OSIRIDE File Transfer Services and Protocol

Paola Bucciarelli
Anna Canepa
Fausto Caneschi
Enrico Zucchelli

Report CNUCE C84-7

Istituto CNUCE
PISA
February 1984

LIST OF FIGURES

First Part

1.0 SERVICE OVERVIEW

1.1 ROLES OF PARTNERS

In a Virtual File Service **VFS** connection between two VFS users, the dialog is always asymmetrical, i.e. the two VFS users play different and complementary roles. The initiator of the VFS connection (called the Primary) is the one that defines the work to be performed on the Virtual Filestore through the connection: it acts on a direct relationship with the end user on behalf of whom it works. The other VFS user (called the Secondary) is the executor of the work proposed by the Primary and reports to it; it resides on the same system as the Real Filestore and has no relation with an end user, except through the Primary.

The actual direction of the data transfer is not related to the concepts of Primary and Secondary; moreover, this direction can vary during different phases of the same File Transfer.
The concept of phase will be explained later on: at this point it is enough to say that a distinction is made between the Sender and the Receiver, and that both Primary and Secondary may be Sender and Receiver, in different phases.

1.2 DYNAMIC STRUCTURING OF A VFS CONNECTION

The VFS allows operation on only one file at a time on a given VFS connection. Multiple files can be handled concurrently through several parallel VFS connection. Furthermore, within one VFS connection, operation on the current file are executed one after the other in the order of submittion. This is necessary to keep total control on the sequence of events.

The work performed on a VFS connection can be dinamically structured as a set of nested enclosures, which are opened in the hierarchical order and closed in the reverse order. If the VFS connection breaks or is abnormally terminated by one VFS user, all the currently opened enclosures are considered as being implicitly closed.

The enclosures are the following, in the hierarchical order, starting from the outmost one:

- Connection enclosure: the VFS connection exists (from establishement to termination of the VFS connection)

- File enclosure: a current file exists (from successful file selection to file release). A connection enclosure contains any number of file enclosure (including none).

- Open enclosure: the current file is ready for data access (from successful file opening to file closing). A file enclosure contains any number of open enclosure (including none).

- Transfer enclosure: file data is being transferred (from transfer begin to normal or abnormal transfer end). For file transfer purpose, only one transfer enclosure for open enclosure is necessary.

1.3 CONNECTION FACILITY

The connection facility service provides for establishment and release of the VFS connection.

At connection establishment, there is a negotiation of the particular class of service to be used and of any user special conventions that may be agreed[1]. Renegotiation is not provided in CSIRIDE Version 1.

Connection termination is normally requested by the Primary, when all work is completed. However, in emergency cases (e.g. system shutdown), the connection can be abnormally terminated by either VFS user at any point of time. The connection can also be accidentally lost: this is reported to both users by the VFS.

1.4 FILE MANAGEMENT

The file management service provides to the Primary all file services whith the exception of file data transfer. This includes:

- Selection of a current file, by designating an existing file .

- Release of the current file when all work on it has been completed.

---

1 Particular conventions may serve two main purposes:

- addition of specific services, file model or data model

- simplification, by assuming implicit prenegotiation of a number of parameters (useful for small systems offering a single choice for each capability).

- Creation of a new file, with specified attributes. This file then becomes the current file. An option specifies what to do if a file with the same name already exists.

- Deletion (and release) of current file.

- Retrieval of a selected attributes of the current file.

- Opening of the current file for data access, with a specified lock. Only sequential data access is provided in OSIRIDE version 1: read the file or write to the file (either after its current content or overwriting it).

- Closing of the current file, with release of the lock.

- Starting of the file data transfer: this enters the next part of the service.

1.5 FILE DATA TRANFER

The file data transfer service provides for the transfer of file data. The data flow is **one way from Sender to Receiver.** Facilities are provided for:

- orderly termination of the data transfer by the Sender (with acknowledgement by the Receiver).

- Abnormal termination of the data transfer by either user.

- Checkpointing and checkpoint acknowledgement.

- Immediate restart of the data transfer to a negotiated previous position. This form of restart can be requested by either user.

1.6 RECOVERY

The purpose of the restart facility is to avoid complete repetition of a transfer which was interrupted before completion. The prime objective is to minimize the amount of data retransmission, while eliminating any loss or duplication of data in the received file.

Restart may be either immediate or deferred.

An immediate restart occurs within a transfer enclosure, on request of either VFS user. In such case, the negotiation of the restart position and retransmission occurs immediately, without exiting from the transfer enclosure.

A deferred restart occurs after the transfer enclosure has been terminated and the file closed and potentially released. The termination can be involuntary (failure of network or either node) or voluntary (shutdown or desire to execute higher priotity work). In any case, all enclosures that have been closed will be reopened before the transfer can be resumed. All file attributes and processing attributes are set to the same values as initially. Restart position is negotiated while entering the open enclosure. A deferred restart can occur within the same or a different VFS connection. A deferred restart is initiated only by the entity which was the Primary of the interrupted activity.

The deferred restart is the only type of restart implemented in CSIRIDE.

1.7 OPEN IDENTIFICATION

Since a restart can be deferred, there is a need to relate several successive transfer enclosures as belonging to the same file transfer activity. As a result, each new file transfer request will be identified by a unique identification, which will be recalled every time a deferred restart is attempted. This identification is supplied by the Primary when entering the open enclosure. It is unique only within the Primary and has to be prefixed by the address of the primary to be globally unique. The open identification is deleted (and can therefore be reused for another activity) once the associated file transfer request is either successfully completed or abandoned (not recoverable failure).

1.8 RESTART POSITION NEGOTIATION

The restart position is designated in terms of record position in the data flow, starting from the beginning of file data transfer. This type of designation is not applicable to unstructured files: for unstructured files, it is expressed in terms of character or octet position in the data flow (according to file-data-type value), starting from the beginning of file data transfer.

The negotiation rules are the following:

- one partner proposes a restart position
- the other can agree or specify an earlier position.

The sender can abstain from specifying a restart position, since restart is normally driven by the receiver. However, the sender may specify a restart position to force retransmission of data.

1.9 CHECKPOINTS

As described above, checkpoints (i.e. the marking of particular points in the data flow) are not required for resynchronization of the data flow. However, they are useful whenever context information (i.e. information necessary to properly restart data transfer after a recoverable failure) is saved: checkpoint acknowledgment by the receiver allows the sender to purge its context information (thus avoiding an uncontrolled growth). Restart at a position before an acknowledged checkpoint is allowed, but might involve full retransmission of the data, depending on the implementation.

Checkpointing is not implemented in OSIRIDE Version 1. This section has been introduced for completeness with respect to the standard ECMA-85.

1.10 VIRTUAL FILE SERVICES

Fig. 1 pag. 7 lists all the services of the VFS. For each service, its type, the user who can initiate it , and its purpose are specified.

The meaning of types is:

1 Service structure with a request primitive and indication primitive

2 Service structure with request, indication, response and confirmation primitives.

3 Service structure with two indication primitives.

The initiators are:

PR Primary

SC Secondary

SN Sender

RC Receiver.

Service	type	init.	Description
		CONNECTION FACILITY	
F-CONNECT	2	PR	Establish VFS connection
F-RELEASE	2	PR	Clean release of VFS connection
F-DISCONNECT	1	PR,SC	Unclean release of VFS connection
F-ABORT	3	-	Loss of presentation connection
		FILE MANAGEMENT	
F-SELECT-FILE	2	PR	Establish current file
F-RELEASE-FILE	2	PR	Release current file
F-CREATE-FILE	2	PR	Create new file
F-DELETE-FILE	2	PR	Delete current file
F-READ-ATTRIBUTES	2	PR	Read attributes of current file
F-OPEN-FILE	2	PR	Open current file for data access
F-CLOSE-FILE	2	PR	Close current file
F-BEGIN-DATA	2	PR	Start transfer of file data
		FILE DATA TRANSFER	
F-DATA	1	SN	Transfer file data
F-END-DATA	2	SN	End transfer of file data
F-ABORT-DATA	2	SN,RC	Abort transfer of file data

Fig. 1. Virtual file services

2.0 PROTOCOL OVERVIEW

2.1 ROLES OF VFS ENTITIES

The asymmetry of the VFS is reflected in the protocol: the two VFS entities play different and complementary roles corresponding to the roles played by their respective users; Primary and Secondary outside of a transfer enclosure, Sender and Receiver within a transfer enclosure (see "1.0 Service Overwiew" pag. 2).

2.2 DESCRIPTIVE MODEL

The Virtual File Protocol is modelled as an abstract machine with protocol structures between the two VFS entities. A **protocol structure** is an elementary dialogue for the purpose of an indivisible operation. As such, it is totally successful or totally unsuccessful, never partly successful. It is composed of a **request**, issued by one VFS entity, and for most (but not all) types of structure, of a **response**, issued by the other VFS entity. Each response or request is a single protocol **message**.

There are two types of protocol structures:

- Type 1 structure: request without response.
- Type 2 structure: request with response

A protocol message contains protocol control information (i.e. one or more parameters) and may in some cases also contain file data.

Dynamic execution of the Virtual File protocol results in an ordered sequence of protocol structures. To describe the protocol, it is sufficient to separately describe each of its structures (or messages), plus any precedence relationship between structures (state transmission).

Structure	type	init.	Description
Select Protocol	2	PR	Initiate VFS connection
Release protocol	2	PR	Release VFS connection
Disconnect Protocol	1	PR,SC	Abnormally terminate a VFS connecti
End group[2]	2	PR	Delimit a structure group
Select File	2	PR	Establish current file
Release File	2	PR	Release current file
Create File	2	PR	Create new file
Delete File	2	PR	Delete current file
Read Attributes	2	PR	Read attributes of current file
Open File	2	PR	Open current file for data access
Close File	2	PR	Close current file
Begin Data	2	PR	Begin transfer of file data
Data	1	SN	Transfer file data
End Transfer	2	SN	End transfer of file data
Abort Transfer	2	SN,RC	Abort transfer of file data
Restart Transfer	2	SN,RC	Resynchr. transfer in progress
Checkpoint	2	SN	Requets acknowledgement

Fig. 2. Protocol structures

2.3 PROTOCOL DESCRIPTION

2.3.1 Notation

This clause provides a narrative description of protocol.
The formal description is provided in "5.3 Formal Description" pag. 41.

Each message is defined by the following items:

- Sender of message
- Function
- List of parameters

2 Not supported in OSIRIDE phase 1

- Resulting states transition(s)
- Relationship with service primitives (sending/receiving)

Note: The "**grouping options**" and "**special conventions**" parameters are not described because they are not implemented in OSIRIDE Version 1.

2.3.2 Select_Protocol_request_(SP)

Sent by : Primary

Function	Initiate a VFS connection (request opening of a VFS connection enclosure)
Content	filestore-name authentication protocol definition = {Protocol-identifier {Protocol-version {class-of-filestore {class-of-service
Parameter description	see "2.3.28 Parameters description" pag. 20
Transition	Dormant --> SP pending
Sending	on F-CONNECT request primitive.
Receiving	generate a F-CONNECT indication primitive. The espected outcome is a F-CONNECT response primitive.

2.3.3 Select_Protocol_response_(SPR)

Sent by : Secondary

Function	response to SP
Content	diagnostic protocol definition = {Protocol-version {class-of-filestore {class-of-service
Parameter description	see "2.3.28 Parameters description" pag. 20
Transition	SP pending --> NO file (successful)

	SP pending --> Dormant (rejected)
Sending	on F-CONNECT response primitive.
Receiving	generate a F-CONNECT confirmation primitive.

2.3.4 Release_Protocol_request_(RP)

Sent by : Primary

Function	request normal termination of VFS connection enclosure
Content	none
Transition	SP No-file --> RP-pending SP pending --> Dormant (rejected)
Sending	on F-RELEASE requets primitive.
Receiving	generate a F-RELEASE indication primitive. The expected outcome is a F-RELEASE response primitive.

2.3.5 Release_Protocol_response_(RPR)

Sent by : Secondary

Function response to RP

Content none

Transition RP pending --> Dormant

Sending on F-RELEASE response primitive.

Receiving generate a F-RELEASE confirmation primitive.

2.3.6 Disconnect_Protocol_request_(DP)

Sent by : Primary/Secondary

Function	Request abnormal termination of VFS connection enclosure
Content	diagnostic

Parameter description	see "2.3.28 Parameters description" pag. 20
Transition	Any state --> Dormant
Sending	on F-DISCONNECT request primitive.
Receiving	generate a F-DISCONNECT indication primitive.

2.3.7 Select_File_request_(SL)

Sent by : Primary

Function	establish an existing file as current (request opening of a file enclosure)
Content	filestore-name file-password
Parameter description	see "2.3.28 Parameters description" pag. 20
Transition	No file --> SL pending
Sending	on F-SELECT-FILE request primitive.
Receiving	generate a F-SELECT-FILE indication primitive. The espected outcome is a F-SELECT-FILE response primitive.

2.3.8 Select_File_response_(SLR)

Sent by : Secondary

Function	response to SL.
Content	diagnostic
Parameter description	see "2.3.28 Parameters description" pag. 20
Transition	SL pending --> file-selected (successful) SL pending --> no-file (rejected)
Sending	on F-SELECT-FILE response primitive.
Receiving	generate a F-SELECT-FILE confirmation primitive.

2.3.9 Release_File_request_(RL)

Sent by : Primary

Function	release the current file. (requests closing of a file enclusure)
Content	none
Transition	File-selected --> RL-pending
Sending	on F-RELEASE-FILE request primitive.
Receiving	generate a F-RELEASE-FILE indication primitive. The espected outcome is a F-RELEASE-FILE response primitive.

2.3.10 Release_File_response_(SLR)

Sent by : Secondary

Function	response to RL.
Content	diagnostic
Parameter description	see "2.3.28 Parameters description" pag. 20
Transition	RL-pending --> no-file
Sending	on F-RELEASE-FILE response primitive.
Receiving	generate a F-RELEASE-FILE confirmation primitive. The espected outcome is a F-RELEASE-FILE response primitive.

2.3.11 Create_file_request_(CR)

Sent by : Primary

Function	Create and establish a new file as current (request opening of a file enclusure)
Content	file-name file-password clash-options reversible-mapping

	file-attributes = {global attributes {record-attributes {key-attributes {field-attributes
Parameter description	see "2.3.28 Parameters description" pag. 20
Transition	No-file --> CR-pending
Sending	on F-CREATE-FILE request primitive.
Receiving	generate a F-CREARTE-FILE indication primitive. The espected outcome is a F-CREATE-FILE response primitive.

2.3.12 Create_File_response_(CRR)

Sent by : Secondary

Function	response to CR
Content	diagnostic
Parameter description	see "2.3.28 Parameters description" pag. 20.
Transition	CR-pending --> File-selected (successful) CR-pending --> No-file (rejected)
Sending	on F-CREATE-FILE response primitive.
Receiving	generate a F-CREATE-FILE confirmation primitive.

2.3.13 Delete_File_request_(DL)

Sent by : Primary

Function	delete and release the current file. (this request closes the file enclosure)
Content	none
Transition	File-selected --> DL-pending
Sending	on F-DELETE-FILE request primitive.
Receiving	generate a F-DELETE-FILE indication primitive.

The espected outcome is a F-DELETE-FILE response primitive.

2.3.14 Delete_File_response (DLR)

Sent by : Secondary

Function	response to DL.
Content	diagnostic
Parameter description	see "2.3.28 Parameters description" pag. 20.
Transition	DL-pending --> no-file SL pending --> no-file (rejected)
Sending	on F-DELETE-FILE response primitive.
Receiving	generate a F-DELETE-FILE confirmation primitive.

2.3.15 Read_attributes_request (RA)

Sent by : Primary

Function	retrieve specified attributes of the current file.
Content	none
Transition	File-selected --> RA-pending
Sending	on F-READ-ATTRIBUTES request primitive.
Receiving	generate a F-READ-ATTRIBUTES indication primitive. The espected outcome is a F-READ-ATTRIBUTES response primitive.

2.3.16 Read_Attributes_response (RAR)

Sent by : Secondary

Function	response to RA.
Content	diagnostic

	file-attibutes {history-attributes {global-attributes {record-attributes {key-attributes {files-attributes
Parameter description	see "2.3.28 Parameters description" pag. 20.
Transition	RA-pending --> File-selected
Sending	on F-READ-ATTRIBUTES response primitive.
Receiving	generate a F-READ-ATTRIBUTES confirmation primitive.

2.3.17 Open File request (OP)

Sent by : Primary

Function	Initiate processing of contens of current file. (request opening an open enclusure)
Content	access-mode processing-mode lock failure-option open-identification restart position
Parameter description	see "2.3.28 Parameters description" pag. 20.
Transition	File-selected --> OP-pending
Sending	on F-OPEN-FILE request primitive.
Receiving	generate a F-OPEN-FILE indication primitive. The espected outcome is a F-OPEN-FILE response primitive.

2.3.18 Open File response (OPR)

Sent by : Secondary

Function	response to CP
Content	diagnostic restart-position

Parameter description	see "2.3.28 Parameters description" pag. 20
Transition	OP-pending --> File-open (successful) OP-pending --> File-selected (rejected)
Sending	on F-OPEN-FILE response primitive.
Receiving	generate a F-OPEN-FILE confirmation primitive.

2.3.19 Close_File_request_(CL)

Sent by : Primary

Function	Terminate processing of contens of current file. (request closing an open enclusure)
Content	none
Transition	File-open --> CL-pending File-aborted --> CL-pending
Sending	on F-CLOSE-FILE request primitive.
Receiving	generate a F-CLOSE-FILE indication primitive. The espected outcome is a F-CLOSE-FILE response primitive.

2.3.20 Close_file_response_(CLR)

Sent by : Secondary

Function	response to CL
Content	diagnostic
Parameter description	see "2.3.28 Parameters description" pag. 20.
Transition	CL-pending --> File-selected
Sending	on F-CLOSE-FILE response primitive.
Receiving	generate a F-CLOSE-FILE confirmation primitive.

2.3.21 Begin_Transfer_request_(BT)

Sent by : Primary

Function	cause transition to file data transfer level. (request opening a transfer enclusure)
Content	none
Transition	File-open --> BT-pending
Sending	on F-BEGIN-DATA request primitive.
Receiving	generate a F-BEGIN-DATA indication primitive. The espected outcome is a F-BEGIN-DATA response primitive.

2.3.22 Begin_Transfer_response_(BTR)

Sent by : Secondary

Function	response to BT
Content	none
Transition	BT-pending --> Data BT-pending --> File open (rejected)
Sending	on F-BEGIN-DATA response primitive.
Receiving	generate a F-BEGIN-DATA confirmation primitive.

2.3.23 Data_request_(DATA)

Sent by : Sender

Function	transfer a file data and / or delimiter.
Content	file-data
Parameter description	see "2.3.28 Parameters description" pag. 20.
Transition	Data --> Data
Sending	on F-DATA request primitive.
Receiving	generate a F-DATA indication primitive.

2.3.24 End_Transfer_request_(ET)

Sent by : Sender

Function	specify termination of the transfer without loss of data (request closing the transfer enclosure)
Content	none
Transition	Data --> ET-pending
Sending	on F-END-DATA request primitive.
Receiving	generate a F-END-DATA indication primitive. The espected outcome is a F-END-DATA response primitive cr a F-RESTART or F-A-BORT-DATA request primitive.

2.3.25 End_Transfer_response_(ETR)

Sent by : Receiver

Function	response to ET
Content	none
Transition	ET-pending --> File-open
Sending	on F-END-DATA response primitive.
Receiving	generate a F-END-DATA confirmation primitive.

2.3.26 Abort_Transfer_request_(AT)

Sent by : Sender/Receiver

Function	specify abnormal termination of the transfer, with possible destruction of data in transit. (this request closes the enclosure).
Content	Diagnostic
Parameter description	see "2.3.28 Parameters description" pag. 20.
Transition	any other data transfer state --> AT-pending

Sending	on F-ABORT-DATA request primitive.
Receiving	generate a F-ABORT-DATA indication primitive. The espected outcome is a F-ABORT-DATA response primitive.

2.3.27 Abort_Transfer_response_(ATR)

Sent by : Sender/Receiver

Function	response to AT.
Content	diagnostic
Parameter description	see "2.3.28 Parameters description."
Transition	AT-pending --> File-aborted
Sending	on F-ABORT-DATA response primitive.
Receiving	generate a F-ABORT-DATA confirmation primitive.

2.3.28 Parameters_description

The parameters used in the protocol messages are described in this section.

Filestore-name global title necessary to establish a connection with the application entity supporting the filestore. There is no default value, because it is provided by the FTF user.

Authentication management information necessary for security and accounting of the connection (user identification and password, account identification).

Protocol-identifier applicable value is **"FTF"**.

Protocol-version designates the VFP version. Negotiable, applicable value for this version **"1"**.

class-of-filestore specifies the class of virtual file model which will be used on this connection, negotiated. See Fig. 7 pag. 32 for values.

Class-of-service specifies the class of the Virtual File Service which will be used on this connection, negotiated. See Fig. 7 pag. 32 for values.

Diagnostic is the parameter that provides for error reporting appearing in response and confirmation primitive. It also appears in a number of request and indication primitives (for disruptive services). The diagnostic parameter conveys up to three elements of information, corresponding to three levels of error analisys:

- severity
- reason
- diagnostic supplement

Each element can be supplied only if the preceding (more synthetic) elements have been supplied. A separate diagnostic parameter is used for each detected error. Limitations specific to some services are indicated within the description of these services.

File-password specifies any password(s) to be used subsquently for protection of the file. The passwords attribute depends on local naming policies. So the Virtual Filestore has to allow any types of passwords.

Special-information Transparent data obeying special conventions between VFS users. It is not used in OSIRIDE.

File-name specifies the identification of the file. Each file within a given Virtual Filestore is identified in an unambiguous way by means of its name. As the neaming policy depends on the local operating system, the Virtual File Service allows any types of names.

File-attributes specifies the value to be assigned to file attributes other then name and password. They can be:

Global attributes

file-structure Defines the internal structuring of the file. Two models are supported: **unstructured** and **flat.** Other models of file structure may be considered in future versions. An unstructured file has no visible internal structure: it is composed only of a sequence of octets or characters (see file-data-type). A flat file is composed of records, without any relationship between records other than sequencing. This category includes most conventional files and the relational model.
The value is **symbolic** (flat/unstructured)

file-data-type This attribute describes the type of data stored in the file.
The value **Heterogeneous** means that field description are provided in the file attributes. Otherwise the file contents are consid-

ered as homogeneous and field descriptions are not supplied.
The value **character** means that the file data is entirely composed of characters.
The value **transparent** means that the file data is entirely composed of octets, whose contents are undefined.
When the file structure is **unstructured**, this attribute cannot have the value **heterogeneous.**
The value is **symbolic** (character/ transparent/ heterogeneous).

file-current-size This attribute specifies the approximate amount of user data currently in the file. The value provided at creation time indicates the minimum amount of space to be allocated to the file. After creation, the value of this attribute is updated locally every time the file grows. The unit of measure is **kilo-octets** if file-data-type is transparent, **kilo-characters** if file-data-type is character, and **records** if data-file-type is heterogeneous.
The value is **numeric.**

file-maximum-size This attribute defines the maximum size to which the file can grow. Crossing this boundary my cause an error. The unit of measure is the same as for the file-current-size.
The value is **numeric.**

Record attributes

record-sequence This attribute describes the order of the records when the file is sequentially accessed. Not applicable to unstructured files.
The value is **symbolic** (by position/by key).
Since different systems may adopt different data syntaxes, it is not possible to guarantee that the key sequence is preserved in a file transfer.

direct-access This attribute describes by which means the record are direcly accessible. It is not applicable to unstructured files.
The value is **symbolic** (by no means/by position/by key).

Record sequence and **direct access** together comprise the so called **file organization.** The table below shows the equivalence between some well known organizations and the value of these attributes.

	record sequence	
direct access	by position	by key
by no means	sequential	-
by position	relative	-
by key	random	index sequential

It has to be noted that the 'by key' value is not supported in OSIRIDE.

record-size-type This attribute defines whether all records have the same size or not. Not applicable to unstructured files.

The value is **symbolic** (fixed/variable).

record-size This attribute defines the maximum or fixed record size. It is applicable only if the file structure is **flat** and the file data type is not **heterogeneous**. If the file structure is **flat** and the file data is **heterogeneous**, the record size is deduced from the field descriptions.

The unit of measure is octets if file data type is **transparent** and characters if file data type is **character**

The value is **numeric**.

Clash-options specifies what to do if the supplied file-name corresponds to an already existing file. The legal values are:

Reject the existing file is kept; diagnostic severity is failure.

Keep the existing file is kept and selected; diagnostic severity is success.

Replace the existing file is replaced by the newly defined file; diagnostic severity is success.

Reversible-mapping specifies that the mapping between virtual and real file must be such that all the attributes supplied with the F-CREATE-FILE are returned unchanged on any subsequent F-READ-ATTRIBUTES (unless changed by the user).

If reversible mapping is requested, the F-CREATE-FILE will be rejected if the Secondary does not implement it.

Legal value: Yes/no

3.0 PROTOCOL ENCODING

3.1 GENERAL CONVENCTIONS

The bits of an octet are identified b1, b2, ...,b8, being b1 leftmost and most important bit. The same convection applies to string of more than one octet, the rightmost bit becoming b16, b24 or b36.

3.1.1 Message structure

Each message of the VFP is composed of two parts:

1. a one octet message header

2. a variable length message content

The message header contains a 8-bit message type. The legal type values are listed in Fig. 3.
The message is composed of the message parameters, in any order.
The representation of parameters is described in Fig. 4 pag. 28.

1 SP	2 SPR
3 SL	4 SLR
5 OP	6 OPR
7 CL	8 CLR
9 RL	10 CLR
11 BT	12 BTR
13 ET	14 ETR
15 DATA	16-31 unassigned
32 RA	33 RAR
34 CR	35 CRR
36 DL	37 DLR
38-255 unassigned	

Fig. 3. Message type value

3.1.2 Parameter encoding

3.1.2.1 Type/Length/Value technique (TLV technique)

The TLV technique is a method for coding an Information Unit.
Every information unit is encoded as a triplet, made of:

1. a type (1st field)

2. a length (2nd field)

3. a value (3rd field)

- **Type field (1 octet)**
  - b1 = 0
  - b2 - b8 = type value (1 to 127, 0 reserved)
- **length field (1 or 2 octets)**
  - b1 = 0 the length is specified on 7 bits (b2-b8)
  - b1 = 1 the length is specified on 15 bits (b2-b16)
  - b2-b8 or b16: binary number of octets of the value field.
- **value field (0 to N octets)**
  - all octets: data

3.1.2.2 Parameter values representation

The following value types are needed to represent the various parameters of the VFP:

C	character string. Characters are coded in ASCII
N	numeric. Unsigned binary integer, with 4 possible sizes: 8, 16, 24 or 32 bits.
S	symbolic. Unsigned binary integer, whose value has specific meanings. Size is 8 bits.
M	bit map. Bit string in which each bit encodes a specific meaning. Size is 8 or 16 bits. Any bit whose role is not defined must be encoded as 0. For bits representing specified options, the bit is set to 1 if the option is requested, 0 otherwise.
D	date (and optionally time). Encoded as specified in the standard ISO/2014 and ISO/3307: YYMMDD(hhmmss), where yy=year, mm=month, DD=day, hh=hour, mm=minute and ss=second. It is encoded as a character string (see above), with a size of either 6 octets (date only) or 12 octets (date and time)
T	transparent. encoded by VFS user.
A	aggregate. Composed of a field (sometimes smaller than 8 bits) of the above types.

3.1.2.3 Value truncation

For economy reasons, the minimum variable length string is used to express values.

- character strings do not contain unnecessary spaces.
- for numeric or symbolic, all unnecessary octets from left containing only zero bits are removed.
- for bit maps, all unnecessary octets from right containing only zero bits are removed.
- for transparent strings no truncation is applied.

3.1.2.4 Parameter encoding

For each parameter of the protocol (see Fig. 4 pag. 28) the following description is given:

- parameter type (unique identifier)
- maximum length of value field ("-" means unlimited)
- Type of value representation (C, N, S, M, D and T, see Fig. 4 pag. 28).
- encoding of all possible values (for S and M only)

The parameters which are directly mapped onto presentation service parameters do not appear in this table. These are : filestore-name and checkpoint-identification.

Default Values

The existence of a default value, to be used in case the parameter has not been explicitly specified, depends only on the parameter value type, see Fig. 4 pag. 28:

- for value types S and M, there is always a default value, equal to zero (0)
- for value types C, N, D and T, there is no default value.

Aggregates

A few parameters contain, instead of an elementary value, an aggregate of values. This aggregate is encoded as a fixed data structure when most elements are always present and there is no need for extendability. Otherwise, it is encoded recursively as a set of TLV items. In this latter case, the range of T's internal to

aggregate can overlap the range of T's defined in Fig. 4 pag. 28, since this represents another level of encoding.
The internal encoding of aggregate parameter is specified in the clauses starting from "3.1.2.5 Value of diagnostic parameter" pag. 29, up to "3.1.2.9 Contents of record-attributes aggregate" pag. 33.

type	parameter	value length	value type	value encoding
1	Diagnostic	-	-	aggregate
2	Protocol-definition	-	-	aggregate
3	File-name	64	C	
4	File-password	32	C	
5	Access-control-list	1	S	0=no; 1=yes
6	Access-mode	1	S	0 = In sequence
7	Processing-mode	1	S	0 = read, 1 = write, 2 = append
8	Lock	1	S	0 = exclusive 1 = shared read 2 = shared update
9	Failure-option	1	M	b1 = rollback
11	Authentication	-	-	aggregate
16	Open-identification	4	N	
17	Restart-option	4	N	
24	Clash-option	1	S	0 = reject, 1 = keep, 2 = replace
25	Reversible-mapping	1	S	0 = no, 1 = yes
26	Requested-attribues	1	M	b1 = history b2 = global b3 = record b4 = key b5 = field
28	History-attribues	-	-	aggregate
29	Global-attribues	-	-	aggregate
30	Record-attribues	-	-	aggregate
31	Key-attribues	-	-	aggregate
32	Field-attribues	-	-	aggregate
63	Special-information	2	T	
-	File-data	-	T	[3]

Fig. 4. Parameter encoding

[3] No type is defined for File-data. See "3.1.2.11 File data encoding" pag. 34.

3.1.2.5 Value of diagnostic parameter

The value field of the diagnostic parameter is a structure defined as follows in BNF:

<diagnostic value> ::= <severity> <reason> {<DS>}
<DS> ::= <DS type> <DS value length> <DS value>

The terminal elements are encoded as follows:

<severity>
4-bit unsigned binary integer (b1-b4 of first octet of the <diagnostic value>. The values are reported into Fig. 5 pag. 30. When the diagnostic parameter is omitted, the default value of <severity> is **success** (0).

<reason>
12 bit unsigned binary integer (b5-b8 of first octet of <diagnostic value>, followed by b1-b8 of second octet). For values see Fig. 6 pag. 31. The table also indicates the corresponding severity (or severities) and the messages where each reason value can appear (except for values applicable to the most messages). When the diagnostic parameter is omitted the default value of <reason> is **no reason provided** (0).

<DS type>
3-bit unsigned binary integer (b1-b3) of first octet of <DS>. legal values :0 - 2.

<DS value length>
5-bit unsigned binary integer (b4-b8 of first octet of <DS>.

<DS value>
Depends on <DS type> value, as follows:

0 character string, up to 31 characters

1 one octet containing a parameter type or 2 octet containing an aggregate type and a parameter type, in the case of parameters recursively encoded within aggregates.

2 one octet containing a message type.

EXCEPTION:

A special encoding is defined for the diagnostic parameter when it is supplied as user data in a P-DISCONNECT for compatibility with ECMA-85, which offers only 3 octets of user data.
In this special case, the following field are omitted: parameter type, parameter length, DS type, DS value length. Only 3 octets are encoded as follows:

- first 2 octets: severity reason (standard encoding)
- third octet: message type (standard value field of DS type 2).

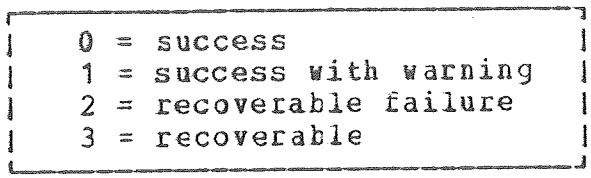

Fig. 5. Severity values

value	sev.	reason	messages
0	0	No reason provided	
1	123	No standard reason	
2	3	Unset parameter value	
3	3	Illegal parameter value	
4	3	Unsupported parameter value	
5	3	Illegally duplicated parameter	
6	3	Illegal parameter type	
7	3	Unsupported parameter type	
8	23	I/O error	AT,ATR
9	23	File space error	AT,ATR
10	23	Transmission error	AT,ATR
11	23	Record size error	AT,ATR
12	23	Presentation error (formatting)	AT,ATR
13	23	Presentation error (compression)	AT,ATR
14	23	Presentation error (encryption)	AT,ATR
15	3	VF protocol violation	VF conn. abort
16	3	Time-out expiration	DP
17	3	Shut-down	DP
18	3	File does not exist	SLR, OPR
19	3	Insufficient permission	
20	3	Isufficient resources	
21	3	File not mountable	SLR, CRR, OPR
22	3	File busy	SLR, CRR, DLR, OPR
23	1	More restrictive lock	OPR
24	3	Rollback not supported	OPR
25	3	File already exists	CRR
26	1	Existing file kept	CRR
27	1	Existing file replaced	CRR
28	3	File mapping not reversible	CRR
29	3	Local filestore error	
30	3	Local filestore restriction	
31	3	Password collision	CRR
32	1	Input file empty	OPR
33	3	Spec. error on local filestore	
34	3	Illeg. param. value duplic.	
35	3	Conflicting parameter value	
36	3	File not prepared for restart	OPR
37	3	File waiting restart	OPR
38	3	Open-identification not unique	OPR
39	3	Open-identification not found	OPR

When no messages are specified, the reason code is locally generated

Fig. 6. Diagnostic reason values

3.1.2.6 Contents of protocol-definition aggregate

Parameters of this aggregate are encoded as a structure containing a fixed number of elements in the specific order. This structure is described in Fig. 7.

type	parameter	value length	value type	value encoding
-	Protocol-identifier	1	S	0 = VFP
-	Protocol-version	1	M	b1 = version 1
-	class-of-filestore	1	M	b1 = unstructured files b2 = field description
-	Class-of-service	1	M	b1 = basic file mgt b2 = restart

Fig. 7. Protocol definition encoding

3.1.2.7 Contents of history-attributes aggregate

Parameters of this aggregate are recursively encoded as a set of TLV's which are defined in Fig. 8.

type	parameter	value length	value type	value encoding
56	Creation-date	12	D	
57	Creation-user-id	8	C	
58	Last-access-mode	12	D	
59	Last-access-user-id	8	C	
60	Last-modified-date	12	D	
61	Last-modified-user-id	8	C	
62	Total-number-of-accesses	4	N	
63	Total-number-of-modifications	4	N	

Fig. 8. History-attributes encoding

3.1.2.8 Contents of global-attributes aggregate

Parameters of this aggregate are recursively encoded as a set of TLV's which are shown in the table of Fig. 9 pag. 33.

type	parameter	value length	value type	value encoding
1	File-structure	1	S	0 = flat 1 = unstructured
2	File-data-type	1	S	0 = character 1 = transparent 2 = heterogeneous
3	File-current-size	4	N	
4	File-maximum-size	4	N	

Fig. 9. Global-attributes encoding

3.1.2.9 Contents of record-attributes aggregate

Parameters of this aggregate are recursively encoded as a set of TLV's which are defined in the table of Fig. 10.

type	parameter	value length	value type	value encoding
5	Record-size-type	1	S	0 = fixed 1 = variable
6	Record-size	4	N	
7	Record-sequence	1	S	0 = by position 1 = by key[4]
8	Direct-access	1	S	0 = by no means 1 = by sequence 2 = by key[4]

Fig. 10. Record-attributes encoding

3.1.2.10 Contents of authentication aggregate

Parameters of this aggregate are recursively encoded as a set of TLV's which are defined in Fig. 11 pag. 34.

[4] Not implemented in CSIRIDE Version 1.

type	parameter	value length	value type	value encoding
1	User-identification	8	C	
2	User-password	8	C	
3	account-identification	8	C	

Fig. 11. Authentication encoding

3.1.2.11 File data encoding

Because the encoding of file data for the tranfer is provided by the OSIRIDE Access Method, the length of this data may vary from one VFS to the other, due to differences in local syntaxes. Therefore, this length cannot be specified in the file-data parameter encoding, which is to be considered as a special type of parameter. Since the file-data parameter is the only parameter of the DATA message, it is encoded as transparent data of the VFP, whithout any parameter Header. File data occupy the totality of the DATA message, with exception of the message header.

4.0 VIRTUAL FILE MODEL SUBSET

The first OSIRIDE implementation is based on the subset named "kernel", with the "unstructured" extention. The attributes and the attributes values which describe the Virtual File in these subsets are:

- file-name
- file-password
- access-control-list
- history attributes
- file structure
- file-data-type
- file-current-size
- file-maximum-size
- record-sequence
- direct-access
- record-size-type
- record-size
- key-position
- key-length

4.1 FILE NAME

This is a **mandatory** parameter, which consists on a string of characters transferred transparently.

It is used to identify the file.

4.2 FILE PASSWORD

This is a **non mandatory** parameter, which **is supported** in OSIRIDE. In OSIRIDE it is used both for carrying security information and for other system dependent information which can be useful for the file access.

4.3 ACCESS CONTROL LIST

This is a **non mandatory** parameter, which is not supported in OSIRIDE.

4.4 HISTORY ATTRIBUTES

These are **non mandatory** parameters, which are not supported in OSIRIDE.

4.5 FILE STRUCTURE

This is a **mandatory** parameter, with value "flat" in the kernel subset. The value "unstructured" is also supported, for all cases when files with a complex structure are transmitted between homogeneous systems.

4.6 FILE DATA TYPE

This is a **mandatory** parameter, with symbolic values "character" and "transparent" in the kernel subset. Both values are supported in OSIRIDE.

4.7 FILE CURRENT SIZE

This is a **non mandatory** parameter, which **is supported** in CSIRIDE.

4.8 FILE MAXIMUM SIZE

This is a **mandatory** parameter.

4.9 RECORD SEQUENCE

This is a **mandatory** parameter, as value "by position", and an **optional** parameter as value "by key".

The value "by key" is not supported in CSIRIDE.

4.10 DIRECT ACCESS

This is a **mandatory** parameter, as value "by no means" and an optional one as values "by position" and "by key".

The value "by key" is **not** supported.

4.11 RECORD SIZE TYPE

This is a **mandatory** parameter, as value "fixed", and an **optional** one, as value "variable".

Both values are supported.

4.12 RECORD SIZE

This is a **mandatory** parameter.

4.13 KEY POSITION

This is a **non mandatory** parameter, which is not supported.

4.14 KEY LENGTH

This is a **non mandatory** parameter, which is not supported.

5.0 VIRTUAL FILE PROTOCOL

The Virtual File Protocol defined in "2.0 Protocol overview" pag. 8 is subsetted into three subsets:

1. Kernel, with the following service primitives:
   - F_CONNECT
   - F_RELEASE
   - F_DISCONNECT
   - F_ABORT
   - F_END_GROUP [5]
   - F_SELECT_FILE
   - F_RELEASE_FILE
   - F_OPEN_FILE
   - F_CLOSE_FILE
   - F_BEGIN_DATA
   - F_DATA
   - F_END_DATA
   - F_ABORT_DATA
2. Basic file management extention, with the following service primitives:
   - F_READ_ATTRIBUTES
   - F_CREATE_FILE
   - F_DELETE_FILE
3. Restart extention, with the following service primitives:
   - F_RESTART
   - F_CHECKPOINT

The OSIRIDE File Transfer Protocol is based on the first two subsets, i.e. the kernel and the basic file management extention, although a recovery mechanism has been studied. In what follows,

---

[5] Support for this primitive is **optional** and **not supported** in OSIRIDE.

the implications of the OSIRIDE recovery mechanism in choosing the File Transfer Protocol options and subsets are explained.

5.1 THE OSIRIDE RECOVERY MECHANISM

According to the ECMA File Transfer Protocol, when flat files are transmitted, the transmission is performed on a record-by-record basis, i.e. only one record is sent a time.

This means that, in case of network failure, the previously interrupted transmission may be easily restarted, if the number of the last sent record has been recorded. There is **no need**, therefore, for an explicit checkpointing mechanism, which is implemented by the F_CHECKPOINT service primitive.
Moreover, the service provided by the F_RESTART primitive is not needed too, at least in the first OSIRIDE implementation.
This led not to implement the Restart extention subset.

On the other hand, the recovery mechanism **must** use the **open-identification** and **restart-position** parameters of the F_OPEN_FILE service primitive, which are supported **only** if the Restart extention is supported.

The choice that has been made for OSIRIDE is:

1. All three subsets are formally supported in OSIRIDE, that is, support for all three of them is indicated in the F_CONNECT service primitive.

2. The implementation of the service primitives which constitute the Restart extention, i.e. F_RESTART and F_CHECKPOINT, is **not mandatory** in OSIRIDE.

3. The OSIRIDE File Transfer **will never** use such primitives.

5.2 STATE TRANSITIONS TABLES

This section is the formal description of the OSIRIDE Virtual File Protocol, that means the state tables of ECMA-85 reduced according to the choices and the subsets selections which have been made in this document.

The tables which follow come from ECMA-85 Appendix D, and adopt the same formalism as there. It has to be noted that there are no tables for what **grouping** and **restart subset** are concerned, because those functions/subsets are not implemented in OSIRIDE.

5.3 FORMAL DESCRIPTION

Fig. 12 lists the states which are used in the formal description. For each entry there is a state code and a brief description.

Fig. 13 lists the events which are used in the formal description. For each entry there is a state code and a brief description.

Fig. 14 lists the acronyms which are used in the formal description to identify messages sent.

Fig. 15 lists the conditions which are used in the formal description. For each entry there is a condition code and a brief description.

STATE-code	State description	State tables
CF..	Connection states	
CF01	Dormant	Fig. 16, Fig. 17
CF02	SP-pending	Fig. 16, Fig. 17
CF03	RP-pending	Fig. 16, Fig. 17
FM..	File Management states	
FM01	No file	Fig. 16, Fig. 17
		Fig. 18, Fig. 19
FM02	SL Pending	Fig. 18, Fig. 19
FM03	CR Pending	Fig. 18, Fig. 19
FM04	File selected	Fig. 18, Fig. 19
		Fig. 20, Fig. 21
FM05	RL Pending	Fig. 18, Fig. 19
FM06	DL Pending	Fig. 18, Fig. 19
FM07	RA Pending	Fig. 18, Fig. 19
FM08	OP Pending	Fig. 20, Fig. 21
FM09	File open	Fig. 20, Fig. 21
FM10	CL pending	Fig. 20, Fig. 21
FM11	BT pending	Fig. 20, Fig. 21
DT..	Data Transfer states	
DT01	Data	Fig. 22, Fig. 23
DT02	ET Pending	Fig. 22, Fig. 23
DT04A	AT Pending (Sender)	Fig. 22, Fig. 23
DT04B	AT Pending (Receiver)	Fig. 22, Fig. 23
DT05	File aborted	Fig. 20, Fig. 21

Fig. 12. Protocol Machine States

Event Code	Event Description
XX	XX request message
XXR	XX response message
XX-RQ	Request primitive associated to XX
XX-IN	Indication primitive associated to XX
XX-RP	Response primitive associated to XX
XX-CF	Confirmation primitive associated to XX

Fig. 13. Protocol Machine Events

Acronym	Message name	State tables
SP	Select protocol	Fig. 16, Fig. 17
RP	Release protocol	Fig. 16, Fig. 17
AB	Abort	Fig. 16, Fig. 17 [6]
DP	Disconnect protocol	Fig. 16, Fig. 17
SL	Select file	Fig. 18, Fig. 19
RL	Release file	Fig. 18, Fig. 19
CR	Create file	Fig. 18, Fig. 19
DL	Delete file	Fig. 18, Fig. 19
RA	Read attributes	Fig. 20, Fig. 21
OP	Open file	Fig. 20, Fig. 21
CL	Close file	Fig. 20, Fig. 21
BT	Begin transfer	Fig. 20, Fig. 21
DATA	Data	Fig. 22, Fig. 23
ET	End Transfer	Fig. 22, Fig. 23
AT	Abort Transfer	Fig. 22, Fig. 23

Fig. 14. List of protocol messages

6 There are only two events related to the acronym AB:

AB: Presentation connection abort indication from layer 6

AB-IN: VFS Connection abort indication

Condition	Meaning
+	Value of diagnostic severity is "success" or "success with warning"
-	Value of diagnostic severity is "failure"
pr	VFS entity is "primary"
sc	VFS entity is "secondary"
CONDV: cond	This intersection is valid only if **cond** is true

Fig. 15. Conditions

	CF01 Dormant	CF02 SP-Pend.	CF03 RP-Pend.	FM01 No File	Any Other State
SP-RQ	SP :CF02				
SPR		SP-CF +:FM01 -:CF01			
RP-RQ				RF :CF03	
RPR			RP-CF :CF01		
DP-RQ		DP :CF01	DP :CF01	DF :CF01	DP :CF01
DPR		DP-IN :CF01	DP-IN :CF01	DP-IN :CF01	DP-IN :CF01
AB		AB-IN :CF01	AB-IN :CF01	AB-IN :CF01	AB-IN :CF01

Fig. 16. Connection Protocol (primary)

	CF01 Dormant	CF02 SP-Pend.	CF03 RP-Pend.	FM01 No File	Any Other State
SP	SP-IN :CF02				
SP-RP		SPR +:FM01 -:CF01			
RP				RP-IN :CF03	
RP-RP			RPR :CF01		
DP-RQ		DP :CF01	DP :CF01	DP :CF01	DP :CF01
DP		DP-IN :CF01	DP-IN :CF01	DP-IN :CF01	DP-IN :CF01
AB		AB-IN :CF01	AB-IN :CF01	AB-IN :CF01	AB-IN :CF01

Fig. 17. Connection Protocol (secondary)

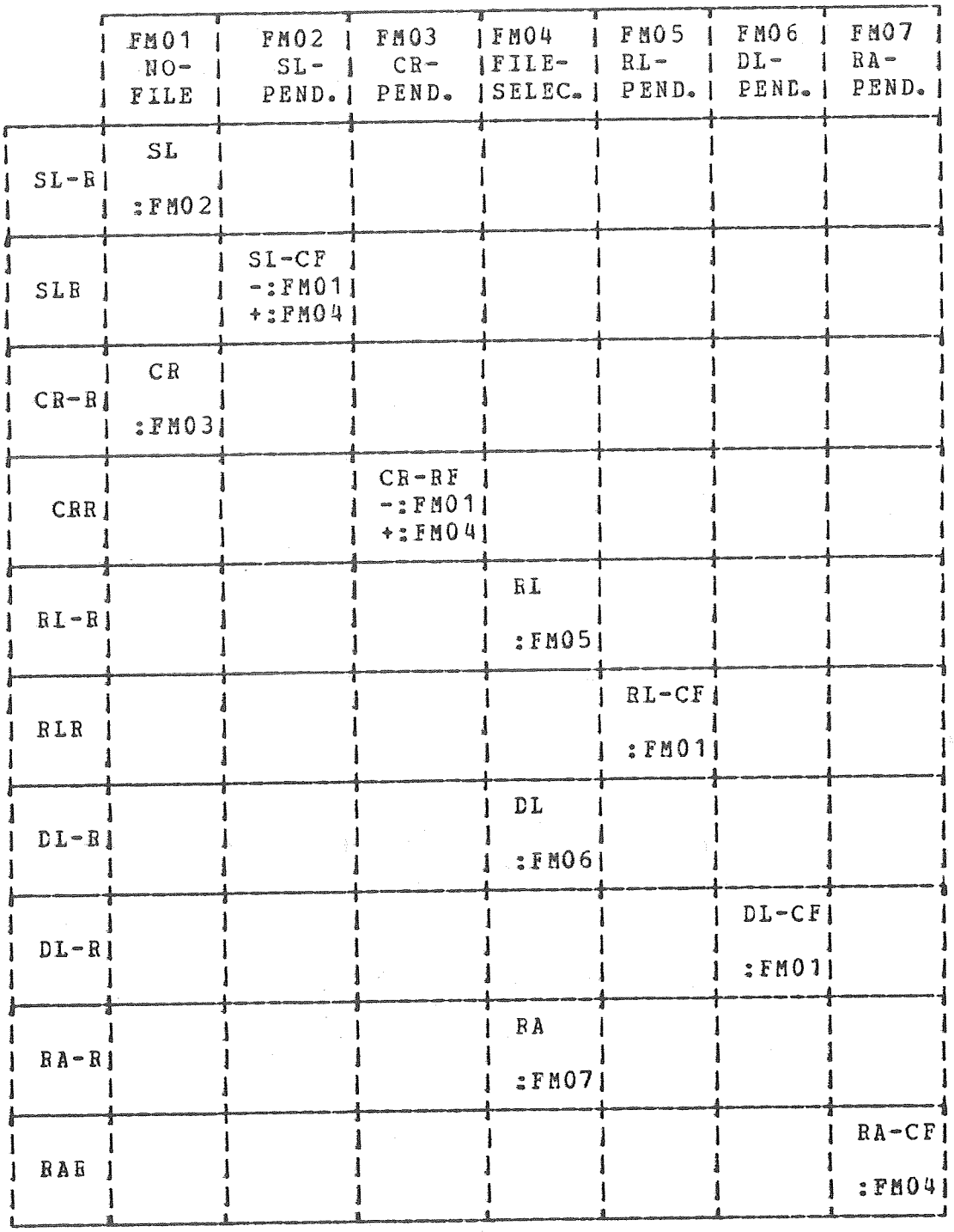

Fig. 18. File Management - File enclosure (primary)

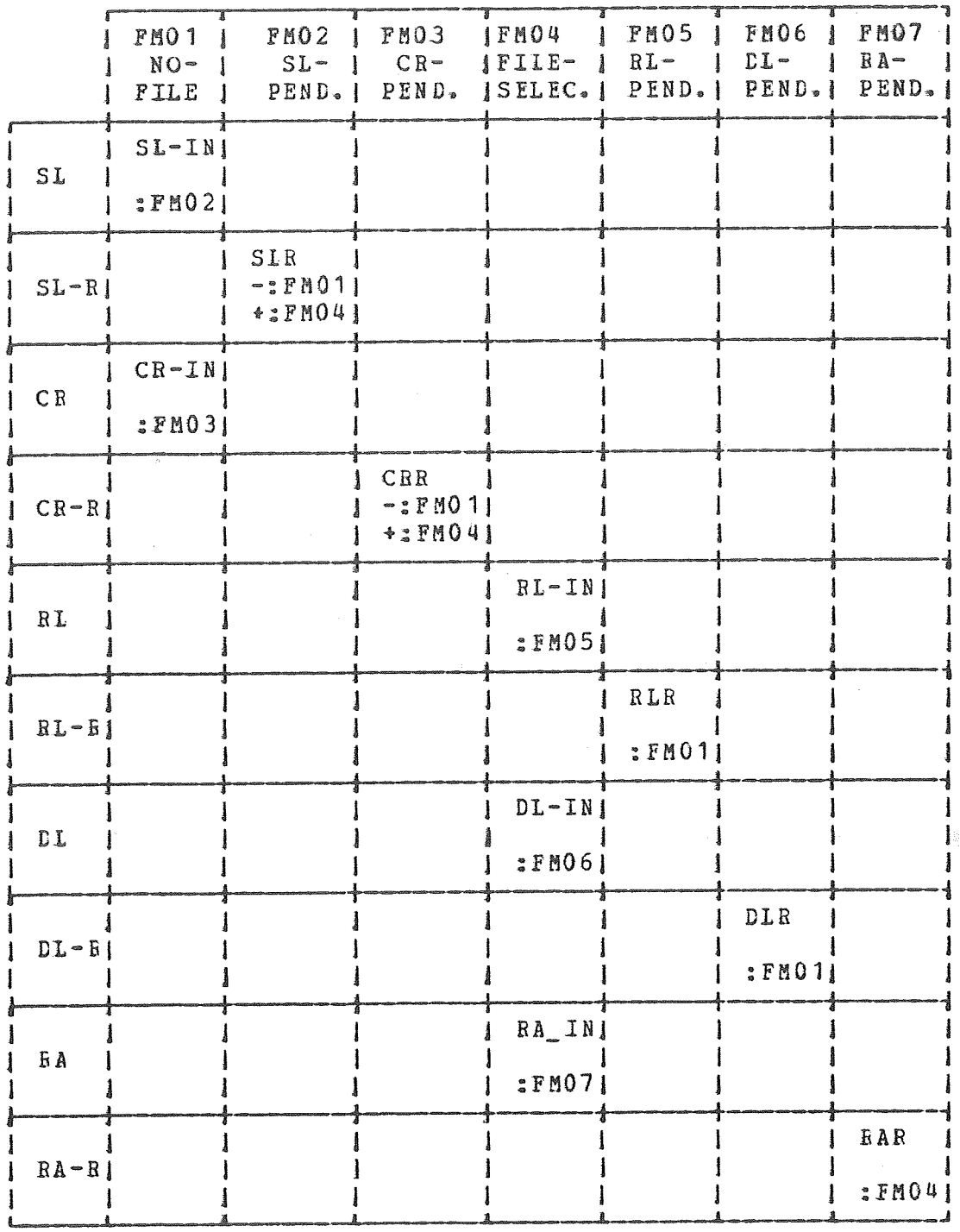

Fig. 19. File Management - File enclosure (secondary)

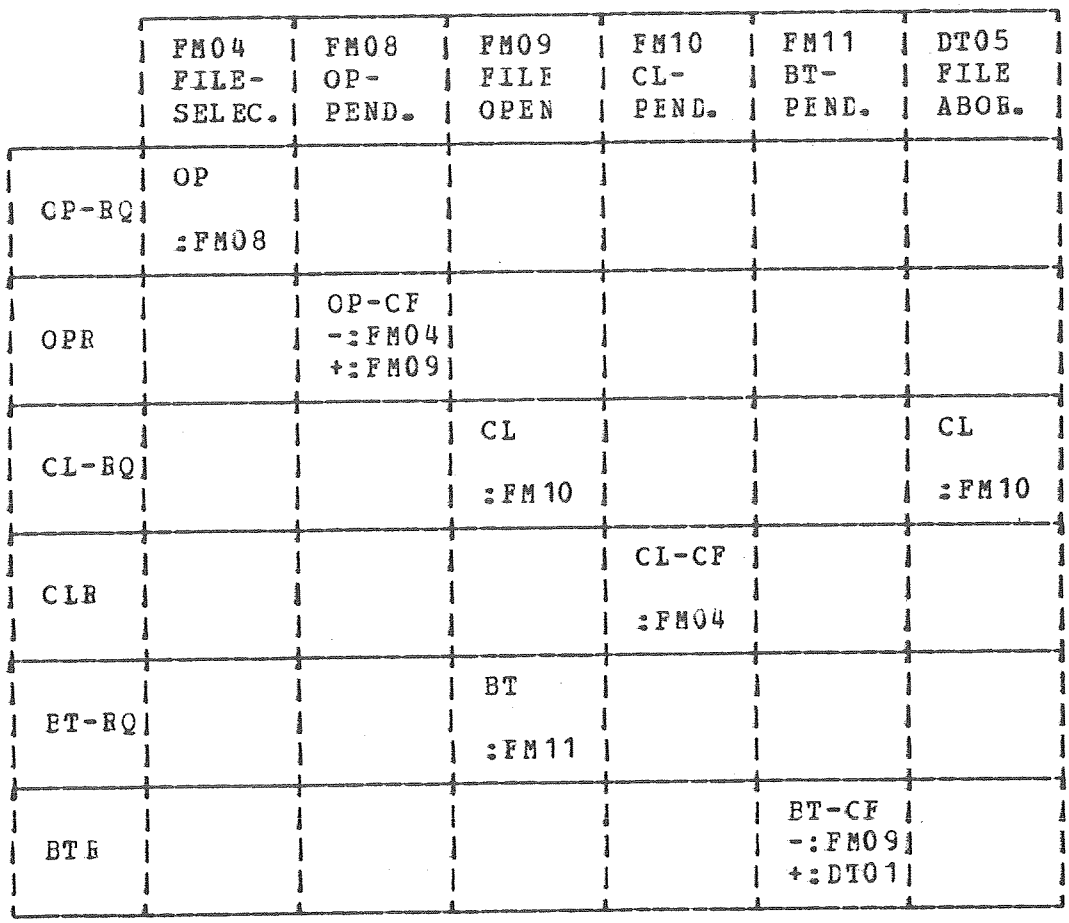

Fig. 20. File management - Open enclosure (primary)

	FM04 FILE-SELEC.	FM08 OP-PEND.	FM09 FILE OPEN	FM10 CL-PEND.	FM11 BT-PEND.	DT05 FILE ABOR.
CP	OP-IN :FM08					
OP-RP		OPR -:FM04 +:FM09				
CL			CL-IN :FM10			CL-IN :FM10
CL-RP				CLR :FM04		
BT			BT-IN :FM11			
BT-RP					BTR -:FM09 +:DI01	

Fig. 21. File management - Cpen enclosure (secondary)

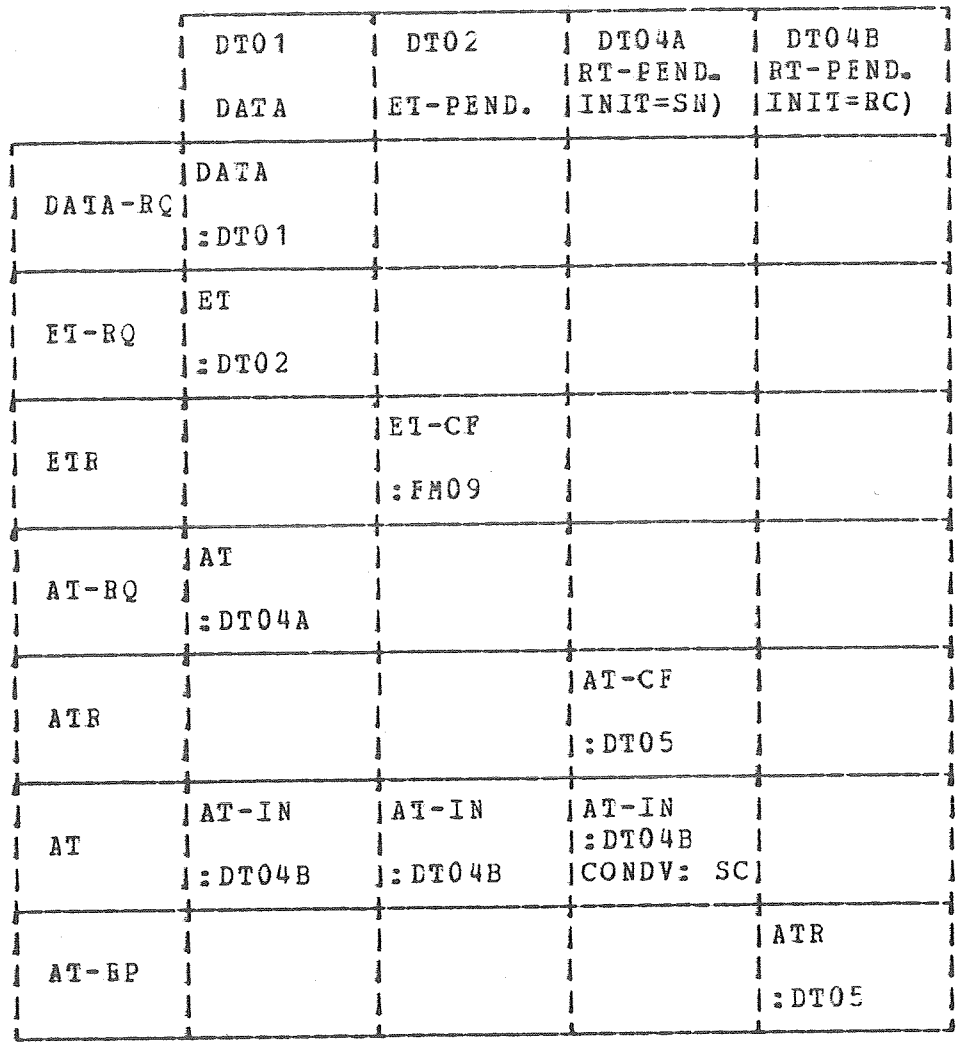

Fig. 22. Data Transfer (sender)

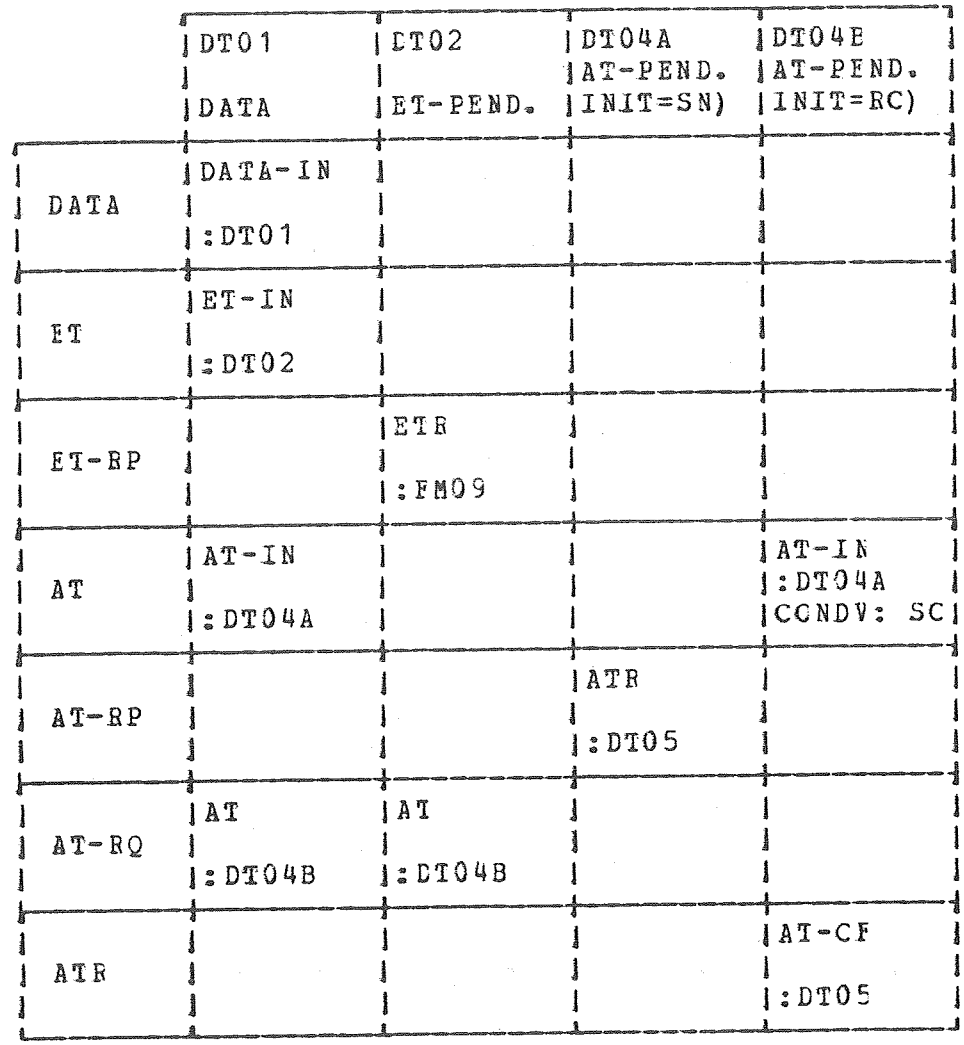

Fig. 23. Data Transfer (receiver)

Second part

1.0 INTERNAL FTF STRUCTURE

1.1 DESIGN HYPOTHESIS

The following design specifications have been written assuming some implementation choices which are not particularly restrictive and, if necessary, can be easily removed and substituted with similar solutions. The choices are:

1. A new instance of FTF Primary process is activated for each user request of SEND or RECEIVE and a new instance of FTF Secondary process is activated for each incoming Connect request. FTF terminates and is de-activated at the normal or abnormal termination of each transfer.

2. When FTF Primary terminates, it stores the result in a record (disk context) of a disk file (context table). The user can delete a disk context with a CANCEL operation.

1.2 GENERAL OVERVIEW OF THE FTF STRUCTURE

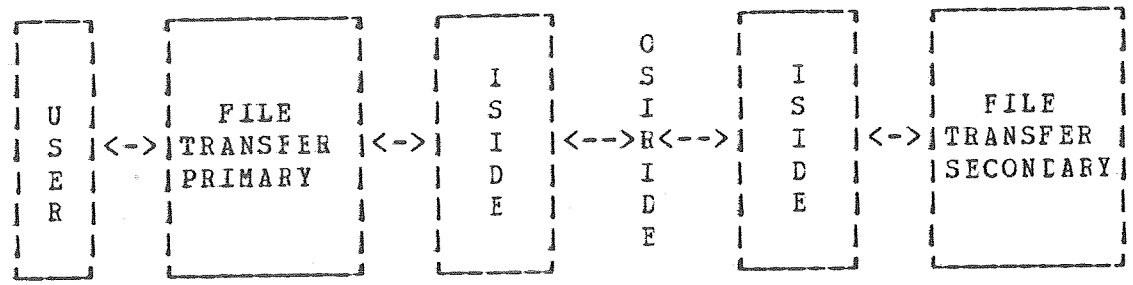

Fig. 24. General Overview of the FTF structure

The FTF is achieved through the cooperation of two different processes, the Primary and the Secondary, which generally reside on different machines and play complementary roles:

- the **Primary**, local to the user, interfaces the user requests and is the "**master**" of the transfer operation, as it activates the secondary and sends commands to it. The communication between the user and the primary task is established by the routines of **FT User Interface** see "1.4 FTF user interface" pag. 55.

- **The secondary**, local to the accessed file [7], executes all commands issued by the Primary. It communicates only with its own file-system and allows the Primary to access this file-system through the **VFS primitives**. These primitives refer to a virtualized format of the '**Virtual File**' model wich has been presented in the first part.

Note that the Primary generally cooperates with a Secondary which is remote with respect to it, i.e. which resides on another machine. The only case in which the Secondary could be local to the Primary is when the accessed file is local to the user. In this case, the File Transfer happens locally: the Primary dialogs with the Secondary and both are on the same machine. This is always possible because the ISIDE services[8] can be used also to perform local inter-process communication. Therefore, it is important to point out that both Primary and Secondary should be implemented on each system to achieve the complete File Transfer functionality. The complete functionality consist of being able to initiate a transfer operation on behalf of a local user and to respond to a transfer proposal coming from a remote system. The tasks of Primary and Secondary on the same system do not interact each other except when they provide a local transfer.

1.3 FTF USER SERVICES

The FTF can be used either by a program or a terminal application and offers the following services:

- FTF performs the transfer of a file: this can occur either between two different systems or locally to the system which requires the operation. The user resides on the system where the request is generated and must be local to one of the two systems involved in the requested transfer. Two types of transfer are possible:

  - **send a file**
  - **receive a file**

These operations can be successfully terminated or interrupted. Interruption of a file transfer can be either reversible or not. In the former case a restart operation will be possible. In the latter case, both parties purge any information related to the transfer (CONTEXT) and the transmitted data also, i.e. they try to re-establish the conditions as before the file transfer was initiated.

---

7 The 'accessed file' is the one which the transfer is addressed to. It is accessed to be read in case of a **RECEIVE** operation, or accessed to be written in case of a **SEND** operation.

8 ISIDE is the OSIRIDE network access method

FTF allows:

- stopping a running transfer operation;
- cancelling an interrupted transfer operation;
- restarting an interrupted restartable transfer operation;
- waiting for the transfer operation completion;
- monitoring the status of the transfer, to provide information concerning the progress of the transfer;
- deleting a file on the remote system.

For each service the FTF returns a diagnostic to the user. A detailed description of diagnostic fields is in "1.8 FTF diagnostic for the user" pag. 61.

1.4 FTF USER INTERFACE

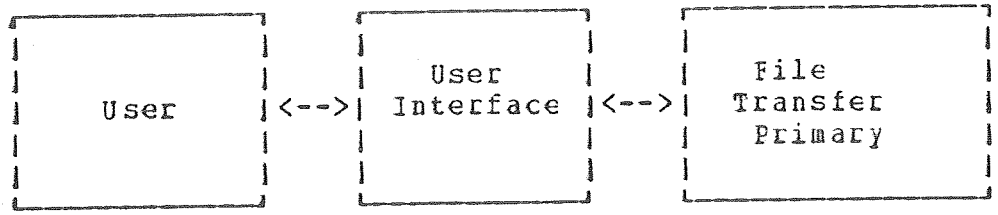

Fig. 25. User Interface

The user interface here described consists of a set of logical commands with related parameters. Each command is used to invoke one service. The specification provided here is indipendent from any specific implementation of the real user interface, from the user environment and the used programming language.

1.4.1 FTF commands

To use FTF services, the user must invoke the commands listed in Fig. 26 pag. 56, with the associated parameters.

For each command, the list of the associated parameter is given, together with the parameter returned from the FTF. The meaning of the parameters is explaned in "1.4.2 Parameters meaning" pag. 56.

command	parameters	returns
SEND	local file name host name remote file name effect password(s)	transfer identifier return code
RECEIVE	local file name host name remote file name effect password(s)	transfer identifier return code
RESTART	transfer identifier	return code
CANCEL	transfer identifier	return code
DELETE	host name remote file name password(s)	return code
STOP	transfer identifier	return code
STATUS	transfer identifier	return code

Fig. 26. User command format

1.4.2 Parameters meaning

The meaning of the parameters is explaned in the following list:

local file name identifier of the local file, both in SEND and RECEIVE operation.

host name name of the remote host involved in the operation.

remote file name identifier of the remote file, both in SEND and RECEIVE operation.

effect indication of the operation type which the user wants to execute on accessed file. The

admitted destination effects are listed in "1.4.3 Parameters values" pag. 57.

password(s) key word for accessing the remote file

transfer identifier identifier of the transfer request. It is assigned by the FTF to the SEND or RECEIVE user request. No algorithms are described for the assignement of the identifier, the only restriction being that it must be unique. It must be issued by the user for any other request concerning the same transfer (e.g. CANCEL, RESTART , etc.)

return code result of the user request. The list of all possible values of return codes is provided in "1.8 FTF diagnostic for the user" pag. 61.

1.4.3 Parameters values

The parameters legal values are described in the following list.

LOCAL FILE NAME The only restriction is that the VFP fixes the maximum length for this parameter to 64 characters.

HOST NAME The only restriction is that the VFP fixes the maximum length for this parameter to 16 characters. This however, has to be a name known in CSIRIDE.

REMOTE FILE NAME The only restriction is that the VFP fixes the max length for this parameter to 64 characters.

EFFECT may assume the following values:

MAKE if the destination file does not exist, it will be created, with the specified file name, the contents and attributes of the source file, as a copy of the source.
If the file exists, an error will be returned.

REPLACE if the file exists, the entire file is rebuilt, according to source file

---

9 No effect on source file can be specified. The source file is always just read and released at the end of transfer.
Simultaneous access in reading is allowed by the protocol and should not be prevented by the implementation.

organization, attributes and contents.
If the file does not exist, an error is returned.

APPEND if the file exist, a set of records is added to it.
If the file does not exist, an error is returned.

If no effect has been specified, the default value is MAKE.

PASSWORD(S) The only restriction is that the VFP fixes the max length for this parameter to 32 characters.

1.4.4 Parameters types

local file name	string of characters
host name	string of characters
remote file name	string of characters
effect	string of characters
password(s)	string of characters
transfer identifier	integer number
return code	string of characters

1.4.5 Explanatory notes

Restart of a transfer

One of the above presented services is the possibility to restart[10] an interrupted file transfer operation. To achieve this service, the Primary stores all information permitting the re-establishement of transfer, so that each interrupted transfer is univocally identified. For this purpose, the **Primary** associates with each transfer a particular set of information (the CONTEXT), which survives to the recoverable transfer interruptions in order to allow future resumption. For each transfer a **CONTEXT** exists in main memory (memory CONTEXT) and a copy of it on the permanent memory (disk CONTEXT). The CONTEXT contents are described in "3.0 Data structures" pag. 97. The CONTEXTs (both 'disk' and 'memory') are created at

10 The RESTART facility is not defined on DELETE, CANCEL, STATUS.

the beginning of a SEND or RECEIVE operation. The disk CONTEXT is used for RESTART, STATUS[11] or CANCEL operation, whereas the memory CONTEXT is examined when a STATUS request is issued during a running transfer. The assigment of CONTEXT name, 'disk' and 'memory', are local implementation choices.

The CONTEXT is identified by means of the Transfer IDdentifier, "**TID**", which is an integer number represented by characters. The TID generation is provided by the file transfer Primary: it is returned to the user to be specified in a RESTART, STATUS, STOP or CANCEL user request. Through this identifier the FTF can retrieve the CONTEXT of the transfer to be restarted, cancelled, tested or stopped. The list of disk CONTEXT names, local to the File Transfer implementation tasks, is stored in a file and referred here as "disk CONTEXT TABLE". Its name is known only to the local implementation of the FTF.

Type of services

Two types of services are provided:

- **not suspensive** services
- **suspensive** services

On the request for **not suspensive** service like SEND, RECEIVE or RESTART, the FTF responds immediately with local acceptance or refusal. The results of these operations may be obtained by means of the STATUS primitive.

In the case when a **suspensive** service was requested, like DELETE, CANCEL and STOP, the diagnostic directly indicates the result.

1.5 TYPES OF INTERRUPTIONS

Three different types of interruption may happen:

1. Voluntary interruption ordered by the user during the 'data transfer enclosure' (see "2.2.2.4 STOP procedure" pag. 75)

2. Voluntary interruption by FTF itself caused by some error or abnormal condition diagnosed during the interaction with:

---

[11] In the following two different types of STATUS request (see "2.0 FTF implementation specifications" pag. 66) are described:

- STATUS request issued during a active FTF process
- STATUS request issued after the termination of FTF process

- local user
- local operating system
- local file-system
- the remote FTF (i.e. protocol errors or VFS errors signalled through the protocol)

3. Loss of connection signalled by lower layers.

As far as restartability of the transfer is concerned, the following rules apply to the above interruption types:

type 1 and 2: not restartable. Any trace of the File Transfer is cancelled by both parties and the conditions preceding the transfer are re-established (except when "replacing" or "appending" a file).

type 3: restartable by the Primary on user request. The partially transferred file is available to the user, therefore the restart may be not possible if modifications have occurred in it.

A power failure occurring on the remote system is known as a **loss of connection.**

Note: to cancel a restartable transfer it is sufficient to cancel its CONTEXT and delete the partially transferred file (which can be local or remote with respect to the user).

1.6 STATES OF THE FILE TRANSFER

Five different states are admitted for a file transfer, i.e. at any moment of its activity, the transfer must be in **one** of the following conditions[12]:

NOT ACTIVE transfer not in progress.

RUNNING transfer in progress.

SUCCESSFULLY TERMINATED no abnormal condition during the transfer has occurred.

RESTARTABLE a transfer interruption of type 3 has occurred.

---

12 These listed 'transfer states' should not be confused with the ones of the VFP ECMA-85, mentioned in "5.0 Virtual File Protocol"of "First Part" and in Fig. 12 pag. 41.

NOT RESTARTABLE a transfer interruption of type 1 or 2 has occurred.

1.7 STATES OF THE FTF PROCESS

The FTF process can be in two states:

ACTIVE FTF process is active, i.e. running or waiting for request

DISACTIVE FTF process is not active, i.e. not running and unable to receive requests.

1.8 FTF DIAGNOSTIC FOR THE USER

The FTF diagnostics are listed in the tables of Fig. 27 pag. 62, Fig. 28 pag. 63, Fig. 29 pag. 64 and Fig. 30 pag. 65.

Type	meaning	FTP Primitive which can return it	Type of interrup
0	Transfer successfully terminated	STATUS[13]	/
1	Request locally accepted and in progress	SEND, RECEIVE, RESTART	/
2	running	STATUS	/
3	Stop down	STOP, STATUS	1
4	Cancel down	CANCEI	/
5	file success. deleted	DELETE	/
6	loss of. connection[14]	STATUS, DELETE	3
7	Power fail[14]	STATUS	3

Fig. 27. VFP diagnostic

13 The diagnostic returned by the STATUS is referred to the operation identified by the TID parameter (transfer or restart). The STATUS operation itself is supposed always successful.

14 These errors can be also returned by an operation on which the restart is not defined, like the delete operation.

Type	meaning	FTF Primitive which can return it	Type of interrup
10	Interface error	STATUS, DELETE, CANCEI	2
11	Protocol Violation	STATUS, DELETE, CANCEI	2
12	Wrong status of protocol	STATUS, DELETE, CANCEI	2
13	Parameters conversion error	STATUS, DELETE, CANCEI	2
14	Diagnostic returned from Secondary	STATUS, DELETE, CANCEI	2
15	Connection failed	STATUS, DELETE, CANCEI	2
16	End of data from Primary	STATUS	2

Fig. 28. VFP diagnostics (continued)

Type	meaning	FTF Primitive which can return it	Type of interrup
17	Attempt to OPEN a not existing file	STATUS	2
18	Attempt to CREATE an already existing file	STATUS	2
19	Privilege violation	STATUS	2
20	File structure invalid	STATUS	2

Fig. 29. Local File System diagnostics

Type	meaning	PTF Primitive which can return it	Type of interrup
21	Local user	ALL PRIMITIVES	2
22	Invalid destination effect	SEND, RECEIVE	2
23	Source file does not exist	STATUS	2
24	Context not found	STATUS, CANCEL,RESTART	2
25	Sintax error in the PTF request	ALL PRIMITIVES	2
26	Missing parameter	ALL PRIMITIVES	2
27	STOP refused (too late)	STOP	2

Fig. 30. PTF general diagnostic

2.0 FTF IMPLEMENTATION SPECIFICATIONS

Fig. 31 shows the detail of the FTF implementation. The Primary is driven by the user's requests, which arrive through the user interface.

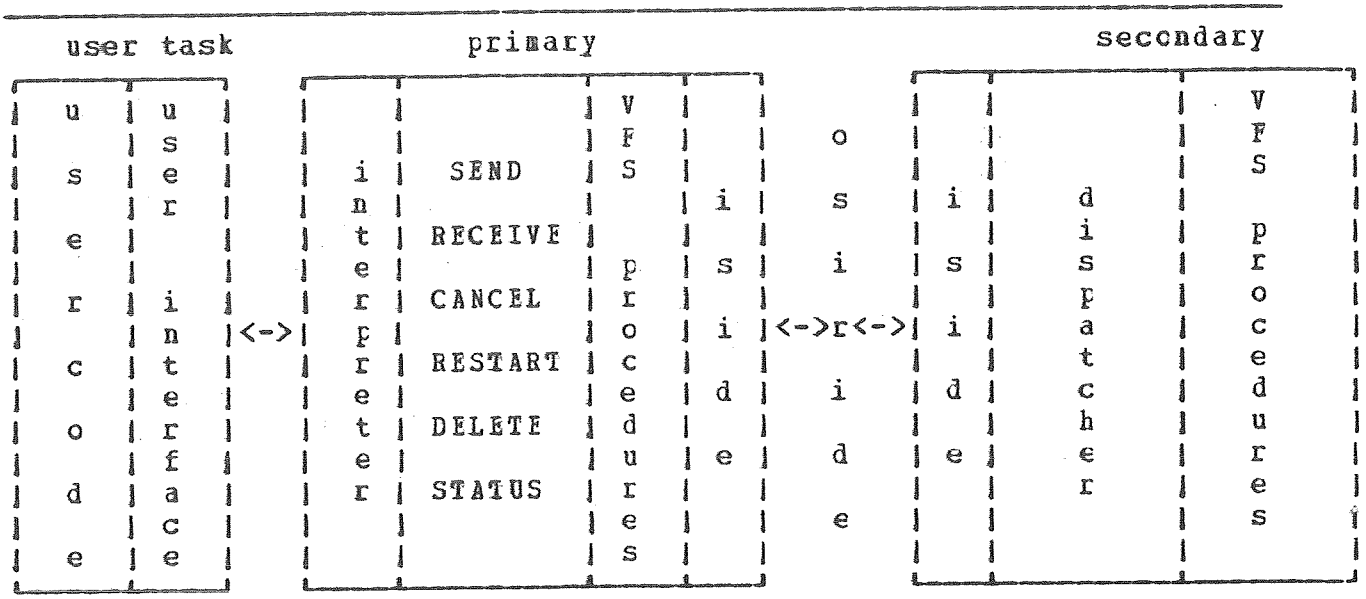

Fig. 31. FTF architecture

The following are the recognized user interface procedures:

SEND activate FTF process
transfer request for SEND
wait for the immediate result
return

RECEIVE activate FTF process
transfer request for RECEIVE
wait for the immediate result
return

RESTART activate FTF process
transfer request for RESTART
wait for the immediate result
return

CANCEL access the disk CONTEXT TABLE
If the TID does not exist then
return the diagnostic 'context not found'
else
cancel the row corresponding to TID, only if the 'transfer status' indicates a termination status

DELETE activate FTF process
transfer request for delete.

```
          Wait for operation completion

STATUS    if FTF is RUNNING then
            transfer 'transfer status'
            wait for response
            return
          else
            access the disk CONTEXT and get the diagnostic of
            transfer whose TID has been specified

STOP      if FTF is NOT ACTIVE then
            return the diagnostic 'stop refused (too late)' else
            transfer request for STOP
            wait for operation completion
```

2.1 FTF STRUCTURE OVERVIEW

The Primary and Secondary tasks have a modular structure. The following is a first, schematic description of how the Primary and the Secondary modules behave. After a FTF user request, either through a application program or a terminal operator (see FTF interface in Fig. 31 pag. 66) the **Primary task** executes the following actions:

1. get user command from the interface routines and examine the syntactic-semantic aspects of the parameters.

2. according to the requested operation, call the procedures which handle the specific operation.

3. the above-mentioned procedures interact with the local file-system and call the VFS Primary procedures.

4. the VFS procedures perform the following functions:

   - checking and updating of the protocol status.
   - encoding[15].
   - interfacing ISIDE services tc send/receive the request/response.

The **Secondary task** functions (see Fig. 31 pag. 66) are the following:

1. check the received message and call the corresponding VFS procedures

2. the VFS procedures

   - test and update the protocol status.

- map input parameters.

- perform the appropriate actions, according to the received message.

- call ISIDE services to send the response.

Each one of the above listed items will be developed in the next paragraphs.

2.2 PRIMARY STRUCTURE

The File Transfer Primary is local to the user, handles his requests and returns the related responses (see "1.2 General overview of the FTF structure" pag. 53). The Primary task provides for different functions, by using the set of procedures described in the following paragraphs:

- interpretation of the user command string as received by the interface, and activation of the proper module. The interpreter module is the 'main' program of the Primary.

- All VFS procedures, which constitute the VFS Primary, communicate with the Secondary task by using the ISIDE services.

---

15 The FTF performs two different type of encoding/decoding operations:

- encoding/decoding to/from the TLV encoding to protocol message procedures. This operation is be called 'encoding' function.

- interfacing the local file-system by executing a 'mapping function' between the real file parameters and operations and Virtual file-system attributes and protocol requests.

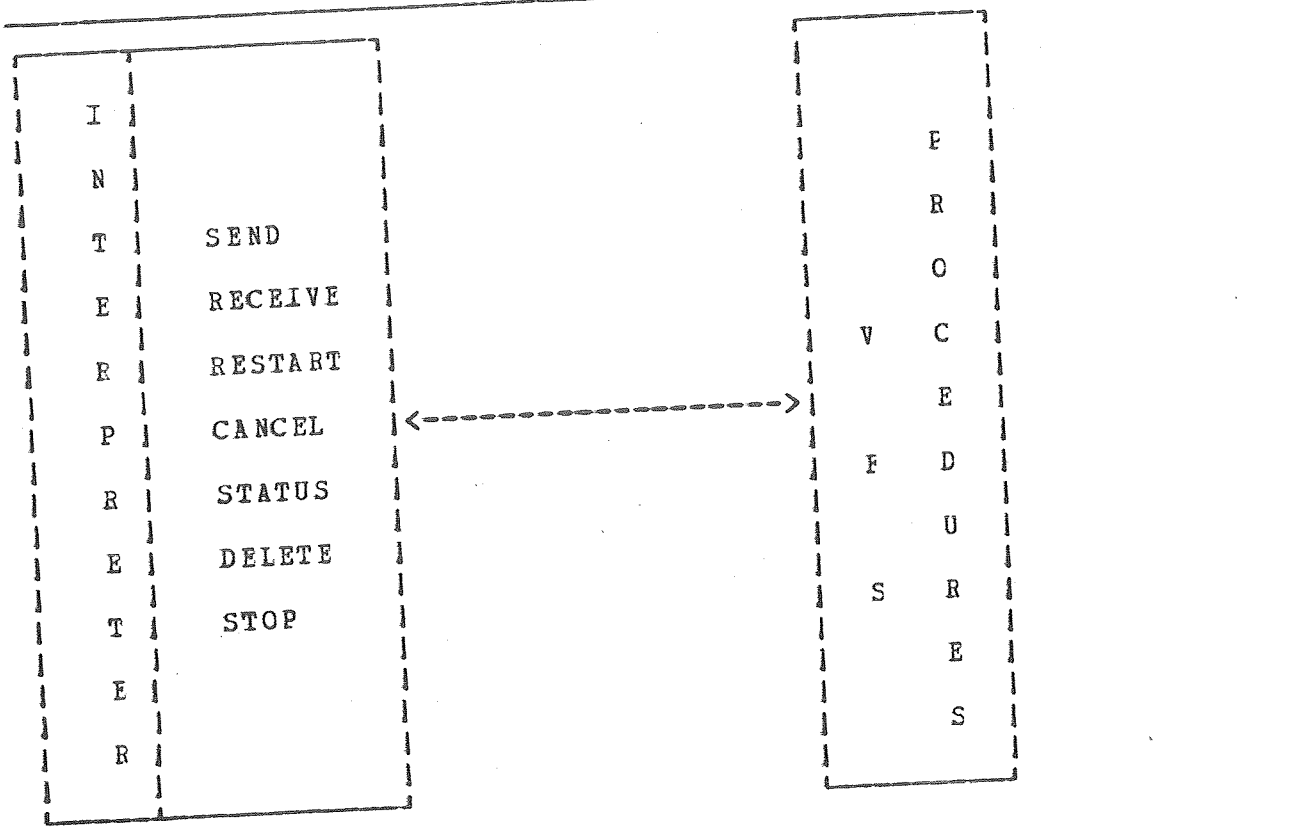

Fig. 32. File Transfer Primary

2.2.1 Primary main program

The main module, called **INTERPRETER** in Fig. 32, executes the following set of operations:

- analyze the command line coming from the user interface, containing the requested operation and parameters (see "1.4 FTF user interface" pag. 55).
  - get the operation code
  - assume default options
  - get the options
  - get the host name
  - if the required service is SEND or RECEIVE
    - then get local and remote file name
  - if the requested service is RESTART, CANCEL, STOP or STATUS

— get the transfer identifier

- if the requested service is DELETE then

  — get the remote file name to be deleted.

- For all requested types, check the supplied parameters

- If there is an error then returns a diagnostic and EXIT.

- If operation is RESTART then

  - search the disk CONTEXT identified by the TID:

    — If it does not exist, return a diagnostic and EXIT

    — else return a diagnostic indicating operation locally accepted and continue

- If operation is SEND or RECEIVE then

  - initiate all global variables of the task

  - generate the TID value for the request [16]

  - create a memory and disk CONTEXT (identified by that TID) and set the transfer status with a value indicating local acceptance.

- if any error occurs in the previous phases then

  - return a diagnostic and EXIT

- else

  - return a diagnostic indicating local acceptance and CONTINUE

- call the procedure corresponding to the user request (SEND, RECEIVE, RESTART, STATUS, STOP or DELETE) and receive the results of the invoked procedure.

- if the operation was STATUS or DELETE then

  - return the result tc the user.

- else

---

[16] The TID value has to be unique in the system among all the TIDs in the CONTEXT TABLE. A simple solution is to use the TID as the numeric key for a record in CONTEXT TABLE and increase the last key to generate a new TID. This solution may require further considerations in case of indipendent copies of FTF sharing the same CONTEXT TABLE.

- write the result in the related disk CONTEXT, update the TRANSFER STATUS field, fill in the DATE OF TERMINATION field and EXIT

- if the operation was STOP

  - the result is both communicated to the user and stored in the disk CONTEXT.

2.2.2 FTF Procedures

Note: All operations on the local system require to be interfaced with the local file-system.

2.2.2.1 SEND procedure

The SEND procedure issues the VFS operations necessary to send a file.

- Fill the disk and memory CONTEXT with all transfer informations (see "3.3 Primary CONTEXT list and description fields" pag. 98), except the 'transferred record numbers', to be computed later on.

- Try to open the local file

- If an abnormal condition occurs (for example, the local file does not exist) then

  - return a diagnostic with the reason of failure and EXIT

  - set the condition to "not restartable"

- Establish a connection with the remote: call the VFS procedure SELECT PHOTOCOL (see "2.2.3.1 SP Primary procedure" pag. 79).

- Set parameters to act on the remote file, according to the destination effect specified by the user by disk CONTEXT in case of RESTART.

- If destination effect is MAKE then

  - Set the parameter CLASH-OPTION to KEEP

- If destination effect is REPLACE then

  - Set the parameter CLASH-OPTION to REPLACE

- If destination effect is APPEND then

  - Set the parameter CLASH-OPTION to REJECT.

- map file attributes from real to virtual format

- If destination effect is APPEND then

  - select the remote file by calling the VFS procedure SELECT FILE (see "2.2.3.2 SL Primary procedure" pag. 80).

  - else try to create the remote file by calling the VFS procedure CREATE FILE (see "2.2.3.3 CR Primary procedure" pag. 80).

- prepare parameters to continue the action on the remote file: set the parameter PROCESSING-MODE to LOAD or APPEND

- try to open the remote file: call the VFS procedure OPEN FILE (see "2.2.3.5 OP Primary procedure" pag. 82). The restart position is specified only if the SEND procedure has been invoked by a RESTART command. In this case, the position is negotiated with the Secondary.

- set the TRANSFER STATUS in the memory and disk CONTEXT with the value RUNNING.

- begin the data transfer: call the VFS procedure BEGIN TRANSFER (see "2.2.3.6 BT Primary procedure" pag. 83).

- get records from local file and send a record a time until end of file is reached: call the VFS procedure DATA (see "2.2.3.7 DATA Primary procedure" pag. 84).

- start the transfer from the record indicated in the CONTEXT after OPEN procedure, and increase a record counter. At the end of the data transfer, this value will be stored in the field 'transferred record numbers' of the memory and disk CONTEXT.

- send end of transfer: call the VFS procedure END TRANSFER (see "2.2.3.8 ET Primary procedure" pag. 84).

- close the remote file: call the VFS procedure CLOSE FILE (see "2.2.3.9 CL Primary procedure" pag. 84).

- release the remote file: call the VFS procedure RELEASE FILE (see "2.2.3.11 RL Primary procedure" pag. 85).

- close the local file

- release the protocol connection: call the VFS procedure RELEASE PROTOCOL (see "2.2.3.12 RP Primary procedure" pag. 86).

- set the TRANSFER STATUS field in the disk CONTEXT to SUCCESSFUL TERMINATION.

- return a diagnostic indicating successful termination.

If during the execution of this procedure, an error of type 1 or 2 ("1.5 Types of interruptions" pag. 59) occurs, then:

- disconnect with a suitable reason code: call the VFS procedure DISCONNECT PROTOCOL (see "2.2.3.13 DP Primary procedure" pag. 86)

- set TRANSFER STATUS field in the disk CONTEXT to NOT RESTARTABLE

- set the condition to not restartable

- return diagnostic and EXIT

If an error of type 3 occurs, then

- disconnect with a suitable reason code

- set TRANSFER STATUS field in the disk CONTEXT to RESTARTABLE

- return diagnostic and EXIT

- set the condition to RESTARTABLE

2.2.2.2 RECEIVE procedure

The RECEIVE procedure issues the VFS operations necessary to receive a file.

- try the local file creation:
  - if the operation is successful completed then a 'new file' has been created.
  - otherwise, if the file already exists then check the destination effect:
    - destination effect = MAKE ====> return a diagnostic with the reason of failure and EXIT setting a not restartable condition.
    - destination effect = REPLACE ====> open the existing local file.
    - destination effect = APPEND =====> open the existing local file and point to the bottom.
- establish a connection with the remote: call the VFS procedure SELECT PROTOCOL (see "2.2.3.1 SP Primary procedure" pag. 79).
- select the remote file: call the VFS procedure SELECT FILE (see "2.2.3.2 SL Primary procedure" pag. 80).
- read the remote file attributes: call the VFS procedure READ ATTRIBUTES (see "2.2.3.4 RA Primary procedure" pag. 81).
- prepare file attributes and parameters to act on the local file: map attributes from virtual format to real one.
- set the parameter PROCESSING-MODE to READ.
- open the remote file: call the VFS procedure OPEN FILE (see "2.2.3.5 OP Primary procedure" pag. 82).
- If the RECEIVE procedure has been called by a RESTART command, the CONTEXT for the requested file already existed. In this case, the restart position is specified in the OPEN request, and the data transfer starts form that position. Otherwise, the data transfer starts from the top of the file.
- set the TRANSFER STATUS field in the memory and disk CONTEXT to RUNNING.
- begin the data transfer: calls the VFS procedure BEGIN TRANSFER (see "2.2.3.6 BT Primary procedure" pag. 83).
- receive data until end of transfer occurs: calls the procedure REC DATA, and copy data in the local file (see "2.2.3.14 REC DATA Primary procedure" pag. 86).
- at the end of transfer, close the local file.

- close the remote file: call the VFS procedure CLOSE FILE (see "2.2.3.9 CL Primary procedure" pag. 84).

- release the remote file: call the VFS procedure RELEASE FILE (see "2.2.3.11 RL Primary procedure" pag. 85)

- release the protocol connection: call the VFS procedure RELEASE PROTOCOL (see "2.2.3.12 RP Primary procedure" pag. 86).

- set the TRANSFER STATUS field in the disk CONTEXT with SUCCESSFUL TERMINATION.

- return a diagnostic indicating successful termination.

If during the execution an abnormal condition occurs, the same error handling as in the SEND procedure takes place.

2.2.2.3 RESTART procedure

The RESTART procedure handles the restart operation for an interrupted and restartable file transfer and may be issued only by the same user who initiated the file transfer.

- get disk CONTEXT contents and restore all variables necessary to restart the operation

- check the operation that has been interrupted:

  - If operation was SEND then call SEND procedure

  - If operation was RECEIVE then call RECEIVE procedure

- return the completion code of the requested operation.

2.2.2.4 STOP procedure

The STOP procedure is invoked on user's request to immediately abort a File Transfer.

- set the TRANSFER STATUS to NOT RESTARTABLE

- send a Disconnect protocol request by calling the VFS procedure DISCONNECT PROTOCOL (see "2.2.3.13 DP Primary procedure" pag. 86).

- return a diagnostic. This may indicate success or failure, which may only happen when the File Transfer has already been completed or was never invoked.

2.2.2.5 STATUS procedure

The STATUS procedure provides information on a File Transfer, by using the disk or memory CONTEXT.

- If the File Transfer is running, information is obtained from the memory CONTEXT.

- If the File Transfer is suspended, or terminated but the disk CONTEXT still exists, information is obtained from the disk CONTEXT.

- If no CONTEXT is found, a diagnostic is returned.

The actual information which is passed as return from the STATUS procedure is implementation dependent.

2.2.2.6 CANCEL procedure

The CANCEL procedure is used to delete all information related to a previously interrupted File Transfer. The procedure tries to re-establish the conditions as before the transfer was initiated.

- get the disk CONTEXT contents:

- if the operation was SEND and destination effect = MAKE then

  - establish a connection with the remote

  - select and delete the remote file whose name is specified in the disk CONTEXT, by calling the VFS procedure SELECT and DELETE file.

  - release the connection with the remote

- if the operation was RECEIVE and destination effect = MAKE then erase the file local to Primary whose name is specified in the disk CONTEXT.

- In any case (both for SEND and RECEIVE) **erase the disk CONTEXT**

- return a diagnostic

Note: In case of APPEND or REPLACE the previous conditions on the destination file cannot be re-established.

2.2.2.7 DELETE procedure

The DELETE procedure is a File Management service, which is used to permanently erase a remote file.

- establish a connection with the remote: call the VFS procedure SELECT PROTOCOL (see "2.2.3.1 SP Primary procedure" pag. 79).

- select the remote file to be deleted: call the VFS procedure SELECT FILE.

- delete the remote file as it is required: call the VFS procedure DELETE FILE

- releases the connection with the remote: calls the VFS procedure RELEASE PROTOCOL (see "2.2.3.12 RP Primary procedure" pag. 86).

- return a diagnostic indicating the successful deletion of the file

If during the execution an abnormal condition occurs, the same error handling as described in the SEND primitive is performed.

2.2.3 VFS Primary Procedures

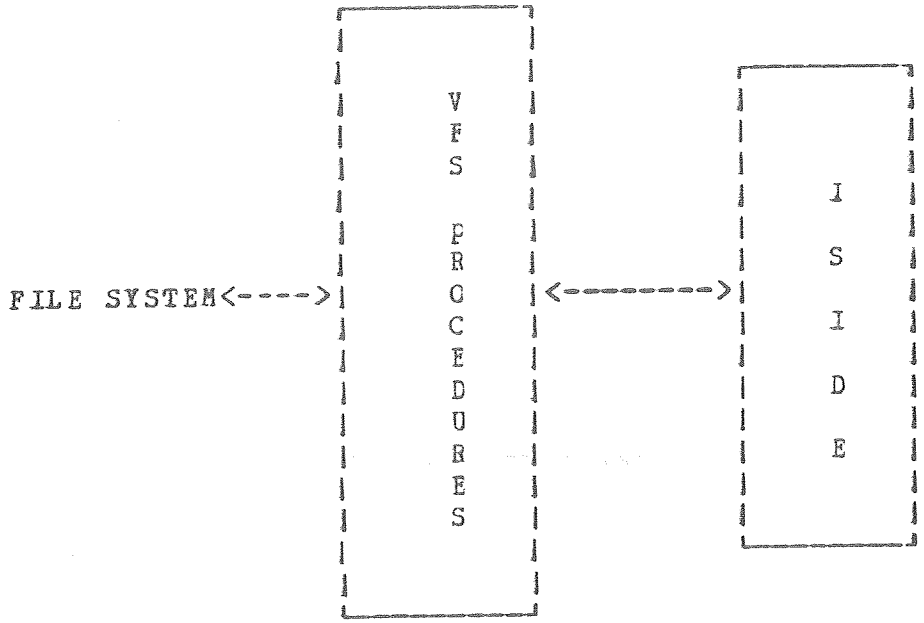

Fig. 33. VFS Primary

The VFS Primary implements the protocol aspects of the FTF, as it interfaces the real file system and the OSIRIDE environment to provide the mapping function function from/to real format of attributes and operation to/from virtual one, and, through the ISIDE services, to send protocol messages. The VFS procedures functions are globally the following:

- initiate a connection with the remote

- establish an existing file as current file

- create and establish a new file as current file

- retrieve specific attributes of the current file

- initiate processing of the contents of the current file
- cause transition to file data transfer level
- transfer file data and/or a delimiter
- specify termination of the transfer without loss of data
- terminate processing of the current file
- delete and release the current file
- release the current file
- specify that an abnormal termination of the transfer has occurred, with possible destruction of data in transit
- request normal termination of the connection
- request abnormal termination of the connection

Each VFS procedure is able to provide one of these functions. The calling sequence of the VFS procedures must follow the rules set by the protocol enclosures, which are the following:

```
SELECT PROTOCOL = SP
        SELECT FILE = SL
        CREATE FILE = CR
        READ ATTRIBUTES = RA
                OPEN FILE = OP
                        BEGIN DATA TRANSFER = BT
                        DATA TRANSFER = DATA
                        ABORT DATA TRANSFER = AT
                        END DATA TRANSFER = ET
                CLOSE FILE = CL
                DELETE FILE = DL
        RELEASE FILE = RL
RELEASE PROTOCOL = RP
DISCONNECT PROTOCOL = DP
```

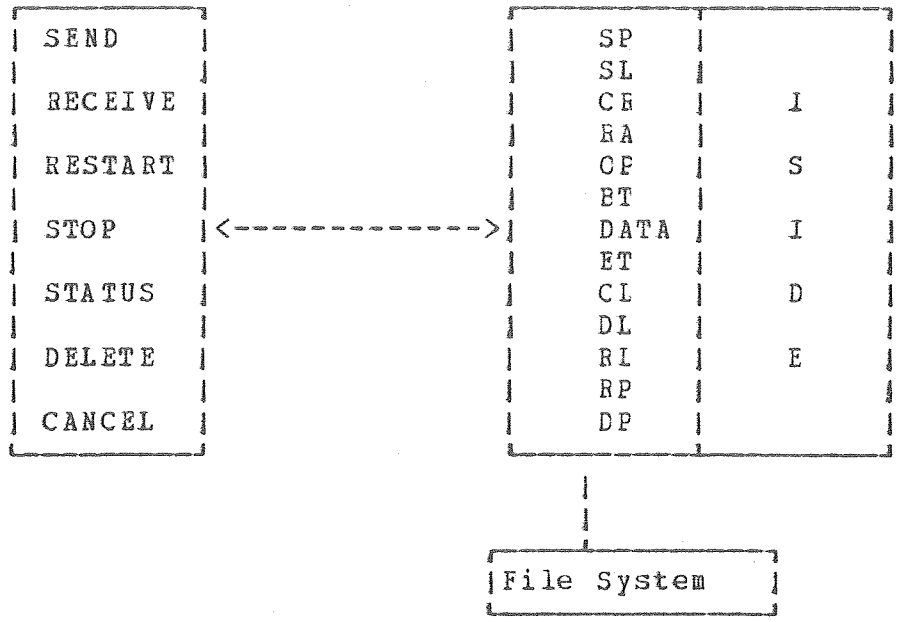

Fig. 34. File Transfer Primary

In the following, the VFS procedures are named with their acronyms as in Fig. 34.

The parameters' encoding is explained in "3.1.2.4 Parameter encoding" pag. 27.

2.2.3.1 SP Primary procedure

The SP procedure establishes a network connection with the remote.

- send select protocol request:
  - codify user data from real format to virtual one
  - use the following ISIDE services to send request to the Secondary:
    - OPEN ("4.1 OPEN primitive" pag. 100)
    - CONNECT ("4.2 CONNECT primitive" pag. 101)
- receive Select Protocol Response from the remote
- decode parameters from virtual format to real one and prepare the variables necessary to access the input buffer[18].

[17] We call always 'user data' the 'data' to be transferred, either a transparent data or a real message.

parameters filestore-name
authentication
protocol-identifier
protocol-version
class-of-filestore
class-of-service

transition protocol state dormant ====> no file (see "2.3.2 Select Protocol request (SP)" of "First Part").

2.2.3.2 SL Primary procedure

The SL procedure establishes a file as the current one

- send Select File request:
- codify user data from real format to virtual one [17]
- use the following ISIDE service to send request to Secondary
  - DATA (see "4.5 DATA primitive" pag. 104)
- receive Select File Response by using the following ISIDE service:
  - RECEIVE (see "4.3 RECEIVE primitive" pag. 103)
- decode parameters from virtual format to real one and prepare the variables necessary to access the input buffer[18].

parameters file-name
file-passwords

transition protocol state no file ====> file selected (see "2.3.7 Select File request (SL)" of "First Part").

2.2.3.3 CR Primary procedure

The CR procedure creates a remote file.

- send Create File request:
  - codify user data from real format to virtual one [17]
  - use the following ISIDE service to send request to Secondary
    - DATA (see "4.5 DATA primitive" pag. 104)
- receive Create File response:

---

[18] Input buffer = set of informations permitting the access to the received message.

- receive a message from the remote through the following ISIDE service:

  — RECEIVE (see "4.3 RECEIVE primitive" pag. 103)

- decode parameters from virtual format to real one and prepare the variables necessary to access the input buffer[18].

parameters
file-name
file-passwords
file-attributes
clash-option
reversible-mapping

transition protocol state: no file ====> file selected (see "2.3.11 Create file request (CR)" of "First Part").

Short description of the most significant parameters:

Clash-option specifies what to do if the supplied file-name corresponds to an already existing file.

- **REJECT** the existing file is kept
- **KEEP** the existing file is kept and selected
- **REPLACE** the existing file is replaced by the newly defined file

In OSIRIDE, the parameter value depends on the '**destination effect**' specified by the user:

effect = MAKE ====> CLASH-OPTION = KEEP
effect = REPLACE ====> CLASH-OPTION = REPLACE
effect = APPEND ====> CLASH-OPTION = REJECT

2.2.3.4 RA Primary procedure

The RA prodedure reads the requested attributes of the current file:

- send Read Attributes request:
  - codify user data from real format to virtual one [17]
  - use the following ISIDE service to send request to Secondary:

    — DATA (see "4.5 DATA primitive" pag. 104)

  - receive the Read Attribute response from the remote through the following ISIDE service:

    — RECEIVE (see"4.3 RECEIVE primitive" pag. 103)

- decode the parameters from virtual format to real one and prepare the variables necessary to access the input buffer[18].

parameters requested-attributes

transition protocol state: file selected ====> file selected (see "2.3.15 Read attributes request (RA)" of "First Part").

2.2.3.5 OP Primary procedure

The OP procedure opens the current file.

- send Open File request:
  - codify user data from real format to virtual one [17]
  - use the following ISIDE service to send request to Secondary:
    - DATA (see "4.5 DATA primitive" pag. 104)
  - receive the Open File response from the remote through the following ISIDE service:
    - RECEIVE (see"4.3 RECEIVE primitive" pag. 103)
  - decode the parameters from virtual format to real one and prepare the variables necessary to access the input buffer[18].

parameters
access-mode
processing-mode
lock
failure-option
open-identification
restart-position

transition protocol state: file selected ====> file open (see "2.3.17 Open File request (OP)" of "First Part").

Short description of the most significant parameters:

processing-mode specifies the intended processing of the file contents.

READ reading of the file contents

LOAD writing into the file from the beginning (overwriting any previous contents)

APPEND writing into the file after its current end (preserving any previous contents)

In OSIRIDE, the parameter value depends on the user requested services:

service = **SEND** ====> PROCESSING-MODE = **LOAD** or **APPEND** [19]
service = **RECEIVE** ====> PROCESSING-MODE = **READ**

open-identification uniquely identifies within the Primary the current file-opening. The same identification must be supplied in case of reopening for restart of an interrupted data transfer. In OSIRIDE the parameter contains the TID value.

restart-position specifies that this open is for restart and negotiates the position where to resume data transfer.

2.2.3.6 BT Primary procedure

The BT procedure alerts the Secondary that the data transfer phase is beginning.

- send Begin Transfer request:
  - codify user data from real format to virtual one [17]
  - use the following ISIDE services to send the request to Secondary:
    - DATA (see "4.5 DATA primitive" pag. 104)
    - SYNC (see "4.7 SYNCHRONIZE primitive" pag. 105)
    - if the operation is RECEIVE then a CONTROL primitive (see "4.6 CONTROL primitive" pag. 104) is also issued, to give the DATA & SYNC tokens.
- receive Begin Transfer response:
  - receive the Begin Transfer response from the remote through the following ISIDE service:
    - RECEIVE (see "4.3 RECEIVE primitive" pag. 103)
  - decode the parameters from virtual format to real one and prepare the variables necessary to access the input buffer[18].

parameters special-information

transition protocol state: file open ====> data (see "2.3.21 Begin Transfer request (BT)" of "First Part").

---

19 Depending on the user's specified parameter in the SEND command.

2.2.3.7 DATA Primary procedure

The DATA procedure is used to send file data.

- codify user data from real format to virtual one [17]
- use the following ISIDE service to transfer data to the Secondary:
  - DATA (see "4.5 DATA primitive" pag. 104)

parameters file-data

transition protocol state: data ====> data (see "2.3.23 Data request (DATA)" of "First Part").

2.2.3.8 ET Primary procedure

The ET procedure alerts the Receiver that the data transfer phase is finished.

- send End Data transfer request:
  - codify user data from real format to virtual one [17]
  - use the following ISIDE services to send request to Secondary
    - DATA (see "4.5 DATA primitive" pag. 104)
    - If the Sender is the Secondary, then the SYNC and CONTROL ISIDE primitive is used, otherwise the SYNC primitive is used (no tokens are GIVEn from the Primary to the Secondary except at the beginning of data transfer phase if necessary).
- receive End Data transfer response:
  - receive message from the remote through the following ISIDE service:
    - RECEIVE (see "4.3 RECEIVE primitive" pag. 103)

transition protocol state: data ====> file open (see "2.3.24 End Transfer request (ET)" of "First Part").

2.2.3.9 CL Primary procedure

The CL procedure closes the current file.

- send Close File request:
  - codify user data from the real to the virtual format [17]

- use the following ISIDE service to send request to the Secondary:

  — DATA (see "4.5 DATA primitive" pag. 104)

- receive the Close File response from the remote through the following ISIDE service:

  - RECEIVE (see "4.3 RECEIVE primitive" pag. 103)

transition protocol state: file open ====> file selected (see "2.3.19 Close File request (CL)" of "First Part").

2.2.3.10 DL Primary procedure

The DL procedure is used to delete the current file

- send Delete File request:

  - codify user data from real format to virtual one [17]

  - use the following ISIDE service to send the request to the Secondary:

    — DATA (see "4.5 DATA primitive" pag. 104)

- receive the Delete File response from the remote through the following ISIDE service:

  - RECEIVE (see hdref refid=isrec.)

transition protocol state: file selected ====> no file (see "2.3.13 Delete File request (DL)" of "First Part").

2.2.3.11 RL Primary procedure

The RL procedure is used to release current file.

- send Release File request:

  - codify user data from the real to the virtual format: [17]

  - use the following ISIDE service to send request to Secondary:

    — DATA (see "4.5 DATA primitive" pag. 104)

- receive the Release File response from the remote through the following ISIDE service:

  - RECEIVE (see "4.3 RECEIVE primitive" pag. 103)

transition protocol state: file selected ====> no file (see "2.3.9 Release File request (RL)" of "First Part").

2.2.3.12 RP Primary procedure

The RP procedure is used to terminate a connection.

- send a Release Protocol request:
  - codify user data from the real to the virtual format: [17]
  - use the following ISIDE service to send the request to the Secondary:
    - FINISH (see "4.10 FINISH primitive" pag. 106)
- receive the Release Protocol response from the remote.

transition protocol state: no file ====> dormant (see "2.3.4 Release Protocol request (RP)" of "First Part").

2.2.3.13 DP Primary procedure

The DP procedure releases a connection in an abnormal manner.

- send Disconnect Protocol request:
  - codifies user data from the real to the virtual format: [17]
  - use the following ISIDE service to send request to Secondary:
    - FINISH (see "4.10 FINISH primitive" pag. 106)
    - The Abort mode of FINISH is selected.

transition protocol state: any state ====> dormant (see "2.3.6 Disconnect Protocol request (DP)" of "First Part").

2.2.3.14 REC DATA Primary procedure

Receives data or advice of End Transfer from the Sender. If an advice of End Transfer is received from the Secondary then an End Transfer response is sent by using the following ISIDE service:

- RESPONSE (see "4.9 RESPONSE primitive" pag. 106)

parameters rec-data-diagnostic

output parameter rec-data-diagnostic

transition protocol state: data =====> data

Note: This procedure is not included in the ECMA-85 VFP, but is a local procedure used when the Primary plays the role of the 'Receiver' in the File Transfer.

2.3 SECONDARY STRUCTURE

The File Transfer Secondary is local to the file where the Primary action (reading, writing or appending) is addressed. This task runs independently of the Primary. As already described, the Secondary does not communicate with the user, but interfaces its local file-system, and executes all Primary commands, sending or receiving file data. The Secondary function is performed by the VFS Secondary procedures and its structure, which will be explained in the following sections, is composed essentially by a 'main' module which invokes the VFS procedures and by the bodies of these procedures. The Secondary may implement an operator interface, which is not described in this document because it is local operating system dependent. This operator interface may be useful whenever the behaviour of the Secondary has to be subject to operating needs. For instance, in many cases it is useful that the operator has the capability of stopping the functioning of the secondary in case of congestion. Similar considerations are also valid for the Primary, with the difference that the Primary already has a command interface.

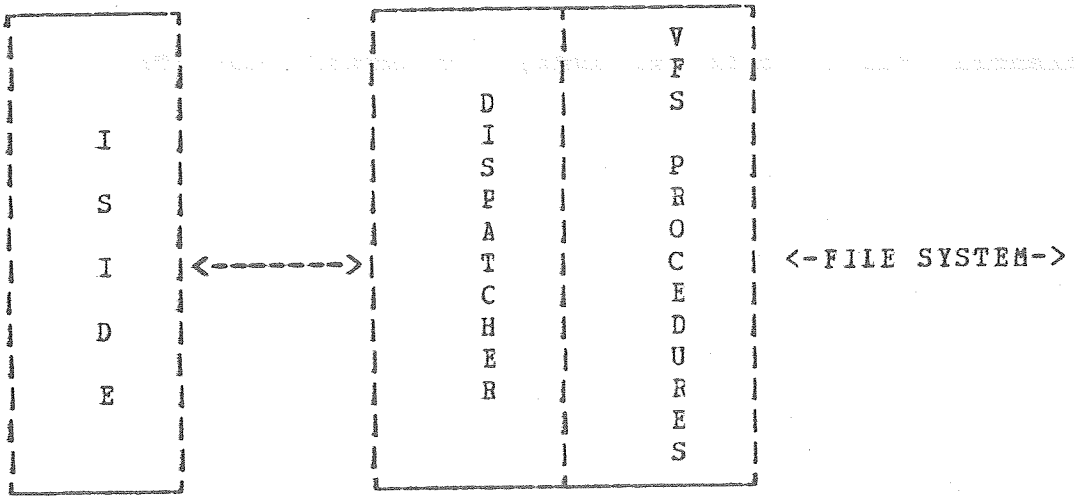

Fig. 35. File Transfer Secondary

2.3.1 Secondary main program

The main module, called DISPATCHER in Fig. 35 pag. 87, executes the following set of operations:

- receive messages from the Primary, through ISIDE services
- check the received messages and call the appropriate VFS procedures:
- if the received message is:
  - a **CONNECT** indication then call the VFS procedure **SELECT PROTOCOL** (see "2.3.2.1 SP Secondary procedure" pag. 90).
  - a **SELECT FILE** request then call the VFS procedure **SELECT FILE** ("2.3.2.2 SL Secondary procedure" pag. 90)
  - a **CREATE FILE** request then call the VFS procedure **CREATE FILE** ("2.3.2.3 CR Secondary procedure" pag. 91)
  - a **READ ATTRIBUTES** request then call the VFS procedure **READ ATTRIBUTES** ("2.3.2.4 RA Secondary procedure" pag. 92)
  - a **OPEN FILE** request then call the VFS procedure **OPEN FILE** ("2.3.2.5 OP Secondary procedure" pag. 92)
  - a **BEGIN TRANSFER** request then call the VFS procedure **BEGIN TRANSFER** ("2.3.2.6 BT Secondary procedure" pag. 93)
  - a **END TRANSFER** request then call the VFS procedure **END TRANSFER** ("2.3.2.8 ET Secondary procedure" pag. 94)
  - a **CLOSE FILE** request then call the VFS procedure **CLOSE FILE** ("2.3.2.9 CL Secondary procedure" pag. 95)
  - a **DELETE FILE** request then call the VFS procedure **DELETE FILE** ("2.3.2.10 DL Secondary procedure" pag. 95)
  - a **RELEASE FILE** request then call the VFS procedure **RELEASE FILE** ("2.3.2.11 RL Secondary procedure" pag. 95)
  - a **RELEASE PROTOCOL** request then call the VFS procedure **RELEASE PROTOCOL** ("2.3.2.12 RP Secondary procedure" pag. 96) for a normal connection termination
  - a **FINISH** request then
    - if the received diagnostic indicates a NOT RESTARTABLE error (type 1 and 2 - see "1.5 Types of interruptions" pag. 59) then re-establish the previous conditions, i.e. delete the received file, if any, and only in the case of MAKE

— if the received diagnostic indicates a RESTARTABLE error (type 3 - see "1.5 Types of interruptions" pag. 59) then the involved files are kept as they are.

— if the received message is different from those above listed messages, then an error occurs: in this case the error is catalogued as a 'protocol error' and causes a NOT RESTARTABLE condition (see "1.5 Types of interruptions" pag. 59): the Secondary uses a ISIDE service to disconnect itself from the Primary.

2.3.2 VFS Secondary Procedures

The notation used for the VFS Secondary procedures is the same as that introduced in "2.2.3 VFS Primary Procedures" pag. 77 for the VFS Primary.

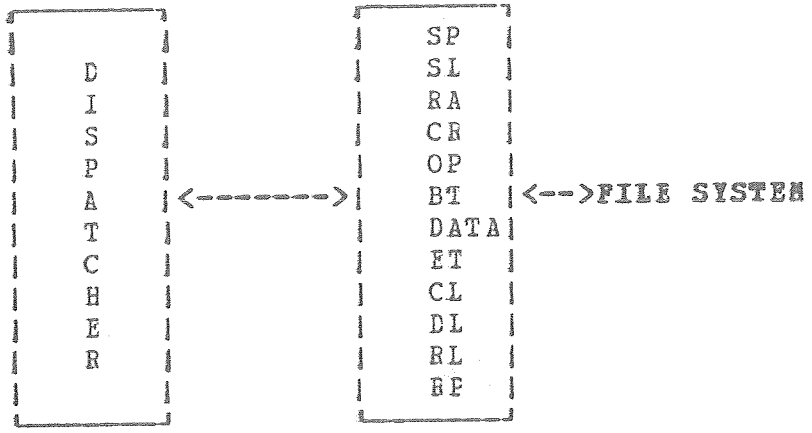

Fig. 36. File Transfer Secondary Procedures

The VFS procedure functions are globally the following:

- response to Select Protocol request
- response to Select File request
- response to Create File request
- response to Read Attributes request
- response to Open File request
- response to Begin Data Transfer request
- Send Data to Primary
- response to End data Transfer request

- response to Close File request
- response to Delete File request
- response to Release File request
- response to Release Protocol request

2.3.2.1 SP Secondary procedure

The Select Protocol procedure is used to accept or reject an incoming connection indication.

- control the received parameters if they are consistent with FTF local version
- send Select Protocol response by using the following ISIDE service:
  - ACCEPT (see "4.8 ACCEPT primitive" pag. 105)

  The **result** parameter of ACCEPT service is set to **yes** if the connection request is to be accepted and to **no** otherwise.

parameters: protocol-version
class-of-filestore
class-of-service
select-protocol-diagnostic

transition protocol state: dormant =====> no file

2.3.2.2 SL Secondary procedure

The Select File procedure opens an existing file, local to the Secondary task.

- decode the input parameters from TLV format to real one
- interface the local file-system
- try to open the local file whose name is specified in the input parameter FILE-NAME
- if the file is not found then an error occurs: the result is set to "no restartable" value
- if the open operation terminates correctly the SL procedure sends a Select File response:
  - map the attributes from real to virtual format
  - send response by using the following ISIDE service:
    - DATA (see "4.5 DATA primitive" pag. 104)

parameters: select-file-diagnostic

transition protocol state: no file =====> file selected

2.3.2.3 CR Secondary procedure

The Create File procedure is used to build a file, local to the Secondary task.

- decode the input parameters from TLV to real format
- interface the local file-system by mapping the virtual attributes on the real format
- check the requested destination effect, which, at this level, is expressed as a value of input parameter 'CLASH-OPTION'
- tries to create the local file with the name specified in the input parameter FILE-NAME:
- if the operation is successful completed, the operation can proceed only if the Destination Effect is MAKE and the CLASH-OPTION is KEEP.
- If the file already exists, then the operation can proceed only if:

    A. the Destination Effect is REPLACE and the CLASH-OPTION is REPLACE.

    or

    B. the Destination Effect is APPEND and the CLASH-OPTION is REJECT.

- Any other condition interrupts the process in a not restartable way.
- if the file creation is successful, the CR procedure sends Create File response:
    - map output parameters from real to virtual format
    - send the response by using the following ISIDE service:
        - DATA (see "4.5 DATA primitive" pag. 104)

parameters: create-file-diagnostic

transition protocol state: no file =====> file selected

2.3.2.4 RA Secondary procedure

The Read Attributes procedure reads the requested attributes of the current file, local to the Secondary task.

- decode the input parameters from TLV to real format
- send Read Attributes response:
  - map output parameters from real to virtual format
  - send response by using the following ISIDE service:
    - DATA (see "4.5 DATA primitive" pag. 104)

parameters: file-attributes
read-attributes-diagnostic

transition protocol state: file selected =====> file selected

2.3.2.5 OP Secondary procedure

The Open File procedure opens the current file, local to the Secondary task.

- decode input parameters from TLV to real format
- check the operation to be performed, which, at this level, is expressed as a value of the input parameter 'PROCESSING-MODE':
  - if the operation is SEND and PROCESSING-MODE is LOAD and transfer status is NO ACTIVE, then go to the beginning of the file.
  - if the operation is SEND and PROCESSING-MODE is APPEND and transfer status is NO ACTIVE, then go to the bottom of the file
  - if the operation is SEND and PROCESSING-MODE is LOAD or APPEND and transfer status is RESTARTABLE, then
    - open the file whose name has been received from the Primary and on which the transfer is to begin
    - go at the interruption point (received by the Primary in the input parameter RESTART-POSITION) and continues the transfer
  - if the operation is RECEIVE and PROCESSING-MODE is READ and transfer status is NO ACTIVE, then point to the appropriate position in the file to read its records
  - if the operation is RECEIVE and PROCESSING-MODE is READ and transfer status = RESTARTABLE, then continues the Receive operation from the interruption point

- after completion of the above mentioned operations, the OP procedure sends Open File response:
  - map output parameters from real to virtual format
  - send response by using the following ISIDE service:
    - DATA (see "4.5 DATA primitive" pag. 104)

parameters: restart-position
open-file-diagnostic

transition protocol state: file selected =====> file open

2.3.2.6 BT Secondary procedure

The begin transfer procedure sends data to Primary until end of file or an interruption occurs.

- decode input parameters from TLV to real format
- send Begin Data transfer response:
  - map output parameters from real to virtual format (TLV)
  - send response by using the following ISIDE service:
    - DATA (see "4.5 DATA primitive" pag. 104)
      RESPONSE (see "4.9 RESPONSE primitive" pag. 106)
- check the operation to be performed:
  - if the operation is RECEIVE
    - the Secondary is the 'sender' in the transfer
    - Until end of file or an error or a voluntary interruption from the Primary occurs repeat:
      - interface the local file-system by reading a record a time in the file
      - map output parameters from real to virtual format
      - send data to Primary by using the following ISIDE service:
        - DATA (see "4.5 DATA primitive" pag. 104)
    - if the transfer has been successfully completed send 'end data transfer' request, i.e. invoke the ET procedure ("2.3.2.8 ET Secondary procedure" pag. 94)
  - if the operation is SEND:
    - the Secondary is the 'receiver' in the transfer

— send a response by using the following ISIDE services:

- RESPONSE (see "4.9 RESPONSE primitive" pag. 106) the **command** parameter of the RESPONSE operation is set to "SYNCHRONIZE"

parameters: begin-data-diagnostic

transition protocol state: file open =====> data

2.3.2.7 DATA Secondary procedure

The DATA procedure is used to transfer file data

- decode input parameters from TLV format to real and returns a diagnostic
- interface the local file-system:
  - write into the current file
  - update all elements used to recognize the data location in the file.

parameters: file-data

transition protocol state: data =====> data

2.3.2.8 ET Secondary procedure

The End Transfer procedure specifies the termination of the transfer without loss of data.

- decode input parameters from TLV to real format and return a diagnostic
- send End Data transfer response:
  - map output parameters from real to virtual format
  - send response by using the following ISIDE services:
    - if the Secondary is Sender:
      - DATA (see "4.5 DATA primitive" pag. 104)
      - RESPONSE (see "4.9 RESPONSE primitive" pag. 106)
      - CONTROL (see "4.6 CONTROL primitive" pag. 104)
    - if the Secondary is Receiver:
      - DATA (see "4.5 DATA primitive" pag. 104)

parameters: none

transition protocol state: data =====> file open

2.3.2.9 CL Secondary procedure

The Close File procedure closes a file enclosure.

- decode input parameters from TLV to real format
- send Close File response:
  - map output parameters from real to virtual format (TLV)
  - send response by using the following ISIDE service:
    - DATA (see "4.5 DATA primitive" pag. 104)

parameters: special-information
close-file-diagnostic

transition protocol state: file open =====> file selected

2.3.2.10 DL Secondary procedure

The Delete File procedure erases the current file.

- decode input parameters from TLV to real format
- erase the current file
- send Delete File response:
  - map output parameters from real to virtual format
  - send the response by using the following ISIDE service:
    - DATA (see "4.5 DATA primitive" pag. 104)

parameters delete-file-diagnostic

transition protocol state: file selected =====> no file

2.3.2.11 RL Secondary procedure

The Release File procedure releases current file.

- decode input parameters from TLV to real format
- close the current file
- send Release File response:

- map output parameters from real to virtual format
- send the response by using the following ISIDE service:
  - DATA (see "4.5 DATA primitive" pag. 104)

parameters: release-file-diagnostic

transition protocol state: file selected =====> no file

2.3.2.12 RP Secondary procedure

The Release Protocol procedure is used for a normal termination of a connection.

- send release protocol response:
  - map output parameters from real to virtual format (TLV)
  - send the response by using the following ISIDE service:
    - RESPONSE (see "4.9 RESPONSE primitive" pag. 106). The **command** parameter of RESPONSE primitive is set to **FINISH**. The FT connection is thus released.

parameters: none

transition protocol state: no file =====> dormant

3.0 DATA STRUCTURES

This section describes the basic data structures of FTF implementation. The most important structures which are included in the FTF implementation are:

- information permitting the recognition in an unambiguous way of the received message and of the message to be sent.
  - input message identifiers set
  - output message identifiers set
- information about the transfer development
  - CONTEXT

Note: As far as the CONTEXT is concerned, this is supposedly located in the Primary. A similar data structure may be developped also for the Secondary. In this latter case, the relevant information for the Secondary is:

- identifier of that specific transfer (TID), as it is received from the Primary
- transfer status received from the Primary
- remote file name received from the Primary
- date and time when the transfer was started
- date and time when the transfer was interrupted or terminated
- number of transferred records at the moment of interruption-time.

The current implementation does not foresee the Secondary CONTEXT use, so this note is only a proposal for further developments.

3.1 INPUT MESSAGE IDENTIFIERS SET

Any legal input message is made of:

- input message code
- input message length: integer value, whose limit is established by the ISIDE services used in conjunction with that message.
- pointer to the input message fields in the record containing the message.

- input message offset: heading which can contain some further information about the message type.

3.2 OUTPUT MESSAGE IDENTIFIERS SET

Any output message is made of:

- output message code
- output message length: integer value, whose limit is established by the ISIDE services used in conjunction with that message.
- pointer to the output message fields in the record containing the message
- output message offset: heading which can contain some further information about the message type.

3.3 PRIMARY CONTEXT LIST AND DESCRIPTION FIELDS

The Primary CONTEXT contains the following information:

identifier(TID): this identifier is supplied by the Primary and is unique for the Primary.
The value is **numeric.**

Transfer status: see "1.6 States of the file transfer" pag. 60.
The value is **character**

user identifier: owner of the transfer identifier
The value is **character.**

host identifier: remote 'host name' specified by the user request
The value is **character.**

required service: see "1.4.1 FTF commands" pag. 55.
The value is **character**

destination effect: see "1.4.3 Parameters values" pag. 57.
The value is **character**

local file name: see "1.4.3 Parameters values" pag. 57.
The value is **character.**

remote file name: see "1.4.3 Parameters values" pag. 57.
The value is **character.**

date and time when the transfer was started
The value is **character.**

date and time when the transfer was interrupted or terminated
The value is **character.**

transferred records number at interruption time
The value is **numeric.**

diagnostic field see "1.8 FTF diagnostic for the user" pag. 61.
The value is **numeric.**

4.0 ISIDE PRIMITIVES

ISIDE is the access method for Applications on the OSIRIDE network. An application, in this case the File Transfer, wishing to exchange data with a partner has to:

- Declare itself to ISIDE
- Open a connection with the partner
- Exchange data
- Close the connection
- Detach itself from ISIDE

What follows is the list of the ISIDE primitives invoked by the FTF, with the related parameters setting.

4.1 OPEN PRIMITIVE

The FTF has to declare itself to the OSIRIDE software before starting. This primitive is invoked by the primary in "2.2.3.1 SP Primary procedure" pag. 79.

parameters

myname this parameter is a character string 16-character long , the local name of FTF.

password is a non mandatory character string, ranging from 1 to 8, decided at moment of implementation and used to give some access rigths (see"4.2 CONNECT primitive" pag. 101)

maxin is a non mandatory number ranging from 1 to 254 [20] which specifies the maximum number of session connection over which an ESTABLISH request will be issued. If not specified the default value is 1. **This value is implementation matter.**

maxout is a non mandatory number ranging from 1 to 254 [20] which specifies the maximum number of session connection over which an ESTABLISH indication will be issued. If not specified this default value is 1. **This value is implementation matter.**

---

20 **in any case** the value <maxin> + <maxout> may not exceed the value of 254.

address this parameter is used to identify a user routine which should take contrul at operation completion. It too is an implementation dependent value.

system this parameter is a character string 16-character long, which uniquely identifies the CSIRIDE system where the Application program runs. **It is not used by the FTF.**

return parameter that brings the return code of the operation.

4.2 CONNECT PRIMITIVE

The connect primitive is the way an OSIRIDE user tries the connection with a partner in the CSIRIDE network and is invoked in "2.2.3.1 SP Primary procedure" pag. 79.

parameters

system name of the remote OSIRIDE system.

user address of the remote application. Value defined is **FTP**

profile non mandatory parameter, which if specified, indicates that the user has requested a particular session profile. This parameter is used to request particular classes of throughput , resilience ("robustness" of the connection against network errors), and other session characteristics. The File Transfer Application will require:

A **Very High Throughput** in the direction of the data transfer,

B **Low Throughput** in the opposite direction, because not very urgent messages are transferred in that direction,

C **Low Resilience**, because the File Transfer Protocol has a recovery mechanism in case of network errors,

data, mark, sync,term are non mandatory parameters that specify the definitions of tokens. These parameters may assume the value:

- **MINE** which states that the specified token is edefined and owned by the issuer of the request.

- **YOURS** which states that the specified token is defined and owned by the receiver cf the request

- **CHOOSE** which states that the specified token is defined, and the receiver of the request should decide about the ownership of it.

In this case the value for these parameters is:

- **DATA** = **MINE**
- **MARK** = **MINE**
- **SYNC** = **MINE**
- **TERM** = not specified because the termination token is not used

type is a presentation-related parameter, which, if used, specifies the type of data which will be dealt with during the session. The possible value is:

- CHARACTER
- OCTET
- BIT

if no value is specified, CHARACTER is the value assumed by default.

syntax specifies whether transformation of code has to be performed in user data. This parameter may assume the value:

- **YES** if translation is performed
- **NO** if no translation is performed

In this implementation the value is: **yes**

comp specifies whether data are to be compressed during the transmission. The legal value are:

- **YES** Compression is performed
- **NO** Compression is not performed

In this first version no compression is performed, so the value is **no**.

Password This parameter may be locally used to check the validity of the request for a given profile.

buffer specifies the address of the buffer where the Select Protocol parameters are to be put.

lenght specifies the lenght of the previously indicated buffer.

address see "4.1 OPEN primitive" pag. 100

identifier is the ISIDE identifier of the connection

return see "4.1 OPEN primitive" pag. 100

4.3 RECEIVE PRIMITIVE

This primitive allows an OSIRIDE user to declare himself ready for receiving data on a previously established connection.
It is used by "2.2.3.4 RA Primary procedure" pag. 81, "2.2.3.5 OP Primary procedure" pag. 82, "2.2.3.6 BT Primary procedure" pag. 83, "2.2.3.8 ET Primary procedure" pag. 84, "2.2.3.9 CL Primary procedure" pag. 84, "2.2.3.10 DL Primary procedure" pag. 85 and "2.2.3.11 RL Primary procedure" pag. 85.

parameters

identifier see "4.2 CONNECT primitive" pag. 101

buffer is the buffer where to receive data

lenght is the length of the buffer

address see "4.1 OPEN primitive" pag. 100

return see "4.1 OPEN primitive" pag. 100

residual is the residual byte count.

4.4 READY PRIMITIVE

This primitive is used by the Secondary in the initialization phase (which is not described in this document) tc declare its readyness to accept a connection establishement indication.

parameters

System specifies a particular system tc be addressed. If this value is not specified any user anywhere may try a connection. It is not used by the FTF.

user specifies a particular remote user residing on a remote system, which is the only one allowed to try a connection. It is not used by the FTF.

broadcast specifies whether the ready information has to be broadcasted through the OSIRIDE network. It is not used by the FTF.

address see "4.1 OPEN primitive" pag. 100.

return is the ISIDE return code

4.5 DATA PRIMITIVE

This primitive is used to transfer a data buffer one the previously established connection.
It is used in "2.2.3.2 SL Primary procedure" pag. 80, "2.2.3.3 CR Primary procedure" pag. 80, "2.2.3.4 RA Primary procedure" pag. 81, "2.2.3.5 OP Primary procedure" pag. 82,"2.2.3.6 BT Primary procedure" pag. 83, "2.2.3.7 DATA Primary procedure" pag. 84, "2.2.3.8 ET Primary procedure" pag. 84,"2.2.3.9 CL Primary procedure" pag. 84, "2.2.3.10 DL Primary procedure" pag. 85, "2.2.3.11 RL Primary procedure" pag. 85,"2.3.2.2 SL Secondary procedure" pag. 90, "2.3.2.3 CR Secondary procedure" pag. 91, "2.3.2.4 RA Secondary procedure" pag. 92,"2.3.2.5 OP Secondary procedure" pag. 92, "2.3.2.6 BT Secondary procedure" pag. 93, "2.3.2.8 ET Secondary procedure" pag. 94,"2.3.2.9 CL Secondary procedure" pag. 95, "2.3.2.10 DL Secondary procedure" pag. 95 and "2.3.2.11 RL Secondary procedure" pag. 95.

parameters

identifier see "4.2 CONNECT primitive" pag. 101.

buffer specifies the data buffer to be sent and contains the VFS protocol message.

length specifies the buffer length.

code see "4.2 CONNECT primitive" pag. 101 and the value is **"yes"**

return is the ISIDE return code

4.6 CONTROL PRIMITIVE

This primitive is used to perform the Token management.
It is used by "2.2.3.6 BT Primary procedure" pag. 83 and "2.3.2.8 ET Secondary procedure" pag. 94.

parameters

identifier see "4.2 CONNECT primitive" pag. 101

mode is a mandatory parameter, which specifies how the token(s) specified in "4.2 CONNECT primitive" pag. 101 should be exchanged. The legal value are:

please to request one or more tokens owned by the partner.

give to release the possession of specific tokens.

type This parameter always has the value **give** It always assumes the value DATA+MARK+SYNC.

address see "4.1 OPEN primitive" pag. 100.

address is the ISIDE return code.

4.7 SYNCHRONIZE PRIMITIVE

This primitive is used to define points in the dialogue between the partners. It is used by "2.2.3.6 BT Primary procedure" pag. 83 and "2.2.3.8 ET Primary procedure" pag. 84.

parameters

identifier see "4.1 OPEN primitive" pag. 100.

type specifies the type of sinchronization point. In the OSIRIDE FTF protocol, it always assumes the value **mayor.**

point is the value that the session layer assignes as a mark serial number

return is the ISIDE return code

4.8 ACCEPT PRIMITIVE

This primitive is used to answer to a CONNECT (see "4.2 CONNECT primitive" pag. 101) request, in order to negotiate the session and presentation parameters. It brings the SP response, and is only used by the Seccndary in "2.3.2.1 SP Secondary procedure" pag. 90.

parameters

identifier see "4.1 OPEN primitive" pag. 100

result this parameter brings the result of operation:

yes if connection is accepted

no if connection is rejected.

for **data, mark, synk, term comp, buffer, length, address** and **return** parameters see "4.2 CONNECT primitive" pag. 101.

4.9 RESPONSE PRIMITIVE

This primitive is used to explicitly answer to an indication coming from the partner. It is used in "2.2.3.14 REC DATA Primary procedure" pag. 86, "2.3.2.6 BT Secondary procedure" pag. 93, "2.3.2.8 ET Secondary procedure" pag. 94 and "2.3.2.12 RP Secondary procedure" pag. 96.

parameters

identifier see "4.1 OPEN primitive" pag. 100.

command specifies the indication which is answered to.

point specifies a synch point to be confirmed.

data, mark, sync and term are never set.

buffer not used (no parameters are transferred together with any response).

length not used.

address see "4.1 OPEN primitive" pag. 100.

return is the ISIDE return code

4.10 FINISH PRIMITIVE

This primitive is used to answer to terminate the FTF connection. It is used by "2.2.3.12 RP Primary procedure" pag. 86 and "2.2.3.13 DP Primary procedure" pag. 86.

parameters

identifier see "4.2 CONNECT primitive" pag. 101.

address see "4.1 OPEN primitive" pag. 100.

mode is the parameter which specifies the type of termination. Its values are:

term for give a "soft" connection termination

abor which means that the session has to be aborted.

return is the ISIDE return code

REFERENCES

[1] Standard ECMA 85 - Virtual File Protocol.

[2] Standard ECMA 84 - Data Presentation Protocol.

[3] F. Caneschi, E. Zucchelli: "The implementation of OSIRIDE File Transfer over ECMA-85" Dcc. OSIRIDE/83/FTP/04 March 1983.