The Consultative Committee for Space Data Systems

Recommendation for Space Data System Standards

# CCSDS FILE DELIVERY PROTOCOL (CFDP)

RECOMMENDED STANDARD

CCSDS 727.0-B-5

BLUE BOOK

JULY 2020

The Consultative Committee for Space Data Systems

Recommendation for Space Data System Standards

# CCSDS FILE DELIVERY PROTOCOL (CFDP)

RECOMMENDED STANDARD

CCSDS 727.0-B-5

BLUE BOOK

JULY 2020

# AUTHORITY

| | |
|---|---|
| Issue: | Recommended Standard, Issue 5 |
| Date: | July 2020 |
| Location: | Washington, DC, USA |

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and represents the consensus technical agreement of the participating CCSDS Member Agencies. The procedure for review and authorization of CCSDS documents is detailed in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4), and the record of Agency participation in the authorization of this document can be obtained from the CCSDS Secretariat at the e-mail address below.

This document is published and maintained by:

CCSDS Secretariat
National Aeronautics and Space Administration
Washington, DC, USA
Email: secretariat@mailman.ccsds.org

## STATEMENT OF INTENT

# FOREWORD

Until relatively recently, the typical storage medium for spacecraft has been the tape recorder, a complex device offering limited data storage and data access. The use of this type of storage has typically been limited to the recording and subsequent dump to the ground of telemetry data. Manipulation from the ground has required significant human intervention and used ad hoc, privately developed protocols.

The introduction of solid state mass memory providing gigabytes of storage with random access opens up a whole new ethos of spacecraft operation in which much of the routine traffic to and from the spacecraft will be in the form of files. Furthermore, because of the random-access nature of the onboard storage medium, it becomes possible to repeat transmission of data lost on the link and thus guarantee delivery of critical information.

To exploit the potential advantages of onboard mass memory, protocol support is required to provide a standard means to move data to and from the onboard storage medium in the form of files.

While the onboard storage medium has rapidly evolved, the essential constraints of space missions remain:

– limited systems resources in terms of computational power and memory capacities;

– environmental restrictions including noisy, bandwidth-limited, asymmetrical, and interrupted communications links, some with very long propagation delay;

– varying user needs including a requirement for early access to transferred data regardless of its quality.

In view of these constraints, it is clear that there is a need for a file delivery service capable of transferring files to and from mass memory located in the space segment. Such a capability must not only operate under the constraints associated with space data communication, but it must also be applicable to the diverse range of mission configurations ranging from single low earth orbiting spacecraft to complex networks of relays, orbiters, and landers.

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Recommended Standard is therefore subject to CCSDS document management and change control procedures, which are defined in the *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4). Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relating to the contents or status of this document should be sent to the CCSDS Secretariat at the email address indicated on page i.

# DOCUMENT CONTROL

| **Document** | **Title** | **Date** | **Status** |
|---|---|---|---|
| CCSDS 727.0-B-1 | CCSDS File Delivery Protocol, Issue 1 | January 2002 | Original issue (superseded) |
| CCSDS 727.0-B-2 | CCSDS File Delivery Protocol, Issue 2 | October 2002 | Issue 2 (superseded) |
| CCSDS 727.0-B-3 | CCSDS File Delivery Protocol, Issue 3 | June 2005 | Issue 3 (superseded) |
| CCSDS 727.0-B-4 | CCSDS File Delivery Protocol (CFDP), Recommended Standard, Issue 4 | January 2007 | Issue 4 (superseded) |
| CCSDS 727.0-B-5 | CCSDS File Delivery Protocol (CFDP), Recommended Standard, Issue 5 | July 2020 | Current issue:<br>– moves Store and Forward Overlay Operations from section 6 to a new normative annex (annex B);<br>– removes Extended Procedures;<br>– adds support for large files, multi-segment records, and per-segment metadata;<br>– adds support for choice of checksum algorithm from a new SANA Checksum Types registry;<br>– updates, clarifies some text based on interoperability testing. |

NOTE – Changes from the previous issue are too numerous to permit meaningful application of change bars.

## CONTENTS

## CONTENTS (continued)

## CONTENTS (continued)

## CONTENTS (continued)

# 1 INTRODUCTION

## 1.1 PURPOSE AND SCOPE

This document defines a CCSDS File Delivery Protocol (CFDP) and associated service for application in the space environment. It is intended for use over the current and envisaged packet delivery services used in the space environment, including:

– CCSDS conventional packet telecommand;

– CCSDS conventional packet telemetry;

– CCSDS Advanced Orbiting Systems (AOS) Path service.

In may also operate over a wide variety of ground network services, including those specified by the CCSDS for cross-support purposes.

The protocol operates in the space-to-ground, ground-to-space, and space-to-space directions of transfer. It may be initiated by the file sending or receiving entity.

In the interests of interoperability, protocol elements are included for generalized forms of standard file manipulation operations based on assumptions of a common model for a 'filestore', or medium used to store files. It is recognized, however, that the precise nature and capabilities of filestore management systems are operating-system dependent and, for that reason, the protocol assumes a virtual filestore and associated services that an implementation must map to the capabilities of the actual filestore used.

It is important to note that the current specification for CFDP omits the Extended Procedures that were defined in the original definition of the protocol. Service classes 3 and 4 have accordingly been deleted as well.

## 1.2 APPLICABILITY

This Recommendation specifies a protocol and associated services that are applicable to space missions with continuous duplex contact, intermittent duplex contact, asymmetrical time-disjoint contact, and simplex contact.

## 1.3 CONVENTIONS AND DEFINITIONS

### 1.3.1 BIT NUMBERING CONVENTION AND NOMENCLATURE

In this document, the following convention is used to identify each bit in an *N*-bit field. The first bit in the field to be transmitted (i.e., the most left justified when drawing a figure) is defined to be 'Bit 0'; the following bit is defined to be 'Bit 1', and so on up to 'Bit *N*–1'. When the field is used to express a binary value (such as a counter), the Most Significant Bit (MSB) shall be the first transmitted bit of the field, that is, 'Bit 0'.

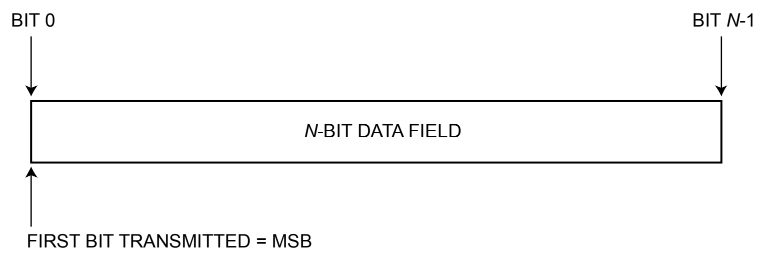


**Figure 1-1: Bit Numbering Convention**

In accordance with modern data communications practice, spacecraft data fields are often grouped into 8-bit ‘words’ which conform to the above convention. Throughout this Recommendation, the following nomenclature is used to describe this grouping:

8-BIT WORD = ‘OCTET’

**Figure 1-2: Octet Convention**

By CCSDS convention, all ‘spare’ bits shall be permanently set to value ‘zero’.

## 1.3.2 ORGANIZATION OF THE RECOMMENDATION

This Recommendation is organized as follows:

– Section 2 provides an overview of the protocol, its intended use, and a description of the main interactions involved in a file transfer.

– Section 3 defines the services provided by the protocol along with the associated primitives and parameters.

– Section 4 is the main body of the protocol specification.

– Section 5 defines the formats for the PDUs.

– Section 6 defines proxy and directory operations.

– Section 7 defines a number of classes of operation designed to meet the most typical operating scenarios.

– Section 8 describes the management information base.

– Annex A provides the PICS Pro Forma for CFDP.

– Annex B provides a specification for standard Store and Forward Overlay operations.

– Annex C provides Security, SANA, and Patent considerations.

– Annex D provides a list of informative references.

– Annex E provides a list of acronyms and definitions.

– Annex F provides an example of file modular checksum calculation.

## 1.3.3 DEFINITIONS

### 1.3.3.1 Definitions from OSI Basic Reference Model

This Recommendation makes use of a number of terms defined in reference [5]. The use of those terms in this Recommendation shall be understood in a generic sense, that is, in the sense that those terms are generally applicable to any of a variety of technologies that provide for the exchange of information between real systems. Those terms are:

– entity;

– Protocol Data Unit (PDU);

– service;

– Service Access Point (SAP);

– Service Data Unit (SDU).

### 1.3.3.2 Definitions from Open Systems Interconnection (OSI) Service Definition Conventions

This Recommendation makes use of a number of terms defined in reference [6]. The use of those terms in this Recommendation shall be understood in a generic sense, that is, in the sense that those terms are generally applicable to any of a variety of technologies that provide for the exchange of information between real systems. Those terms are:

– Indication;

– Primitive;

– Request;

– Response.

### 1.3.3.3 Terms Defined in This Recommendation

Within the context of this document the following definitions apply:

A *CFDP protocol entity* (or *CFDP entity*) is a functioning instance of an implementation of CFDP, roughly analogous to an Internet protocol ‘host’.

The functional concatenation of a file and related *metadata* transmitted between two CFDP entities is termed a *File Delivery Unit* (FDU); in this context, the term ‘metadata’ is used to refer to any data exchanged between CFDP entities in addition to file content, typically either additional application data (such as a ‘message to user’) or data that aid the recipient entity in effectively utilizing the file (such as file name). It should be noted that an FDU may consist of metadata only. Note also that the term ‘file’ is frequently used in this specification as an abbreviation for ‘file delivery unit’; only when the context clearly indicates that only actual files are being discussed, for example, in the explanation of the segmentation control parameter or the source and destination file name parameters of the CFDP Service Definition, should the term ‘file’ not be read as ‘file delivery unit’.

The individual, bounded, self-identifying items of CFDP data transmitted between CFDP entities are termed *CFDP Protocol Data Units* (*CFDP PDU*s). Unless otherwise noted, in this document the term ‘PDU’ always means ‘CFDP PDU’. CFDP PDUs are of two general types: *File Data PDUs* convey the contents of the files being delivered, while *File Directive PDUs* convey only metadata and other non-file information that advances the operation of the protocol.

A *transaction* is the end-to-end transmission of a single FDU between two CFDP entities. A single transaction normally entails the transmission and reception of multiple PDUs. Each transaction is identified by a unique transaction ID; all elements of any single FDU, both file content and metadata, are tagged with the same CFDP transaction ID.

Any single end-to-end file transmission task has two associated entities: the entity that has the file at the beginning of the task (the *source*) and the entity that has a copy of the file when the task is completed (the *destination*).

Each end-to-end file transmission task comprises one or more point-to-point file copy operations. A file copy operation has two associated entities: the entity that has a copy of the file at the beginning of the operation (the *sender* or *sending entity*) and the entity that has a copy of the file when the operation is completed (the *receiver* or *receiving entity*). In the simplest case, the only sender of the file is the source, and the only receiver is the destination. In more complex cases (the general case), there are additional ‘*waypoint*’ entities that receive and send copies of the file; the source is the first sender, and the destination is the last receiver.

*Filestore* is a generic term referring to the medium used to store files.

The term *offset* is used in a familiar way: the offset of a given octet of file data is the number of data octets that precede this octet in the file.

The file delivery *progress* represented by a given File Data PDU is the sum of the offset of the PDU’s file data content (the offset of the content’s first octet) and the length of that file data content.

The *transmission progress* for a given transaction, delivering some file, is the maximum progress value over all File Data PDUs sent so far in the course of this transaction.

Similarly, the *reception progress* for a given transaction is the maximum progress value over all File Data PDUs received so far in the course of this transaction. If and only if no data are lost in transmission, reception progress is equal to the number of file data octets received.

The term *CFDP user* is used to refer to the software task that causes the local entity to initiate a transaction or the software task that is notified by the local entity of the progress or completion of a transaction. The CFDP user local to the source entity is referred to as the *source CFDP user*. The CFDP user local to the destination entity is referred to as the *destination CFDP user*. The CFDP user may be operated by a human or by another software process. Unless otherwise noted the term *user* always refers to the CFDP user.

## 1.4 REFERENCES

The following documents contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All documents are subject to revision, and users of this Recommendation are encouraged to investigate the possibility of applying the most recent editions of the documents indicated below. The CCSDS Secretariat maintains a register of currently valid CCSDS Recommendations.

[1] *AOS Space Data Link Protocol*. Issue 3. Recommendation for Space Data System Standards (Blue Book), CCSDS 732.0-B-3. Washington, D.C.: CCSDS, September 2015.

[2] *TM Space Data Link Protocol*. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 132.0-B-2. Washington, D.C.: CCSDS, September 2015.

[3] *Space Packet Protocol*. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 133.0-B-2. Washington, D.C.: CCSDS, June 2020.

[4] *TC Space Data Link Protocol*. Issue 3. Recommendation for Space Data System Standards (Blue Book), CCSDS 232.0-B-3. Washington, D.C.: CCSDS, September 2015.

[5] *Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model*. 2nd ed. International Standard, ISO/IEC 7498-1:1994. Geneva: ISO, 1994.

[6] *Information Technology—Open Systems Interconnection—Basic Reference Model—Conventions for the Definition of OSI Services*. International Standard, ISO/IEC 10731:1994. Geneva: ISO, 1994.

[7] *Operation of CFDP over Encapsulation Service*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 722.1-M-1. Washington, D.C.: CCSDS, March 2014.

[8] *Encapsulation Packet Protocol*. Issue 3. Recommendation for (Blue Book), CCSDS 133.1-B-3. Washington, D.C.: CCSDS, April 2020.

[9] *CCSDS Bundle Protocol Specification*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 734.2-B-1. Washington, D.C.: CCSDS, September 2015.

[10] “Checksum Identifiers.” Space Assigned Numbers Authority. https://sanaregistry.org/r/checksum_identifiers.

The latest issues of CCSDS documents may be obtained from the CCSDS Secretariat at the address indicated on page i.

# 2 OVERVIEW

## 2.1 GENERAL

This Recommendation defines a protocol suitable for the transmission of files to and from spacecraft data storage. In addition to the purely file-delivery-related functions, the protocol also includes file management services to allow control over the storage medium.

The protocol is capable of operating in a wide variety of mission configurations, from relatively simple low earth orbit spacecraft to complex arrangements of orbiters and landers supported by multiple ground facilities and transmission links. The protocol provides a *core* file delivery capability operating across a single pairwise communication association. When the underlying unitdata transfer protocol provides network functionality, the core file delivery capability of CFDP can deliver files across an arbitrary network containing multiple links with disparate availability.

The protocol is independent of the technology used to implement data storage and requires only a few fundamental filestore capabilities in order to operate. It assumes two filestores, one within the spacecraft and one on the ground, and operates by copying data between the two filestore locations.

The protocol makes no assumptions about the information being transferred and can be utilized for a wide range of applications involving the loading, dumping, and control of spacecraft storage.

The protocol has been specifically designed to minimize the resources required for operation. It is also scalable, so that only those elements required to fulfill the selected options are required to be implemented.

Although the protocol can operate over a wide range of underlying communication services, this Recommendation assumes the use of CCSDS services as defined in references [1], [2], [3], [4], [7], [8],and [9].

## 2.2 ARCHITECTURE ELEMENTS

### 2.2.1 GENERAL

The architectural elements involved in the file delivery protocol are depicted in figure 2-1 and described below.

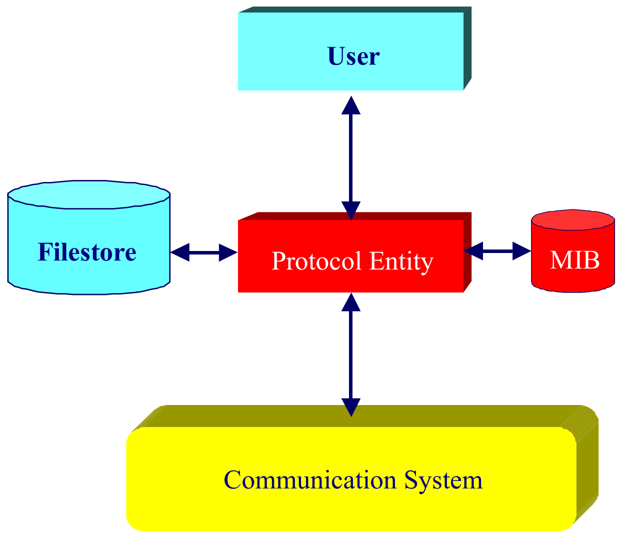


**Figure 2-1: Architectural Elements of the File Delivery Protocol**

### 2.2.2 USER

The protocol operates at the request of the CFDP user. The user interacts with the protocol using the service primitives defined in section 3.

A CFDP user is always a software task, which may or may not be operated by a human.

Each CFDP entity has at most one user.

### 2.2.3 PROTOCOL ENTITY

The protocol entity consists of implementations of the *core delivery procedures*, which allow immediate file delivery and manipulation over a single communication association between two entities. A single service interface is presented to the user.

### 2.2.4 FILESTORE

The protocol operates by copying files from storage medium to storage medium, and it is therefore assumed that all CFDP entities have access to a local storage capability. As the ways in which the storage capability is provided will vary, the protocol is built on the premise that any file or organized set of files (i.e., a filestore) can be described in terms of a single standard representation. This representation, called a '**virtual filestore**' is assigned a standard set of attributes that are then used by the protocol to manage the file delivery process. In an implementation, the virtual filestore must be mapped to and from actual hardware and software that constitute the real filestore. This approach allows complete independence from the technology used to implement the filestore.

### 2.2.5 UNDERLYING COMMUNICATION SYSTEM

The protocol assumes the availability of a single conceptual underlying communication system, referred to as the '**Unitdata Transfer (UT) layer**', to which all CFDP entities in a given CFDP addressing domain have access. In order that the protocol may operate over a wide range of implementations, the services required of the UT layer have intentionally been kept as simple as possible; those services are assumed to be made available to any single CFDP entity at only a single conceptual service access point.

Because of the potential diversity of underlying services in use, no reuse of underlying protocol features is utilized beyond the addressing required to identify localities within the domain of the underlying communication system ('**UT addresses**', to which CFDP entities' names may be mapped using information contained in the management information base) and the delimitation of the CFDP PDU.

The relative independence of CFDP specifically implies that all multiplexing to support multiple file-delivery transactions occurs internally in the CFDP entity and that sequence auditing and error detection are provided by CFDP. PDU length information is incorporated in the CFDP PDU header to give independence from the underlying protocol and to simplify protocol processing.

It should be noted that although CFDP can operate over a service in which data errors, data loss, and out-of-sequence delivery occur, it is not intended to compensate for networks in which these effects are prevalent. Severe performance reductions will result if such an approach is taken. A simplified analysis of CFDP performance is detailed in the CFDP Green Book (CCSDS 720.1-G-1). This analysis will allow implementers to gauge likely performance based on file size, file segment size and bit error probability.

### 2.2.6 MANAGEMENT INFORMATION BASE

To perform a file delivery, a significant amount of information must be passed by the local user to its local CFDP entity and by the local CFDP entity to the remote CFDP entity. Typically, this data is static and may be stored within the CFDP entities as system tables, referred to as the Management Information Base (MIB). The MIB contains such information as default values for user communications requirements, for example, for address mapping, and for communication timer settings. The MIB is formally defined as part of the protocol specification in section 8.

## 2.3 GENERAL CHARACTERISTICS

### 2.3.1 GENERAL

The CFDP enables the moving of a file from one filestore to another, in which the two filestores are, generally, resident in separate data systems and often with an intervening space link. Functionally attached to each such file may be metadata of variable size, format, and semantics. The metadata consists of information associated with the file or the file transfer such as file names, filestore requests, or messages to the CFDP user. The functional concatenation of a file and all relevant metadata is termed an FDU, but it is also permissible for an FDU to consist of metadata only.

FDU transmission is accomplished by CFDP entities, each of which has access to exactly one filestore. A transaction is the end-to-end transmission of a single FDU between two CFDP entities. The initiation of the transaction causes a file copy operation to occur between two entities, the sender and the receiver.

As depicted in figure 2-2, the protocol procedures constitute the interaction between two protocol entities. The sending entity is the entity from which the file is copied in a file copy operation. The receiving entity is the entity to which the file is copied in a file copy operation.

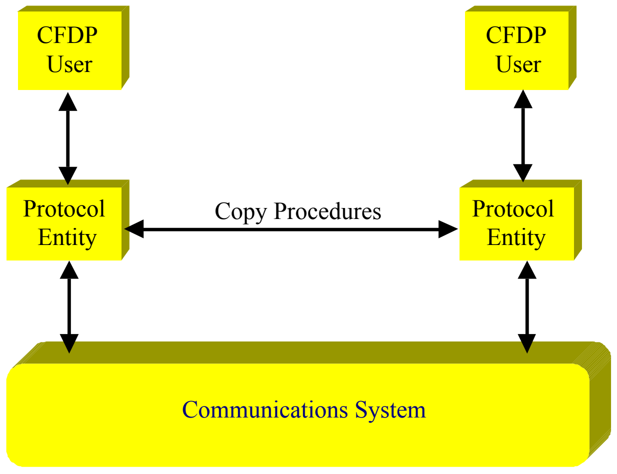


**Figure 2-2: CFDP Procedures**

The reliability of a transaction is determined by whether the transaction is chosen to operate in unacknowledged mode or one of the acknowledged modes. In unacknowledged mode, data delivery failures are not reported to the sender and therefore cannot be repaired. Reception of the complete file is therefore not guaranteed by CFDP.

NOTE – The use of a reliable underlying communication system may in practice guarantee complete file delivery even when file transfer is performed in unacknowledged mode. This consideration is outside the scope of the CFDP specification.

In acknowledged mode, the receiver informs the sender of any undelivered file segments or ancillary data. These are then retransmitted, guaranteeing complete file delivery.

Each transaction results in the copying of a single FDU from source to destination. Within any single transaction, the subsidiary actions cancel, suspend, resume, and report may be performed.

The transaction is terminated when one of the following conditions applies:

– the file has been successfully transferred;

– a sending or receiving entity has disallowed the transaction;

– the transaction has been canceled because a fault was detected;

– the source or destination CFDP user has canceled the transaction.

### 2.3.2 USER OPERATIONS

The term 'user operations' refers to the use of the CFDP services offered by the local CFDP entity to cause the CFDP user of a remote CFDP entity to initiate additional CFDP transactions. User operations are implemented using the 'Message to User' capability of the protocol to forward an 'order' to the remote CFDP user, which will in turn initiate a transaction with its local CFDP entity.

Five standard user operations are defined: proxy operations, remote status report operations, remote suspend and resume operations, and directory operations. Directory operations are used to request a listing of the contents of a specified directory in the remote user's local filestore. Remote status report operations are used to request a report of the status of a specified CFDP transaction at the remote entity. Remote suspend and resume operations are used to request the suspension and resumption of a specified CFDP transaction at the remote entity. Proxy operations are used to initiate the delivery of a file from a remote CFDP entity to some other user, either to the local user itself (in which case, the proxy operation functions as a 'Get') or to the user of some third CFDP entity. The FDU transmitted in a proxy operation normally contains a file but may contain only metadata, such as filestore directives or a Message to User containing an order to another remote CFDP user.

User operations are described in section 6.

### 2.3.3 ADDRESSING

Within any single CFDP network (i.e., any set of CFDP entities that is closed under communication), each CFDP entity is assigned a unique name. As each CFDP entity has access to exactly one filestore, identification of a CFDP entity implicitly identifies the associated filestore. At each CFDP entity location, address look-up capabilities are provided using information contained in the associated MIB. This look-up capability provides translation between the network-unique name of a CFDP entity and the corresponding UT address, which may in reality be an Internet address, radio device buffer, APID, virtual channel number, or other implementation-specific mechanism.

### 2.3.4 RETRANSMISSION STRATEGIES

The quality of service offered by the protocol is selectable, according to mission requirements and transmission capability, and ranges from an unacknowledged option, whereby a file is transmitted with no attempt at completeness should errors occur (errors will be detected and data discarded), to a fully acknowledged option providing error recovery through retransmission. For the acknowledged modes of operation, several sub-options may be selected by the receiver. These sub-options relate to release time of any Negative Acknowledgments (NAKs) and range from immediate release to deferred release (whereby any NAKs are stored until the assumed end of the transmission). The extent of available retransmission strategies is more completely described in section 4.

### 2.3.5 VIRTUAL FILESTORE

The virtual filestore concept provides for a mapping of protocol filestore directives to actual filestore manipulation. The way in which this mapping is performed in an implementation is a local matter. Allowances are made in the protocol for inclusion of additional filestore directives using the Type-Length-Value (TLV) capability of the Metadata PDU.

To enable interoperability, this Recommendation assumes a minimum set of capabilities from the virtual filestore as follows:

– create file;

– delete file;

– rename file;

– append file;

– replace file;

– create directory;

– remove directory;

– list directory.

In some circumstances, it is advantageous for CFDP to be able to recognize record boundaries within the file. If this option is to be used, the filestore must have the capability to make the distinction between such files and those which are to be treated as a stream of octets.

## 2.4 OVERVIEW OF INTERACTIONS

Figure 2-3 shows the details of a Copy operation. (In all following figures in this section, dashed lines denote optional actions.)

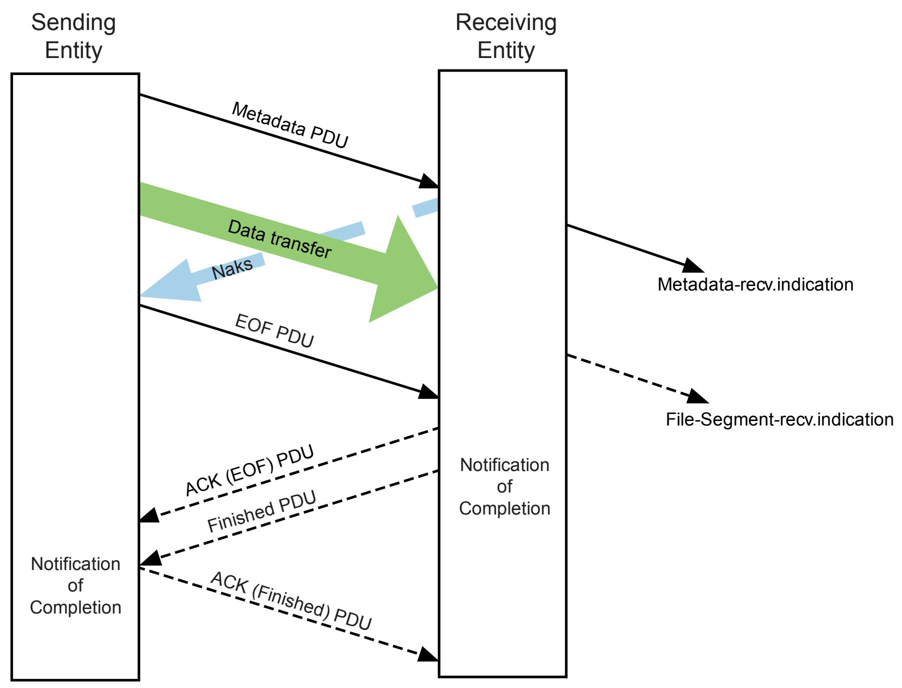


**Figure 2-3: Copy Operations, Sequence of Events**

All copy operations start with transmission of the metadata, followed by transmission of the file segments and a PDU to indicate the End Of File (EOF). Optional service primitives may be issued at the receiving entity to announce receipt of individual file segments; notification of receipt of metadata is mandatory.

For reliable transactions, a number of features are added. NAKs are used to request retransmission of lost data. Receipt of the EOF PDU is ensured via an EOF acknowledgement. Since lost data may still be outstanding after the EOF sequence, a Finished PDU is sent when all file data has been successfully assembled. Delivery of this PDU is ensured through use of the Finished acknowledgement PDU.

Optionally, a Finished PDU may also be sent upon completion of a transaction in Unacknowledged mode. This option, termed ‘transaction closure’, enables the source entity to be informed of the outcome of a transaction that did not require acknowledgments, possibly because it was conducted using an underlying communication system that was itself known to be reliable.

When the file transmission task is completed, the sending and receiving entities issue Notices of Completion. These Notices of Completion are abstract internal events that are not exposed to the CFDP user. They are used to inform the Put procedures (described later in this section) of completion of the file transmission task.

The Put transaction is initiated by a `Put.request` from a CFDP user to the source entity. The `Put.request` results in initiation of the FDU transmission procedure and also of the copy operation. The `Put.request` also results in the notification of a Transaction identifier to the application so that the transaction may subsequently be unambiguously identified. A Notice of Completion at the source entity causes it to issue a `Transaction-Finished.indication`. This process is optional at the destination entity. The Put transaction is illustrated in figure 2-4.

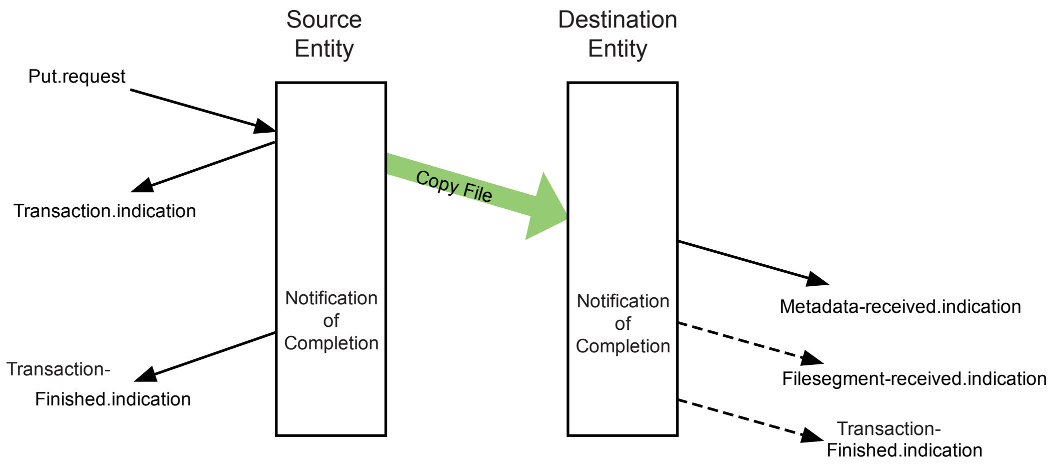

Source Entity
Destination Entity
Put.request
Transaction.indication
Copy File
Notification of Completion
Notification of Completion
Metadata-received.indication
Filesegment-received.indication
Transaction-Finished.indication
Transaction-Finished.indication

**Figure 2-4: Put Operations, Sequence of Events**

When the source of the file to be transferred is other than the local filestore, the requesting user must initiate a Put transaction containing a message to the user of the remote CFDP entity, the ‘proxy user’. This message requests that the proxy user in turn submit a `Put.request` to that entity. The destination of the requested Put transaction may be the original requesting user (in which case the proxy operation functions as a ‘Get’) or some third CFDP user. On completion of the second transaction, the proxy user initiates a third transaction to notify the original requesting user of the completion of the proxy operation. A proxy put operation is illustrated in figure 2-5.

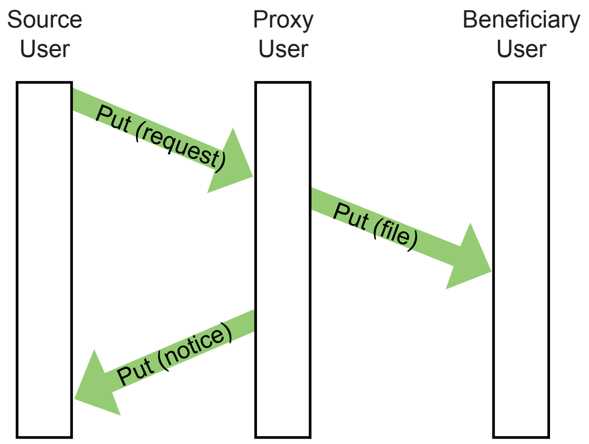


**Figure 2-5: Proxy Put Operations, Sequence of Events**

# 3 SERVICE DESCRIPTION

## 3.1 SERVICES AT THE USER INTERFACE

**3.1.1** The services provided by the protocol shall be made available to the CFDP user and shall include the following:

a) initiating the transmission of files between filestores;

b) initiating filestore operations on remote filestores;

c) receiving events related to the operation of current transactions;

d) requesting status information related to the current transactions;

e) sending or receiving messages associated with the current transactions;

f) suspending, resuming, or canceling the transmission of a current transaction.

**3.1.2** The CFDP entity shall be implemented such that virtually any number of transactions may be concurrently in various stages of transmission or reception at a single CFDP entity.

NOTE – To clarify: the implementation must be able to accept a **`Put.request`** primitive, and thereupon initiate a new transaction, prior to the completion of previously initiated transactions. The completion of a transaction encompasses the initial transmission of file data by the source entity, the reception of some or all of that data by the destination entity, and any consequent acknowledgment and retransmission data exchange that may be necessary between the two entities. The requirement for concurrent transaction support therefore does not necessarily imply that the implementation must be able to begin initial transmission of file data for one transaction while initial transmission of file data for one or more other transactions is still in progress (but neither is support for this functional model precluded).

## 3.2 SUMMARY OF PRIMITIVES

**3.2.1** The CFDP service shall consume the following request primitives:

a) **`Put.request`**;

b) **`Report.request`**;

c) **`Cancel.request`**;

d) **`Suspend.request`**;

e) **`Resume.request`**.

**3.2.2** The CFDP service shall deliver the following indication primitives:

a) **`Transaction.indication`**;

b) **`EOF-Sent.indication`**;

c) **`Transaction-Finished.indication`**;

d) **`Metadata-Recv.indication`**;

e) **`File-Segment-Recv.indication`**;

f) **`Report.indication`**;

g) **`Suspended.indication`**;

h) **`Resumed.indication`**;

i) **`Fault.indication`**;

j) **`Abandoned.indication`**;

k) **`EOF-Recv.indication`**.

**3.2.3** Summary of Parameters

NOTE – The availability and use of parameters for each primitive are indicated in 3.4, in which parameters that are optional are identified with square brackets [thus]. The following definitions apply.

**3.2.4** The *destination CFDP entity ID* parameter shall identify the CFDP entity **to** which the FDU is to be sent.

**3.2.5** The *source CFDP entity ID* parameter shall identify the CFDP entity **from** which the FDU is to be sent.

**3.2.6** The *source file name* parameter

a) shall contain the full path name at which the file to be copied is located at the filestore associated with the source entity;

b) shall be omitted when the FDU to be Put contains only metadata, such as a message to a user or a standalone filestore request.

**3.2.7** The *destination file name* parameter

a) shall contain the full path name to which the file to be copied will be placed at the filestore associated with the destination entity;

b) shall be omitted when the FDU to be Put contains only metadata, such as a message to a user or a standalone filestore request.

**3.2.8** The *segmentation control* parameter

a) shall indicate whether the file being delivered is to be segmented as an array of octets or as an array of variable-length records;

b) shall be omitted when local and remote file names are omitted.

**3.2.9** The *Transaction ID* parameter shall uniquely identify a single instance of FDU delivery and shall contain the ID of the source CFDP entity together with a sequence number that is specific to that entity.

NOTES

1 At any moment, any given transaction ID is unique within the CFDP addressing domain that encompasses the source CFDP entity.

2 The entity ID and transaction sequence number are variable in length to accommodate differing mission requirements. A common parsing scheme is defined to allow full interoperability despite differing parameter lengths.

**3.2.10** The *Condition code* parameter shall provide additional information on some change in transaction status.

**3.2.11** The *Status report* parameter shall indicate the status of the indicated file delivery transaction. The format and scope of the *status report* parameter are specific to the implementation. It could contain information such as:

a) whether the transaction is finished, canceled, suspended, or active;

b) what extents of the FDU are known to have been successfully received by the receiving CFDP entity;

c) what extents of the FDU are known to have been transmitted by the sending CFDP entity.

**3.2.12** *Fault handler overrides*:

a) If included, the optional *fault handler overrides* shall indicate the actions to be taken upon detection of one or more types of fault condition. Each fault handler override shall identify both a type of fault condition to be handled and the action to be taken in the event that a fault of this type is detected; each such action shall be one of the following:

   – cancel the file delivery transaction,

   – suspend the transaction,

   – ignore the fault but issue a **`Fault.indication`** primitive to the local CFDP user, or

   – abandon the transaction and issue an **`Abandoned.indication`** primitive to the local CFDP user.

b) The fault conditions for which handler overrides may be specified are a subset of the conditions listed in table 5-5, later in this specification. The listed conditions that are *not* fault conditions are ‘No error’, ‘Suspend request received’, and ‘Cancel request received’.

c) For each type of fault condition for which no *fault handler override* is included among the service request parameters, a default handler shall be selected according to the contents of the MIB.

**3.2.13** If included, the optional *transmission mode* parameter shall override the default transmission mode. The values of the transmission mode parameter are ‘acknowledged’ or ‘unacknowledged’.

**3.2.14** If included, the optional *Messages to User* parameter shall be transmitted at the beginning of the transaction and delivered to the destination CFDP user upon receipt. Certain messages are defined in the User Operations section to allow remote initiation of CFDP transactions.

**3.2.15** *Filestore requests*:

a) if included, the optional filestore requests shall be transmitted at the beginning of the transaction and shall be acted upon by the destination entity when all data transfer activities of the transaction are completed;

b) to indicate the action to be taken, each filestore request shall include sub-parameters as follows:

   – *action*,

   – *file name 1*,

   – *file name 2* (present for certain values of the *action* parameter);

c) the *action* sub-parameter may take the values of:

   – *create file*,

   – *delete file*,

   – *rename file (file name 2* present*)*,

   – *append file* (*file name 2* present),

   – *replace file* (*file name 2* present),

   – *create directory*,

   – *remove directory*,

   – *‘deny’ file* (like *delete*, but does not fail if file does not exist),

   – *‘deny’ directory* (like *remove*, but does not fail if directory does not exist);

d) when the value of the *file name 1* or *file name 2* sub-parameter is equal to the value of the *destination file name* parameter, the file to which the filestore request pertains shall be the newly delivered file;

e) in the case of multiple *filestore requests* in the same transaction, the source entity shall preserve the order in which the requests were submitted, and the destination entity shall execute them in this same order.

**3.2.16** The *Filestore Responses* parameter shall indicate whether the indicated *filestore requests* in the indicated transaction succeeded or failed and shall include the following additional information for each filestore request:

– if it failed, an explanation;

– if it succeeded, a report in action-specific format (if relevant).

**3.2.17** The *Filestore Responses* parameter shall not be used to convey directory listing information; the transfer of directory listing information is achieved via Directory Operations transactions.

**3.2.18** The *flow label* parameter may optionally be used to support prioritization and preemption schemes.

NOTE – The *flow label* parameter should be taken as a hint to the order in which PDUs should be transmitted when the opportunity arises, but the manner in which flow labels are used is strictly an implementation matter.

**3.2.19** The *offset* parameter shall indicate a displacement from the beginning of the file, expressed as a number of octets.

**3.2.20** The *length* parameter shall indicate the number of octets of file data received.

**3.2.21** The *progress* parameter shall report on current file transmission or reception progress, as defined earlier in 1.3.3.3.

**3.2.22** If included, the optional *closure requested* parameter shall override the 'transaction closure requested' setting in the MIB. The values of the closure requested parameter are 'true' (indicating that transaction closure is requested) or 'false' (indicating that transaction closure is not requested).

**3.2.23** The file size parameter shall indicate the length in octets of the file that is being conveyed by means of the indicated transaction.

**3.2.24** The optional *segment metadata* parameter shall comprise from 0 to 63 octets of metadata pertaining to a received file segment. The length and nature of this metadata are an implementation matter.

**3.2.25** The optional *segment metadata length* parameter shall indicate the number of octets of segment-specific metadata contained in a received file segment.

**3.2.26** The optional *record continuation state* parameter shall indicate the manner in which the content of a received file segment aligns with the record structure of the file. The value of this parameter shall indicate one of the following conditions:

– the first octet of the segment is the first octet of a record, and the end of that record is not contained in this segment;

– the last octet of the segment is the last octet of a record, and the beginning of that record is not contained in this segment;

– the first octet of the segment is the first octet of a record, and the last octet of the segment is the last octet of a record (either the same record or a subsequent record);

– the segment continues a record; that is, the segment contains neither the first octet nor the last octet of any record;

– the file has no record structure.

## 3.3 SERVICES REQUIRED OF THE UNDERLYING COMMUNICATION SYSTEM

The service primitives and parameters required to access the services of the Unitdata Transfer (UT) layer are:

| | |
|---|---|
| `UNITDATA.request` | (UT_SDU, UT Address) |
| `UNITDATA.indication` | (UT_SDU, UT Address) |

NOTES

1 The service assumed of the underlying layer is as simple as possible to allow CFDP to be as generally applicable as possible. For the purposes of this specification, the service is referred to as the UT service, and the delimited data unit it transfers is the UT Service Data Unit (UT_SDU). Although the UT layer is conceptually a single service, in practice, CFDP might use several different underlying communication systems at different times or even concurrently, depending on the remote entities with which it is in communication. The sole service may, for example, be the CCSDS Encapsulation service, and the UT_SDU would be the data field of an Encapsulation packet. Alternatively, the sole service may be the Delay-Tolerant Networking (DTN) Bundle Protocol, in which case the UT_SDU would be the payload of a DTN bundle.

2 The format and contents of the UT Address parameter depend on the addressing capabilities and conventions of the underlying service. Information in the MIB must enable translation between CFDP entity names and the corresponding UT addresses.

3 CFDP operates over a single conceptual UT service access point. In practice, the implementation of the UT layer is not constrained to provide all services through a single service access point. Different service access points would necessarily be provided if different underlying communication systems were in use concurrently. A given implementation might even maintain a different service access point for each remote entity with which CFDP is currently (at any moment) in direct communication.

4 The above list of mandatory UT-layer service primitives does not imply that the UT layer is prohibited from supporting additional services for the convenience of the CFDP implementation. Possible additional primitives that might prove valuable include:

– An indication whenever the UT layer is prepared to accept another PDU for transmission to the UT layer instance at an identified UT address. (This indication could be used to implement data flow control. Destination UT address might be needed in the indication because the UT layer might be transmitting concurrently to multiple UT addresses.)

– An indication whenever the UT layer completes transmission of a UT_SDU whose content is a CFDP PDU to which Positive Acknowledgment procedures apply. (This indication could be used to start positive acknowledgment timers.)

5 The assumed minimum underlying quality of service is:

– with possible errors in the delivered UT_SDUs;

– incomplete, with some UT_SDUs missing;

– in sequence; that is, the delivered UT_SDUs are delivered in the order in which they were transmitted. (In-sequence arrival is specifically assumed in the design of the immediate, asynchronous, and prompted Lost Segment Detection modes and the Keep Alive mechanism.)

CFDP can operate over a service in which data errors and data loss occur; and when no Incremental Lost Segment Detection procedures are in effect, even out-of-sequence data arrival can be tolerated. However, the protocol is not intended to compensate for networks in which these effects are prevalent. Severe performance reductions will result if such an approach is taken.

6 CFDP includes no security mechanisms of any kind but is of course subject to the same confidentiality, data integrity, authentication, and availability concerns as any other protocol. Given the absence of security mechanisms built into CFDP itself, the UT-layer protocol stack over which CFDP functions needs to provide whatever services are required in order to ensure the secure operation of the protocol as deployed. The details of these services are necessarily opaque to CFDP and are beyond the scope of this Recommendation.

## 3.4 CFDP SERVICE PRIMITIVES

### 3.4.1 `Put.request`

#### 3.4.1.1 Function

The **`Put.request`** primitive shall be used by the application to request delivery of a file from the source filestore to a destination filestore.

#### 3.4.1.2 Semantics

**`Put.request`** shall provide parameters as follows:

```
Put.request    (destination CFDP entity ID,
               [source file name],
               [destination file name],
               [segmentation control],
               [fault handler overrides],
               [flow label],
               [transmission mode],
               [closure requested],
               [messages to user],
               [filestore requests])
```

#### 3.4.1.3 When Generated

**`Put.request`** is generated by the source CFDP user at any time.

#### 3.4.1.4 Effect on Receipt

Receipt of **`Put.request`** shall cause the CFDP entity to initiate source entity put procedures.

#### 3.4.1.5 Additional Comments

None.

## 3.4.2 `Cancel.request`

### 3.4.2.1 Function

The `Cancel.request` primitive shall be used to request that a transaction be canceled.

### 3.4.2.2 Semantics

`Cancel.request` shall provide parameters as follows:

`Cancel.request` (transaction ID)

### 3.4.2.3 When Generated

`Cancel.request` is generated by any CFDP user at any time during the lifetime of any transaction.

### 3.4.2.4 Effect on Receipt

Receipt of `Cancel.request` shall cause a Notice of Cancellation at the local entity.

### 3.4.2.5 Additional Comments

None.

### 3.4.3 `Suspend.request`

#### 3.4.3.1 Function

The `Suspend.request` primitive shall be used to request that a transaction be suspended.

#### 3.4.3.2 Semantics

`Suspend.request` shall provide parameters as follows:

`Suspend.request` (transaction ID)

#### 3.4.3.3 When Generated

`Suspend.request` is generated by any CFDP user at any time during the lifetime of any transaction, except in the case of a transaction sent in unacknowledged mode, for which it can only be generated by the transaction's source user.

#### 3.4.3.4 Effect on Receipt

Receipt of `Suspend.request` shall cause a Notice of Suspension at the local entity.

#### 3.4.3.5 Additional Comments

None.

### 3.4.4 `Resume.request`

#### 3.4.4.1 Function

The `Resume.request` primitive shall be used to request that a suspended transaction be resumed.

#### 3.4.4.2 Semantics

`Resume.request` shall provide parameters as follows:

`Resume.request` (transaction ID)

#### 3.4.4.3 When Generated

`Resume.request` is generated by any CFDP user at any time during the lifetime of any transaction, except in the case of a transaction sent in unacknowledged mode, for which it can only be generated by the transaction's source user.

#### 3.4.4.4 Effect on Receipt

Receipt of `Resume.request` shall have no effect on any transaction that is not currently suspended. For a transaction that is currently suspended, it shall cause the local CFDP entity to initiate Resume procedures.

#### 3.4.4.5 Additional Comments

User application developers may find it advisable to provide automated support for one special case of transaction resumption, which applies to any transaction the user application can determine is currently suspended solely due to transaction inactivity (i.e., as a result of an Inactivity fault). On receipt of any service indication produced because of arrival of a PDU for such a transaction, the user application might infer that the transaction's inactivity is ended and thereupon automatically resume the transaction.

### 3.4.5 `Report.request`

#### 3.4.5.1 Function

The `Report.request` primitive shall be used to request a report on the status of a transaction.

#### 3.4.5.2 Semantics

`Report.request` shall provide parameters as follows:

`Report.request` (transaction ID)

#### 3.4.5.3 When Generated

`Report.request` is generated by any CFDP user at any time during the lifetime of any transaction.

#### 3.4.5.4 Effect on Receipt

Receipt of `Report.request` shall cause the CFDP entity to return a `Report.indication` primitive to the CFDP user.

#### 3.4.5.5 Additional Comments

None.

### 3.4.6 `Transaction.indication`

#### 3.4.6.1 Function

The `Transaction.indication` primitive shall be used to indicate the Transaction identifier to the source CFDP user.

#### 3.4.6.2 Semantics

`Transaction.indication` shall provide parameters as follows:

`Transaction.indication` (transaction ID)

#### 3.4.6.3 When Generated

`Transaction.indication` shall be generated on receipt of a `Put.request` primitive.

#### 3.4.6.4 Effect on Receipt

The effect on receipt of `Transaction.indication` by a CFDP user is undefined.

#### 3.4.6.5 Additional Comments

The Transaction ID parameter is returned by the CFDP entity and provides the source CFDP user with a means of uniquely identifying the associated transaction thereafter.

## 3.4.7 `EOF-Sent.indication`

### 3.4.7.1 Function

The **`EOF-Sent.indication`** primitive shall be used to notify the source CFDP user of the initial transmission of a transaction's EOF PDU.

### 3.4.7.2 Semantics

**`EOF-Sent.indication`** shall provide parameters as follows:

**`EOF-Sent.indication`** (transaction ID)

### 3.4.7.3 When Generated

**`EOF-Sent.indication`** shall be generated on initial transmission of an EOF PDU, at the conclusion of initial transmission of a transaction's metadata and file data.

### 3.4.7.4 Effect on Receipt

The effect on receipt of **`EOF-Sent.indication`** by a CFDP user is undefined.

### 3.4.7.5 Additional Comments

Generation of **`EOF-Sent.indication`** is optional. The user may choose to base a flow control system in part on this mechanism, for example, to refrain from submitting a new **`Put.request`** primitive until the **`EOF-Sent.indication`** resulting from the previous one has been received.

### 3.4.8 `Transaction-Finished.indication`

#### 3.4.8.1 Function

The `Transaction-Finished.indication` primitive shall be used to indicate to the source or destination CFDP user that the transaction is complete and that the source CFDP user is authorized to modify or delete any retransmission buffer (file) that it was sharing with the protocol entity to conserve persistent storage space.

#### 3.4.8.2 Semantics

`Transaction-Finished.indication` shall provide parameters as follows:

| | |
|---|---|
| `Transaction-Finished.indication` | (transaction ID,<br>[filestore responses],<br>[status report],<br>condition code,<br>file status,<br>delivery code) |

#### 3.4.8.3 When Generated

`Transaction-Finished.indication` is generated on Notice of Completion of a file transmission procedure either at the source CFDP entity or, optionally, at the destination CFDP entity.

#### 3.4.8.4 Effect on Receipt

The effect on receipt of `Transaction-Finished.indication` by an application is undefined.

#### 3.4.8.5 Additional Comments

`Transaction-Finished.indication` is always generated by the source CFDP entity and can also be generated by the destination CFDP entity. The condition code indicates the condition under which the transaction was finished.

### 3.4.9 `Metadata-Recv.indication`

#### 3.4.9.1 Function

The **`Metadata-Recv.indication`** primitive shall be used to indicate to the destination CFDP user that the metadata associated with a transaction has been received.

#### 3.4.9.2 Semantics

**`Metadata-Recv.indication`** shall provide parameters as follows:

| | |
|---|---|
| **`Metadata-Recv.indication`** | (transaction ID,<br>source CFDP entity ID,<br>[file size],<br>[source file name],<br>[destination file name],<br>[messages to user]) |

#### 3.4.9.3 When Generated

**`Metadata-Recv.indication`** is generated on receipt of a Metadata PDU.

#### 3.4.9.4 Effect on Receipt

The effect on receipt of **`Metadata-Recv.indication`** by the destination CFDP user is undefined.

#### 3.4.9.5 Additional Comments

None.

### 3.4.10 `File-Segment-Recv.indication`

#### 3.4.10.1 Function

The `File-Segment-Recv.indication` primitive shall be used to indicate the receipt of individual file data (non-metadata) segments to the destination CFDP user.

#### 3.4.10.2 Semantics

`File-Segment-Recv.indication` shall provide parameters as follows:

| | |
|---|---|
| `File-Segment-Recv.indication` | (transaction ID,<br>offset,<br>length,<br>[record continuation state,<br>length of segment metadata,<br>segment metadata]) |

#### 3.4.10.3 When Generated

`File-Segment-Recv.indication` is generated on receipt of a File Data PDU.

#### 3.4.10.4 Effect on Receipt

The effect on receipt of `File-Segment-Recv.indication` by a CFDP user is undefined.

#### 3.4.10.5 Additional Comments

Generation of `File-Segment-Recv.indication` is optional. Offset is the number of octets in the file prior to the first octet of the received File Data PDU's content.

### 3.4.11 `Suspended.indication`

#### 3.4.11.1 Function

The **`Suspended.indication`** primitive shall be used to indicate to the CFDP user that the transaction has been suspended.

#### 3.4.11.2 Semantics

**`Suspended.indication`** shall provide parameters as follows:

| | |
|---|---|
| **`Suspended.indication`** | (transaction ID,<br>condition code) |

#### 3.4.11.3 When Generated

**`Suspended.indication`** is generated on Notice of Suspension of a file transmission procedure.

#### 3.4.11.4 Effect on Receipt

The effect on receipt of **`Suspended.indication`** by a CFDP user is undefined.

#### 3.4.11.5 Additional Comments

None.

### 3.4.12 `Resumed.indication`

#### 3.4.12.1 Function

The **`Resumed.indication`** primitive shall be used to indicate to the CFDP user that the transaction has been resumed.

#### 3.4.12.2 Semantics

**`Resumed.indication`** shall provide parameters as follows:

| | |
|---|---|
| **`Resumed.indication`** | (transaction ID,<br>progress) |

#### 3.4.12.3 When Generated

**`Resumed.indication`** shall be generated in response to **`Resume.request`**.

#### 3.4.12.4 Effect on Receipt

The effect on receipt of **`Resumed.indication`** by a CFDP user is undefined.

#### 3.4.12.5 Additional Comments

Progress indicates how far the issuing CFDP entity had progressed, in sending or receiving the transaction's transmitted file, as of the moment at which the **`Resumed.indication`** was generated.

### 3.4.13 `Report.indication`

#### 3.4.13.1 Function

The `Report.indication` primitive shall be used to report to the CFDP user on the status of a transaction.

#### 3.4.13.2 Semantics

`Report.indication` shall provide parameters as follows:

```
Report.indication   (transaction ID,
                    status report)
```

#### 3.4.13.3 When Generated

`Report.indication` shall be generated on receipt of the `Report.request` primitive.

#### 3.4.13.4 Effect on Receipt

The effect on receipt of `Report.indication` by a CFDP user is undefined.

#### 3.4.13.5 Additional Comments

The format and scope of the *status report* parameter are specific to the implementation.

### 3.4.14 `Fault.indication`

#### 3.4.14.1 Function

The `Fault.indication` primitive shall be used to indicate to the CFDP user the occurrence of a fault condition for which the designated fault handler was 'Ignore'.

#### 3.4.14.2 Semantics

`Fault.indication` shall provide parameters as follows:

| | |
|---|---|
| `Fault.indication` | (transaction ID,<br>condition code,<br>progress) |

#### 3.4.14.3 When Generated

`Fault.indication` shall be generated upon detection of a fault condition for which the designated fault handler is 'Ignore'.

#### 3.4.14.4 Effect on Receipt

The effect on receipt of `Fault.indication` by a CFDP user is undefined.

#### 3.4.14.5 Additional Comments

The progress parameter indicates how far the issuing CFDP entity had progressed, in sending or receiving the transaction's transmitted file, as of the moment at which the `Fault.indication` was generated.

### 3.4.15 `Abandoned.indication`

#### 3.4.15.1 Function

The **`Abandoned.indication`** primitive shall be used to indicate to the CFDP user the occurrence of a fault condition for which the designated fault handler was ‘Abandon’.

#### 3.4.15.2 Semantics

**`Abandoned.indication`** shall provide parameters as follows:

```
Abandoned.indication  (transaction ID,
                       condition code,
                       progress)
```

#### 3.4.15.3 When Generated

**`Abandoned.indication`** shall be generated upon detection of a fault condition for which the designated fault handler is ‘Abandon’.

#### 3.4.15.4 Effect on Receipt

The effect on receipt of **`Abandoned.indication`** by a CFDP user is undefined.

#### 3.4.15.5 Additional Comments

The progress parameter indicates how far the issuing CFDP entity had progressed in sending or receiving the transaction’s transmitted file as of the moment when the **`Abandoned.indication`** was generated.

### 3.4.16 `EOF-Recv.indication`

#### 3.4.16.1 Function

The `EOF-Recv.indication` primitive shall be used to indicate to the destination CFDP user that the EOF PDU associated with a transaction has been received.

#### 3.4.16.2 Semantics

`EOF-Recv.indication` shall provide parameters as follows:

`EOF-Recv.indication` (transaction ID)

#### 3.4.16.3 When Generated

`EOF-Recv.indication` is generated on initial receipt, at the destination entity of a transaction, of the EOF PDU for the transaction.

#### 3.4.16.4 Effect on Receipt

The effect on receipt of `EOF-Recv.indication` by the destination CFDP user is undefined.

#### 3.4.16.5 Comments

Generation of `EOF-Recv.indication` is optional.

# 4 PROTOCOL SPECIFICATION

## 4.1 CRC PROCEDURES

### 4.1.1 CRC PROCEDURES AT THE PDU TRANSMITTING ENTITY

If the CRC option is active, the PDU-transmitting entity shall set the CRC flag to 'true' and calculate and insert the CRC for each outgoing PDU.

### 4.1.2 CRC PROCEDURES AT THE PDU RECEIVING ENTITY

If the CRC flag is set to 'true' in the incoming PDU, the PDU-receiving entity shall calculate the CRC and discard the PDU if it fails the CRC validation procedure.

### 4.1.3 CRC VALIDATION PROCEDURE

**4.1.3.1** The CRC computation algorithm shall be the standard CCSDS Telecommand CRC algorithm specified in 4.2.1.3 of the CCSDS Telecommand Recommendation (reference [4]).

**4.1.3.2** The CRC value shall be placed in the final octets of the PDU data field, and its length shall be included in the PDU data field length. The CRC algorithm shall be applied from the first octet of the PDU header to the last octet of the PDU data field prior to the CRC value.

## 4.2 CHECKSUM PROCEDURES

### 4.2.1 GENERAL

**4.2.1.1** Each file conveyed by CFDP shall be accompanied by a checksum, the purpose of which is to protect the integrity of the file.

NOTE – In the event that the checksum accompanying a given file is computed by means of the null checksum defined in 4.2.2.4 below, the checksum will not protect the integrity of the file.

**4.2.1.2** The checksum shall be 32 bits in length.

### 4.2.2 CHECKSUM COMPUTATION PROCEDURE

**4.2.2.1** The checksum shall be the result of executing the applicable checksum computation algorithm, which shall be determined as described below.

**4.2.2.2** The set of *available* checksum computation algorithms shall comprise all checksum computation algorithms that

a) are implemented by the checksum computing entity; and

b) are also among the first 16 algorithms enumerated in the SANA Checksum Identifiers registry (reference [10]).

**4.2.2.3** The modular checksum computation algorithm (identified by checksum type 0) described below shall be implemented.

**4.2.2.4** The null checksum algorithm (identified by checksum type 15) shall be implemented. The null checksum algorithm shall be simply to set the value of the checksum to zero.

**4.2.2.5** Checksum algorithms identified by checksum types 1 through 14 may be implemented.

NOTE – It is recommended that one of the checksum computation algorithms that are stronger than the modular checksum algorithm be implemented for improved protection of the integrity of large files, as required.

**4.2.2.6** The *preferred* checksum computation algorithm shall be determined as described in 4.2.3.2 and 4.2.4.2, depending on whether the checksum is to be computed by the sending entity or by the receiving entity.

**4.2.2.7** If the preferred checksum computation algorithm is one of the available checksum computation algorithms, then the applicable checksum computation algorithm shall be the preferred checksum computation algorithm.

**4.2.2.8** If the preferred checksum computation algorithm is not one of the available checksum computation algorithms, an Unsupported Checksum Type fault shall be raised, and the applicable checksum computation algorithm shall be selected as follows:

**4.2.2.8.1** If the checksum is to be computed by the sending entity, the applicable checksum computation algorithm shall be the modular checksum calculation algorithm defined in 4.2.5.

**4.2.2.8.2** If the checksum is to be computed by the receiving entity, the applicable checksum computation algorithm shall be the null checksum algorithm defined in 4.2.2.4.

## 4.2.3 CHECKSUM ASSERTION PROCEDURE

**4.2.3.1** The checksum shall be computed over the file data and inserted into the EOF (No error) PDU by the sending entity.

**4.2.3.2** For checksum computation at the sending entity, the preferred checksum computation algorithm shall be the algorithm identified by the checksum type specified in the MIB remote entity configuration information pertaining to the receiving entity unless overridden for mission purposes, an implementation matter.

### 4.2.4 CHECKSUM VERIFICATION PROCEDURE

**4.2.4.1** The checksum shall be recomputed and verified by the receiving entity.

**4.2.4.2** For checksum computation at the receiving entity, the preferred checksum computation algorithm shall be the algorithm identified by the checksum type specified in the Metadata PDU.

### 4.2.5 MODULAR CHECKSUM

The modular checksum (identified by checksum type zero) shall be calculated by the following method (see annex F for an example):

a) the checksum shall initially be set to all 'zeroes';

b) it shall be calculated by modulo $2^{32}$ addition of all 4-octet words, aligned from the start of the file;

c) each 4-octet word shall be constructed by copying some octet of file data, whose offset within the file is an integral multiple of 4 (such as 0, 4, 8, 12, etc.), into the first (high-order) octet of the word and copying the next three octets of file data into the next three octets of the word;

d) the results of the addition shall be carried into each available octet of the checksum unless the addition overflows the checksum length, in which case, carry shall be discarded.

NOTES

1 In order to include in a checksum the content of a file-data PDU whose offset is not an integral multiple of 4, it is necessary to align the data properly before adding 4-octet blocks of it to the checksum. Data at offset $Q$ may be aligned by inserting $N$ octets of value 'zero' before the first octet of the data, where $N = Q \bmod 4$ (the remainder obtained upon dividing $Q$ by 4).

2 In order to include in a checksum a sequence of $M$ octets (the first of which is at a file offset that is an integral multiple of 4) where $M$ is less than 4, it is necessary to pad the data to length 4 before adding it to the checksum. The data may be padded by inserting $(4 - M)$ octets of value 'zero' after the last octet of the data. This condition can apply only at the end of the file.

## 4.3 PUT PROCEDURES

Receipt of a `Put.request` primitive shall cause the source CFDP entity to initiate

a) a Transaction Start Notification procedure; and

b) a Copy File procedure, for which

   1) any fault handler overrides shall be derived from the `Put.request`,

   2) any Messages to User or filestore requests shall be derived from the `Put.request`,

   3) omission of source and destination filenames shall indicate that only metadata will be delivered,

   4) the transmission mode (acknowledged or unacknowledged) is defined by the existing content of the MIB unless overridden by the transmission mode parameter of the `Put.request`,

   5) the transaction closure requested indication (true or false) is defined by the existing content of the MIB unless overridden by the closure requested parameter of the Put.request.

## 4.4 TRANSACTION START NOTIFICATION PROCEDURE

**4.4.1** The transaction identifier shall be

a) issued sequentially by the source CFDP entity in response to a service request; and

b) used for all subsequent references to the transaction.

**4.4.2** No two transactions for which there is any contemporaneous protocol activity may have the same transaction identifier.

NOTE – Duplicate transaction identifiers will inevitably result in erroneous protocol behavior.

**4.4.3** The source CFDP entity shall issue a `Transaction.indication` primitive that notifies the source CFDP user of the transaction identifier.

## 4.5 PDU FORWARDING PROCEDURES

**4.5.1** Each outgoing PDU shall be directed to the appropriate CFDP entity using the addressing capabilities of the underlying Unitdata Transfer service.

**4.5.2** The remote Unitdata Transfer (UT) SAP to which the PDU is to be directed shall be obtained from the MIB, based on the identifier of the CFDP entity to which the PDU is to be delivered.

**4.5.3** Depending on the type of PDU, the identifier of the CFDP entity to which the PDU is to be delivered shall be one of the following:

a) for an ACK PDU, the ID of the CFDP entity that sent the PDU that is being acknowledged;

b) for a File Data, Metadata, EOF, or Prompt PDU, the receiver entity ID for that file copy operation;

c) for a NAK, Finished, or Keep Alive PDU, the sender entity ID for that file copy operation.

NOTE – The sender and receiver of the file are the source and destination entities as identified in the PDU header.

**4.5.4** Any PDU whose destination entity ID differs from the identifier of the CFDP entity to which it is delivered shall be discarded.

## 4.6 COPY FILE PROCEDURES

NOTES

1 In the following subsections, the name of an EOF PDU or a Finished PDU includes, in parentheses, either the name of the specific condition code that the PDU carries (such as ‘No error’), the general condition name ‘complete’ indicating that the PDU carries a condition code that signals successful transaction completion (either ‘No error’ or ‘Unsupported checksum type’), or the general condition name ‘cancel’ indicating that the PDU carries some condition code other than ‘No error’, ‘Unsupported checksum type’, or ‘Suspend request received’.

2 The name of an ACK PDU includes, in parentheses, the type of PDU that is being acknowledged: EOF or Finished.

3 The sender of the file is always the transaction’s source entity, and the receiver of the file is always the transaction’s destination entity.

### 4.6.1 GENERAL PROCEDURES

#### 4.6.1.1 Copy File Procedures at the Sending Entity

**4.6.1.1.1** If the Transaction mode is unacknowledged, unacknowledged procedures shall apply at the sending entity. If the Transaction mode is acknowledged, acknowledged procedures shall apply at the sending entity.

**4.6.1.1.2** Initiation of the Copy File procedures shall cause the sending CFDP entity to forward a Metadata PDU to the receiving CFDP entity.

**4.6.1.1.3** The Metadata PDU shall contain

a) the size of the file if known (i.e., for a bounded file);

b) the source and destination names (path names) of the file;

c) optional fault handler overrides, Messages to User, filestore requests, and/or flow label;

d) an indication of whether or not transaction closure is requested;

e) the checksum type number identifying the algorithm used to compute the checksum for this file.

NOTE – Assuring that all relevant metadata for the transaction are contained within a single Metadata PDU is an implementation responsibility.

**4.6.1.1.4** For transactions that deliver more than just metadata, Copy File initiation also shall cause the sending CFDP entity to retrieve the file from the sending filestore and to transmit it in File Data PDUs.

**4.6.1.1.5** Each segment of data shall be forwarded in a File Data PDU, which also shall contain the offset, in octets, from the beginning of the file at which the segment is located.

**4.6.1.1.5.1** A File Data PDU may additionally contain information characterizing the manner in which the PDU's content aligns with the record structure of the file. This information, termed 'record continuation state', shall be included whenever the segmentation control service parameter has requested that record boundaries be respected; when observance of record boundaries has not been requested, record continuation state information may optionally be included, but the record continuation state shall always be 00. When record continuation state is included in a File Data PDU, the Segment Metadata flag in the PDU header shall be set to 1, the optional segment administrative octet shall be present, and the first (high-order) 2 bits of the administrative octet shall indicate the record continuation state as defined in 5.3 below.

**4.6.1.1.5.2** In addition, the File Data PDU may contain *N* octets of segment metadata, where *N* is in the range 0 to 63.

NOTE – The decision on whether or not to include segment metadata in a File Data PDU and, if so, the length and nature of that metadata is a local implementation matter.

**4.6.1.1.5.3** Whenever segment metadata is included in a File Data PDU, the Segment Metadata flag in the PDU header shall be set to '1', and the optional segment administrative octet shall be present, immediately followed by the number of segment metadata octets indicated by the integer value expressed in the last (low-order) 6 bits of the administrative octet.

**4.6.1.1.5.4** Record continuation state and segment metadata are not mutually exclusive: either one, both, or neither may be present in any File Data PDU.

**4.6.1.1.6** If the *segmentation control* service parameter has requested that record boundaries be respected, the Segmentation Control flag of the header of each File Data PDU shall be set to '1', and record continuation state shall be included in every File Data PDU, as discussed above. Multiple complete records may occupy a single segment.

NOTE – The method by which record boundaries in a file are detected is a local implementation matter. Agreement on such a method is not necessary for interoperability among CFDP entities. (However, it may be required for the correct operation of the source and destination CFDP user applications.)

**4.6.1.1.7** If the *segmentation control* service parameter was omitted, record boundaries shall by default not be respected. When record boundaries are not respected, either by default or by explicit request in the Put.request primitive, the Segmentation Control flag of the header of each File Data PDU shall be set to '0'.

NOTE – The segmentation algorithm is otherwise not specified in this Recommendation.

**4.6.1.1.8** If the *segmentation control* service parameter has requested that record boundaries be respected and there is no observable record structure to the file, an Invalid File Structure fault shall be declared.

NOTE – The algorithm for observing record structure in a file is not specified in this Recommendation. It is an implementation matter at the sender and is not relevant at the receiver.

**4.6.1.1.9** When the Metadata PDU and all File Data PDUs have been issued (that is, when either (A) the file to be sent is bounded and the last octet of that file has been sent, (B) the file to be sent is unbounded and the transmission session for that file has concluded, or (C) no file is to be sent),

a) the sending CFDP entity shall issue an EOF (No error) PDU;

b) the sending CFDP entity may optionally issue an `EOF-Sent.indication`, depending on a setting in the MIB;

c) the EOF (No error) PDU shall contain the checksum of the original file and the length of the file (which might not have been known at the time the Metadata PDU was issued).

**4.6.1.1.10** The flow label may optionally be used to support prioritization and preemption schemes.

NOTE – The use of the flow label is implementation specific.

### 4.6.1.2 Copy File Procedures at the Receiving Entity

**4.6.1.2.1** If the Transaction mode is unacknowledged, unacknowledged procedures shall apply at the receiving entity. If the Transaction mode is acknowledged, acknowledged procedures shall apply at the receiving entity.

**4.6.1.2.2** The receiving CFDP entity shall store the value of the transmission mode bit contained in the first received PDU of a transaction and use it for subsequent processing of the transaction. If the receiving CFDP entity is incapable of operating in this mode, an Invalid Transmission Mode fault shall be declared.

**4.6.1.2.3** The receiving CFDP entity shall store fault handler overrides, file size, flow label, and file name information contained in the Metadata PDU and use it for subsequent processing of the transaction.

**4.6.1.2.4** Any repeated Metadata PDU shall be discarded.

**4.6.1.2.5** For transactions that deliver more than just metadata, if and when the receiving CFDP entity determines that the filestore cannot accept the file, a Filestore Rejection fault shall be declared.

NOTE – Determination of filestore inability to accept a file is implementation specific.

**4.6.1.2.6** If the receiver is the transaction's destination, receipt of a Metadata PDU shall cause the destination entity to issue a **`Metadata-Recv.indication`**:

a) if the Metadata PDU contains one or more messages to user, they shall be passed as a parameter of the **`Metadata-Recv.indication`**;

b) if the Metadata PDU contains filestore requests, these shall be stored for execution at the conclusion of delivery of the transaction's file data, if any.

**4.6.1.2.7** On reception of File Data PDUs by the receiver, the file data shall be assembled into a file using the offset and segment size information of the File Data PDU:

a) any repeated data shall be discarded;

b) if the segmentation control flag indicates that record boundaries have been respected, the alignment of records in the received File Data PDU may be used to assist in record reconstruction;

c) if the sum of the File Data PDU's offset and segment size exceeds the file size indicated in the first previously received EOF (No error) PDU, if any, then a File Size Error fault shall be declared.

**4.6.1.2.8** At the earliest time at which the transaction's Metadata, File Data (if any), and EOF (No error) PDUs have all been received by the receiving entity:

a) a checksum shall be calculated for the delivered file by means of the applicable checksum algorithm, determined as described in 4.2.2 above;

b) the calculated and received file checksums shall be compared;

c) if the compared checksums are equal or the applicable checksum algorithm is the null checksum algorithm, *file delivery shall be deemed Complete*;

d) Otherwise, a File Checksum Failure fault shall be declared.

NOTE – The action taken upon such error need not necessarily entail discarding the delivered file. The default handler for File Checksum Failure faults may be Ignore, causing the discrepancy to be announced to the user in a **`Fault.indication`** but permitting the completion of the Copy File procedure at the receiving entity. This configuration setting might be especially appropriate for transactions conducted in unacknowledged mode.

**4.6.1.2.9** Upon initial receipt of the EOF (No error) PDU, the file size indicated in the PDU shall be compared to the transaction reception progress and a File Size Error fault declared if the progress exceeds the file size.

**4.6.1.2.10** The flow label may optionally be used to support prioritization and preemption schemes.

NOTE – The use of the flow label is implementation specific.

## 4.6.2 OPTIONAL COPY FILE PROCEDURES AT THE RECEIVING ENTITY

Receipt of a File Data PDU may optionally cause the receiving CFDP, if it is the transaction's destination, to issue a **`File-Segment-Recv.indication`**. Initial receipt of the EOF PDU for a transaction may optionally cause the receiving CFDP, if it is the transaction's destination, to issue an **`EOF-Recv.indication.`**

## 4.6.3 UNACKNOWLEDGED MODE PROCEDURES

### 4.6.3.1 Transmission Paths

Unacknowledged mode procedures may be exercised over simplex transmission paths unless the transaction closure option is invoked, in which case duplex transmission paths are required.

### 4.6.3.2 Unacknowledged Mode Procedures at the Sending Entity

**4.6.3.2.1** Transmission of an EOF (No error) PDU shall cause the sending CFDP entity to issue a Notice of Completion (Completed) unless transaction closure was requested.

**4.6.3.2.2** In the latter case, a transaction-specific Check timer shall be started. The expiry period of the timer shall be determined in an implementation-specific manner.

**4.6.3.2.3** Reception of a Finished PDU shall cause

a) the Check timer of the associated transaction to be turned off; and

b) a Notice of Completion (either Completed or Canceled, depending on the nature of the Finished PDU) to be issued.

**4.6.3.2.4** If the timer expires prior to reception of a Finished PDU for the associated transaction, a Check Limit Reached fault shall be declared.

### 4.6.3.3 Unacknowledged Mode Procedures at the Receiving Entity

Upon receipt of an EOF (No error) PDU:

a) If file reception is deemed complete as explained in 4.6.1.2.8 above, the receiving CFDP entity shall issue a Notice of Completion (Completed). Moreover, if the Closure Requested flag in the transaction’s Metadata PDU is set to ‘1’ and the receiving entity is the transaction’s destination, the receiving CFDP entity shall issue a Finished complete) PDU. Any filestore responses shall be carried as parameters of the Finished (complete) PDU. The condition code in the Finished (complete) PDU shall be ‘No error’ if file reception was deemed complete following determination that the calculated and received checksums were equal, but ‘Unsupported checksum type’ otherwise.

b) Otherwise, a transaction-specific Check timer shall be started. The timer shall have an implementation-specific expiry period, and there shall be an implementation-specific limit on the number of times the Check timer for any single transaction may expire. When the timer expires, the receiving entity shall determine whether or not file reception is now deemed complete. If so, the receiving entity shall issue a Notice of Completion (Completed) and, if the Closure Requested flag in the transaction’s Metadata PDU is set to ‘1’ and the receiving entity is the transaction’s destination, shall additionally issue a Finished (complete) PDU as described above. Otherwise, if the Check timer expiration limit has been reached, then a Check Limit Reached fault shall be declared; otherwise the Check timer shall be reset.

NOTE – Although it is expected that the PDUs of a transaction in Unacknowledged mode will normally arrive in the order in which they were transmitted, and therefore normally no File Data PDUs will arrive following reception of the EOF (No error) PDU, this expectation may be incorrect in some circumstances; for example, the PDUs might be traversing a network which might send packets over multiple routes, potentially reordering their arrival. The Check timer is intended to ensure successful transaction completion under such conditions.

## 4.6.4 ACKNOWLEDGED MODE PROCEDURES

### 4.6.4.1 Transmission Path

Acknowledged mode procedures may be exercised only over duplex transmission paths.

### 4.6.4.2 Acknowledged Mode Procedures at the Sending Entity

**4.6.4.2.1** The sending CFDP entity shall respond to all received NAK PDUs by retransmitting the requested Metadata PDU and/or the extents of the data file defined by the start and end offsets of the segment requests in the NAK PDU.

NOTE – The retransmission buffer for a File Copy procedure may be private to the protocol or (for greater efficiency in non-volatile storage space management) may be a publicly visible file. This is an implementation matter, as is the manner in which the buffer is released.

**4.6.4.2.2** Segmentation of the retransmitted data shall not be constrained by the segmentation of the original transmission or by the start and end offsets in the NAK PDU. The constraints imposed by the segmentation control parameter, however, remain in effect.

**4.6.4.2.3** Receipt of the Finished (complete) PDU shall cause the sending CFDP entity to issue a Notice of Completion (Completed).

**4.6.4.2.4** Positive Acknowledgment procedures shall be applied to the EOF (No error) and Finished (complete) PDUs with the Expected Responses being ACK (EOF) PDU and ACK (Finished) PDU, respectively.

### 4.6.4.3 Acknowledged Mode Procedures at the Receiving Entity

**4.6.4.3.1** A file data gap shall be detected when any of the following conditions is true:

a) the offset of the first octet of the first extent of received file data is not zero;

b) the offset of the last octet of some contiguous extent of file data differs from the offset of the first octet of the next contiguous extent of data of the same file by more than one;

c) the offset of the last octet of the last extent of file data differs from the length of the file as stated in the EOF (No error) PDU by more than one.

**4.6.4.3.2** Lost metadata shall be detected upon receipt of the first non-Metadata PDU (File Data or EOF (No error)) of a transaction for which no Metadata PDU has yet been received.

**4.6.4.3.3** In accordance with any Lost Segment Detection procedures that are in effect, the receiving CFDP entity shall issue NAK PDUs as follows:

a) Each NAK PDU shall identify the subset of file data to which it pertains, that is, the *scope* of the NAK PDU. The scope of a NAK PDU is expressed as two offsets within the file, indicating the start and end of the scope.

b) In addition to its scope, each NAK PDU shall contain zero or more segment requests. The segment request(s) in a NAK PDU shall identify the start offsets and end offsets of all extents of file data within its scope that have not yet been received, and shall also identify missing metadata, if any.

c) The start offsets and end offsets of segment requests do not need to relate in any way to the original segmentation of the file.

d) One *NAK sequence* shall be issued for each event upon which gap reporting is deemed necessary by any Lost Segment Detection procedures that are in effect.

   1) Each NAK sequence shall pertain to some subset of file data, that is, the *scope* of the NAK sequence.

   2) The start of scope for a NAK sequence shall be the end of scope of the previous NAK sequence issued for the same transaction, or zero if there has been no prior NAK sequence or if the event causing issuance of the NAK sequence is expiration of a transaction-specific NAK timer, as described later.

   3) The end of scope for a NAK sequence shall be the entire length of the file if the transaction's EOF (No error) PDU has been received. Otherwise, it shall be the current reception progress at the time of the event that caused issuance of the NAK sequence.

   4) Each NAK sequence shall comprise as many NAK PDUs as are required, taking into consideration any limits on the maximum size of the content of a UT_SDU, to contain all the segment requests that must be sent in order to request retransmission of all the missing file data and metadata within the scope of the NAK sequence.

   NOTE – Normally each NAK sequence will comprise only a single NAK PDU.

   5) The start of scope for the first NAK PDU in a sequence shall be the start of scope for the NAK sequence; the start of scope for every other NAK PDU in the sequence shall be the end of scope of the prior NAK PDU. The end of scope for the last NAK PDU in the sequence shall be the end of scope for the NAK sequence itself; the end of scope for every other NAK PDU in the sequence shall be the end offset of the NAK PDU's last segment request.

**4.6.4.3.4** At the moment that file delivery is deemed Complete, as explained in 4.6.1.2.8 above, the entity shall:

a) issue a Notice of Completion (Completed);

b) if the receiving entity is the transaction's destination, issue a Finished (complete) PDU:

    1) any filestore responses shall be carried as parameters of the Finished (complete) PDU;

    2) the condition code in the Finished (complete) PDU shall be 'No error' if file reception was deemed complete following determination that the calculated and received checksums were equal, but 'Unsupported checksum type' otherwise.

**4.6.4.3.5** Positive Acknowledgment procedures shall be applied to the EOF (No error) PDU and to any Finished (complete) PDU produced, with the Expected Responses being ACK (EOF) PDU and ACK (Finished) PDU respectively.

### 4.6.4.4 Incremental Lost Segment Detection Procedures at the Receiving Entity

#### 4.6.4.4.1 General

**4.6.4.4.1.1** The various Incremental Lost Segment Detection procedures are not mutually exclusive. Selection of Incremental Lost Segment Detection procedures to implement at a given entity is a local matter.

**4.6.4.4.1.2** None of the Incremental Lost Segment Detection procedures shall remain in effect, for a given transaction, after initial reception of the EOF (No error) PDU.

#### 4.6.4.4.2 Immediate

In Immediate Mode, each file data gap or lost metadata detected prior to reception of the EOF (No error) PDU shall cause the receiving CFDP entity to issue a NAK sequence immediately.

#### 4.6.4.4.3 Prompted

In Prompted mode, upon receipt of a Prompt PDU prior to reception of the EOF (No error) PDU, the receiving CFDP entity shall issue a NAK sequence. If there are no missing file data or metadata, the NAK sequence shall comprise a single NAK PDU containing no segment requests.

NOTE – The processing of a Prompt PDU received subsequent to reception of the EOF (No error) PDU is undefined, an implementation matter. For example, the receiving CFDP entity might simply discard the Prompt PDU; alternatively, it might respond by setting the expiration time of the transaction-specific NAK timer (discussed later) to the current time, causing re-examination of the file to check for data reception completeness and potential issuance of a NAK sequence.

##### 4.6.4.4.4 Asynchronous

In Asynchronous mode, in response to an implementation-specific external event occurring prior to reception of the EOF (No error) PDU, the receiving CFDP entity shall issue a NAK sequence. If there are no missing file data or metadata, the NAK sequence shall comprise a single NAK PDU containing no segment requests.

NOTE – Examples of external events are the opening of a window of opportunity, for missions having sporadic contact, or a signal from a periodic timer at the receiving CFDP entity.

#### 4.6.4.5 Incremental Lost Segment Detection Procedures at the Sending Entity

In response to an implementation-specific external event, the sending CFDP entity may issue a Prompt (NAK) PDU.

#### 4.6.4.6 Deferred Lost Segment Detection Procedures at the Receiving Entity

Upon initial receipt of the EOF (No error) PDU for a transaction, the receiving entity shall determine whether or not any of the transaction's file data or metadata have yet to be received. If so:

a) If any file data gaps or lost metadata are detected for which no segment requests were included in previously issued NAK PDUs, the receiving CFDP entity shall issue a NAK sequence.

b) A transaction-specific NAK timer shall be started. The timer shall have an implementation-specific expiry period. When the timer expires, the receiving entity shall determine whether or not any of the transaction's file data or metadata have yet to be received. If so, the receiving entity shall issue a NAK sequence whose scope begins at zero and extends through the entire length of the file, and the timer shall be reset.

   NOTE – Because the expiry period of the NAK timer is wholly implementation-specific, it is not required to be the same on successive iterations of this data completeness determination cycle; in fact, it might even be dynamically altered (possibly multiple times) prior to any single expiration. In effect, re-examination of the file to check for data reception completeness and potentially initiate issuance of a NAK sequence may occur at any time following reception of the EOF (No error) PDU, at the sole discretion of the implementation.

#### 4.6.4.7 NAK Limit

An implementation-specific measure of NAK activity shall be maintained for each transaction, and, if an implementation-specific limit is reached for a given transaction, a NAK Limit Reached fault shall be declared.

NOTE – A typical implementation of NAK activity limit is a limit on the number of successive times the NAK timer is allowed to expire without intervening reception of file data and/or metadata that had not previously been received.

### 4.6.5 KEEP ALIVE PROCEDURES

#### 4.6.5.1 Overview

The Keep Alive mechanism is provided as a means by which a sending entity may monitor the progress of file data reception at the receiving entity so that problems may be detected prior to acknowledgment of the EOF (No error) PDU.

#### 4.6.5.2 Keep Alive Procedures at the Receiving Entity

**4.6.5.2.1** In all acknowledged modes, the receiving CFDP entity may periodically send a Keep Alive PDU to the sending CFDP entity reporting on the transaction’s reception progress so far at this entity.

**4.6.5.2.2** The Keep Alive PDU shall also be sent in response to receipt of a Prompt (Keep Alive) PDU.

**4.6.5.2.3** However, transmission of Keep Alive PDUs shall cease upon receipt of the transaction’s EOF (No error) PDU.

#### 4.6.5.3 Keep Alive Procedures at the Sending Entity

**4.6.5.3.1** At the sending CFDP entity, if the discrepancy between the reception progress reported by the Keep Alive PDU, and the transaction’s transmission progress so far at this entity exceeds a preset limit, the sending CFDP entity may optionally declare a Keep Alive Limit Reached fault.

**4.6.5.3.2** In response to an implementation-specific stimulus, the sending Entity may generate a Prompt (Keep Alive) PDU to force the receiving entity to send a Keep Alive PDU.

### 4.6.6 CANCEL RESPONSE PROCEDURES

#### 4.6.6.1 Cancel Response Procedures at the Receiving Entity

**4.6.6.1.1** Receipt of an EOF (cancel) PDU shall cause the receiving CFDP entity to issue a Notice of Completion (Canceled).

**4.6.6.1.2** If an acknowledged mode is in effect, Positive Acknowledgement procedures shall be applied to the EOF (cancel) PDU with the Expected Response being an ACK (EOF) PDU.

#### 4.6.6.2 Cancel Response Procedures at the Sending Entity

**4.6.6.2.1** Receipt of a Finished (cancel) PDU shall cause the sending CFDP entity to issue a Notice of Completion (Canceled).

**4.6.6.2.2** If an acknowledged mode is in effect, Positive Acknowledgement procedures shall be applied to the Finished (cancel) PDU with the Expected Response being an ACK (Finished) PDU.

### 4.6.7 RESUME PROCEDURES

#### 4.6.7.1 General

Resume procedures apply upon receipt of a **`Resume.request`** primitive submitted by the CFDP user. However:

a) a **`Resume.request`** primitive shall be ignored if it pertains to a transaction that is not currently suspended;

b) if the transaction to which a **`Resume.request`** primitive pertains is currently not only suspended but also frozen (as defined in 4.12), then the transaction shall be considered no longer suspended but the only applicable procedure shall be the issuance of a **`Resumed.indication`**.

NOTE – Transaction resumption at the sending entity may result in unsatisfactory communication behavior if the transaction is not concurrently resumed at the receiving entity for the same transaction, and vice versa. To this end, it is recommended that a Remote Resume request message (described later) be transmitted to the peer entity whenever a transaction is locally resumed.

#### 4.6.7.2 Resume Procedures at the Sending Entity

**4.6.7.2.1** On receipt of a **`Resume.request`** primitive, the sending CFDP entity shall

a) resume transmission of Metadata PDU, file segments, and EOF PDU;

b) issue a **`Resumed.indication`**.

**4.6.7.2.2** If operating in acknowledged mode,

a) any suspended transmission of Prompt PDUs shall be resumed;

b) the application of Positive Acknowledgment Procedures to PDUs previously issued by this entity shall be resumed.

#### 4.6.7.3 Resume Procedures at the Receiving Entity

**4.6.7.3.1** On receipt of a **`Resume.request`** primitive, the receiving CFDP entity shall

a) resume transmission of NAK PDUs;

b) resume any suspended transmission of Keep Alive PDUs;

c) issue a **`Resumed.indication`**.

**4.6.7.3.2** The application of Positive Acknowledgment Procedures to PDUs previously issued by this entity shall be resumed.

### 4.6.8 REPORT PROCEDURES

Upon receipt of a **`Report.request`** primitive during a Copy File procedure, the CFDP entity shall issue a **`Report.indication`** primitive providing information on the progress of the transaction.

NOTE – Since there is no interaction with any other protocol entity, there is no protocol associated with this procedure.

## 4.7 POSITIVE ACKNOWLEDGEMENT PROCEDURES

### 4.7.1 POSITIVE ACKNOWLEDGEMENT PROCEDURES AT PDU SENDING END

If Positive Acknowledgement procedures apply to a PDU,

a) upon issuing the PDU, the sending CFDP entity shall start a timer and retain the PDU for retransmission as necessary;

b) if the Expected Response is not received before expiry of the timer, the sending CFDP entity shall reissue the original PDU;

c) the sending CFDP entity shall keep a tally of the number of transmission retries;

d) if a preset limit is exceeded, the sending CFDP entity shall declare a Positive ACK Limit Reached fault;

e) receipt of the Expected Response shall cause the sending CFDP to release the retained PDU.

### 4.7.2 POSITIVE ACKNOWLEDGEMENT PROCEDURES AT PDU RECEIVING END

Receipt of a PDU to which Positive Acknowledgement procedures are applied shall cause the receiving CFDP entity immediately to issue the Expected Response.

NOTES

1 By issuing the Expected Response, the CFDP entity **only** confirms receipt of the PDU; issuance of the Expected Response does not imply that the receiving CFDP entity has taken any other action as a result. For example, production of an ACK (EOF (Cancel)) PDU implies that an EOF (Cancel) PDU was received but not necessarily that the referenced transaction was canceled.

2 The receiving CFDP entity is **always** required to issue the Expected Response upon receipt of a PDU to which Positive Acknowledgment procedures are applied. The purpose of the Expected Response is to turn off the PDU retransmission timer at the sending end. This purpose must be served regardless of the status of the transaction: undefined, active, terminated, or unrecognized.

## 4.8 FAULT HANDLING PROCEDURES

**4.8.1** Whenever a fault condition is detected and declared, the appropriate fault handler shall be invoked. A default handler shall be specified in the MIB for each type of fault condition, but for any particular transaction, a fault handler override may have been stipulated for one or more types of fault condition. Fault handler overrides, when present, shall be invoked in preference to the corresponding default fault handlers.

NOTE – Until the metadata for a transaction is received, the receiving CFDP entity cannot know what fault handler overrides have been stipulated and therefore can only invoke the default fault handlers. Stipulating a fault handler override in a Put request therefore does not guarantee that the receiving entity will behave in the stipulated manner.

**4.8.2** Invocation of a fault handler shall result in one of the following:

a) issuance of a Notice of Cancellation;

b) issuance of a Notice of Suspension, but only if the affected transaction was sent in ‘acknowledged’ transmission mode, or the fault condition was declared at the source entity of the transaction;

c) no protocol action; that is, the fault condition is ignored (it should be noted that this may result in unpredictable protocol behavior), and a `Fault.indication` with condition code identifying the fault condition is issued to the CFDP user;

d) abandonment of the transaction, resulting in an `Abandoned.indication` with condition code identifying the fault condition.

**4.8.3** All faults shall be logged, even if no action is taken.

## 4.9 FILESTORE PROCEDURES

**4.9.1** Filestore requests shall be executed only if any associated Copy File procedure proceeded to completion with no error.

NOTE – When a Copy File procedure results in a fault of any kind, no associated filestore requests are executed and no Filestore Responses are generated.

**4.9.2** Filestore requests shall be transmitted in the Directive Parameter field of the Metadata PDU in the order in which they were submitted in the Put primitive (see 3.4.1).

**4.9.3** Execution of filestore requests is mandatory. Filestore requests shall be executed in the order in which they are received in the Directive Parameter field of the Metadata PDU.

**4.9.4** Execution of a filestore request shall result in the generation of a Filestore Responses parameter. If acknowledged mode is in effect or transaction closure is requested, the Filestore Responses parameter shall be transmitted to the originator of the request via the Finished (complete) PDU.

**4.9.5** If any filestore request obtained from a given Metadata PDU does not succeed, no subsequent filestore requests from the same Metadata PDU shall be executed. For each of these non-executed subsequent filestore requests, the filestore request status code returned in the resulting Filestore Responses parameter shall be 'not performed'.

**4.9.6** Failure of a filestore request shall **not** result in the declaration of a fault of any kind.

## 4.10 INACTIVITY MONITOR PROCEDURES

**4.10.1** For a particular transaction, if there is a cessation of PDU reception for a specified time period (the transaction inactivity limit), then an Inactivity fault condition shall be declared.

NOTE – The 'cessation of PDU reception' that indicates inactivity is not limited to the cessation of File Data PDU reception at the receiving entity. The cessation of expected File Directive PDU reception (dependent upon the transaction's transmission mode) at either the sending or the receiving entity may likewise indicate inactivity. Detection of this condition is a local implementation matter.

**4.10.2** This requirement does not apply to the sending entity of an unacknowledged mode transfer.

**4.10.3** The default handler for Inactivity faults shall be to Cancel the transaction.

## 4.11 INTERNAL PROCEDURES

### 4.11.1 NOTICE OF COMPLETION PROCEDURES

NOTES

1 In the following subsections, the name of a Notice of Completion may include, in parentheses, the name of the specific disposition of the affected Copy File procedure and either 'Completed', indicating successful completion of the procedure, or 'Canceled', indicating cancellation of the procedure. Disposition name may be omitted from the name of a Notice of Completion when it is irrelevant in the context in which the name is used.

2 It is possible for a Notice of Completion (Canceled) to be issued after a Notice of Completion (Completed) has been issued for the same transaction. This could be caused, for example, by the receiving entity's repeated failure to receive an ACK PDU in response to its Finished (complete) PDU, resulting in the declaration of a Positive ACK Limit Reached fault. The standard Notice of Completion Procedures continue to apply in this event. At the receiving entity, the received file will already have been delivered to the user for normal processing by the time the Notice of Completion (Canceled) is issued.

#### 4.11.1.1 Notice of Completion Procedures at the Sending Entity

**4.11.1.1.1** On Notice of Completion of the Copy File procedure, the sending CFDP entity shall

a) release all unreleased portions of the file retransmission buffer;

b) stop transmission of file segments and metadata.

**4.11.1.1.2** If sending in acknowledged mode,

a) any transmission of Prompt PDUs shall be terminated;

b) the application of Positive Acknowledgment Procedures to PDUs previously issued by this entity shall be terminated.

**4.11.1.1.3** In any case,

a) if the sending entity is the transaction's source, it shall issue a **`Transaction-Finished.indication`** primitive indicating the condition in which the transaction was completed;

b) if all the following are true:

   1) the sending entity is the transaction's source,

   2) the file was sent in acknowledged mode,

   3) the procedure disposition cited in the Notice of Completion is 'Completed', and

   4) the Finished PDU whose arrival completed the transaction contained a Filestore Responses parameter,

   then that Filestore Responses parameter shall be passed in the `Transaction-Finished.indication` primitive.

#### 4.11.1.2 Notice of Completion Procedures at the Receiving Entity

**4.11.1.2.1** If receiving in acknowledged mode,

a) transmission of NAK PDUs, whether in response to NAK timer expiration or in response to any other events, shall be terminated;

b) any transmission of Keep Alive PDUs shall be terminated;

c) the application of Positive Acknowledgment Procedures to PDUs previously issued by this entity shall be terminated.

**4.11.1.2.2** In any case,

a) if the receiving entity is the transaction's destination, and the procedure disposition cited in the Notice of Completion is 'Completed', the receiving CFDP entity shall execute any filestore requests conveyed by the Put procedure;

b) if the procedure disposition cited in the Notice of Completion is 'Canceled', and the receiving entity is the transaction's destination, then the incomplete data shall be either discarded or retained according to the option set in the MIB;

NOTE – On Notice of Completion (Canceled) for a transaction for which a Notice of Completion (Completed) was previously declared, all file data have necessarily already been received, and therefore there are no incomplete data to discard or retain.

c) if the receiving entity is the transaction's destination, then it may optionally issue a `Transaction-Finished.indication` primitive indicating the condition in which the transaction was completed;

d) if the receiving entity is the transaction's destination, Filestore Responses and/or a Status Report shall be passed as parameters of the `Transaction-Finished.indication` primitive as available.

### 4.11.2 NOTICE OF CANCELLATION PROCEDURES

#### 4.11.2.1 General

At any time during a Copy File procedure, either the sending CFDP entity or receiving CFDP entity may issue a Notice of Cancellation.

NOTE – The Notice of Cancellation may be issued in reaction to the declaration of a fault or to receipt of a `Cancel.request` primitive submitted by the CFDP user.

#### 4.11.2.2 Notice of Cancellation Procedures at the Sending Entity

**4.11.2.2.1** On Notice of Cancellation of the Copy File procedure, the sending CFDP entity shall

a) issue a Notice of Completion (Canceled);

b) issue an EOF (cancel) PDU indicating the reason for transaction termination: `Cancel.request` received or the condition code of the fault whose declaration triggered the Notice of Cancellation. The file size field in the EOF (cancel) PDU shall contain the transaction’s current transmission progress, and the checksum in this PDU shall be the computed checksum over all File Data PDUs sent so far in the course of this transaction.

**4.11.2.2.2** If sending in acknowledged mode,

a) Positive Acknowledgment procedures shall be applied to the EOF (cancel) PDU with the Expected Response being an ACK (EOF) PDU with condition code equal to that of the EOF (cancel) PDU;

b) any PDU received after issuance of the EOF (cancel) PDU and prior to receipt of the Expected Response shall be ignored, except that all Positive Acknowledgment procedures remain in effect.

**4.11.2.2.3** Any fault declared in the course of transferring the EOF (cancel) PDU must result in abandonment of the transaction.

#### 4.11.2.3 Notice of Cancellation Procedures at the Receiving Entity

**4.11.2.3.1** On Notice of Cancellation of the Copy File procedure, the receiving CFDP entity shall issue a Notice of Completion (Canceled).

**4.11.2.3.2** If receiving in acknowledged mode,

a) the receiving CFDP entity shall issue a Finished (cancel) PDU indicating the reason for transaction termination: `Cancel.request` received or the condition code of the fault whose declaration triggered the Notice of Cancellation;

b) Positive Acknowledgment procedures shall be applied to the Finished (cancel) PDU with the Expected Response being an ACK (Finished) PDU with condition code equal to that of the Finished (cancel) PDU;

c) any PDU received after issuance of the Finished (cancel) PDU and prior to receipt of the Expected Response shall be ignored, except that all Positive Acknowledgment procedures remain in effect;

d) any fault declared in the course of transferring this PDU must result in abandonment of the transaction.

**4.11.2.3.3** If receiving in unacknowledged mode, and if the transaction's Metadata PDU has been received and the Closure Requested flag is set to '1' in that PDU, then the receiving CFDP entity shall issue a Finished (cancel) PDU indicating the reason for transaction termination: Cancel.request received or the condition code of the fault whose declaration triggered the Notice of Cancellation.

#### 4.11.2.4 Notice of Suspension Procedures

#### 4.11.2.5 General

**4.11.2.5.1** At any time during a Copy File procedure, either the sending CFDP entity or the receiving CFDP entity may issue a Notice of Suspension.

NOTE

1 A Notice of Suspension may be issued in reaction to the declaration of a fault or to receipt of a `Suspend.request` primitive submitted by the CFDP user.

2 Transaction suspension at the sending entity may result in unsatisfactory communication behavior if the transaction is not concurrently suspended at the receiving entity for the same transaction, and vice versa. To this end, it is recommended that a Remote Suspend request message (described later) be transmitted to the peer entity whenever a transaction is locally suspended.

**4.11.2.5.2** However, a `Notice of Suspension` shall be ignored if it pertains to a transaction that is already suspended or if it is issued by the receiving CFDP entity for a transaction sent in Unacknowledged mode.

**4.11.2.5.3** The following lists of the effects of transaction suspension at the sending and receiving entities are exhaustive; no additional effects should be inferred. In particular, received EOF and Finished PDUs shall be acknowledged and processed in accordance with Positive Acknowledgment Procedures and the relevant Copy File procedures, regardless of any transaction suspension that may be in effect.

NOTE – It follows from the above that transaction suspension has no effect whatsoever on the disposition of File Data PDUs that are received at the receiving entity. No received PDUs are discarded, and in fact it is possible, and valid, for the entire file to be reassembled and delivered to the user application while the transaction is suspended.

#### 4.11.2.6 Notice of Suspension Procedures at the Sending Entity

**4.11.2.6.1** On Notice of Suspension of the Copy File procedure, the sending CFDP entity shall

a) suspend transmission of Metadata PDU, file segments, and EOF PDU;

b) save the status of the transaction.

**4.11.2.6.2** If operating in acknowledged mode,

a) any transmission of Prompt PDUs shall be suspended;

b) the inactivity timer shall be suspended;

c) the application of Positive Acknowledgment Procedures to PDUs previously issued by this entity shall be suspended.

**4.11.2.6.3** The sending entity shall issue a **`Suspended.indication`**.

#### 4.11.2.7 Notice of Suspension Procedures at the Receiving Entity

On Notice of Suspension of the Copy File procedure, the receiving CFDP entity shall

a) suspend transmission of NAK PDUs;

b) suspend any transmission of Keep Alive PDUs;

c) suspend the inactivity timer;

d) suspend transmission of Finished (complete) PDUs;

e) suspend the application of Positive Acknowledgment Procedures to PDUs previously issued by this entity;

f) issue a **`Suspended.indication`** if so configured in the MIB (see table 8-1);

g) save the status of the transaction.

## 4.12 LINK STATE CHANGE PROCEDURES

### 4.12.1 GENERAL

Link state change procedures shall apply upon notification by the MIB of the start and end of opportunities to transmit PDUs to, or receive PDUs from, specific remote CFDP entities.

NOTE – The start of a transmission opportunity nominally corresponds to some moment at which the remote entity will begin listening for PDUs from the local CFDP entity (e.g., will have pointed its antenna appropriately and applied power to its receiver), less the applicable one-way light time between the local and remote entities; similarly, the end of a transmission opportunity nominally precedes, by one-way light time, the moment at which the remote entity will cease listening for PDUs from the local entity. A reception opportunity likewise nominally begins at some moment at which the remote entity will begin sending PDUs to the local entity, plus one-way light time, and ends at the moment at which the remote entity will cease transmission, again, plus one-way light time. However, the manner in which these notifications are initiated and delivered to CFDP is wholly a matter of implementation.

### 4.12.2 TRANSMISSION OPPORTUNITY PROCEDURES

**4.12.2.1** On notification of the end of an opportunity to transmit to a specified remote CFDP entity, the CFDP entity shall ‘freeze’ transmission for all transactions for which it is the sending entity and the specified remote entity is the receiving entity.

**4.12.2.2** The freezing of transmission for a transaction shall have the same effects as suspension of that transaction by the sending entity (see 4.11.2.6), except that no **`Suspended.indication`** shall be issued and the transaction shall not be considered suspended.

**4.12.2.3** On notification of the start of an opportunity to transmit to a specified remote CFDP entity, the CFDP entity shall ‘thaw’ transmission for all transactions for which it is the sending entity and the specified remote entity is the receiving entity.

**4.12.2.4** The thawing of transmission for a transaction shall have the same effects as resumption of that transaction by the sending entity (see 4.6.7.2.1), except that (a) no **`Resumed.indication`** shall be issued and (b) thawing transmission for a suspended transaction shall have no effect whatsoever.

#### 4.12.2.5 Reception Opportunity Procedures

**4.12.2.6** On notification of the end of an opportunity to receive from a specified remote CFDP entity, the CFDP entity shall freeze reception for all transactions for which it is the receiving entity and the specified remote entity is the sending entity, except those that are in Unacknowledged mode.

**4.12.2.7** The freezing of reception for a transaction shall have the same effects as suspension of that transaction by the receiving entity (see 4.11.2.7), except that no `Suspended.indication` shall be issued, and the transaction shall not be considered suspended.

**4.12.2.8** On notification of the start of an opportunity to receive from a specified remote CFDP entity, the CFDP entity shall thaw reception for all transactions for which it is the receiving entity and the specified remote entity is the sending entity, except those that are in Unacknowledged mode.

**4.12.2.9** The thawing of reception for a transaction shall have the same effects as resumption of that transaction by the receiving entity (see 4.6.7.3), except that (a) no `Resumed.indication` shall be issued, and (b) thawing reception for a suspended transaction shall have no effect whatsoever.

# 5 PDU FORMATS

## 5.1 GENERAL

**5.1.1** PDUs shall consist of a fixed-format PDU header followed by a fixed- or variable-format PDU data field.

**5.1.2** The PDU header shall consist of the fields shown in table 5-1. Optional fields are present only when so indicated by the corresponding flag bit.

**5.1.3** The PDU header fields shall be transmitted in the order of presentation in table 5-1.

NOTE – Every PDU is associated with a unique transaction by the source Entity ID together with the Transaction sequence number.

**Table 5-1: Fixed PDU Header Fields**

| Field | Length (bits) | Values | Comment |
|---|---|---|---|
| Version | 3 | '001' | For the second version of the protocol. |
| PDU type | 1 | '0' — File Directive<br>'1' — File Data | |
| Direction | 1 | '0' — toward file receiver<br>'1' — toward file sender | Used to perform PDU forwarding. |
| Transmission Mode | 1 | '0' — acknowledged<br>'1' — unacknowledged | |
| CRC Flag | 1 | '0' — CRC not present<br>'1' — CRC present | |
| Large File flag | 1 | '0' — small file<br>'1' — large file | Every file whose size can't be represented in an unsigned 32-bit integer shall be flagged 'large'. All files of unbounded size shall be flagged 'large'. All other files shall be flagged 'small'. |
| PDU Data field length | 16 | | In octets. |
| Segmentation control | 1 | '0' — record boundaries are not preserved in file data segmentation<br>'1' — record boundaries are preserved in file data segmentation | Always '0' (and ignored) for File Directive PDUs. |
| Length of entity IDs | 3 | | Number of octets in entity ID less one; that is, '0' means that entity ID is one octet. Applies to all entity IDs in the PDU header. |
| Segment metadata flag | 1 | '0' — segment metadata not present in PDU<br>'1' — segment metadata present in PDU | Always '0' (and ignored) for File Directive PDUs. |

| Field | Length (bits) | Values | Comment |
|---|---|---|---|
| Length of Transaction sequence number | 3 | | Number of octets in sequence number less one; that is, ‘0’ means that sequence number is one octet. |
| Source entity ID | Variable | | Identifies the entity that originated the transaction. An unsigned binary integer. |
| Transaction sequence number | Variable | | Uniquely identifies the transaction, among all transactions originated by this entity. An unsigned binary integer. |
| Destination entity ID | Variable | | Identifies the entity that is the final destination of the transaction’s metadata and file data. An unsigned binary integer. |

**5.1.4** Every entity ID in any CFDP PDU shall be represented as an unsigned binary integer of length, as indicated by an entity ID length field elsewhere in the PDU. Every transaction sequence number in any CFDP PDU shall be represented as an unsigned binary integer of length as indicated by a transaction sequence number length elsewhere in the PDU.

**5.1.5** The sequence number shall be sequentially allocated by the CFDP Entity responding to the `Put.request`.

**5.1.6** Two PDU types are defined:

- File Directive;
- File Data.

**5.1.7** The PDU Data field can contain data items in any one of five formats:

- fixed-format data;
- Length, Value (LV) format;
- TLV format;
- variable-length file data;
- File Size Sensitive (FSS).

NOTES

1 Fixed-format data is precisely defined in this document and is optimized for efficiency of the protocol for which the format of the data is not required to be variable. Fixed format data fields are fixed in their position in the PDU and in their length.

2 Each PDU field referred to as an offset is encoded as the number of octets in the file prior to the octet described by that field; for example, the offset of the first octet of the file is encoded as all ‘zeroes’.

3 Entity IDs may be of different lengths in different PDUs, as may transaction sequence numbers, but these differences in length have no semantic significance. Two entity IDs of different lengths are compared by, in effect, inserting leading ‘zeroes’ to pad the shorter entity ID to the same length as the longer entity ID and then comparing the resulting values. The same procedure applies to comparisons between transaction sequence numbers of different lengths.

**5.1.8** The general format of LV objects shall be as shown in table 5-2.

**Table 5-2: LV Object Format**

| Field | Size (bits) | Values | Comment |
|---|---|---|---|
| Length | 8 | 0 to 255 | Length of value. |
| Value | 8 × Length | | |

NOTE – LV format data is variable in its length but is invariable in its position in the PDU. It can, by virtue of its length field, be parsed by the protocol.

**5.1.9** The format of a TLV object shall be as shown in table 5-3.

**Table 5-3: TLV Object Format**

| Field | Size (bits) | Values | Comment |
|---|---|---|---|
| Type | 8 | (See 5.4, below.) | Nature of value. |
| Length | 8 | 0 to 255 | Length of value. |
| Value | 8 × Length | | |

NOTES

1 TLV format data is variable both in its length and in its position in the PDU. It can, however, by virtue of its type and length fields, be parsed by the protocol.

2 In the following subsections, parameters are characterized as either fixed parameters (indicated by a specific field length), LV parameters (indicated by the notation ‘LV’), or TLV parameters (indicated by the notation ‘TLV hh’, where hh is the hexadecimal value in the Type field of the TLV object).

**5.1.10** File-Size Sensitive (FSS) data items are integer values whose size in bits shall depend on the value of the Large File flag in the PDU header. When the value of the Large File flag is zero, the size of each FSS data item in the PDU shall be 32 bits. When the value of the Large File flag is 1, the size of each FSS data item in the PDU shall be 64 bits.

## 5.2 FILE DIRECTIVE PDUs

### 5.2.1 GENERAL

**5.2.1.1** The data field of File Directives shall consist of a Directive Code octet followed by a Directive Parameter field.

**5.2.1.2** The Directive Code shall indicate one of the directives shown in table 5-4.

**Table 5-4: Directive Codes**

| Directive Code (hexadecimal) | Directive |
|---|---|
| 00 | Reserved |
| 01 | Reserved |
| 02 | Reserved |
| 03 | Reserved |
| 04 | EOF PDU |
| 05 | Finished PDU |
| 06 | ACK PDU |
| 07 | Metadata PDU |
| 08 | NAK PDU |
| 09 | Prompt PDU |
| 0C | Keep Alive PDU |
| 0D–FF | Reserved |

NOTE – The Directive Parameter fields for the above actions are described, in order of transmission, in the following subsections.

**5.2.1.3** In several cases, the Directive Parameter field of a File Directive includes a four-bit Condition Code. The Condition Code shall in each case indicate one of the conditions shown in table 5-5.

**Table 5-5: Condition Codes**

| Condition Code (binary) | Condition |
|---|---|
| ‘0000’ | No error |
| ‘0001’ | Positive ACK limit reached |
| ‘0010’ | Keep alive limit reached |
| ‘0011’ | Invalid transmission mode |
| ‘0100’ | Filestore rejection |
| ‘0101’ | File checksum failure |
| ‘0110’ | File size error |
| ‘0111’ | NAK limit reached |
| ‘1000’ | Inactivity detected |
| ‘1001’ | Invalid file structure |
| ‘1010’ | Check limit reached |
| ‘1011’ | Unsupported checksum type |
| ‘1100’ – ‘1101’ | (reserved) |
| ‘1110’ | `Suspend.request` received |
| ‘1111’ | `Cancel.request` received |

### 5.2.2 END-OF-FILE PDU

The contents of the Parameter field for a File Directive having a Code of EOF PDU shall be as shown in table 5-6.

**Table 5-6: End-of-File PDU Contents**

| Parameter | Length (bits) | Values | Comments |
|---|---|---|---|
| Condition Code | 4 | (See table 5-5.) | |
| Spare | 4 | | All 'zeroes'. |
| File Checksum | 32 | | (See 4.2, Checksum Procedures.) |
| File size | FSS | | In octets. This value shall be the total number of file data octets transmitted by the sender, regardless of the condition code (i.e., it shall be supplied even if the condition code is other than 'No error'). |
| Fault Location | TLV | An entity ID TLV (see 5.4.6). | Omitted if condition code is 'No error'. Otherwise, entity ID in the TLV is the ID of the entity at which transaction cancellation was initiated. |

### 5.2.3 FINISHED PDU

The contents of the Parameter field for a File Directive having a Code of Finished PDU shall be as shown in table 5-7.

**Table 5-7: Finished PDU Contents**

| Parameter | Length (bits) | Values | Comments |
|---|---|---|---|
| Condition Code | 4 | (See table 5-5.) | |
| Spare | 1 | | |
| Delivery Code | 1 | '0' — Data Complete<br>'1' — Data Incomplete | 'Data Complete' means that metadata, all file data, and EOF have been received, and the checksum has been verified. |

| Parameter | Length (bits) | Values | Comments |
|---|---|---|---|
| File Status | 2 | ‘00’ — Delivered file discarded deliberately<br>‘01’ — Delivered file discarded due to filestore rejection<br>‘10’ — Delivered file retained in filestore successfully<br>‘11’ — Delivered file status unreported | File status is meaningful only when the transaction includes the transmission of file data. |
| Filestore Responses | TLVs | (See table 5-17.) | A filestore response TLV must be included for each filestore request TLV of the Metadata PDU. |
| Fault Location | TLV | An entity ID TLV (see 5.4.6). | Omitted if condition code is ‘No error’ or ‘Unsupported checksum type’. Otherwise, entity ID in the TLV is the ID of the entity at which transaction cancellation was initiated. |

### 5.2.4 ACK PDU

The contents of the Parameter field for a File Directive having a Code of ACK PDU shall be as shown in table 5-8. The possible values of the Transaction Status parameter are as follows (all values in binary notation):

‘00’ – Undefined. The transaction to which the acknowledged PDU belongs is not currently active at this entity, and the CFDP implementation **does not** retain transaction history. The transaction might be one that was formerly active and has been terminated, or it might be one that has never been active at this entity.

‘01’ – Active. The transaction to which the acknowledged PDU belongs is currently active at this entity.

‘10’ – Terminated. The transaction to which the acknowledged PDU belongs is not currently active at this entity, the CFDP implementation **does** retain transaction history, and the transaction is thereby known to be one that was formerly active and has been terminated.

‘11’ – Unrecognized. The transaction to which the acknowledged PDU belongs is not currently active at this entity, the CFDP implementation **does** retain transaction history, and the transaction is thereby known to be one that has never been active at this entity.

**Table 5-8: ACK PDU Contents**

| Parameter | Length (bits) | Values | Comments |
|---|---|---|---|
| Directive code of acknowledged PDU | 4 | (See table 5-4.) Only EOF and Finished PDUs are acknowledged. | Directive code of the PDU that this ACK PDU acknowledges. |
| Directive subtype code | 4 |  | Values depend on the directive code of the PDU that this ACK PDU acknowledges. For ACK of Finished PDU: binary ‘0001’. Binary ‘0000’ for ACKs of all other file directives. |
| Condition code | 4 | (See table 5-5.) | Condition code of the acknowledged PDU. |
| Spare | 2 |  |  |
| Transaction status | 2 |  | Status of the transaction in the context of the entity that is issuing the acknowledgment. |

### 5.2.5 METADATA PDU

The contents of the Parameter field for a File Directive having a Code of Metadata PDU shall be as shown in table 5-9.

**Table 5-9: Metadata PDU Contents**

| Parameter | Length (bits) | Values | Comments |
|---|---|---|---|
| Reserved for future use | 1 | | Set to ‘0’. |
| Closure requested | 1 | ‘0’ – Transaction closure not requested<br>‘1’ – Transaction closure is requested | If transaction is in Acknowledged mode, set to ‘0’ and ignored. |
| Reserved for future use | 2 | | Set to all ‘zeroes’. |
| Checksum type | 4 | Checksum algorithm identifier as registered in the SANA Checksum Types Registry | Value zero indicates use of the legacy modular checksum. |
| File size | FSS | Length of File | In octets. Set to all ‘zeroes’ for a file of unbounded size. |
| Source file name | LV | | When there is no associated file, for example, messages used for Proxy operations, the LV Length field indicates zero length, and the LV value field is omitted. |
| Destination file name | LV | | When there is no associated file, for example, messages used for Proxy operations, the LV Length field indicates zero length, and the LV value field is omitted. |
| Options | TLVs | | Filestore requests.<br>Messages to user.<br>Fault Handler overrides.<br>Flow Label. |

### 5.2.6 NAK PDU

**5.2.6.1** The contents of the Parameter field for a File Directive having a Code of NAK PDU shall be as shown in table 5-10.

**Table 5-10: NAK PDU Contents**

| Parameter | Length (bits) | Values | Comments |
|---|---|---|---|
| Start of scope | FSS | (See 4.6.4.3.3.) | |
| End of scope | FSS | (See 4.6.4.3.3.) | |
| Segment Requests (× *N*) | 64 × *N* | (See following table.) | |

**5.2.6.2** Each Segment Request shall be as shown in table 5-11.

**Table 5-11: Segment Request Form**

| Parameter | Length (bits) | Values | Comments |
|---|---|---|---|
| Start offset | FSS | Data — Offset of start of requested segment<br>Metadata — ‘zero’ | In octets. |
| End Offset | FSS | Data — Offset of first octet after end of requested segment<br>Metadata — ‘zero’ | In octets. |

### 5.2.7 PROMPT PDU

The contents of the Parameter field for a File Directive having a Code of Prompt PDU shall be as shown in table 5-12.

**Table 5-12: Prompt PDU Contents**

| Parameter | Length (bits) | Values | Comments |
|---|---|---|---|
| Response required | 1 | ‘0’ — NAK<br>‘1’ — Keep Alive | |
| Spare | 7 | | |

### 5.2.8 KEEP ALIVE PDU

The contents of the Parameter field for a File Directive having a Code of Keep Alive PDU shall be as shown in table 5-13.

**Table 5-13: Keep Alive PDU Contents**

| Parameter | Length (bits) | Values | Comments |
|---|---|---|---|
| Progress | FSS | offset from start of file | In octets. |

## 5.3 FILE DATA PDU

The contents of the data field of a File Data PDU shall be as shown in table 5-14. Values of the record continuation state parameter shall be as follows:

‘00’— File data of this PDU contains neither the start nor the end of any record. If the value of the PDU header’s segmentation control flag is ‘1’, then this value indicates that the file data is the continuation of a record begun in a prior PDU; otherwise, it indicates that the file has no records.

‘01’— The first octet of the file data of this PDU is the first octet of a record, and the end of that record is not within the file data of this PDU.

‘10’— The last octet of the file data of this PDU is the last octet of a record, and the beginning of that record is not within the file data of this PDU.

‘11’— The first octet of the file data of this PDU is the first octet of a record, and the last octet of the file data of this PDU is the last octet of that record or of a subsequent record. If the file data field contains multiple complete records, identification of the boundaries of these records is an application matter; one possible approach would be to encode record offsets in the segment metadata field.

**Table 5-14: File Data PDU Contents**

| Parameter | Length (bits) | Values | Comments |
|---|---|---|---|
| Record continuation state | 2 | (See above.) | Present if and only if the value of the segment metadata flag in the PDU header is 1. |
| Segment metadata length | 6 | 0 to 63 | Present if and only if the value of the segment metadata flag in the PDU header is 1. |
| Segment metadata | Variable, from 0 to 504 | | Present if and only if the value of the segment metadata flag in the PDU header is 1 and the segment metadata length is greater than zero. Values are application-specific. |
| Offset | FSS | | In octets. |
| File data | Variable | | |

## 5.4 TLV PARAMETERS

### 5.4.1 FILESTORE REQUEST TLV

**5.4.1.1** The Type of the Filestore Request TLV shall be 00 hex.

**5.4.1.2** The Value of the Filestore Request TLV shall be as shown in table 5-15.

**Table 5-15: Filestore Request TLV Contents**

| Parameter | Length (bits) | Values | Comments |
|---|---|---|---|
| Action Code | 4 | (See table 5-16.) | |
| Spare | 4 | | |
| First File Name | LV | | |
| Second File Name | LV | | Only present for some action codes. |

**5.4.1.3** The Action codes and their parameters are as shown in table 5-16.

**Table 5-16: Filestore Request TLV Action Codes**

| Action Code | Action | Second File Name Present |
|---|---|---|
| ‘0000’ | Create File | N |
| ‘0001’ | Delete File | N |
| ‘0010’ | Rename File | Y |
| ‘0011’ | Append File | Y |
| ‘0100’ | Replace File | Y |
| ‘0101’ | Create Directory | N |
| ‘0110’ | Remove Directory | N |
| ‘0111’ | Deny File (delete if present) | N |
| ‘1000’ | Deny Directory (remove if present) | N |

### 5.4.2 FILESTORE RESPONSE TLV

**5.4.2.1** The Type of the Filestore Response TLV shall be 01 hex.

**5.4.2.2** The Value of the Filestore Response TLV shall be as shown in table 5-17.

**Table 5-17: Filestore Response TLV Contents**

| Parameter | Length (bits) | Values | Comments |
|---|---|---|---|
| Action Code | 4 | | As for Filestore Request TLV. |
| Status Code | 4 | (See table 5-18.) | |
| First File name | LV | | |
| Second File Name | LV | | Only present for some action codes, as for Filestore Request TLV. |
| Filestore Message | LV | | LV length field indicates zero length, and LV value field is omitted when there is no Filestore Message parameter present. Filestore message provides additional implementation-specific information regarding the result of performing the indicated filestore action. |

**Table 5-18: Filestore Response Status Codes**

| Action Code | Action | Use of Filenames | Status Codes |
|---|---|---|---|
| ‘0000’ | Create | First — filename to be created | ‘0000’ — Successful<br>‘0001’ — Create not allowed<br>‘1111’ — Not performed |
| ‘0001’ | Delete | First — filename to be deleted | ‘0000’ — Successful<br>‘0001’ — File does not exist<br>‘0010’ — Delete not allowed<br>‘1111’ — Not performed |
| ‘0010’ | Rename | First — old filename<br>Second — new filename | ‘0000’ — Successful<br>‘0001’ — Old File Name does not exist<br>‘0010’ — New File Name already exists<br>‘0011’ — Rename not allowed<br>‘1111’ — Not performed |
| ‘0011’ | Append | First — name of file whose contents form the first part of the new file and name of the new file<br>Second — name of file whose contents will form the second part of the new file | ‘0000’ — Successful<br>‘0001’ — File Name 1 does not exist<br>‘0010’ — File Name 2 does not exist<br>‘0011’ — Append not allowed<br>‘1111’ — Not performed |

| Action Code | Action | Use of Filenames | Status Codes |
|---|---|---|---|
| '0100' | Replace | First — filename whose contents are to be replaced<br>Second — filename whose contents will replace the contents of the first filename | '0000' — Successful<br>'0001' — File Name 1 does not exist<br>'0010' — File Name 2 does not exist<br>'0011' — Replace not allowed<br>'1111' — Not performed |
| '0101' | Create Directory | First — name of directory to be created | '0000' — Successful<br>'0001' — Directory cannot be created<br>'1111' — Not performed |
| '0110' | Remove Directory | First — name of directory to be deleted | '0000' — Successful<br>'0001' — Directory does not exist<br>'0010' — Delete not allowed<br>'1111' — Not performed |
| '0111' | Deny File | First — filename to be deleted | '0000' — Successful<br>'0010' — Delete not allowed<br>'1111' — Not performed |
| '1000' | Deny Directory | First — name of directory to be deleted | '0000' — Successful<br>'0010' — Delete not allowed<br>'1111' — Not performed |

### 5.4.3 MESSAGE TO USER TLV

The Type of the Message to User TLV shall be 02 hex; the format of the Value (the message) is not defined.

NOTE – To enable interoperability, it will typically be necessary for each message to contain some indication of its nature. The message identifier for the standard CFDP Proxy and Directory Operations messages (see section 6 below) is the presence of the ASCII characters 'cfdp' in the first four octets of each message; that message identification bit pattern is reserved for this purpose.

### 5.4.4 FAULT HANDLER OVERRIDE TLV

The Type of the Fault Handler Override TLV shall be 04 hex; the one-byte Value[1] shall be encoded as shown in table 5-19.

[1] The Value field of this TLV may be extended in future versions of this specification.

**Table 5-19: Fault Handler Override Field Encoding**

| Parameter | Length (bits) | Values | Comments |
|---|---|---|---|
| Condition code | 4 | (See table 5-5.) | The condition codes for 'No error', 'Suspend.request received', and 'Cancel.request received' are not applicable; these are not faults. |
| Handler code | 4 | '0000' — reserved for future expansion<br>'0001' — issue Notice of Cancellation<br>'0010' — issue Notice of Suspension<br>'0011' — Ignore error<br>'0100' — Abandon transaction<br>'0101'–'1111' — reserved | |

NOTE – If the Metadata PDU is not the first received PDU of a transaction, the fault handler settings in the MIB will be applied until the arrival of the Metadata PDU.

### 5.4.5 FLOW LABEL TLV

The Type of the Flow Label TLV shall be 05 hex; the format of the Value is not defined.

### 5.4.6 ENTITY ID TLV

The Type of the Entity ID TLV shall be 06 hex; the Value shall be an Entity ID as discussed in 5.1.

# 6 USER OPERATIONS

NOTE – This section describes standard interoperable procedures that use the services provided by CFDP.

## 6.1 RESERVED CFDP MESSAGE FORMAT

**6.1.1** CFDP User Operations shall be accomplished through the exchange of Reserved CFDP Messages between CFDP users. These messages shall be transported as the values of Message to User TLV parameters in the metadata of standard CFDP transactions.

**6.1.2** Each such message shall consist of a Reserved CFDP Message header formatted according to table 6-1, followed by message content whose format varies with the type of the message.

**Table 6-1: Reserved CFDP Message Header**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Message identifier | 32 | The ASCII characters ‘cfdp’. | Distinguishes Reserved CFDP Messages from all other user messages. |
| Message type | 8 | (See following tables.) | |

**6.1.3** The fields in the content of a Reserved CFDP Message shall contain data items as described in the CFDP Protocol Specification.

**6.1.4** The fields shall be in fixed format, in variable format, or in LV format, as defined in 5.1 above.

**6.1.5** The **Originating Transaction ID** message is common to all User operations. Its message type shall contain the hexadecimal value 0A, and its content shall be constructed as indicated in table 6-2.

**Table 6-2: Originating Transaction ID Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Reserved for future use | 1 | set to ‘0’ | |
| Length of entity ID | 3 | | Number of octets in source entity ID less one; that is, ‘0’ means that source entity ID is one octet. |
| Reserved for future use | 1 | set to ‘0’ | |
| Length of Transaction sequence number | 3 | | Number of octets in sequence number less one; that is, ‘0’ means that sequence number is one octet. |
| Source entity ID | Variable | | Identifies the entity that originated the transaction. An unsigned binary integer. |
| Transaction sequence number | Variable | | Uniquely identifies the transaction, among all transactions originated by this entity. An unsigned binary integer. |

## 6.2 PROXY OPERATION

NOTE – The term ‘proxy operation’ refers to the use of CFDP services by some CFDP user (referred to as the ‘originator’ of the operation) to initiate the transmission of a file by some remote CFDP entity’s user (referred to as the ‘respondent’) to some other entity’s user (referred to as the ‘beneficiary’). The beneficiary of a proxy operation might be either the originator itself (in which case the proxy operation functions as a ‘Get’) or some third CFDP entity’s user. It should be noted that while the core CFDP file transmission service may be invoked by the Proxy mechanism, the Store and Forward Overlay file transmission service (see Appendix A) may not.

### 6.2.1 GENERAL

To enable interoperability, the following mandatory behavior shall be observed by CFDP users that are in compliance with the CFDP proxy operations specification.

### 6.2.2 PROXY OPERATIONS MESSAGE TYPES

The message type field for each Reserved CFDP Message used in Proxy operations shall contain one of the values specified in table 6-3.

**Table 6-3: Proxy Operations Message Types**

| Message Type (hexadecimal) | Interpretation |
|---|---|
| 00 | Proxy Put Request |
| 01 | Proxy Message to User |
| 02 | Proxy Filestore Request |
| 03 | Proxy Fault Handler Override |
| 04 | Proxy Transmission Mode |
| 05 | Proxy Flow Label |
| 06 | Proxy Segmentation Control |
| 07 | Proxy Put Response |
| 08 | Proxy Filestore Response |
| 09 | Proxy Put Cancel |
| 0B | Proxy Closure Request |

### 6.2.3 INITIATING A PROXY OPERATION

#### 6.2.3.1 General

**6.2.3.1.1** When required to initiate a proxy operation, the CFDP user shall use the CFDP `Put.request` primitive to request delivery of a file delivery unit whose metadata contains one or more of the Reserved CFDP Messages, defined in 6.2.3.2 through 6.2.3.8, which correspond to the parameters of the `Put.request` primitive (3.4.1).

**6.2.3.1.2** The CFDP user that is the destination of the file delivery unit (the respondent) shall in turn use the contents of these messages to formulate a second `Put.request` primitive that accomplishes the desired remote file delivery.

**6.2.3.1.3** The transaction ID assigned to the original request, as reported to the originating CFDP user by the corresponding `Transaction.indication` primitive, shall be cited in the transaction that constitutes the respondent's eventual report on the results of the requested proxy operation.

#### 6.2.3.2 Proxy Put Request

The Proxy Put Request message is mandatory and shall be constructed as indicated in table 6-4.

**Table 6-4: Proxy Put Request Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Destination entity ID | Variable | LV | The ID of the beneficiary's CFDP entity. An unsigned binary integer. |
| Source file name | Variable | LV | Length is zero if parameter is omitted. |
| Destination file name | Variable | LV | Length is zero if parameter is omitted. |

#### 6.2.3.3 Proxy Message to User

One or more Proxy Message to User messages may be optionally included and shall be constructed as indicated in table 6-5.

**Table 6-5: Proxy Message to User Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Message text | Variable | LV | (See 5.4.3.) |

#### 6.2.3.4 Proxy Filestore Request

One or more Proxy Filestore Request messages may be optionally included and shall be constructed as indicated in table 6-6.

**Table 6-6: Proxy Filestore Request Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Length (octets) | 8 | Number of octets in the request field. | |
| Request | 8 × length | The value of this field is a single CFDP filestore request as defined in table 5-15. | |

#### 6.2.3.5 Proxy Fault Handler Override

One or more Proxy Fault Handler Override messages may optionally be included and shall be constructed as indicated in table 6-7.

**Table 6-7: Proxy Fault Handler Override Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Fault handler override | 8 | (See 5.4.4.) | |

#### 6.2.3.6 Proxy Transmission Mode

The Proxy Transmission Mode message is optional and shall be constructed as indicated in table 6-8.

**Table 6-8: Proxy Transmission Mode Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Spare | 7 | All 'zeroes'. | |
| Transmission mode | 1 | (See table 5-1.) | |

#### 6.2.3.7 Proxy Flow Label

A Proxy Flow Label message is optional and shall be constructed as indicated in table 6-9.

**Table 6-9: Proxy Flow Label Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Flow label | Variable | LV | (See 5.4.5.) |

#### 6.2.3.8 Proxy Segmentation Control

A Proxy Segmentation Control message is optional and shall be constructed as indicated in table 6-10.

**Table 6-10: Proxy Segmentation Control Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Spare | 7 | All 'zeroes'. | |
| Segmentation control | 1 | '0' — Record boundaries respected;<br>'1' — Record boundaries not respected. | |

### 6.2.3.9 Proxy Closure Request

A Proxy Closure Request message is optional and shall be constructed as indicated in table 6-11.

**Table 6-11: Proxy Closure Request Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Spare | 7 | All 'zeroes'. | |
| Closure requested | 1 | '0' — Transaction closure not requested<br>'1' — Transaction closure is requested | |

### 6.2.4 RESPONDING TO A PROXY PUT REQUEST

Upon receipt of a Proxy Put Request message, the respondent CFDP user shall use the CFDP **`Put.request`** primitive to request delivery of a file delivery unit with the following parameter(s):

- destination entity ID as specified in the Proxy Put Request message;
- source file name as specified in the Proxy Put Request message (if any);
- destination file name as specified in the Proxy Put Request message (if any);
- segmentation control as specified in the Proxy Segmentation Control message that was received in the same **`Metadata-recv.indication`** (if any);
- transmission mode as specified in the Proxy Transmission Mode message that was received in the same **`Metadata-recv.indication`** (if any);
- closure request as specified in the Proxy Closure Request message that was received in the same **`Metadata-recv.indication`** (if any);
- flow label as specified in the Proxy Flow Label message that was received in the same **`Metadata-recv.indication`** (if any);

- one fault handler override for each Proxy Fault Handler Override message that was received in the same `Metadata-recv.indication`;
- one Message to User for each Proxy Message to User message that was received in the same `Metadata-recv.indication`;
- one filestore request for each Proxy Filestore Request message that was received in the same `Metadata-recv.indication`;
- one Reserved CFDP Message of type Originating Transaction ID, citing the transaction ID of its triggering Proxy Put Request transaction.

### 6.2.5 REPORTING ON THE RESULTS OF A PROXY PUT REQUEST

#### 6.2.5.1 General

**6.2.5.1.1** Upon receipt of a `Transaction-Finished.indication` indicating the completion of a transaction that was initiated in response to a Proxy Put Request message, the respondent CFDP user that requested that transaction shall use the CFDP `Put.request` primitive to request delivery of a file delivery unit (to the originator of the proxy operation) whose metadata contains one each of the Reserved CFDP Messages of message types Proxy Put Response (defined in 6.2.5.2) and Originating Transaction ID, together with zero or more Reserved CFDP Messages of message type Proxy Filestore Response (defined in 6.2.5.3).

**6.2.5.1.2** If for any reason the `Put.request` cannot be honored, the respondent CFDP user shall take no further action with regard to this proxy operation.

#### 6.2.5.2 Proxy Put Response

The Proxy Put Response message is mandatory and shall be constructed as indicated in table 6-12.

**Table 6-12: Proxy Put Response Message**

| Field | Size (bits) | Values | Comment |
|---|---|---|---|
| Condition code | 4 | (See table 5-5.) | |
| Spare | 1 | All ‘zeroes’. | |
| Delivery Code | 1 | (See table 5-7.) | |
| File Status | 2 | (See table 5-7.) | |

#### 6.2.5.3 Proxy Filestore Response

If the `Transaction-Finished.indication` included filestore responses, then for each such response, one Proxy Filestore Response message shall be included and shall be constructed as indicated in table 6-13.

**Table 6-13: Proxy Filestore Response Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Length | 8 | Number of octets in the response field. | |
| Response | 8 *length | The value of this field is a single CFDP filestore response as defined in table 5-17. | |

### 6.2.6 CANCELING A PROXY PUT REQUEST

#### 6.2.6.1 General

When required to cancel a proxy operation, the proxy operation originator shall use the CFDP `Put.request` primitive to request delivery of a file delivery unit whose metadata contains one each of the Reserved CFDP Messages of message types Proxy Put Cancel, defined in 6.2.6.2, and Originating Transaction ID.

#### 6.2.6.2 Proxy Put Cancel

A Proxy Put Cancel message is mandatory. It has no content.

### 6.2.7 RESPONDING TO A PROXY PUT CANCEL

Upon receipt of a Proxy Put Cancel message, the respondent CFDP user shall use the `Cancel.request` primitive to request cancellation of the transaction that was initiated in response to the one identified by the accompanying Originating Transaction ID message. The results of this action shall be reported according to 6.2.5.

## 6.3 DIRECTORY OPERATIONS

### 6.3.1 GENERAL

To enable interoperability, the following mandatory behavior shall be observed by CFDP users that are in compliance with the CFDP directory operations specification.

### 6.3.2 DIRECTORY OPERATIONS MESSAGE TYPES

The message type field for each Reserved CFDP Message used in Directory operations shall contain one of the values specified in table 6-14.

**Table 6-14: Directory Operations Message Types**

| Message Type (hexadecimal) | Interpretation |
|---|---|
| 10 | Directory Listing Request |
| 11 | Directory Listing Response |

### 6.3.3 INITIATING A DIRECTORY LISTING

#### 6.3.3.1 General

When required to initiate a directory listing operation, the CFDP user shall use the CFDP `Put.request` primitive to request delivery of a file delivery unit whose metadata contains a Reserved CFDP Message of message type Directory Listing Request, defined in 6.3.3.2.

#### 6.3.3.2 Directory Listing Request

The Directory Listing Request message shall be constructed as indicated in table 6-15.

**Table 6-15: Directory Listing Request Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Directory Name | Variable | LV | |
| Directory File Name | Variable | LV | The file name and path at the filestore local to the requesting CFDP user in which the responding CFDP user should put the directory listing. |

### 6.3.4 RESPONDING TO A DIRECTORY LISTING REQUEST

#### 6.3.4.1 General

**6.3.4.1.1** Upon receipt of a Directory Listing Request message, the remote CFDP user shall use the CFDP `Put.request` primitive to request delivery of a file delivery unit to the entity which sent the original Directory Listing Request message; the file data of this file delivery unit shall be a file that contains the directory listing, while the metadata of this file delivery unit shall contain one each of the Reserved CFDP Messages of message types Directory Listing Response (defined in 6.3.4.2) and Originating Transaction ID.

**6.3.4.1.2** If the remote CFDP user is unable to provide a directory listing file, then the responding file delivery unit shall contain no file, and the listing response code in the Directory Listing Response message shall indicate that the directory listing request was unsuccessful.

### 6.3.4.2 Directory Listing Response

The Directory Listing Response message shall be constructed as indicated in table 6-16.

**Table 6-16: Directory Listing Response Message**

| Field | Size (bits) | Values (hexadecimal) | Comments |
|---|---|---|---|
| Listing Response Code | 1 | 0 — successful<br>1 — unsuccessful | |
| Spare | 7 | All 'zeroes'. | |
| Directory Name | Variable | LV | The name of the directory being listed, taken from the directory listing request. |
| Directory File Name | Variable | LV | The file name and path at the filestore local to the requesting CFDP in which the listing has been put, taken from the directory listing request. |

### 6.3.5 RETRIEVING A DIRECTORY LISTING

Upon receiving the Reserved CFDP Message with a Directory Listing Response message type, the requesting CFDP user may retrieve the listing from the file specified in the message.

## 6.4 REMOTE STATUS REPORT OPERATIONS

### 6.4.1 GENERAL

To enable interoperability, the following mandatory behavior shall be observed by CFDP users that are in compliance with the CFDP remote status report operations specification.

### 6.4.2 REMOTE STATUS REPORT OPERATIONS MESSAGE TYPES

The Remote Status Report Message to User message type field shall contain one of the values specified in table 6-17.

**Table 6-17: Remote Status Report Operations Message Types**

| Message Type (hexadecimal) | Interpretation |
|---|---|
| 20 | Remote Status Report Request |
| 21 | Remote Status Report Response |

### 6.4.3 INITIATING A REMOTE STATUS REPORT

#### 6.4.3.1 General

When required to initiate a remote status report operation, the CFDP user shall use the CFDP `Put.request` primitive to request delivery of a file delivery unit whose metadata contains the Reserved CFDP Message defined in 6.4.3.2.

#### 6.4.3.2 Remote Status Report Request

The Remote Status Report Request message shall be constructed as indicated in table 6-18.

**Table 6-18: Remote Status Report Request Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Reserved for future use | 1 | set to ‘0’ | |
| Length of entity ID | 3 | | Number of octets in source entity ID less one; that is, ‘0’ means that source entity ID is one octet. |
| Reserved for future use | 1 | set to ‘0’ | |
| Length of Transaction sequence number | 3 | | Number of octets in sequence number less one; that is, ‘0’ means that sequence number is one octet. |
| Source entity ID | Variable | | Identifies the entity that originated the transaction for which a status report is requested. An unsigned binary integer. |
| Transaction sequence number | Variable | | Uniquely identifies the transaction, among all transactions originated by this entity. An unsigned binary integer. |
| Report File Name | Variable | LV | The file name and path at the filestore local to the requesting CFDP user in which the responding CFDP user should put the status report. |

## 6.4.4 RESPONDING TO A REMOTE STATUS REPORT REQUEST

### 6.4.4.1 General

**6.4.4.1.1** Upon receipt of a Remote Status Report Request message, the remote CFDP user shall use the CFDP `Put.request` primitive to request delivery of a file delivery unit to the entity that sent the original Status Report Request message. The file data of this file delivery unit shall be a file that contains the status report for the indicated transaction, while the metadata of this file delivery unit shall contain one each of the Reserved CFDP Messages of message types Remote Status Report Response (defined in 6.4.4.2) and Originating Transaction ID.

**6.4.4.1.2** If the remote CFDP user is unable to provide a status report file, then the responding file delivery unit shall contain no file, and the report response code in the Remote Status Report Response message shall indicate that the remote status report request was unsuccessful.

**6.4.4.1.3** If the remote CFDP user is able to provide a status report file, the nature of the status report shall be as described in 3.2.11.

### 6.4.4.2 Remote Status Report Response

The Remote Status Report Response message shall be constructed as indicated in table 6-19.

**Table 6-19: Remote Status Report Response Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Transaction status | 2 | Same values as defined for transaction status in the ACK PDU (see table 5-8) | |
| Spare | 5 | All ‘zeroes’ | |
| Report response code | 1 | ‘0’ — unsuccessful<br>’1’ — successful | |
| Spare | 1 | ‘0’ | This field and the following five fields identify the transaction whose status is being reported; they are taken from the status report request. |
| Length of Entity ID | 3 | | |
| Spare | 1 | ‘0’ | |
| Length of Sequence Number | 3 | | |
| Source Entity ID | Variable | | An unsigned binary integer. |
| Transaction Sequence Number | Variable | | An unsigned binary integer. |

### 6.4.5 RETRIEVING A REMOTE STATUS REPORT

Upon receiving the Reserved CFDP Message with a Remote Status Report Response message type, the requesting CFDP user may retrieve the report from the file specified in the message.

## 6.5 REMOTE SUSPEND OPERATIONS

### 6.5.1 GENERAL

To enable interoperability, the following mandatory behavior shall be observed by CFDP users that are in compliance with the CFDP remote suspend operations specification.

### 6.5.2 REMOTE SUSPEND OPERATIONS MESSAGE TYPES

The Remote Suspend Message to User message type field shall contain one of the values specified in table 6-20.

**Table 6-20: Remote Suspend Operations Message Types**

| Message Type (hexadecimal) | Interpretation |
|---|---|
| 30 | Remote Suspend Request |
| 31 | Remote Suspend Response |

### 6.5.3 INITIATING A REMOTE SUSPEND

#### 6.5.3.1 General

**6.5.3.1.1** When required to initiate a remote suspend operation, the CFDP user shall use the CFDP `Put.request` primitive to request delivery of a file delivery unit whose metadata contains the Reserved CFDP Message defined in 6.5.3.2.

**6.5.3.1.2** The transaction mode of the `Put.request` primitive shall be Acknowledged.

#### 6.5.3.2 Remote Suspend Request

The Remote Suspend Request message shall be constructed as indicated in table 6-21.

**Table 6-21: Remote Suspend Request Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Reserved for future use | 1 | set to '0' | |
| Length of entity ID | 3 | | Number of octets in source entity ID less one; that is, '0' means that source entity ID is one octet. |
| Reserved for future use | 1 | set to '0' | |
| Length of Transaction sequence number | 3 | | Number of octets in sequence number less one; that is, '0' means that sequence number is one octet. |
| Source entity ID | Variable | | Identifies the entity that originated the transaction that is to be suspended. An unsigned binary integer. |
| Transaction sequence number | Variable | | Uniquely identifies the transaction among all transactions originated by this entity. An unsigned binary integer. |

### 6.5.4 RESPONDING TO A REMOTE SUSPEND REQUEST

#### 6.5.4.1 General

**6.5.4.1.1** Upon receipt of a Remote Suspend Request message, the remote CFDP user shall assure the suspension of the indicated transaction if possible; the CFDP **`Suspend.request`** primitive will nominally be used for this purpose. It shall then use the CFDP **`Put.request`** primitive to request delivery of a file delivery unit to the entity that sent the original Suspend Request message. The metadata of this file delivery unit shall contain one each of the Reserved CFDP Messages of message types Remote Suspend Response (defined in 6.5.4.2) and Originating Transaction ID.

**6.5.4.1.2** It is possible for there to be multiple concurrent motivations for suspending a transaction; for example, a CFDP entity might receive a Remote Suspend Request for a transaction that is already suspended due to a locally detected fault condition. The CFDP user application is responsible for managing a queue of such suspension orders in such a way that a suspended transaction remains suspended until all such orders have been explicitly reversed by corresponding resume orders. In particular, a transaction for which a suspension-inducing fault condition was detected must remain suspended until the CFDP user application determines that this fault condition no longer holds, and a transaction for which a remote suspend request was received from a given CFDP entity's user must remain suspended until a remote resume request for that transaction is received from the same user. The manner in which this is accomplished is an implementation matter.

#### 6.5.4.2 Remote Suspend Response

The Remote Suspend Response message shall be constructed as indicated in table 6-22.

**Table 6-22: Remote Suspend Response Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Suspension indicator | 1 | ‘0’ — not suspended<br>‘1’ — suspended | |
| Transaction status | 2 | Same values as defined for transaction status in the ACK PDU (see table 5-8) | |
| Spare | 5 | All ‘zeroes’ | |
| Spare | 1 | ‘0’ | This field and the following five fields identify the transaction whose suspension was requested; they are taken from the suspend request. |
| Length of Entity ID | 3 | | |
| Spare | 1 | ‘0’ | |
| Length of Sequence Number | 3 | | |
| Source Entity ID | Variable | | An unsigned binary integer. |
| Transaction Sequence Number | Variable | | An unsigned binary integer. |

## 6.6 REMOTE RESUME OPERATIONS

### 6.6.1 GENERAL

To enable interoperability, the following mandatory behavior shall be observed by CFDP users that are in compliance with the CFDP remote resume operations specification.

## 6.6.2 REMOTE RESUME OPERATIONS MESSAGE TYPES

The Remote Resume Message to User message type field shall contain one of the values specified in table 6-23.

**Table 6-23: Remote Resume Operations Message Types**

| Message Type (hexadecimal) | Interpretation |
|---|---|
| 38 | Remote Resume Request |
| 39 | Remote Resume Response |

## 6.6.3 INITIATING A REMOTE RESUME

### 6.6.3.1 General

**6.6.3.1.1** When required to initiate a remote resume operation, the CFDP user shall use the CFDP **`Put.request`** primitive to request delivery of a file delivery unit whose metadata contains the Reserved CFDP Message defined in 6.6.3.2.

**6.6.3.1.2** The transaction mode of the **`Put.request`** primitive shall be Acknowledged.

#### 6.6.3.2 Remote Resume Request

The Remote Resume Request message shall be constructed as indicated in table 6-24.

**Table 6-24: Remote Resume Request Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Reserved for future use | 1 | set to ‘0’ | |
| Length of entity ID | 3 | | Number of octets in source entity ID less one; that is, ‘0’ means that source entity ID is one octet. |
| Reserved for future use | 1 | set to ‘0’ | |
| Length of Transaction sequence number | 3 | | Number of octets in sequence number less one; that is, ‘0’ means that sequence number is one octet. |
| Source entity ID | Variable | | Identifies the entity that originated the transaction that is to be resumed. An unsigned binary integer. |
| Transaction sequence number | Variable | | Uniquely identifies the transaction, among all transactions originated by this entity. An unsigned binary integer. |

### 6.6.4 RESPONDING TO A REMOTE RESUME REQUEST

#### 6.6.4.1 General

Upon receipt of a Remote Resume Request message, the remote CFDP user shall assure the resumption of the indicated transaction if possible; the CFDP `Resume.request` primitive will nominally be used for this purpose. It shall then use the CFDP `Put.request` primitive to request delivery of a file delivery unit to the entity which sent the original Resume Request message. The metadata of this file delivery unit shall contain one each of the Reserved CFDP Messages of message types Remote Resume Response (defined in 6.6.4.2) and Originating Transaction ID.

NOTE – Suspension order queue management requirements are specified in 6.5.4.1.2.

#### 6.6.4.2 Remote Resume Response

The Remote Resume Response message shall be constructed as indicated in table 6-25.

**Table 6-25: Remote Resume Response Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Suspension indicator | 1 | ‘0’ — not suspended<br>‘1’ — suspended | Because multiple motivations for suspending a transaction may be concurrently valid, the successful processing of a Remote Resume Request may not actually change the suspension status of the affected transaction. |
| Transaction status | 2 | Same values as defined for transaction status in the ACK PDU (see table 5-8) | |
| Spare | 5 | All ‘zeroes’ | |
| Spare | 1 | ‘0’ | This field and the following five fields identify the transaction whose resumption was requested; they are taken from the resume request. |
| Length of Entity ID | 3 | | |
| Spare | 1 | ‘0’ | |
| Length of Sequence Number | 3 | | |
| Source Entity ID | Variable | | An unsigned binary integer. |
| Transaction Sequence Number | Variable | | An unsigned binary integer. |

# 7 CFDP SERVICE CLASSES

NOTE – This section describes the minimum functionality of CFDP required to realize interoperable implementations of certain identified application services. The classes are, as yet, neither definitive nor complete. For each class, an event diagram is included to illustrate the sequence of events associated with the use of that protocol class.

## 7.1 DEFINED CLASSES

The defined CFDP classes are:

a) Class 1—Unreliable Transfer;

b) Class 2—Reliable Transfer.

NOTES

1 State Transition Diagrams and State Tables for each of the Service Classes are provided in reference [D4].

2 The following subsections describe the functions and procedures, contained in section 4 of this document, that are applicable to each Class. In order to implement the Class, it is only necessary to make reference to the General Characteristics overview of 2.3, the subsections specified in the Class procedures, and any PDU descriptions (section 5) referenced in the procedures.

## 7.2 FUNCTIONS OF CLASS 1—UNRELIABLE TRANSFER

NOTE – Class 1 provides for the unreliable delivery of bounded or unbounded data files from the source to the destination over simplex or duplex paths. Class-1 procedures may be exercised over simplex transmission paths unless transaction closure is requested, in which case, duplex transmission paths are required.

### 7.2.1 PROCEDURES FOR CLASS 1—SOURCE

For Class 1 functionality, the source CFDP entity shall use the procedures specified in table 7-1.

**Table 7-1: Class 1 Source Procedures**

| Procedure | Subsection | Notes |
|---|---|---|
| CRC Procedures at the PDU Transmitting Entity | 4.1.1 | |
| CRC Validation Procedure | 4.1.3 | |
| Checksum Procedures | 4.2 | |
| Put Procedures | 4.3 | |
| Transaction Start Notification Procedure | 4.4 | |
| PDU Forwarding Procedures | 4.5 | Applicable to Metadata, File Data, and EOF PDUs only. |
| Copy File Procedures at the Sending Entity | 4.6.1.1 | Transaction closure may be requested if the path to the receiving entity is duplex but shall not be requested if the path to the receiving entity is simplex. |
| Unacknowledged Mode Procedures at the Sending Entity | 4.6.3.2 | |
| Resume Procedures | 4.6.7 | Procedures applicable to the sending entity in unacknowledged mode. |
| Report Procedures | 4.6.8 | |
| Fault Handling Procedures | 4.8 | Issuance of Notice of Cancellation is basic interoperable action. |
| Filestore Procedures | 4.9 | Only procedures applicable to the sending entity. |
| Internal Procedures | 4.11 | Procedures applicable to the sending entity in unacknowledged mode. |
| Transmission Opportunity Procedures | 4.12.2 | |

## 7.2.2 PROCEDURES FOR CLASS 1—DESTINATION

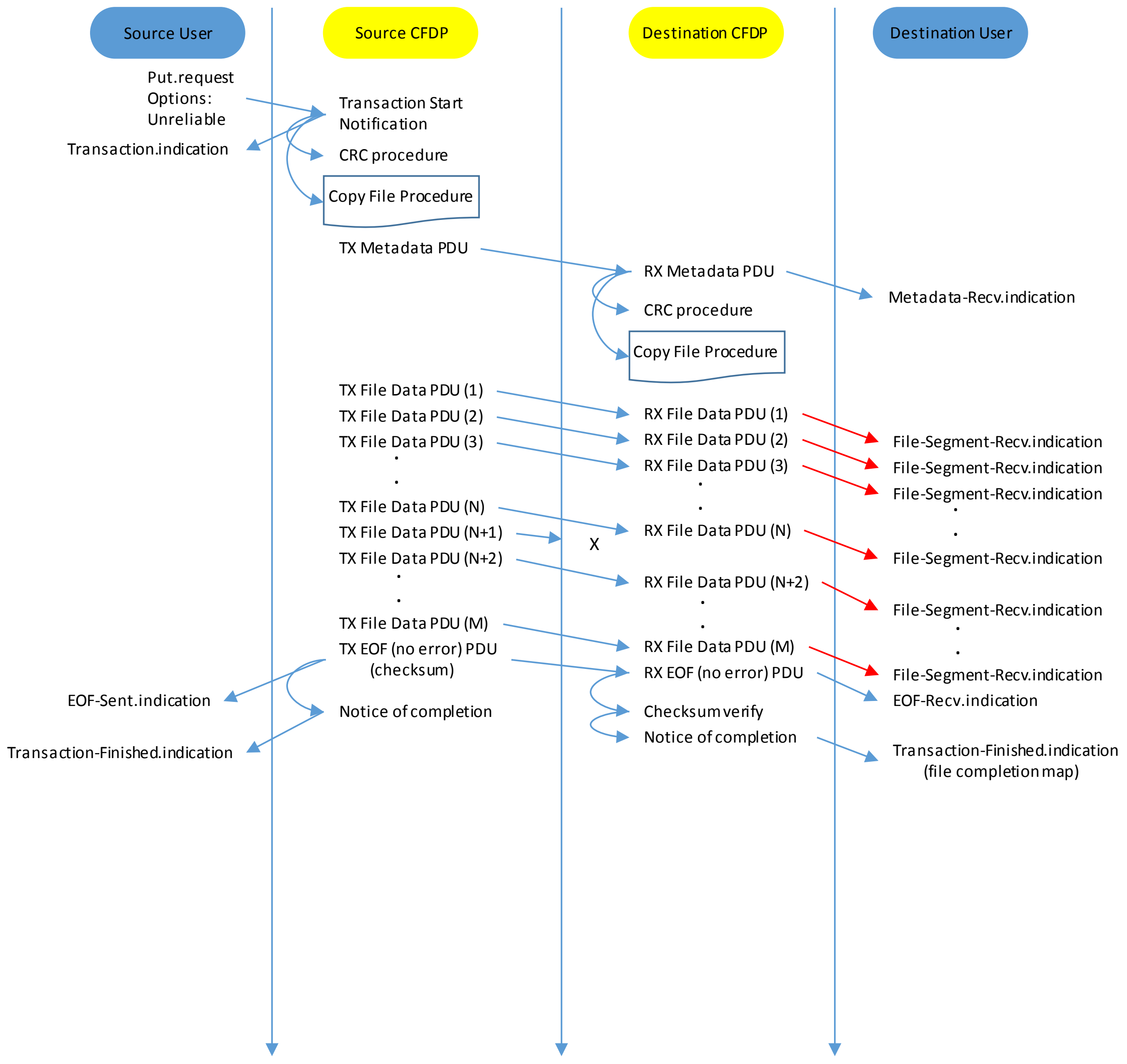
For Class 1 functionality, the destination CFDP entity shall use the procedures specified in table 7-2.

**Table 7-2: Class 1 Destination Procedures**

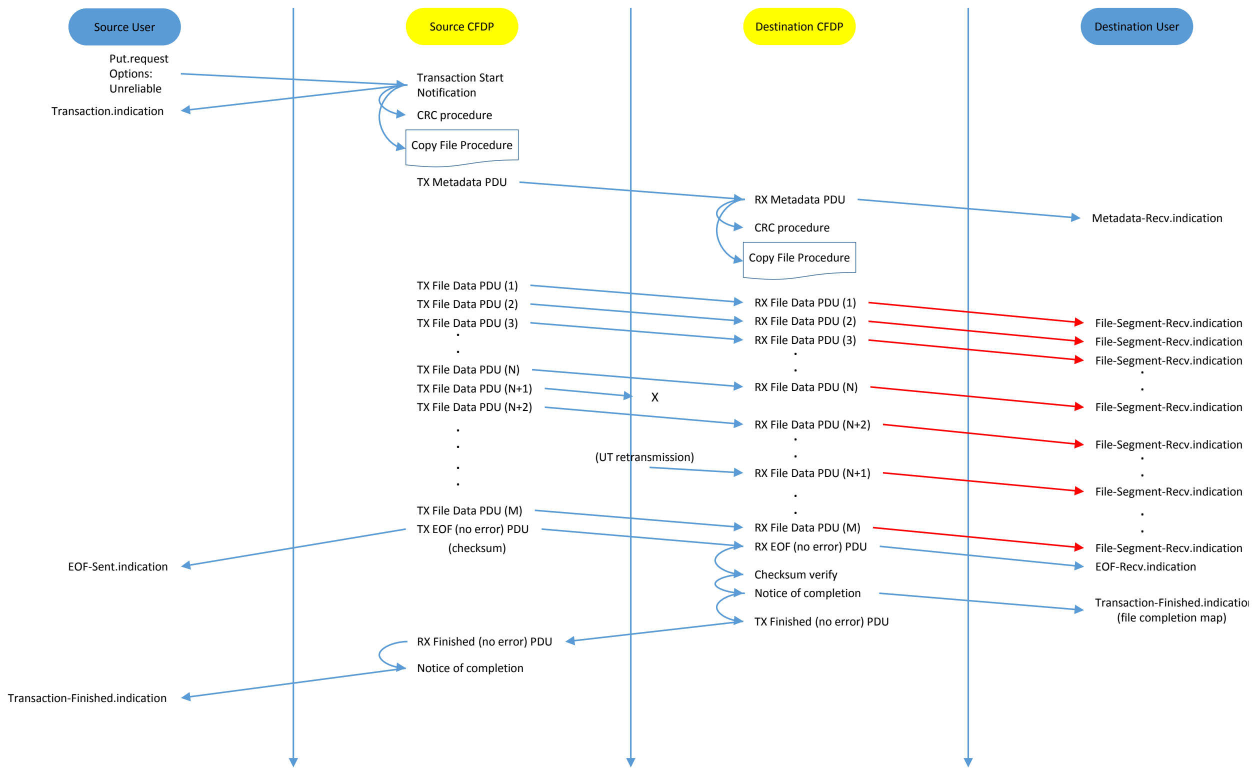
| Procedure | Subsection | Notes |
|---|---|---|
| CRC Procedures at the PDU Receiving Entity | 4.1.2 | |
| CRC Validation Procedure | 4.1.3 | |
| Checksum Procedures | 4.2 | |
| PDU Forwarding Procedures | 4.5 | Only the discarding of mis-delivered PDUs (4.5.4) is applicable. |
| Copy File Procedures at the Receiving Entity | 4.6.1.2 | |
| Unacknowledged Mode Procedures at the Receiving Entity | 4.6.3.3 | |
| Cancel Response Procedures at the Receiving Entity | 4.6.6.1 | Positive Acknowledgement not required. |
| Report Procedures | 4.6.8 | |
| Fault Handling Procedures | 4.8 | Abandonment of transaction is basic interoperable action. |
| Filestore Procedures | 4.9 | Only procedures applicable to the receiving entity. |
| Inactivity Monitor Procedures | 4.10 | |
| Internal Procedures | 4.11 | Incomplete data forwarded. Procedures applicable to the receiving entity in unacknowledged mode. |

## 7.2.3 EVENT DIAGRAMS FOR CLASS 1

Transaction Closure Not Requested.


Source User
Source CFDP
Destination CFDP
Destination User
Put.request Options: Unreliable
Transaction Start Notification
Transaction.indication
CRC procedure
Copy File Procedure
TX Metadata PDU
RX Metadata PDU
Metadata-Recv.indication
CRC procedure
Copy File Procedure
TX File Data PDU (1)
TX File Data PDU (2)
TX File Data PDU (3)
TX File Data PDU (N)
TX File Data PDU (N+1)
TX File Data PDU (N+2)
TX File Data PDU (M)
TX EOF (no error) PDU (checksum)
X
RX File Data PDU (1)
RX File Data PDU (2)
RX File Data PDU (3)
RX File Data PDU (N)
RX File Data PDU (N+2)
RX File Data PDU (M)
RX EOF (no error) PDU
File-Segment-Recv.indication
File-Segment-Recv.indication
File-Segment-Recv.indication
File-Segment-Recv.indication
File-Segment-Recv.indication
File-Segment-Recv.indication
EOF-Recv.indication
EOF-Sent.indication
Notice of completion
Transaction-Finished.indication
Checksum verify
Notice of completion
Transaction-Finished.indication (file completion map)

### 7.2.3.1 Transaction Closure Requested



## 7.3 FUNCTIONS OF CLASS 2—RELIABLE TRANSFER

NOTE – Class 2 provides for the reliable delivery of bounded or unbounded data files from the source to the destination.

### 7.3.1 PROCEDURES FOR CLASS 2—SOURCE

For Class 2 functionality, the source CFDP entity shall use the procedures specified in table 7-3.

**Table 7-3: Class 2 Source Procedures**

| Procedure | Subsection | Notes |
|---|---|---|
| CRC Procedures | 4.1 | |
| Checksum Procedures | 4.2 | |
| Put Procedures | 4.3 | |
| Transaction Start Notification Procedure | 4.4 | |
| PDU Forwarding Procedures | 4.5 | |
| Copy File Procedures at the Sending Entity | 4.6.1.1 | |
| Acknowledged Mode Procedures at the Sending Entity | 4.6.4.2 | |
| Incremental Lost Segment Detection Procedures at the Sending Entity | 4.6.4.5 | |
| Keep Alive Procedures at the Sending Entity | 4.6.5.3 | |
| Cancel Response Procedures at the Sending Entity | 4.6.6.2 | |
| Resume Procedures | 4.6.7 | Only procedures applicable to the sending entity. |
| Report Procedures | 4.6.8 | |
| Positive Acknowledgement Procedures at PDU Sending End | 4.7.1 | |
| Positive Acknowledgement Procedures at PDU Receiving End | 4.7.2 | |
| Fault Handling Procedures | 4.8 | Only issuance of Notice of Cancellation is required. |
| Filestore Procedures | 4.9 | Only procedures applicable to the sending entity. |
| Inactivity Monitor Procedures | 4.10 | |
| Internal Procedures | 4.11 | Only procedures applicable to the sending entity. |
| Link State Change Procedures | 4.12 | |

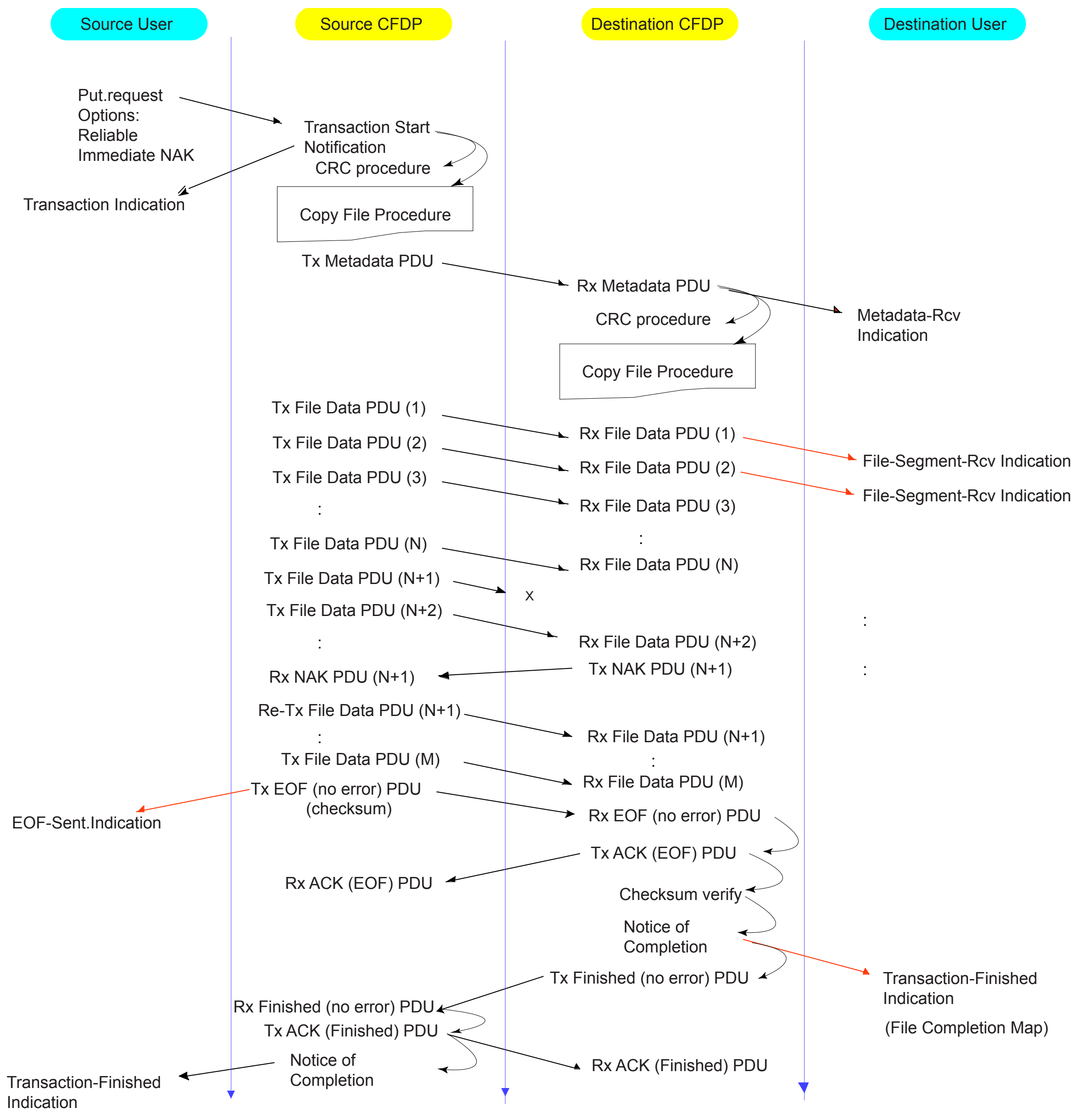
### 7.3.2 PROCEDURES FOR CLASS 2—DESTINATION

For Class 2 functionality, the destination CFDP entity shall use the procedures specified in table 7-4.

**Table 7-4: Class 2 Destination Procedures**

| Procedure | Subsection | Notes |
|---|---|---|
| CRC Procedures | 4.1 | |
| Checksum Procedures | 4.2 | |
| PDU Forwarding Procedures | 4.5 | |
| Copy File Procedures at the Receiving Entity | 4.6.1.2 | |
| Acknowledged Mode Procedures at the Receiving Entity | 4.6.4.3 | |
| Incremental Lost Segment Detection Procedures at the Receiving Entity | 4.6.4.4 | Deferred Mode required only for interoperability. |
| Keep Alive Procedures at the Receiving Entity | 4.6.5.2 | |
| Cancel Response Procedures at the Receiving Entity | 4.6.6.1 | Positive Acknowledgement required.<br>Incomplete data forwarded. |
| Resume Procedures | 4.6.7 | Only procedures applicable to the receiving entity. |
| Report Procedures | 4.6.8 | |
| Positive Acknowledgement Procedures at PDU Sending End | 4.7.1 | |
| Positive Acknowledgement Procedures at PDU Receiving End | 4.7.2 | |
| Fault Handling Procedures | 4.8 | Only issuance of Notice of Cancellation is required. |
| Filestore Procedures | 4.9 | Only procedures applicable to the receiving entity. |
| Inactivity Monitor Procedures | 4.10 | |
| Internal Procedures | 4.11 | Incomplete data forwarded. Only procedures applicable to the receiving entity. |
| Link State Change Procedures | 4.12 | |

## 7.3.3 EVENT DIAGRAM FOR CLASS 2



# 8 MANAGEMENT INFORMATION BASE

## 8.1 GENERAL

The operation of each CFDP entity shall be supported by a single MIB comprising the items of information described below.

NOTES

1 This section describes the MIB.

2 Representation of, and mechanisms for access to, MIB items will be implementation matters. In particular, determination of which items will be static and which will be dynamic is a matter of implementation.

## 8.2 LOCAL ENTITY CONFIGURATION INFORMATION

For each item of local entity configuration information, a single value shall apply that shall pertain to the entire CFDP entity.

**Table 8-1: Local Entity Configuration Information**

| Item | Comment |
|---|---|
| Local Entity ID | |
| `EOF-Sent.indication` required | True or false. |
| `EOF-Recv.indication` required | True or false. |
| `File-Segment-Recv.indication` required | True or false. |
| `Transaction-Finished.indication` required when acting as receiving entity | True or false. |
| `Suspended.indication` required when acting as receiving entity | True or false. |
| `Resumed.indication` required when acting as receiving entity | True or false. |
| Default fault handlers | For each type of fault condition, a default handler (as enumerated in 4.8.2). |

## 8.3 REMOTE ENTITY CONFIGURATION INFORMATION

For each item of remote entity configuration information, a single value shall apply for each remote entity with which the local CFDP entity may be in direct communication.

**Table 8-2: Remote Entity Configuration Information**

| Item | Comment |
|---|---|
| Remote entity ID | |
| Protocol version number | CFDP protocol version implemented at this entity. |
| UT address | UT Address to use when transmitting to this entity. |
| Positive ACK timer interval | Expressed as a time interval, or N/A. |
| NAK timer interval | Expressed as a time interval, or N/A. |
| Keep Alive interval | Expressed as a time interval, or N/A. |
| Immediate NAK mode enabled | True or false. |
| Default transmission mode | Acknowledged or unacknowledged. |
| Transaction closure requested | True or false. |
| Check limit | For use in determining imputed end of transaction. |
| Default type of checksum to calculate for all file transmission to this remote entity | As defined in the SANA Checksum Types registry. |
| Disposition of incomplete received file on transaction cancellation | Discard or retain. |
| CRCs required on transmission | True or false. |
| Maximum file segment length | In octets. |
| Keep Alive discrepancy limit | Expressed as a number of octets, or N/A. |
| Positive ACK timer expiration limit | Number of expirations. |
| NAK timer expiration limit | Number of expirations. |
| Transaction inactivity limit | A time limit. |
| Start of transmission opportunity | A signal produced by the operating environment. |
| End of transmission opportunity | A signal produced by the operating environment. |
| Start of reception opportunity | A signal produced by the operating environment. |
| End of reception opportunity | A signal produced by the operating environment. |

# ANNEX A

# PROTOCOL IMPLEMENTATION CONFORMANCE STATEMENT PROFORMA

# (NORMATIVE)

## A1 INTRODUCTION

### A1.1 GENERAL

This annex provides the Protocol Implementation Conformance Statement (PICS) Requirements List (RL) for implementations of the *CCSDS File Delivery Protocol (CFDP)*, CCSDS 727.0-B-5, July 2020. The PICS for an implementation is generated by completing the RL in accordance with the instructions below. An implementation shall satisfy the mandatory conformance requirements of the base standards referenced in the RL.

An implementation's completed RL is called the PICS. The PICS states which capabilities and options of the protocol have been implemented. The following can use the PICS:

– the protocol implementer, as a checklist to reduce the risk of failure to conform to the standard through oversight;

– the supplier and acquirer or potential acquirer of the implementation, as a detailed indication of the capabilities of the implementation, stated relative to the common basis for understanding provided by the standard PICS proforma;

– the user or potential user of the implementation, as a basis for initially checking the possibility of interoperating with another implementation.

– a protocol tester, as the basis for selecting appropriate tests against which to assess the claim for conformance of the implementation.

### A1.2 NOTATION

The following are used in the RL to indicate the status of features:

#### A1.2.1 Status Symbols

| | |
|---|---|
| M | mandatory. |
| O | optional. |

#### A1.2.2 Support Column Symbols

The support of every item as claimed by the implementer is stated by entering the appropriate answer (Y, N, or N/A) in the support column:

| | |
|---|---|
| Y | Yes, supported by the implementation. |
| N | No, not supported by the implementation. |
| N/A | Not applicable. |

### A1.3 INSTRUCTIONS FOR COMPLETING THE RL

An implementer shows the extent of compliance to the protocol by completing the RL; that is, the RL shows compliance to all mandatory requirements and lists all options that are not supported. The resulting completed RL is called a PICS. In the Support column, each response shall either be a response code selected from the indicated set of responses or else comprise one or more parameter values, as appropriate. If a conditional requirement is inapplicable, N/A should be used. If a mandatory requirement is not satisfied, exception information must be supplied by entering a reference Xi, where i is a unique identifier to an accompanying rationale for the noncompliance.

### A1.4 REFERENCED BASE STANDARDS

The base standards referenced in the RL are:

– CCSDS File Delivery Protocol (this document).

### A1.5 IDENTIFICATION OF PICS

| Ref | Question | |
|---|---|---|
| 1 | Date of Statement (DD/MM/YYYY) | |
| 2 | PICS serial number | |
| 3 | System Conformance statement cross-reference | |

## A2 IDENTIFICATION OF THE IMPLEMENTATION

| | |
|---|---|
| Implementation Name | |
| Implementation Version | |
| Special Configuration | |
| Supplier | |
| Contact Point for Queries | |
| Implementation Name(s) and Versions | |
| Other Information Necessary for full identification, for example, Project developed for, name(s) and version(s) for machines and/or operating systems, system name(s). | |

## A3 IDENTIFICATION OF THE PROTOCOL

| | |
|---|---|
| Protocol Name | CCSDS FILE DELIVERY PROTOCOL |
| Protocol Version | |
| Addenda Implemented | |
| Amendments Implemented | |
| Have any exceptions been required?<br><br>(Note: A YES answer means that the implementation does not conform to the protocol. Non-supported mandatory capabilities are to be identified in the PICS, with an explanation of why the implementation is non-conforming. | Yes_______________ No_______________ |
| Date of Statement | |

## A4 ICS PROFORMA TABLES

## A4.1 SERVICE CLASS 1 – SENDER

### A4.1.1 CFDP Protocol Data Units

| Type ID | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| CFDP-1S-01 | General | 5.1, tables 5-1, 5-2, 5-3<br>5.2.1, tables 5-4, 5-5 | M | |
| CFDP-1S-02 | End-of-file | 5.2.2, table 5-6 | M | |
| CFDP-1S-03 | Finished | 5.2.3, table 5-7<br>5.4, tables 5-17 and 5-18 | M | |
| CFDP-1S-04 | Metadata | 5.2.5, table 5-9<br>5.4, tables 5-15, 5-16, 5-19 | M | |
| CFDP-1S-05 | File data | 5.3, table 5-14 | M | |

### A4.1.2 CFDP Procedures

| Item | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| CFDP-1S-06 | CRC procedures | 4.1 | M | |
| CFDP-1S-07 | Integrity protection | 4.2.1.1 | M | |
| CFDP-1S-21 | Checksum length | 4.2.1.2 | M | |
| CFDP-1S-22 | Applicable checksum algorithm | 4.2.2.1 | M | |
| CFDP-1S-23 | Available checks algorithms | 4.2.2.2 | M | |
| CFDP-1S-24 | Modular checksum algorithm implemented | 4.2.2.3 | M | |
| CFDP-1S-25 | Null checksum algorithm implemented | 4.2.2.4 | M | |
| CFDP-1S-26 | Other checksum algorithms implemented | 4.2.2.5 | O | |
| CFDP-1S-27 | Preferred checksum algorithm | 4.2.2.6 | M | |
| CFDP-1S-28 | Preferred checksum algorithm is available | 4.2.2.7 | M | |
| CFDP-1S-29 | Preferred checksum algorithm not available | 4.2.2.8 | M | |
| CFDP-1S-30 | insertion of checksum into EOF PDU | 4.2.3.1 | M | |
| CFDP-1S-31 | Preferred checksum algorithm at sending entity | 4.2.3.2 | M | |
| CFDP-1S-32 | Verification of checksum received in EOF PDU | 4.2.4.1 | M | |
| CFDP-1S-33 | Preferred checksum algorithm at receiving entity | 4.2.4.2 | M | |
| CFDP-1S-34 | Modular checksum algorithm definition | 4.2.5 | M | |
| CFDP-1S-08 | Put procedures | 4.3 | M | |
| CFDP-1S-09 | Transaction start notification procedure | 4.4 | M | |
| CFDP-1S-10 | PDU forwarding procedures | 4.5 | M | |
| CFDP-1S-11 | Copy file general procedures | 4.6 | M | |
| CFDP-1S-12 | Copy file procedures at the sending entity | 4.6.1.1 except 4.6.1.1.5.1 and 4.6.1.1.5.2 | M | |
| CFDP-1S-13 | Record continuation state and segment metadata procedures | 4.6.1.1.5.1, 4.6.1.1.5.2 | O | |
| CFDP-1S-14 | Unacknowledged mode procedures | 4.6.3 | M | |
| CFDP-1S-15 | Resume procedures | 4.6.7 | M | |
| CFDP-1S-16 | Report procedures | 4.6.8 | M | |
| CFDP-1S-17 | Fault handling procedures | 4.8 | M | |
| CFDP-1S-18 | Filestore procedures | 4.9 | M | |
| CFDP-1S-19 | Internal procedures | 4.11 | M | |
| CFDP-1S-20 | Link state change procedures | 4.12 | M | |

If other checksum algorithms are implemented (CFDP-1S-26), the algorithms that are implemented shall be reported in the PICS.

### A4.1.3 Management Information Base Local Entity Configuration Parameters

| Item | Protocol Feature | Reference | Status | Parameter Value |
|---|---|---|---|---|
| LMIB-1S-01 | Local Entity ID | 8.2 | M | |
| LMIB-1S-02 | `EOF-Sent.indication` required | 8.2 | M | |
| LMIB-1S-03 | Default fault handlers | 8.2 | M | |

### A4.1.4 Management Information Base Remote Entity Configuration Parameters

| Item | Protocol Feature | Reference | Status | Parameter Value |
|---|---|---|---|---|
| RMIB-1S-01 | Remote entity ID | 8.3 | M | |
| RMIB-1S-02 | Protocol version number | 8.3 | M | |
| RMIB-1S-03 | UT address | 8.3 | M | |
| RMIB-1S-04 | Default transmission mode | 8.3 | M | |
| RMIB-1S-05 | Transaction closure requested | 8.3 | M | |
| RMIB-1S-06 | Check limit | 8.3 | M | |
| RMIB-1S-07 | Type of checksum to calculate for all file transmission to this remote entity | 8.3 | M | |
| RMIB-1S-08 | CRCs required on transmission | 8.3 | M | |
| RMIB-1S-09 | Maximum file segment length | 8.3 | M | |
| RMIB-1S-10 | Start of transmission opportunity | 8.3 | M | |
| RMIB-1S-11 | End of transmission opportunity | 8.3 | M | |
| RMIB-1S-12 | Start of reception opportunity | 8.3 | M | |
| RMIB-1S-13 | End of reception opportunity | 8.3 | M | |

## A4.2 SERVICE CLASS 1 – RECEIVER

### A4.2.1 CFDP Protocol Data Units

| Type ID | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| CFDP-1R-01 | General | 5.1, tables 5-1, 5-2, 5-3<br>5.2.1, tables 5-4, 5-5 | M | |
| CFDP-1R-02 | End-of-file | 5.2.2, table 5-6 | M | |
| CFDP-1R-03 | Finished | 5.2.3, table 5-7<br>5.4, tables 5-17 and 5-18 | M | |
| CFDP-1R-04 | Metadata | 5.2.5, table 5-9<br>5.4, tables 5-15, 5-16, 5-19 | M | |
| CFDP-1R-05 | File data | 5.3, table 5-14 | M | |

### A4.2.2 CFDP Procedures

| Item | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| CFDP-1R-06 | CRC procedures | 4.1 | M | |
| CFDP-1R-07 | Integrity protection | 4.2.1.1 | M | |
| CFDP-1R-19 | Checksum length | 4.2.1.2 | M | |
| CFDP-1R-20 | Applicable checksum algorithm | 4.2.2.1 | M | |
| CFDP-1R-21 | Available checks algorithms | 4.2.2.2 | M | |
| CFDP-1R-22 | Modular checksum algorithm implemented | 4.2.2.3 | M | |
| CFDP-1R-23 | Null checksum algorithm implemented | 4.2.2.4 | M | |
| CFDP-1R-24 | Other checksum algorithms implemented | 4.2.2.5 | O | |
| CFDP-1R-25 | Preferred checksum algorithm | 4.2.2.6 | M | |
| CFDP-1R-26 | Preferred checksum algorithm is available | 4.2.2.7 | M | |
| CFDP-1R-27 | Preferred checksum algorithm not available | 4.2.2.8 | M | |
| CFDP-1R-28 | insertion of checksum into EOF PDU | 4.2.3.1 | M | |
| CFDP-1R-29 | Preferred checksum algorithm at sending entity | 4.2.3.2 | M | |
| CFDP-1R-30 | Verification of checksum received in EOF PDU | 4.2.4.1 | M | |
| CFDP-1R-31 | Preferred checksum algorithm at receiving entity | 4.2.4.2 | M | |
| CFDP-1R-32 | Modular checksum algorithm definition | 4.2.5 | M | |
| CFDP-1R-08 | PDU forwarding procedures | 4.5 | M | |
| CFDP-1R-09 | Copy file procedures at the receiving entity | 4.6.1.2 | M | |
| CFDP-1R-10 | Optional copy file procedures at the receiving entity | 4.6.2 | O | |
| CFDP-1R-11 | Unacknowledged mode procedures at the receiving entity | 4.6.3.3 | M | |
| CFDP-1R-12 | Cancel response procedures | 4.6.6 | M | |
| CFDP-1R-13 | Report procedures | 4.6.8 | M | |
| CFDP-1R-14 | Fault handling procedures | 4.8 | M | |
| CFDP-1R-15 | Filestore procedures | 4.9 | M | |
| CFDP-1R-16 | Inactivity monitor procedures | 4.10 | M | |
| CFDP-1R-17 | Internal procedures | 4.11 | M | |
| CFDP-1R-18 | Link state change procedures | 4.12 | M | |

If other checksum algorithms are implemented (CFDP-1R-24), the algorithms that are implemented shall be reported in the PICS.

### A4.2.3 Management Information Base Local Entity Configuration Parameters

| Item | Protocol Feature | Reference | Status | Parameter Value |
|---|---|---|---|---|
| LMIB-1R-01 | Local Entity ID | 8.2 | M | |
| LMIB-1R-02 | `EOF-Recv.indication` required | 8.2 | M | |
| LMIB-1R-03 | `File-Segment-Recv.indication` required | 8.2 | M | |
| LMIB-1R-04 | `Transaction-Finished.indication` required when acting as receiving entity | 8.2 | M | |
| LMIB-1R-05 | `Suspended.indication` required when acting as receiving entity | 8.2 | M | |
| LMIB-1R-06 | `Resumed.indication` required when acting as receiving entity | 8.2 | M | |
| LMIB-1R-07 | Default fault handlers | 8.2 | M | |

### A4.2.4 Management Information Base Remote Entity Configuration Parameters

| Item | Protocol Feature | Reference | Status | Parameter Value |
|---|---|---|---|---|
| RMIB-1R-01 | Remote entity ID | 8.3 | M | |
| RMIB-1R-02 | Protocol version number | 8.3 | M | |
| RMIB-1R-03 | UT address | 8.3 | M | |
| RMIB-1R-04 | Default transmission mode | 8.3 | M | |
| RMIB-1R-05 | Transaction closure requested | 8.3 | M | |
| RMIB-1R-06 | Check limit | 8.3 | M | |
| RMIB-1R-08 | Disposition of incomplete received file on transaction cancellation | 8.3 | M | |
| RMIB-1R-09 | CRCs required on transmission | 8.3 | M | |
| RMIB-1R-10 | Transaction inactivity limit | 8.3 | M | |
| RMIB-1R-11 | Start of transmission opportunity | 8.3 | M | |
| RMIB-1R-12 | End of transmission opportunity | 8.3 | M | |
| RMIB-1R-13 | Start of reception opportunity | 8.3 | M | |
| RMIB-1R-14 | End of reception opportunity | 8.3 | M | |

## A4.3 SERVICE CLASS 2 – SENDER

### A4.3.1 CFDP Protocol Data Units

| Type ID | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| CFDP-2S-01 | General | 5.1, tables 5-1, 5-2, 5-3<br>5.2.1, tables 5-4, 5-5 | M | |
| CFDP-2S-02 | End-of-file | 5.2.2, table 5-6 | M | |
| CFDP-2S-03 | Finished | 5.2.3, table 5-7<br>5.4, tables 5-17 and 5-18 | M | |
| CFDP-2S-04 | ACK | 5.2.4, table 5-8 | M | |
| CFDP-2S-05 | Metadata | 5.2.5, table 5-9<br>5.4, tables 5-15, 5-16, 5-19 | M | |
| CFDP-2S-06 | NAK | 5.2.6, tables 5-10 and 5-11 | M | |
| CFDP-2S-07 | Prompt | 5.2.7, table 5-12 | M | |
| CFDP-2S-08 | Keep-alive | 5.2.8, table 5-13 | M | |
| CFDP-2S-09 | File data | 5.3, table 5-14 | M | |

### A4.3.2 CFDP Procedures

| Item | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| CFDP-2S-10 | CRC procedures | 4.1 | M | |
| CFDP-2S-11 | Integrity protection | 4.2.1.1 | M | |
| CFDP-2S-30 | Checksum length | 4.2.1.2 | M | |
| CFDP-2S-31 | Applicable checksum algorithm | 4.2.2.1 | M | |
| CFDP-2S-32 | Available checks algorithms | 4.2.2.2 | M | |
| CFDP-2S-33 | Modular checksum algorithm implemented | 4.2.2.3 | M | |
| CFDP-2S-34 | Null checksum algorithm implemented | 4.2.2.4 | M | |
| CFDP-2S-35 | Other checksum algorithms implemented | 4.2.2.5 | O | |
| CFDP-2S-36 | Preferred checksum algorithm | 4.2.2.6 | M | |
| CFDP-2S-37 | Preferred checksum algorithm is available | 4.2.2.7 | M | |
| CFDP-2S-38 | Preferred checksum algorithm not available | 4.2.2.8 | M | |
| CFDP-2S-39 | insertion of checksum into EOF PDU | 4.2.3.1 | M | |
| CFDP-2S-40 | Preferred checksum algorithm at sending entity | 4.2.3.2 | M | |
| CFDP-2S-41 | Verification of checksum received in EOF PDU | 4.2.4.1 | M | |
| CFDP-2S-42 | Preferred checksum algorithm at receiving entity | 4.2.4.2 | M | |
| CFDP-2S-42 | Modular checksum algorithm definition | 4.2.5 | M | |
| CFDP-2S-12 | Put procedures | 4.3 | M | |
| CFDP-2S-13 | Transaction start notification procedure | 4.4 | M | |
| CFDP-2S-14 | PDU forwarding procedures | 4.5 | M | |
| CFDP-2S-15 | Copy file general procedures | 4.6 | M | |
| CFDP-2S-16 | Copy file procedures at the sending entity | 4.6.1.1 except 4.6.1.1.5.1 and 4.6.1.1.5.2 | M | |
| CFDP-2S-17 | Record continuation state and segment metadata procedures | 4.6.1.1.5.1, 4.6.1.1.5.2 | O | |
| CFDP-2S-18 | Acknowledged mode procedures at the sending entity | 4.6.4.2 | M | |
| CFDP-2S-19 | Incremental Lost Segment Detection procedures at the sending entity | 4.6.4.5 | O | |
| CFDP-2S-20 | Keep alive procedures at the sending entity | 4.6.5.3 | M | |
| CFDP-2S-21 | Cancel response procedures at the sending entity | 4.6.6.2 | M | |
| CFDP-2S-22 | Resume procedures | 4.6.7 | M | |
| CFDP-2S-23 | Report procedures | 4.6.8 | M | |
| CFDP-2S-24 | Positive acknowledgment procedures | 4.7 | M | |
| CFDP-2S-25 | Fault handling procedures | 4.8 | M | |
| CFDP-2S-26 | Filestore procedures | 4.9 | M | |
| CFDP-2S-27 | Inactivity monitor procedures | 4.10 | M | |
| CFDP-2S-28 | Internal procedures | 4.11 | M | |
| CFDP-2S-29 | Link state change procedures | 4.12 | M | |

If other checksum algorithms are implemented (CFDP-2S-35), the algorithms that are implemented shall be reported in the PICS.

### A4.3.3 Management Information Base Local Entity Configuration Parameters

| Item | Protocol Feature | Reference | Status | Parameter Value |
|---|---|---|---|---|
| LMIB-2S-01 | Local Entity ID | 8.2 | M | |
| LMIB-2S-02 | `EOF-Sent.indication` required | 8.2 | M | |
| LMIB-2S-03 | Default fault handlers | 8.2 | M | |

### A4.3.4 Management Information Base Remote Entity Configuration Parameters

| Item | Protocol Feature | Reference | Status | Parameter Value |
|---|---|---|---|---|
| RMIB-2S-01 | Remote entity ID | 8.3 | M | |
| | Protocol version number | 8.3 | M | |
| RMIB-2S-02 | UT address | 8.3 | M | |
| RMIB-2S-03 | Positive ACK timer interval | 8.3 | M | |
| RMIB-2S-04 | Default transmission mode | 8.3 | M | |
| RMIB-2S-05 | Type of checksum to calculate for all file transmission to this remote entity | 8.3 | M | |
| RMIB-2S-06 | CRCs required on transmission | 8.3 | M | |
| RMIB-2S-07 | Maximum file segment length | 8.3 | M | |
| RMIB-2S-08 | Positive ACK timer expiration limit | 8.3 | M | |
| RMIB-2S-09 | NAK timer expiration limit | 8.3 | M | |
| RMIB-2S-10 | Transaction inactivity limit | 8.3 | M | |
| RMIB-2S-11 | Start of transmission opportunity | 8.3 | M | |
| RMIB-2S-12 | End of transmission opportunity | 8.3 | M | |
| RMIB-2S-13 | Start of reception opportunity | 8.3 | M | |
| RMIB-2S-14 | End of reception opportunity | 8.3 | M | |

## A4.4 SERVICE CLASS 2 – RECEIVER

### A4.4.1 CFDP Protocol Data Units

| Type ID | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| CFDP-2R-01 | General | 5.1, tables 5-1, 5-2, 5-3<br>5.2.1, tables 5-4, 5-5 | M | |
| CFDP-2R-02 | End-of-file | 5.2.2, table 5-6 | M | |
| CFDP-2R-03 | Finished | 5.2.3, table 5-7<br>5.4, tables 5-17 and 5-18 | M | |
| CFDP-2R-04 | ACK | 5.2.4, table 5-8 | M | |
| CFDP-2R-05 | Metadata | 5.2.5, table 5-9<br>5.4, tables 5-15, 5-16, 5-19 | M | |
| CFDP-2R-06 | NAK | 5.2.6, tables 5-10 and 5-11 | M | |
| CFDP-2R-07 | Prompt | 5.2.7, table 5-12 | M | |
| CFDP-2R-08 | Keep-alive | 5.2.8, table 5-13 | M | |
| CFDP-2R-09 | File data | 5.3, table 5-14 | M | |

### A4.4.2 CFDP Procedures

| Item | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| CFDP-2R-10 | CRC procedures | 4.1 | M | |
| CFDP-2R-11 | Integrity protection | 4.2.1.1 | M | |
| CFDP-2R-29 | Checksum length | 4.2.1.2 | M | |
| CFDP-2R-30 | Applicable checksum algorithm | 4.2.2.1 | M | |
| CFDP-2R-31 | Available checks algorithms | 4.2.2.2 | M | |
| CFDP-2R-32 | Modular checksum algorithm implemented | 4.2.2.3 | M | |
| CFDP-2R-33 | Null checksum algorithm implemented | 4.2.2.4 | M | |
| CFDP-2R-34 | Other checksum algorithms implemented | 4.2.2.5 | O | |
| CFDP-2R-35 | Preferred checksum algorithm | 4.2.2.6 | M | |
| CFDP-2R-36 | Preferred checksum algorithm is available | 4.2.2.7 | M | |
| CFDP-2R-37 | Preferred checksum algorithm not available | 4.2.2.8 | M | |
| CFDP-2R-38 | insertion of checksum into EOF PDU | 4.2.3.1 | M | |
| CFDP-2R-39 | Preferred checksum algorithm at sending entity | 4.2.3.2 | M | |
| CFDP-2R-40 | Verification of checksum received in EOF PDU | 4.2.4.1 | M | |
| CFDP-2R-41 | Preferred checksum algorithm at receiving entity | 4.2.4.2 | M | |
| CFDP-2R-42 | Modular checksum algorithm definition | 4.2.5 | M | |
| CFDP-2R-12 | PDU forwarding procedures | 4.5 | M | |
| CFDP-2R-13 | Copy file general procedures | 4.6 | M | |
| CFDP-2R-14 | Copy file procedures at the receiving entity | 4.6.1.2 | M | |
| CFDP-2R-15 | Optional copy file procedures at the receiving entity | 4.6.2 | O | |
| CFDP-2R-16 | Acknowledged mode procedures at the receiving entity | 4.6.4.3 | M | |
| CFDP-2R-17 | Incremental Lost Segment Detection procedures at the receiving entity | 4.6.4.4 | M | |
| CFDP-2R-18 | Keep alive initiation at the receiving entity | 4.6.5.2.1 | O | |
| CFDP-2R-19 | Keep alive procedures at the receiving entity | 4.6.5.2.2, 4.6.5.2.3 | M | |
| CFDP-2R-20 | Cancel response procedures at the receiving entity | 4.6.6.1 | M | |
| CFDP-2R-21 | Resume procedures | 4.6.7 | M | |
| CFDP-2R-22 | Report procedures | 4.6.8 | M | |
| CFDP-2R-23 | Positive acknowledgment procedures | 4.7 | M | |
| CFDP-2R-24 | Fault handling procedures | 4.8 | M | |
| CFDP-2R-25 | Filestore procedures | 4.9 | M | |
| CFDP-2R-26 | Inactivity monitor procedures | 4.10 | M | |
| CFDP-2R-27 | Internal procedures | 4.11 | M | |
| CFDP-2R-28 | Link state change procedures | 4.12 | M | |

If other checksum algorithms are implemented (CFDP-2R-34), the algorithms that are implemented shall be reported in the PICS.

### A4.4.3 Management Information Base Local Entity Configuration Parameters

| Item | Protocol Feature | Reference | Status | Parameter Value |
|---|---|---|---|---|
| LMIB-2R-01 | Local Entity ID | 8.2 | M | |
| LMIB-2R-02 | `EOF-Recv.indication` required | 8.2 | M | |
| LMIB-2R-03 | `File-Segment-Recv.indication` required | 8.2 | M | |
| LMIB-2R-04 | `Transaction-Finished.indication` required when acting as receiving entity | 8.2 | M | |
| LMIB-2R-05 | `Suspended.indication` required when acting as receiving entity | 8.2 | M | |
| LMIB-2R-06 | `Resumed.indication` required when acting as receiving entity | 8.2 | M | |
| LMIB-2R-07 | Default fault handlers | 8.2 | M | |

### A4.4.4 Management Information Base Remote Entity Configuration Parameters

| Item | Protocol Feature | Reference | Status | Parameter Value |
|---|---|---|---|---|
| RMIB-2R-01 | Remote entity ID | 8.3 | M | |
| RMIB-2R-02 | Protocol version number | 8.3 | M | |
| RMIB-2R-03 | UT address | 8.3 | M | |
| RMIB-2R-04 | Positive ACK timer interval | 8.3 | M | |
| RMIB-2R-05 | NAK timer interval | 8.3 | M | |
| RMIB-2R-06 | Keep Alive interval | 8.3 | M | |
| RMIB-2R-07 | Immediate NAK mode enabled | 8.3 | M | |
| RMIB-2R-09 | CRCs required on transmission | 8.3 | M | |
| RMIB-2R-10 | Positive ACK timer expiration limit | 8.3 | M | |
| RMIB-2R-11 | NAK timer expiration limit | 8.3 | M | |
| RMIB-2R-12 | Transaction inactivity limit | 8.3 | M | |
| RMIB-2R-13 | Start of transmission opportunity | 8.3 | M | |
| RMIB-2R-14 | End of transmission opportunity | 8.3 | M | |
| RMIB-2R-15 | Start of reception opportunity | 8.3 | M | |
| RMIB-2R-16 | End of reception opportunity | 8.3 | M | |

## A4.5 STORE AND FORWARD OVERLAY – SENDER

### A4.5.1 SFO Messages

| Type ID | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| SFOS-01 | General | B2.1, table B-1 | M | |
| SFOS-02 | Request message | B2.4.2, table B-2 | M | |
| SFOS-03 | Message to User message | B2.4.3, table B-3 | M | |
| SFOS-04 | Filestore request message | B2.4.4, table B-4 | M | |
| SFOS-05 | Fault Handler override message | B2.4.5, table B-5 | M | |
| SFOS-06 | Flow label message | B2.4.6, table B-6 | M | |
| SFOS-07 | Report message | B2.4.6, table B-7 | M | |
| SFOS-08 | Filestore response message | B2.4.6, table B-8 | M | |

### A4.5.2 SFO Procedures

| Item | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| SFOS-09 | Routing procedures | B2.3 | M | |
| SFOS-10 | SFO transmission initiation procedures | B2.4 | M | |

## A4.6 STORE AND FORWARD OVERLAY – RELAY

### A4.6.1 SFO Messages

| Type ID | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| SFOX-01 | General | B2.1, table B-1 | M | |
| SFOX-02 | Request message | B2.4.2, table B-2 | M | |
| SFOX-03 | Message to User message | B2.4.3, table B-3 | M | |
| SFOX-04 | Filestore request message | B2.4.4, table B-4 | M | |
| SFOX-05 | Fault Handler override message | B2.4.5, table B-5 | M | |
| SFOX-06 | Flow label message | B2.4.6, table B-6 | M | |
| SFOX-07 | Report message | B2.4.6, table B-7 | M | |
| SFOX-08 | Filestore response message | B2.4.6, table B-8 | M | |

### A4.6.2 SFO Procedures

| Item | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| SFOX-09 | Routing procedures | B2.3 | M | |
| SFOX-10 | Relaying an SFO transmission file data unit | B2.5 | M | |
| SFOX-11 | Relaying an SFO transmission report | B2.7 | M | |

## A4.7 STORE AND FORWARD OVERLAY – RECEIVER

### A4.7.1 SFO Messages

| Type ID | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| SFOR-01 | General | B2.1, table B-1 | M | |
| SFOR-02 | Request message | B2.4.2, table B-2 | M | |
| SFOR-03 | Message to User message | B2.4.3, table B-3 | M | |
| SFOR-04 | Filestore request message | B2.4.4, table B-4 | M | |
| SFOR-05 | Fault Handler override message | B2.4.5, table B-5 | M | |
| SFOR-06 | Flow label message | B2.4.6, table B-6 | M | |
| SFOR-07 | Report message | B2.4.6, table B-7 | M | |
| SFOR-08 | Filestore response message | B2.4.6, table B-8 | M | |

### A4.7.2 SFO Procedures

| Item | Protocol Feature | Reference | Status | Support |
|---|---|---|---|---|
| SFOR-09 | Routing procedures | B2.3 | M | |
| SFOR-10 | Reporting on the results of an SFO file transmission | B2.6 | M | |

# ANNEX B

# STORE AND FORWARD OVERLAY OPERATIONS

# (NORMATIVE)

## B1 OVERVIEW

The CFDP Store and Forward Overlay (SFO) system is a mechanism for transmitting files between users of CFDP entities that may never be in direct communication.

Each transmitted file is received, stored, and forwarded in a hop-by-hop manner by intermediate *waypoint users* until it finally reaches a user, termed the *agent*, whose CFDP entity can directly communicate with that of the *destination* user. The file to be transmitted and all associated metadata are encapsulated in an *SFO transmission file delivery unit* for transmission to each waypoint. The SFO transmission file delivery unit metadata additionally contains an 'SFO Request' Message to User that identifies and characterizes the transmission.

The agent user transmits the content of the received SFO transmission file delivery unit, the original file together with the original associated metadata as extracted from the SFO transaction's own metadata, to the destination user in a conventional single-hop CFDP transaction termed an *SFO final delivery transaction*. This is similar to the operation of the 'respondent' in a Proxy file transfer, except that the SFO Request message is delivered to the destination user in addition to all the original metadata. The agent is then responsible for transmitting an *SFO transmission report file delivery unit* (again, via waypoint users) back to the user which was the original *source* of the file.

Errors in routing or relaying and (optionally) successful relay operations are likewise reported back to the original source user in SFO transmission report file delivery units. Relaying failures and (optionally) successes are additionally reported to the final destination user when possible.

In the illustration of SFO operations in figure B-1, the thick right arrows represent SFO transmission file delivery units, while the double right arrow represents the file delivery unit of an SFO final delivery transaction. The thin arrows represent SFO transmission report file delivery units, generated either by the agent waypoint to report on final file delivery or by any waypoint to report on routing or relay errors, or, optionally, relay success (i.e., 'trace' progress). Reports are sent toward both the original source entity and the final destination entity, and the reports themselves are relayed as necessary.

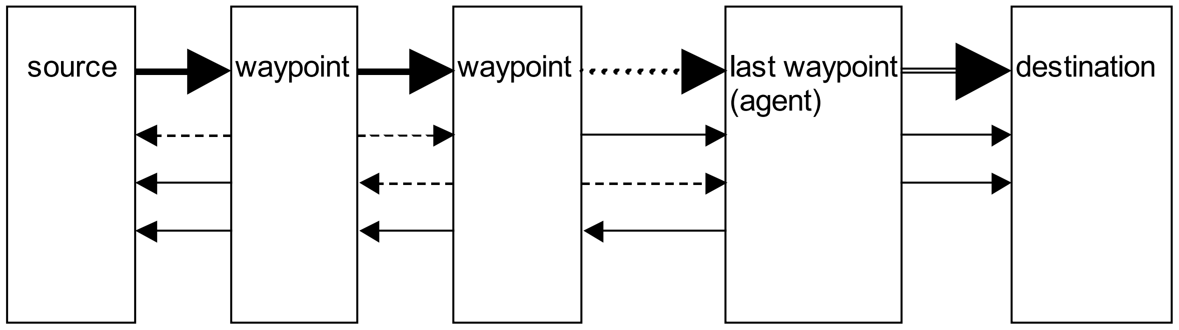


**Figure B-1: SFO Operations**

## B2 REQUIREMENTS

### B2.1 GENERAL

To enable interoperability, mandatory behavior specified in subsections B2.2 through B2.7 shall be observed by CFDP user applications that are in compliance with the Store and Forward Overlay specification.

### B2.2 SFO MESSAGE TYPES

The message type field for each Reserved CFDP Message used in SFO operations shall contain one of the values specified in table B-1.

**Table B-1: SFO Message Types**

| Message Type (hexadecimal) | Interpretation |
|---|---|
| 40 | SFO Request |
| 41 | SFO Message to User |
| 42 | SFO Flow Label |
| 43 | SFO Fault Handler Override |
| 44 | SFO Filestore Request |
| 45 | SFO Report |
| 46 | SFO Filestore Response |

## B2.3 ROUTING

The SFO Routing procedure shall be as follows:

a) The user application shall have access to an information base that associates with each known CFDP entity and the IDs of all other entities from which that entity can receive files directly by means of CFDP.

b) The user application shall compute from this information base the shortest possible *route* to the final destination user. A route shall be a list of CFDP entities, beginning with the user application's own local entity and ending with the local entity of the final destination user, such that every entity in the list is able to receive files directly via CFDP from the preceding entity in the list.

## B2.4 INITIATING AN SFO TRANSMISSION

### B2.4.1 General

**B2.4.1.1** In order to initiate SFO transmission of a file and associated metadata, the CFDP user shall invoke the SFO Routing procedure (see B2.3) to determine a route from its local CFDP entity to that of the final destination user.

**B2.4.1.2** If the computed route is an empty list, then transmission is impossible. The attempt to initiate SFO transmission shall therefore immediately fail.

**B2.4.1.3** If the computed route contains only the local CFDP entity, then the user application is the final destination of the file. The attempt to initiate SFO transmission shall therefore immediately fail.

**B2.4.1.4** If the computed route contains only the local CFDP entity and that of the final destination user, then the user application shall simply use the CFDP `Put.request` primitive to request direct delivery of the file and associated metadata to the final destination. No further SFO functionality shall apply.

**B2.4.1.5** If the computed route contains one or more other CFDP entities (waypoints) in addition to the local CFDP entity and the final destination user's CFDP entity, then the user application shall use the CFDP `Put.request` primitive to request delivery of an SFO transmission file delivery unit to the first waypoint in the route. The file transmitted in the SFO transmission FDU shall be the file that is to be delivered to the final destination user, if any. The metadata of the SFO transmission FDU shall include a single SFO Request message, defined in B2.4.2, together with zero or more of the Reserved CFDP Messages defined in B2.4.3 through B2.4.6; the contents of these messages shall be obtained from the metadata associated with the file. The destination file name specified in the `Put.request` primitive shall be formed as follows:

*SFOsourceEntityID_SFOrequestLabel*.sfo

where *SFOsourceEntityID* is the ASCII character representation of the source entity ID noted in the SFO Request message, with all leading zeroes omitted, and *SFOrequestLabel* is as noted in the SFO Request message.

NOTES

1 Path name is omitted from this file name to avoid incompatibility with path naming rules imposed by the filestores at waypoints.

2 A CFDP user application that implements SFO should reserve space in the local filestore for temporary storage of files that are to be forwarded rather than delivered locally. The action taken with regard to any file received for SFO forwarding that, if retained, would cause this space allocation to be exceeded is implementation-specific.

The transmission mode, segmentation control, fault handler overrides, and flow label that constrain the CFDP transaction resulting from this `Put.request` shall be selected at the discretion of the requesting user application.

NOTE – At any time after the transaction finishes, the user application may at its option delete its local copy of the transmitted file.

### B2.4.2 SFO Request

The SFO Request message is mandatory and shall be constructed as indicated in table B-2.

**Table B-2: SFO Request Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Trace control flag | 2 | 0 (no trace requested), 1 (trace toward source only), 2 (trace toward destination only), 3 (trace in both directions). | Controls the transmission of waypoints' reports of relay transaction success. |
| Transmission mode | 1 | As defined in the CFDP Recommendation. | Specifies the transmission mode for transmission from the agent to the final destination. |
| Segmentation control | 1 | As defined in the CFDP Recommendation. | Specifies file segmentation control for transmission from the agent to the final destination. |
| Closure request | 1 | As defined in the CFDP Recommendation. | Controls closure notices at waypoints. |
| Spare | 3 | | |
| Prior waypoints count | 8 | Set to zero by original source user, incremented by every waypoint. | Indicates to the receiver of the request the number of times the request has been forwarded, so far, by waypoint entities. |
| SFO request label | Variable | LV | A transaction identifier constructed in implementation-specific manner by the original source user. May be used for transaction accounting purposes by the original source user. An array of printable ASCII text characters. |
| Source entity ID | Variable | LV | The ID of the original source user's CFDP entity. A binary integer. |
| Destination entity ID | Variable | LV | The ID of the final destination user's CFDP entity. A binary integer. |
| Source file name | Variable | LV | Length is zero if parameter is omitted. |
| Destination file name | Variable | LV | Length is zero if parameter is omitted. |

### B2.4.3 SFO Message to User

One or more SFO Message to User messages may be optionally included and shall be constructed as indicated in table B-3.

**Table B-3: SFO Message to User Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Message text | Variable | LV | Encapsulates the text of a message to user that is to be delivered to the final destination user. |

### B2.4.4 SFO Filestore Request

One or more SFO Filestore Request messages may be optionally included and shall be constructed as indicated in table B-4.

**Table B-4: SFO Filestore Request Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Length (octets) | 8 | Number of octets in the request field. | |
| Request | 8 × length | The value of this field is a single CFDP filestore request as defined in the CFDP Recommendation. | Encapsulates a filestore request that is to be executed by the local CFDP entity of the final destination user. |

### B2.4.5 SFO Fault Handler Override

One or more SFO Fault Handler Override messages may optionally be included and shall be constructed as indicated in table B-5.

**Table B-5: SFO Fault Handler Override Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Fault handler override | 8 | As defined in the CFDP Recommendation. | Applies to transmission from the agent to the final destination. |

#### B2.4.6 SFO Flow Label

The SFO Flow Label message is optional and shall be constructed as indicated in table B-6.

**Table B-6: SFO Flow Label Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Flow label | Variable | LV, as defined in the CFDP Recommendation. | Applies to transmission from the agent to the final destination. |

### B2.5 RELAYING AN SFO TRANSMISSION FILE DATA UNIT

**B2.5.1** Upon receipt of a **`Transaction-Finished.indication`** indicating successful reception of a file data unit whose metadata included an SFO Request message, the CFDP user shall invoke the SFO Routing procedure (see B2.3) to determine a route from its local CFDP entity to that of the final destination user.

**B2.5.2** If the computed route is an empty list, then relaying is impossible. The CFDP user shall use the SFO Reporting procedure (see B2.6) to notify the original source user application of the routing failure.

**B2.5.3** If the computed route contains only the local CFDP entity, then the user application is the final destination of the file. No further SFO functionality shall apply.

**B2.5.4** If the computed route contains one or more waypoints in addition to the local CFDP entity and the final destination user's CFDP entity, then a relay transaction is in order.

**B2.5.5** If the Prior Waypoints Count in the SFO Request message is equal to the maximum possible value for this field (that is, 255), then relaying is disallowed. The CFDP user shall use the SFO Reporting procedure (see B2.6) to notify the original source user application that the maximum number of waypoints was exceeded for this request.

**B2.5.6** If the destination file name specified in the file data unit's metadata does not conform to the convention defined in B2.4.1.5 above, then relaying is disallowed. The CFDP user shall use the SFO Reporting procedure (see B2.6) to notify the original source user application that the intermediate destination file name was invalid.

**B2.5.7** Otherwise, the user application shall use the CFDP **`Put.request`** primitive to request delivery of the received file (if any) and all received SFO Request, SFO Flow Label, SFO Fault Handler Override, SFO Message to User, and SFO Filestore Request messages to the first waypoint in the route. The destination file name specified in the **`Put.request`** primitive shall be the destination file name specified in the received file data unit's metadata. The transmission mode, segmentation control, fault handler overrides, and flow label that constrain the resulting transaction shall be selected at the discretion of the requesting user application.

**B2.5.8** If the relay transaction finishes in any condition other than 'No error', then the CFDP user shall use the SFO Reporting procedure (see B2.6) to notify the original source and final destination user applications of the relay transaction's final condition.

NOTE – This notification is unconditional, regardless of the value of the trace control flag field in the received SFO Request message.

**B2.5.9** If the relay transaction finishes in 'No error' condition, provided the Trace Control flag in the SFO Request Message is non-zero, the user application shall use the SFO Reporting procedure (see B2.6) to report to the original source and/or final destination user applications (as indicated by the value of the Trace Control flag) on the success of the relay transaction.

NOTE – At any time after the relay transaction finishes, the user application may at its option delete its local copy of the transmitted file.

**B2.5.10** If the computed route contains only the local CFDP entity and that of the final destination user, then the user application, now serving as the agent of the SFO transmission operation, shall initiate an *SFO final delivery transaction*. To do so, it shall use the CFDP **`Put.request`** primitive to request delivery of a file delivery unit with the following parameters:

– destination entity ID as specified in the SFO Request message;

– source file name as specified in the SFO Request message (if any);

– destination file name as specified in the SFO Request message (if any);

– segmentation control as specified in the SFO Request message;

– transmission mode as specified in the SFO Request message;

– flow label as specified in the SFO Flow Label message that was received in the same transaction's **`Metadata-recv.indication`** (if any);

– one fault handler override for each SFO Fault Handler Override message that was received in the same transaction's **`Metadata-recv.indication`**;

– one Message to User for each SFO Message to User message that was received in the same transaction's **`Metadata-recv.indication`**;

– one filestore request for each SFO Filestore Request message that was received in the same transaction's **`Metadata-recv.indication`**;

– the entire SFO Request message, as an additional Message to User. It should be noted that this Message to User is, again, a Reserved CFDP Message as defined in B2.4.2.

When the resulting transaction finishes, the agent user application shall use the SFO Reporting procedure (see B2.6) to report to the original source user application on the success or failure of the final file delivery transaction.

NOTE – At any time after the transaction finishes, the user application may at its option delete its local copy of the transmitted file.

## B2.6 SFO REPORTING

### B2.6.1 General

**B2.6.1.1** When required to report on the results of an SFO file transmission, the user application shall invoke the SFO Routing procedure (see B2.3) to determine routes from its local CFDP entity to those of the original source user and/or final destination user as applicable.

**B2.6.1.2** If the computed route to the source is an empty list, then reporting to the source is impossible. No further SFO functionality shall apply.

**B2.6.1.3** If the computed route to the source contains only the local CFDP entity, then the user application is the original source of the file. Therefore reporting to the source is unnecessary, so no further SFO functionality shall apply.

**B2.6.1.4** If the computed route to the source contains more than one entity, then the user application shall use the CFDP `Put.request` primitive to request delivery of an *SFO transmission report* file delivery unit with the following parameters:

– destination entity ID set to the ID of the second entity in the computed route (the first one following the local CFDP entity);

– no file, source file name, or destination file name;

– a single SFO Report message, defined in B2.6.2;

– zero or more SFO Filestore Response messages, defined in B2.6.3, as applicable.

When the resulting transaction finishes, or if for any reason the `Put.request` cannot be honored, the user application shall take no further action with regard to this SFO transmission.

**B2.6.1.5** If the computed route to the destination is an empty list, then reporting to the destination is impossible. No further SFO functionality shall apply.

**B2.6.1.6** If the computed route to the destination contains only the local CFDP entity, then the user application is the final destination of the file. Therefore reporting to the destination is unnecessary, so no further SFO functionality shall apply.

**B2.6.1.7** If the computed route to the destination contains more than one entity, then the user application shall use the CFDP `Put.request` primitive to request delivery of an *SFO transmission report* file delivery unit with the following parameters:

– destination entity ID set to the ID of the second entity in the computed route (the first one following the local CFDP entity);

– no file, source file name, or destination file name;

– a single SFO Report message, defined in B2.6.2;

– zero or more SFO Filestore Response messages, defined in B2.6.3, as applicable.

When the resulting transaction finishes, or if for any reason the **`Put.request`** cannot be honored, the user application shall take no further action with regard to this SFO transmission.

### B2.6.2 SFO Report

The SFO Report message is mandatory in every SFO transmission report and shall be constructed as indicated in table B-7. If the SFO transmission report is the result of completion of an SFO final delivery transaction, then the message's condition code, delivery code, and file status shall be obtained from the **`Transaction-Finished.indication`** that signaled that completion; otherwise, the values of all three fields shall be undefined.

**Table B-7: SFO Report Message**

| Field | Size (bits) | Values | Comment |
|---|---|---|---|
| SFO request label | Variable | LV | Copied from the request label in the SFO Request message for this SFO transmission. |
| Source entity ID | Variable | LV | The ID of the original source user's CFDP entity for this SFO transmission. A binary integer. |
| Destination entity ID | Variable | LV | The ID of the final destination user's CFDP entity for this SFO transmission. A binary integer. |
| Reporting entity ID | Variable | LV | The ID of the CFDP entity of the user which originally issued this SFO transmission report. A binary integer. |
| Prior waypoints count | 8 | Set to zero by the reporting user application, incremented by every waypoint. | Indicates to the receiver of the report the number of times the report has been forwarded, so far, by waypoint entities. |
| Prior waypoints count of SFO request | 8 | Copied from the prior waypoints count in the SFO Request message for this SFO transmission. | |

| Field | Size (bits) | Values | Comment |
|---|---|---|---|
| Report code | 8 | 1, 2, 3, 4, 5, 6, or 7 | 1: final file delivery transaction completed.<br>2: relay transaction failed.<br>3: routing failed; relay was not possible.<br>4: relay transaction succeeded.<br>5: final file delivery transaction failed.<br>6: maximum number of waypoints exceeded.<br>7: invalid intermediate destination file name; relay was not possible. |
| Condition code | 4 | (See the CFDP Recommendation.) | |
| Direction | 1 | 0: report is destined for the original source user application.<br>1: report is destined for the final destination user application. | |
| Delivery Code | 1 | (See the CFDP Recommendation.) | |
| File Status | 2 | (See the CFDP Recommendation.) | |

### B2.6.3 SFO Filestore Response

If the SFO transmission report is the result of completion of an SFO final delivery transaction, and if the **`Transaction-Finished.indication`** signaling this transaction completion included one or more filestore responses, then for each such response, one SFO Filestore Response message shall be included. Each such message shall be constructed as indicated in table B-8.

**Table B-8: Proxy Filestore Response Message**

| Field | Size (bits) | Values | Comments |
|---|---|---|---|
| Length | 8 | Number of octets in the response field. | |
| Response | 8 * length | The value of this field is a single CFDP filestore response as defined in the CFDP Recommendation. | Encapsulates a filestore request that was produced by the local CFDP entity of the final destination user. |

## B2.7 RELAYING AN SFO TRANSMISSION REPORT

**B2.7.1** Upon receipt of a `Transaction-Finished.indication` indicating successful reception of a file data unit whose metadata included an SFO Report message, the CFDP user shall invoke the SFO Routing procedure (see B2.3) to determine a route from its local CFDP entity to that of the user application to which the report is directed (as indicated by the Direction noted in the SFO Report message).

**B2.7.2** If the computed route is an empty list, then relaying is impossible. The user application shall take no further action with regard to this SFO transmission.

**B2.7.3** If the computed route contains only the local CFDP entity, then the user application is the original source of the file. No further SFO functionality shall apply.

**B2.7.4** If the computed route contains more than one entity, then a relay transaction is in order.

**B2.7.5** If the Prior Waypoints Count in the SFO Report Message is equal to the maximum possible value for this field (that is, 255), then relaying is disallowed. No further SFO functionality shall apply.

**B2.7.6** Otherwise, the user application shall use the CFDP `Put.request` primitive to request delivery of the received SFO Report and SFO Filestore Response message(s) to the second entity in the computed route (the first one following the local CFDP entity). When the resulting transaction finishes, or if for any reason the `Put.request` cannot be honored, the user application shall take no further action with regard to this SFO transmission.

# ANNEX C

# SECURITY, SANA, AND PATENT CONSIDERATIONS

# (INFORMATIVE)

## C1 SECURITY

CFDP was designed prior to the emergence of awareness within CCSDS that communication protocols need to be secured against attack or compromise. Consequently, CFDP includes no security mechanisms of any kind.

CFDP is of course subject to the same confidentiality, data integrity, authentication, and availability concerns as any other protocol. Given the absence of security mechanisms built into CFDP itself, the UT-layer protocol stack over which CFDP functions MUST provide whatever services are required in order to ensure the secure operation of the protocol as deployed. The details of these services are necessarily opaque to CFDP and are beyond the scope of this Recommendation.

## C2 SANA CONSIDERATIONS

The recommendations of this document have created the following SANA registry:

CFDP Checksum Identifiers

Registry Description: The registry named 'CFDP Checksum Identifiers' lists code numbers that identify CFDP checksum calculation algorithms as discussed in 4.1.2 above. The first 16 entries in this registry, checksum identifiers 0 through 15, are reserved for the identification of checksum algorithms utilized by CFDP. Only checksum calculation algorithms that produce 32-bit checksum values shall be included in this registry. Checksum identifier 0 is explicitly reserved for the Modular Checksum calculation defined in 4.1.2.3.

NOTE – At the time of publication of this document, not all of the 16 checksum identifiers reserved for CFDP have yet been defined.

This registry consists of a table of parameters:

ChecksumID: An unsigned integer that uniquely identifies a checksum calculation algorithm.

Name: The human-readable name of the checksum algorithm (CRC-32, etc.).

References: One or more references to documents that define this checksum calculation algorithm.

The initial registry was filled with the following values:

| ChecksumID | Name | References |
|---|---|---|
| 0 (0x00) | Modular Checksum | (*) |
| 1 (0x01) | Proximity-1 CRC32 | (**) |
| 2 (0x02) | CRC-32c | (***) |
| 3 (0x03) | IEEE Std 802.3-2018 FCS | (****) |
| 4 (0x04) | Reserved for CFDP | (*) |
| 5 (0x05) | Reserved for CFDP | (*) |
| 6 (0x06) | Reserved for CFDP | (*) |
| 7 (0x07) | Reserved for CFDP | (*) |
| 8 (0x08) | Reserved for CFDP | (*) |
| 9 (0x09) | Reserved for CFDP | (*) |
| 10 (0x0A) | Reserved for CFDP | (*) |
| 11 (0x0B) | Reserved for CFDP | (*) |
| 12 (0x0C) | Reserved for CFDP | (*) |
| 13 (0x0D) | Reserved for CFDP | (*) |
| 14 (0x0E) | Reserved for CFDP | (*) |
| 15 (0x0F) | NULL | (*****) |

(*) CCSDS 727.0-B-5, *CCSDS File Delivery Protocol (CFDP)*, 4.2 (this document).

(**) CCSDS 211.2-B-3, *Proximity-1 Space Link Protocol—Coding and Synchronization Sublayer*, annex C CRC-32 Coding Procedures (reference [D6]).

(***) RFC 4960, Stream Control Transmission Protocol, Appendix B CRC32c Checksum Calculation (reference [D7]).

(****) IEEE Std 802.3-2018, IEEE Standard for Ethernet, section 3.2.9 Frame Check Sequence (FCS) field (reference [D8]).

(*****) A null checksum. If used, the file checksum value shall be set to 0x00 as specified in 4.2.2.4.

The registration rule for new values of this registry is that engineering review by the SIS area is required, including verification and documentation of correct operation on a representative set of files. The request must come from the Agency Representative of a CCSDS space agency, or other registered organization, that is a member of the CCSDS.

This registry is located at https://sanaregistry.org/r/checksum_identifiers.

## C3 PATENTS

The CCSDS File Delivery Protocol and its normative references are not protected by any known patents.

# ANNEX D

# INFORMATIVE REFERENCES

# (INFORMATIVE)

[D1] *Organization and Processes for the Consultative Committee for Space Data Systems*. Issue 4. CCSDS Record (Yellow Book), CCSDS A02.1-Y-4. Washington, D.C.: CCSDS, April 2014.

[D2] *TC Synchronization and Channel Coding—Summary of Concept and Rationale*. Issue 2. Report Concerning Space Data System Standards (Green Book), CCSDS 230.1-G-2. Washington, D.C.: CCSDS, November 2012.

[D3] *CCSDS File Delivery Protocol (CFDP)—Part 1: Introduction and Overview*. Issue 3. Report Concerning Space Data System Standards (Green Book), CCSDS 720.1-G-3. Washington, D.C.: CCSDS, April 2007.

[D4] *CCSDS File Delivery Protocol (CFDP)—Part 2: Implementers Guide*. Issue 4. Report Concerning Space Data System Standards (Green Book), CCSDS 720.2-G-4. Forthcoming.

[D5] *Specification and Description Language—Overview of SDL-2010*. ITU-T Recommendation Z.100. Geneva: ITU, December 2011.

[D6] *Proximity-1 Space Link Protocol—Coding and Synchronization Sublayer*. Issue 3. Recommendation for Space Data System Standards (Blue Book), CCSDS 211.2-B-3. Washington, D.C.: CCSDS, October 2019.

[D7] R. Stewart, ed. *Stream Control Transmission Protocol*. RFC 4960. Reston, Virginia: ISOC, September 2007.

[D8] *IEEE Standard for Ethernet*. Revision of IEEE Std 802.3-2012. IEEE Std 802.3-2018. New York: IEEE, 2012.

# ANNEX E

# ABBREVIATIONS AND ACRONYMS

# (INFORMATIVE)

| **Term** | **Meaning** |
|---|---|
| ACK | positive acknowledgement |
| AOS | Advance Orbiting Systems |
| APID | application process identifier |
| CCSDS | Consultative Committee for Space Data Systems |
| CFDP | CCSDS File Delivery Protocol |
| EOF | end of file |
| FDU | file delivery unit |
| FSS | file size sensitive |
| LV | length-value |
| MIB | management information base |
| MSB | most significant bit |
| NAK | negative acknowledgement |
| NCC | network control center |
| OSI | Open Systems Interconnection |
| PDU | protocol data unit |
| SAP | service access point |
| SDL | specification and description language |
| SDU | service data unit |
| TC | telecommand |
| TCP | Transmission Control Protocol |
| TLV | type-length-value |
| TM | telemetry |
| UDP | User Datagram Protocol |
| UT | unitdata transfer |

# ANNEX F

# EXAMPLE OF MODULAR CHECKSUM CALCULATION

# (INFORMATIVE)

The following is an example of file modular checksum calculation, provided to illustrate the discussion in 4.2. In order to facilitate the discussion in this annex, the notes from 4.2 are repeated here:

NOTE 1 – In order to include in a modular checksum the content of a File Data PDU whose offset is not an integral multiple of 4, it is necessary to align the data properly before adding 4-octet blocks of it to the checksum. Data at offset *Q* may be aligned by inserting *N* octets of value 'zero' before the first octet of the data, where $N = Q$ mod 4 (the remainder obtained upon dividing *Q* by 4).

NOTE 2 – In order to include in a modular checksum a sequence of *M* octets (the first of which is at a file offset that is an integral multiple of 4) where *M* is less than 4, it is necessary to pad the data to length 4 before adding it to the checksum. The data may be padded by inserting $(4 – M)$ octets of value 'zero' after the last octet of the data. This condition can apply only at the end of the file.

For a file that is 15 bytes long:

00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e

If, for illustrative purposes, the maximum data segment size is 6, then the file will be sent in three File Data PDUs as follows:

| Offset | Length | Content |
|---|---|---|
| 0 | 6 | 00 01 02 03 04 05 |
| 6 | 6 | 06 07 08 09 0a 0b |
| 12 | 3 | 0c 0d 0e |

To compute the checksum, the incoming file data is divided into 4-octet words and summed. Division always takes place at offsets that are integral multiples of 4. So the first word is constructed by first copying the byte at offset 0 (which is an integral multiple of 4) into the high-order octet of the word:

00 __ __ __

Then the NEXT (not previous) three octets of file data are copied into the next three octets of the word:

00 01 02 03

To construct the second word, the byte at offset 4 is copied into the high-order octet of the word:

04 __ __ __

The next three octets of file data are then copied into the next three octets of the word:

04 05 06 07

To construct the third word, the byte at offset 8 is copied into the high-order octet of the word:

08 __ __ __

Then the next three octets of file data are copied into the next three octets of the word:

08 09 0a 0b

To construct the last word, where the remaining sequence of file data octets has length 3, the byte at offset 12 is copied into the high-order octet of the word:

0c __ __ __

Then the remaining octets of file data are copied into the next two octets of the word:

0c 0d 0e __

And finally (per NOTE 2), $(4-3)=1$ octets of value 'zero' are inserted after the last octet of the data:

0c 0d 0e 00

So the checksum is computed by adding these four 32-bit words:

```
  00010203
+ 04050607
+ 08090a0b
+ 0c0d0e00
```

Carrying is done in the usual way, except that anything that would be carried out of the high-order octet is simply discarded.

Below is an example of computing the checksum incrementally as each file data segment arrives, when the segments arrive out of order as follows:

| Offset | Length | Content |
|---|---|---|
| 0 | 6 | 00 01 02 03 04 05 |
| 12 | 3 | 0c 0d 0e |
| 6 | 6 | 06 07 08 09 0a 0b |

From the first file data segment received, the following words are constructed:

00 01 02 03

04 05 00 00 (another application of NOTE 2)

From the second file data segment received, the following words are constructed:

0c 0d 0e 00 (again, see NOTE 2)

From the third file data segment received, words are constructed in a slightly different way, invoking the algorithm in NOTE 1. The offset of the File Data PDU, 6, is not an integral multiple of 4, so the data has to be aligned before dividing it into words. To align data at offset *Q*, *N* preceding octets of value 'zero' are inserted before the first octet of the data, where *N* = *Q* modulo 4. Since 6 modulo 4 is 2, two octets of value 'zero' have to be inserted before the first octet of file data, causing the following change:

06 07 08 09 0a 0b

to

00 00 06 07 08 09 0a 0b

Now the following words from this realigned file data are constructed:

00 00 06 07

08 09 0a 0b

So the checksum is computed by adding these words together:

  00010203 (same as first word in previous case)

+ 04050000

+ 0c0d0e00 (same as fourth word in previous case)

+ 00000607

+ 08090a0b (same as third word in previous case)

Since 04050000 + 00000607 = 04050607, which was the second word in the previous case, it is clear that the order of arrival of the file data segment is immaterial to the incremental computation of the checksum.