfers is part of two profiles in PeSIT :

- the ETEBAC5 profile,
- the secure Non-SIT profile.

The ETEBAC5 profile resulted from the work of the ETEBAC5 study groups ("transport" and "security" groups) of the CFONB. The complete description of the use of the security in the ETEBAC5 standard may be found in the document "Computer Information Exchange between Banks and their Customers" published by the GSIT. It should be noted that use of the ETEBAC5 profile presumes the use of appropriate certificates whose production and distribution to the ETEBAC5 partners is controlled by the CFONB.

The secure Non-SIT profile came about by a wish to implement security functions outside the ETEBAC5 environment. This profile uses the same security parameters as were defined for ETEBAC5. However it has been designed to allow the different functions : reciprocal authentication, integrity, confidentiality to be implemented with only the DES (Data Encryption Standard) algorithm whereas the ETEBAC5 profile has mandatory use of the RSA (Rivest Shamir Adleman) algorithm as well.

The readers attention is drawn to the fact that the use of cryptographic devices and algorithms on data to be transmitted across public networks must comply with the local legislation.

2. IMPLEMENTATION OF THE ALGORITHMS

2.1 DES Encryption

The implementation of the DES encryption is described in the ISO DP 10126 document. An initial chain length of 8 bytes is used. Padding is used to obtain encryption fields of a length which is a multiple of 8 bytes.

2.2 MAC computation

The implementation of the DES MAC computation is described in the ISO DIS 8731 document. The DES is used in CBC mode, with an initial nul chaining value, and padding using 0 binary.

2.3 Order of execution

The MAC computation is carried out on the plain text data.

The encryption is carried out on plain text data (the MAC is not part of the data).

The encryption and the MAC computation may be executed article by article, or on the whole file. In the article-by-article mode paddding is added at the end of each article, in the whole-file mode padding is only added at the end of the file. The mode chosen is indicated in the "operating mode" bytes of the "MAC computation type" and "Encryption type" parameters.
The encryption and the MAC computation are only applied to the file data which implies

that in the case of an FPDU multi-article the length bytes of the articles are not included in either the encryption or the MAC computation.

If the MAC computation is carried out article by article, the MAC of the article n is used as the initial value to calculate the MAC of the article n+1. If partial MACs are transmitted, they are sent at the same time as the checkpoints and so may apply to one or more articles. The final MAC - or total file MAC - is necessarily the MAC of the last article in the file.

The compression is done after encryption and MAC computation. Since the data stream after encryption is nearly a random string of bytes, compression after encryption is probably useless. We may conclude that encryption and compression are effectively mutually exclusive.

2.4 Security and recovery

If a transfer is recovered, the same encryption and MAC computation elements must be used for the following tries. These elements need not be sent again when a recovery takes place. If they are retransmitted then they should be identical to those elements previously transmitted. In particular the initialisation vectors should be the initial one and, consequently, are not significant during a recovery. A recovery always takes place from a checkpoint, thus at an article boundary, so both the sender and the receiver should re-initialise the MAC computation or the encryption algorithm using the appropriate values corresponding with the checkpoint.

2.5. RSA Algorithm

The RSA keys and modulus, as well as all RSA encrypted fields, are transported as numerical values (N), the most significant byte first and the least significant last.

In the certificates, the couple modulus and public key are presented in this order : modulus (64 bytes, with binary zero padding in the most significant bits if necessary) followed by the public key (2 bytes).

There is no redundancy before RSA encryption. All the data to be encrypted by RSA are right justified and binary zero padded.

Example 1 : format of the PI 76 "encryption elements"

Before RSA encryption this field is 16 bytes long :

Bytes 1 to 8 : Encryption key (MSB byte 1, LSB byte 8)
Bytes 9 to 16 : initialisation vector (MSB byte 9, LSB byte 16)

For RSA encryption this field is considered to be a numerical value of 64 bytes whose 48 most significant bytes are zero. The most significant byte that may be non-zero is the most significant byte of the encryption key.

Example 2 : format of the PI 79 "Digital signature"

Before RSA encryption this field is 16 bytes long :

Bytes 1 to 8 : FID MAC (MSB byte 1, LSB byte 8)
Bytes 9 to 16 : FID MAC and data (MSB byte 9, LSB byte 16)

For RSA encryption this field is considered to be a numerical value of 64 bytes whose 48 most significant bytes are zero. The most significant byte that may be non-zero is the most significant byte of the FID MAC.

Example 3 : format of the PI 81 "Acknowledgement of Digital signature"

Before RSA encryption this field is 30 bytes long :

Bytes 1 to 8 : FID MAC (MSB byte 1, LSB byte 8)
Bytes 9 to 16 : FID MAC and data (MSB byte 9, LSB byte 16)
Bytes 17 to 28 : date and time (YYMMDDhhmmss)
Bytes 29 to 30 : ACK/NAK

For RSA encryption this field is considered to be a numerical value of 64 bytes whose 34 most significant bytes are zero. The most significant byte that may be non-zero is the most significant byte of the FID MAC.

2.6. Transformation*

A transformation is sometimes applied to the data in operating modes where only DES is used, thus not in the ETEBAC5 standard. For example it can be applied to "random numbers for authentication" which contribute to the reciprocal authentication, or for the encryption initialisation vectors to allow immediate verification that the two correspondants are using the same key encrypting keys.

This transformation if denoted *. The convention is :

B* is equivalent to "B XOR FFFFFFFF00000000" where :

B is a string of 8 bytes
XOR is the exclusive or operation
FFFFFFFF00000000 is the an 8 byte number whose 4 most significant bytes have all their bits = 1 and the four least significant bytes have all their bits = 0.

3. USE OF CERTIFICATES

The exchange of certificates allows a correspondant's public key, as certified by the Authority, to be sent to his partner.

It is necessary to distinguish the key-pairs (public key, secret key) thus the certificates containing the certified public keys, used for different security functions.

The different uses of the RSA key-pairs are :

1. exchange of random numbers (RN) for reciprocal authentication : use of the sender's secret key.

2. exchange of the RSA encrypted MAC computation elements : use of the receiver's public key.

3. exchange of the RSA encrypted encryption elements : use of the receiver's public key.

4. exchange of the digital signature : use of the sender's secret key.

5. exchange of the second digital signature : use of the sender's secret key.

6. exchange of the acknowledgement of the digital signature : use of the sender's secret key.

To transmit data encrypted under the receiver's public key requires his certificate to have

been received previously.

To transmit data encrypted under the sender's secret key requires the sender's certificate to have been sent previously.

The security constraints may imply the use of different key-pairs, and therefore different certificates, for the certification functions and for the key transport and signature functions.

Write file case (Caller-sender to server-receiver)

Let's consider a caller and a server irrespective of their statute within the ETEBAC5 standard (Customer, Customer operator, Bank, Bank operator).

The caller holds a key-pair Pd1, Sd1 which corresponds with the certificate **A<<Pd1>>**, used to encrypt the Random number n.

The caller holds a key-pair Pd2, Sd2 which corresponds with the certificate **A<<Pd2>>**, used for the first (or only) digital signature.

The caller holds a key-pair Pd3, Sd3 which corresponds with the certificate **A<<Pd3>>**, used for the second digital signature, if such exists.

The server holds a key-pair Ps1, Ss1 which corresponds with the certificate **A<<Ps1>>**, used to encrypt the Random number n.

The server holds a key-pair Ps2, Ss2 which corresponds with the certificate **A<<Ps2>>**, used by the caller to transport the different encryption and MAC computation elements, and by the server to transport the acknowledgement of the digital signature.

The following diagram summarises these exchanges while only showing the parameters related to the RSA keys. The case taken is with reciprocal authentication, reciprocal non-repudiation and confidentiality.

FILE WRITING

CALLER **SERVER**

FPDU.CREATE : PI 72 = RN1
PI 80 = A<<Pd1>>

FPDU.ACK(CREATE) : PI 72 = (ALEA1)Ss1
RN2
PI 80 = A<<Ps1>>
PI 83 = A<<Ps2>>

FPDU.ORF : PI 72 = (RN2)Sd1
PI 74 = (K2)Ps2
PI 76 = (K1)Ps2
PI 80 = A<<Pd2>>
PI 83 = A<<Pd3>> *

FILE TRANSFER

FPDU.DTF.END : PI 79 = (MACs)Sd2
PI 82 = (MACs)Sd3 *

FPDU.ACK(TRANS.END) : PI 81 = (MACs, ACK/NAK)Ss2

Read file case (Caller-receiver to server-sender)

Let's consider a caller and a server irrespective of their statute within the ETEBAC5 standard (Customer, Customer operator, Bank, Bank operator).

The caller holds a key-pair Pd1, Sd1 which corresponds with the certificate **A<<Pd1>>**, used to encrypt the Random number n.

The caller holds a key-pair Pd2, Sd2 which corresponds with the certificate **A<<Pd2>>**, used by the server to transport the different encryption and MAC computation elements, and by the caller to transport the acknowledgement of the digital signature.

The server holds a key-pair Ps1, Ss1 which corresponds with the certificate **A<<Ps1>>**, used to encrypt the Random number n.

The server holds a key-pair Ps2, Ss2 which corresponds with the certificate **A<<Ps2>>**, used for the single digital signature (there is only one digital signature for a read transfer).

The following diagram summarises these exchanges while only showing the parameters related to the RSA keys. The case taken is with reciprocal authentication, reciprocal non-repudiation and confidentiality.

FILE READING

CALLER **SERVER**

FPDU.SELECT : PI 72 = RN1
PI 80 = A<<Pd1>>

FPDU.ACK(SELECT) : PI 72 = (RN1)Ss1
RN2
PI 80 = A<<Ps1>>
PI 83 = A<<Ps2>>

FPDU.ORF : PI 72 = (RN2)Sd1
PI 80 = A<<Pd2>>

FPDU.ACK(ORF) : PI 74 = (K2)Pd2
(K1)Pd2

FILE TRANSFER

FPDU.DTF.END : PI 79 = (MACs)Ss2

FPDU.TRANS.END : PI 81 = (MACs, ACK/NAK)Sd2

ANNEXE 4 : ERROR DIAGNOSTICS

Diagnostic code			
Error type	**Cause Code**	**Cause**	**Service element concerned**
0	000	"Success" : no error	All
3	300	Local "communication system" saturation	F.CONNECT
3	301	Unknown called party identification	
3	302	Called party not connected to a SSAP	
3	303	Distant "communication system" satured (too many connections)	
3	304	Unauthorized caller identification (security)	
3	305	Negociation failure : - SELECT	
	306	- RESYN	
	307	- SYNC	
3	308	Version number not supported	
3	309	Too many connections already open for a processing center	
3	321	Call the back-up number	
3	322	Call back later	
3	399	Other	
3	312	Service termination requested by the user	F.RELEASE
3	313	Connection broken after inactivity time-out Td	
3	314	Unused connection broken to accept a new connection	
3	316	Connection broken by administrative request	
3	399	Other	
3	304	Unauthorized caller identification (security)	F.ABORT
3	309	Too many connections already open for a processing center	
3	310	Network incident	
3	311	Distant PeSIT protocol error	
3	312	Service termination requested by the user	
3	313	Connection broken after inactivity time-out Td	
3	314	Unused connection broken to accept a new connection	
3	315	Negociation failure	
3	316	Connection broken by administrative request	
3	317	Time-out expired	
3	318	Mandatory PI missing or illegal PI contents	
3	319	Byte count or article count incorrect	
3	320	Excessive number of restarts during the transfer	
3	399	Other	

Diagnostic code			
Error type	**Cause Code**	**Cause**	**Service element concerned**
2	200	Insufficient file characteristics	F.CREATE
2	201	System resources temporarily insufficient	F.SELECT
2	202	User resources temporarily insufficient	
2	203	Low priority tranfer	
2	204	File already exists	
2	205	File does not exists	
2	206	File reception would cause disk quota overflow	
2	207	File busy	
2	208	File too old (prior to D-2 in SIT terms)	
2	209	This message type not accepted by the installation refered to	
2	226	Transfer refused	
2	299	Other	
3	304	Unauthorized caller identification (security)	
3	321	Call the back-up number	
3	322	Call back later	
3	399	Other	
2	210	Presentation context negociation failure	F.OPEN
2	211	File cannot be opened	
2	299	Other	
2	212	Normal file closure impossible	F.CLOSE
2	299	Other	
2	213	Unresolvable I/O error	F.READ
2	214	Restart negociation failure	
2	299	Other	
2	213	Unresolvable I/O error	F.WRITE
2	299	Other	
2	213	Unresolvable I/O error	F.DATA.END
2	214	Restart negociation failure	F.CANCEL
2	215	Internal system error	
2	216	Voluntary abrupt termination	
2	217	Too many unacknowledged checkpoints	
2	218	Restart impossible	
2	219	File space overflow	
2	220	Article length exceeds expected length	
2	221	End of transmission time-out expired	
2	222	Excess data between checkpoints	
2	299	Other	

Diagnostic code			
Error type	**Cause Code**	**Cause**	**Service element concerned**
2	223	Abnormal end of transfer	F.TRANSFER.END
2	224	The size of the file transmitted exceeds the size given in the F.CREATE	F.DESELECT
2	225	Congestion in the station application software : the file has been correctly received but SCRS cannot pass it on to the station application software	
2	299	Other	
1	100	Transmission error	F.RESTART
2	299	Other	

ANNEXE 5 : SUMMARY OF THE PROTOCOL UNITS AND THEIR PARAMETERS

FPDU LIST

Code	FPDU LIST
00	FPDU.DTF
01	FPDU.READ
02	FPDU.WRITE
03	FPDU.SYN
04	FPDU.DTF.END
05	FPDU.RESYN
06	FPDU.IDT
08	FPDU.TRANS.END
11	FPDU.CREATE
12	FPDU.SELECT
13	FPDU.DESELECT
14	FPDU.ORF
15	FPDU.CRF
16	FPDU.MSG
17	FPDU.MSGDM
18	FPDU.MSGMM
19	FPDU.MSGFM
20	FPDU.CONNECT
21	FPDU.ACONNECT
22	FPDU.RCONNECT
23	FPDU.RELEASE
24	FPDU.RELCONF
25	FPDU.ABORT
30	FPDU.ACK(CREATE)
31	FPDU.ACK(SELECT)
32	FPDU.ACK(DESELECT)
33	FPDU.ACK(ORF)
34	FPDU.ACK(CRF)
35	FPDU.ACK(READ)
36	FPDU.ACK(WRITE)
37	FPDU.ACK(TRANS.END)
38	FPDU.ACK(SYN)
39	FPDU.ACK(RESYN)
3A	FPDU.ACK(IDT)
3B	FPDU.ACK(MSG)
40	FPDU.DTFMA
41	FPDU.DTFDA
42	FPDU.DTFFA

PARAMETER LIST

PI code	Parameter description
1	CRC usage
2	Diagnostics
3	Caller identification
4	Server identification
5	Access control
6	Version number
7	Option : checkpointing
9	File identifier
11	File type
12	File name
13	Transfer identifier
14	Requested attributes
15	Recovered transfer
16	Data coding
17	Transfer priority
18	Recovery point
19	End of transfer code
20	Checkpoint number
21	Compression
22	Access type
23	Restarting
25	Maximum size of a data element
26	Protocol monitoring time-out
27	Number of data bytes
28	Number of articles
29	Diagnostic complements
30 (PGI)	Logical attributes
31	Article format
32	Article length
33	File attributes
34	Use of the signature
36	SIT MAC
37	File Label
38	Key length
39	Key offset
40 (PGI)	Physical attributes
41	Storage reservation unit
42	Maximum reserved space
50 (PGI)	Historical attributes
51	Date and time of creation
52	Date and time of last access
61	Customer identifier
62	Bank identifier
63	File access control
64	Server date and time
71	Authentication type
72	Authentication elements
73	MAC computation type
74	MAC computation elements
75	Encryption type
76	Encryption elements
77	Digital signature type
78	MAC
79	Digital signature
80	Certificate
81	Acknowledgement of Digital signature
82	Second digital signature
83	Second certificate
91	Datagram
99	Free text