The Consultative Committee for Space Data Systems

Recommendation for Space Data System Practices

SPACECRAFT ONBOARD INTERFACE SERVICES—FILE AND PACKET STORE SERVICES

RECOMMENDED PRACTICE

CCSDS 873.0-M-1

MAGENTA BOOK

September 2012

The Consultative Committee for Space Data Systems

Recommendation for Space Data System Practices

SPACECRAFT ONBOARD INTERFACE SERVICES—FILE AND PACKET STORE SERVICES

RECOMMENDED PRACTICE

CCSDS 873.0-M-1

MAGENTA BOOK

September 2012

AUTHORITY

Issue:	Recommended Practice, Issue 1
Date:	September 2012
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-3), and the record of Agency participation in the authorization of this document can be obtained from the CCSDS Secretariat at the address below.

This document is published and maintained by:

CCSDS Secretariat
Space Communications and Navigation Office, 7L70
Space Operations Mission Directorate
NASA Headquarters
Washington, DC 20546-0001, USA

STATEMENT OF INTENT

The Consultative Committee for Space Data Systems (CCSDS) is an organization officially established by the management of its members. The Committee meets periodically to address data systems problems that are common to all participants, and to formulate sound technical solutions to these problems. Inasmuch as participation in the CCSDS is completely voluntary, the results of Committee actions are termed **Recommendations** and are not in themselves considered binding on any Agency.

CCSDS Recommendations take two forms: **Recommended Standards** that are prescriptive and are the formal vehicles by which CCSDS Agencies create the standards that specify how elements of their space mission support infrastructure shall operate and interoperate with others; and **Recommended Practices** that are more descriptive in nature and are intended to provide general guidance about how to approach a particular problem associated with space mission support. This **Recommended Practice** is issued by, and represents the consensus of, the CCSDS members. Endorsement of this **Recommended Practice** is entirely voluntary and does not imply a commitment by any Agency or organization to implement its recommendations in a prescriptive sense.

No later than five years from its date of issuance, this **Recommended Practice** will be reviewed by the CCSDS to determine whether it should: (1) remain in effect without change; (2) be changed to reflect the impact of new technologies, new requirements, or new directions; or (3) be retired or canceled.

In those instances when a new version of a **Recommended Practice** is issued, existing CCSDS-related member Practices and implementations are not negated or deemed to be non-CCSDS compatible. It is the responsibility of each member to determine when such Practices or implementations are to be modified. Each member is, however, strongly encouraged to direct planning for its new Practices and implementations towards the later version of the Recommended Practice.

FOREWORD

This document is a technical Recommended Practice for use in developing flight and ground systems for space missions and has been prepared by the Consultative Committee for Space Data Systems (CCSDS). The File and Packet Store Services described herein is intended for missions that are cross-supported between Agencies of the CCSDS, in the framework of the Spacecraft Onboard Interface Services (SOIS) CCSDS area.

This Recommended Practice specifies a set of related services to be used by space missions to access and manage files and packets within a spacecraft subnetwork. The SOIS File and Packet Store Services provide a common service interface regardless of the particular type of data link or protocol being used for communication.

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Recommended Practice 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-3). 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 addressed to the CCSDS Secretariat at the address indicated on page i.

At time of publication, the active Member and Observer Agencies of the CCSDS were:

Member Agencies

- Agenzia Spaziale Italiana (ASI)/Italy.
- Canadian Space Agency (CSA)/Canada.
- Centre National d'Etudes Spatiales (CNES)/France.
- China National Space Administration (CNSA)/People's Republic of China.
- Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Federal Space Agency (FSA)/Russian Federation.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- Japan Aerospace Exploration Agency (JAXA)/Japan.
- National Aeronautics and Space Administration (NASA)/USA.
- UK Space Agency/United Kingdom.

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Belgian Federal Science Policy Office (BFSPO)/Belgium.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- China Satellite Launch and Tracking Control General, Beijing Institute of Tracking and Telecommunications Technology (CLTC/BITTT)/China.
- Chinese Academy of Sciences (CAS)/China.
- Chinese Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- CSIR Satellite Applications Centre (CSIR)/Republic of South Africa.
- Danish National Space Center (DNSC)/Denmark.
- Departamento de Ciência e Tecnologia Aeroespacial (DCTA)/Brazil.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Geo-Informatics and Space Technology Development Agency (GISTDA)/Thailand.
- Hellenic National Space Committee (HNSC)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Institute of Space Research (IKI)/Russian Federation.
- KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- Korea Aerospace Research Institute (KARI)/Korea.
- Ministry of Communications (MOC)/Israel.
- National Institute of Information and Communications Technology (NICT)/Japan.
- National Oceanic and Atmospheric Administration (NOAA)/USA.
- National Space Agency of the Republic of Kazakhstan (NSARK)/Kazakhstan.
- National Space Organization (NSPO)/Chinese Taipei.
- Naval Center for Space Technology (NCST)/USA.
- Scientific and Technological Research Council of Turkey (TUBITAK)/Turkey.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.
- United States Geological Survey (USGS)/USA.

DOCUMENT CONTROL

Document	Title	Date	Status
CCSDS 873.0-M-1	Spacecraft Onboard Interface Services—File and Packet Store Services, Recommended Practice, Issue 1	September 2012	Current issue

CONTENTS

Section Page

1 INTRODUCTION 1-1

1.1 PURPOSE AND SCOPE OF THIS DOCUMENT 1-1
1.2 APPLICABILITY 1-1
1.3 RATIONALE 1-1
1.4 DOCUMENT STRUCTURE 1-1
1.5 CONVENTIONS AND DEFINITIONS 1-2
1.6 HOW THIS DOCUMENT FITS INTO THE SOIS DOCUMENTATION TREE . 1-4
1.7 CONVENTIONS 1-4
1.8 REFERENCES 1-4

2 SERVICE CONCEPT 2-1

2.1 OVERVIEW 2-1
2.2 FILE STORE OVERVIEW 2-2
2.3 PACKET STORE OVERVIEW 2-4
2.4 PURPOSE AND OPERATION OF THE FILE AND PACKET STORE SERVICES 2-8
2.5 EXAMPLE DEPLOYMENT 2-10

3 SERVICE DEFINITION 3-1

3.1 PROVIDED SERVICE 3-1
3.2 SERVICES EXPECTED FROM THE UNDERLYING LAYERS 3-4
3.3 SERVICE PARAMETERS 3-9
3.4 SERVICE INTERFACE 3-15

4 MANAGEMENT INFORMATION BASE 4-1

4.1 DISCUSSION 4-1
4.2 SPECIFICATIONS 4-1
4.3 MIB GUIDANCE 4-1
4.4 MAXIMUM FILE SIZE 4-1
4.5 MAXIMUM PACKET STORE SIZE 4-1

ANNEX A FILE AND PACKET STORE SERVICES PROTOCOL IMPLEMENTATION CONFORMANCE STATEMENT PROFORMA (NORMATIVE) A-1
ANNEX B SECURITY CONSIDERATIONS (INFORMATIVE) B-1
ANNEX C ACRONYMS (INFORMATIVE) C-1
ANNEX D INFORMATIVE REFERENCES (INFORMATIVE) D-1
ANNEX E TYPICAL API—POSIX (INFORMATIVE) E-1

CONTENTS (continued)

Figure Page

2-1 File and Packet Store Services Context 2-1
2-2 Random-Access Packet Store Logical Structure 2-6
2-3 Dumping Packets from the Packet Store System to the Spacecraft Telemetry Chain 2-7
2-4 Example Deployment of FPSS and File and Packet Store Systems 2-9
2-5 Multiple Distributed Mass Memories in a File Store 2-10
2-6 Example Interfacing from SSMM-based Packet Store System to Telemetry System 2-11
2-7 Example Interfacing from OBC-based Packet Store System to Telemetry System 2-11

1 INTRODUCTION

1.1 PURPOSE AND SCOPE OF THIS DOCUMENT

This document defines the Spacecraft Onboard Interface Services (SOIS) File and Packet Store Services (FPSS). The definition encompasses specification of the service interface exposed to onboard software (user applications and libraries) as well as the conceptual mapping of the FPSS primitives to the protocols implementing such services.

The SOIS File and Packet Store Services are for use by onboard software to:

- Access, and manage files residing in a file store. The files residing in the file store could contain any type of data, including for example telemetry, commands and command sequences, software updates, imagery, and other science observations.
- Access and manage packets residing in a packet store. It should be noted that the packet store may or may not be aware of, and its actions may or may not be informed by, the contents of a packet.

To achieve this, the FPSS comprise the following services:

- File Access Service (FAS);
- File Management Service (FMS);
- Packet Store Access Service (PSAS);
- Packet Store Management Service (PSMS).

It should be noted that the SOIS File and Packet Store Services do NOT define the file and packet stores themselves, but only their minimum provided service. The initialization and configuration of the file and packet stores is out of scope for this document.

1.2 APPLICABILITY

This document applies to any mission or equipment claiming to provide CCSDS SOIS-compatible File and Packet Store Services.

1.3 RATIONALE

SOIS provides service interface specifications in order to promote commonality of functionality amongst systems implementing well-defined services. These interfaces do not dictate implementation of interfaces or protocols supporting the services.

1.4 DOCUMENT STRUCTURE

This document has four major sections:

- section 1, this section, containing administrative information, definitions, and references;

- section 2, containing the general concepts and assumptions;
- section 3, containing the File and Packet Store Services, in terms of the services provided, services expected from underlying layers, and the service interface;
- section 4, containing the Management Information Base (MIB) for this service.

In addition, one normative and four informative annexes are provided:

- annex A, comprising a Service Conformance Statement Proforma;
- annex B, containing a discussion security considerations with respect to the specifications of this document;
- annex C, containing a list of acronyms;
- annex D, containing a list of informative references;
- annex E, describing aspects of the POSIX API file functions which might be useful to implementers of the File Access and Management Services.

1.5 CONVENTIONS AND DEFINITIONS

1.5.1 BIT NUMBERING CONVENTION AND NOMENCLATURE

In accordance with modern data communications practice, spacecraft data fields are often grouped into eight-bit 'words' widely known as bytes. Throughout this Recommended Practice, such an eight-bit word is called an 'octet'. The numbering for octets within a data structure starts with zero.

By CCSDS convention, any 'spare' bits shall be permanently set to '0'.

1.5.2 DEFINITIONS

1.5.2.1 Definitions from the Open Systems Interconnection (OSI) Reference Model

The document is defined using the style established by the Open Systems Interconnection (OSI) Basic Reference Model (reference [D1]). This model provides a common framework for the development of standards in the field of systems interconnection.

The following terms used in this Recommended Practice are adapted from definitions given in (reference [D1]):

layer: A subdivision of the architecture, constituted by subsystems of the same rank.

(N)-protocol: A set of rules and formats (semantic and syntactic) which determines the communication behavior of (N)-entities in the performance of (N)-functions.

service: A capability of a layer, and the layers beneath it (service providers), provided to the service users at the boundary between the service providers and the service users.

(N)-service-access-point, (N)-SAP: The point at which (N)-services are provided by an (N)-entity to an (N+l)-entity. Within the spacecraft, a SOIS User Service Access Point. As a minimum it locates a data system and an application within that data system.

1.5.2.2 Terms defined in this Recommended Practice

For the purposes of this Recommended Practice, the following definitions also apply:

application: Any component of the onboard software that makes use of the File and Packet Store Services. This includes flight software applications and higher-layer services.

file: An ordered collection of arbitrary information, or resource for storing information, residing in some kind of durable storage, and with an associated name.

file system: A method of storing and organizing files and the data they contain to make it easy to find and access them. File systems may use a local data storage device or provide access to data on a remote server over a network. Optionally, there may be a hierarchy of files, i.e., a folder or directory tree.

file store: A file system and its storage media. A file store comprises:

- one or more *mass memories* in which files reside;
- an associated *file system* providing services for managing the files stored in the mass memories. This document does not intend to define a file system; it is therefore assumed that a file system interfacing to one or more mass memories is already present.

NOTE – No assumption is made about persistence of files in the file store or any file replication strategies.

packet: At a minimum level of complexity, a delimited data structure. Examples of such packets are CCSDS Space Packets (reference [D4]), CCSDS Encapsulation Packets (reference [D5]), IPv4 (reference [D6]) and IPv6 (reference [D7]) packets. Packets are time-stamped with the time the packet was stored in the packet store. For CCSDS packets, primary header fields are accessible to the PSAS. These include packet identifiers and protocol identifiers.

packet store: A collection of packets and its storage media. A packet store comprises:

- one or more *mass memories* in which packets reside;
- an associated *packet storage system* providing services for managing the packets stored in the mass memories.

NOTES

1 No assumption is made about persistence of packets in the packet store or any replication strategies.

2 No assumption is made about where the packet store is located (remotely or locally).

octet: An eight-bit word.

value: A formatted atomic unit of data that is stored in a file or a packet.

1.6 HOW THIS DOCUMENT FITS INTO THE SOIS DOCUMENTATION TREE

This document conforms to the principles set out in the Spacecraft Onboard Interface Services Green Book (reference [D2]) and should not be applied without first consulting this reference.

1.7 CONVENTIONS

1.7.1 NOMENCLATURE

The following conventions apply for the normative specifications in this Recommended Practice:

a) the words ‘shall’ and ‘must’ imply a binding and verifiable specification;

b) the word ‘should’ implies an optional, but desirable, specification;

c) the word ‘may’ implies an optional specification;

d) the words ‘is’, ‘are’, and ‘will’ imply statements of fact.

NOTE – These conventions do not imply constraints on diction in text that is clearly informative in nature.

1.7.2 INFORMATIVE TEXT

In the normative sections of this document (sections 3-4 and annex A), informative text is set off from the normative specifications either in notes or under one of the following subsection headings:

– Overview;

– Background;

– Rationale;

– Discussion.

1.8 REFERENCES

This document contains no normative references. Informative references are contained in annex D.

2 SERVICE CONCEPT

2.1 OVERVIEW

The SOIS File and Packet Store Services (FPSS) are defined within the context of the overall SOIS architecture (see reference [D2]) as one of the services of the Application Support Layer, as illustrated in figure 2-1.

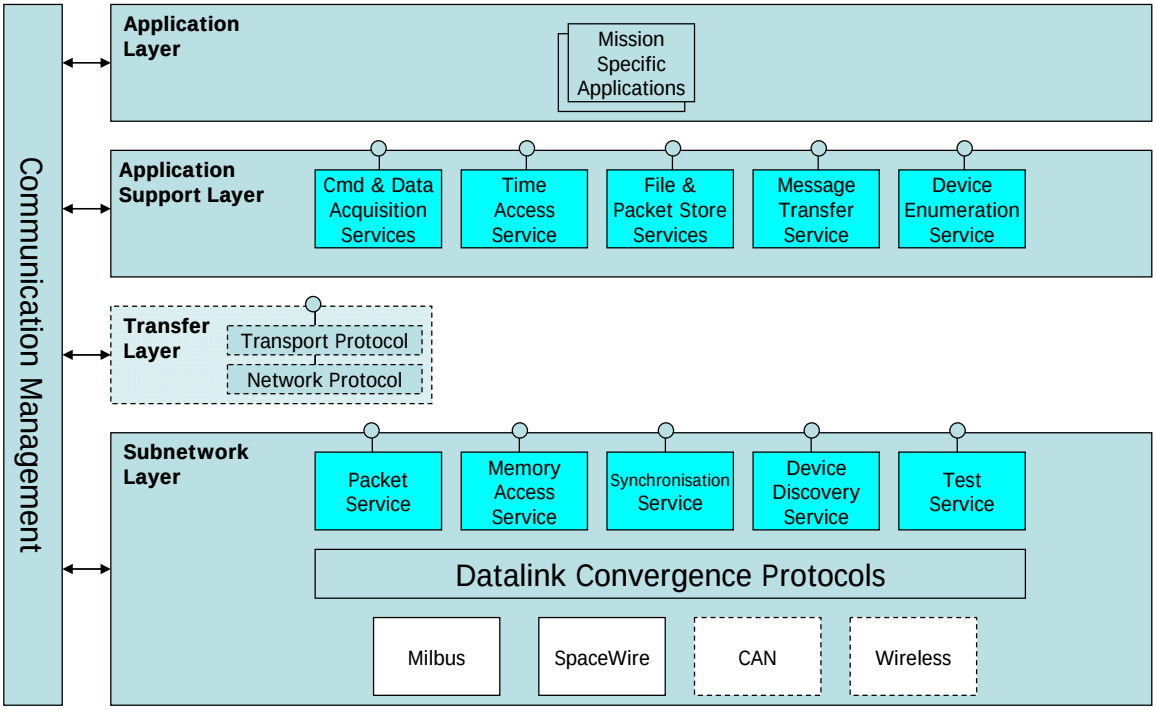

Figure 2-1: File and Packet Store Services Context

The SOIS File and Packet Store Services provide a standard interface to allow onboard software to request:

- access to files resident in a file store;
- modifications to files in a file store;
- management of files within a file store (e.g., create, delete, rename);
- storing, retrieving, and deleting packets in a packet store;
- management of packet stores.

The basic concept underlying the service is that the onboard software should be able to access files in a file store or packets in a packet store independently of the precise physical location or address of the store within the spacecraft's communication architecture, and without requiring detailed knowledge of the mechanism used to access or implement the store. A standard interface makes it easier to develop onboard software, enables configuration changes in the spacecraft design to be easily tolerated, and increases the re-use potential of the software.

2.2 FILE STORE OVERVIEW

2.2.1 TYPES OF FILE STORES

A file store is considered to be a file system and the associated storage medium.

At the highest level a file system is a way to organize, store, retrieve, and manage information on a permanent storage medium. File stores manage permanent storage and form an integral part most onboard spacecraft data handling systems.

Different types of file systems (and file stores) may be considered:

a) **Flat file systems**. Designed to not have subdirectories. Everything is stored in the root level of the storage media.

b) **Hierarchical file systems**. Designed to organize information in directories in a tree-like structure.

Either of these may be used to support the FPSS services and no restriction is imposed on the type of file store. This approach allows complete independence from the technology used to implement the file store. The way in which this mapping is performed is implementation-specific.

The FPSS allow for multiple client processes on multiple machines for both accessing and updating files. Hence updates to the file from one client should not interfere with access and updates from other clients. It is assumed that *concurrency control* or *locking* is handled by the file store itself. In support of this, file locking functionality is provided. An application (the 'lock owner') may lock a file to various degrees of access (exclusive read-only, read-only by all, exclusive access) from other user entities. Any user entity may unlock the file.

NOTE – It is implementation-specific if unlocking of files is restricted to, for example, a 'super-user' application.

2.2.2 CURRENT DIRECTORY

The **current directory** within a file store for each application is maintained by the file system. A number of operations are performed relative to an application's current directory, e.g., creating a file. By default, upon initialization, the application's current directory is at the root level of the storage media for flat file systems and at the root level of the tree for hierarchical file systems. For hierarchical file systems, each application's current directory can be changed by request.

2.2.3 TYPES OF FILES

A file is where a user stores information. Files are logically organized into one-dimensional arrays of octets. The format of a file is defined by its content since a file is solely a container

for data. The size of data stored is arbitrary and may range from only a few octets to the entire capacity of the entire mass memory in which files reside. Files are not limited in size (subject to the limitations of the mass memory in which the files reside). A file store should be able to hold a potentially large number of files, where 'large' ranges from tens of thousands to millions.

Different types of files may be considered:

a) **Regular**. Regular files are the most common files and are used to contain data, for example, information stored in ASCII format text and readable by the user, or information in computer-readable formats (e.g., software images).

b) **Directory**. A directory is a file system structure that contains information that the system needs to access all types of files, but directories do not contain actual file data. Directories permit convenient management of the file system structure. Directories are created and controlled by a separate set of commands.

It is assumed that the file store has regular files. Directory files are optional for SOIS purposes.

2.2.4 BASIC FILE OPERATIONS

There is a set of basic operations that may be performed upon a file in a file system: open, read, write, append, seek, and close.

The **current position** within an open file for each application is maintained by the file system. By default, upon opening a file, the application's current position in the file is initialized to the start of the file. Reading data from and writing data to files is performed at the application's current position, and the application's current position is incremented by the length of data read or written.

Writing to a file overwrites the original data with new data, unless the application's current position is at the end of the file (i.e., beyond the last data in the file), in which case the new data is appended to the end of the file. Reading when the application's current position is at the end of the file returns no data.

The application's current position can be moved to a specified offset from the start of the file using a seek operation, thereby providing random access to the file. This may, for example, be used to overwrite part of a file's contents or to read from a part of a file.

Finally, upon opening a file, the application may indicate that the file is being opened to be appended to, in which case the application's current position in the file is initialized to the end of the file.

Although this Recommended Practice does not exclude a file's being open for simultaneous writing by multiple user entities, it is strongly advised that only a single application at any one time should have write access.

2.3 PACKET STORE OVERVIEW

2.3.1 GENERAL

Any storage technology that is suitable for spacecraft missions may be used to implement a packet store. This approach allows complete independence from the technology used to implement the packet store. The way in which this mapping is performed is implementation-specific.

The FPSS should provide for multiple client processes on multiple machines accessing the same packet store. Hence access from one client should not interfere with accesses from other clients. It is assumed that concurrency control or locking is handled by the packet store itself.

The packet store may provide a separate interface to the spacecraft telemetry system that permits packets to be 'dumped' (i.e., sent directly) to the ground using the spacecraft telemetry system (see 2.3.4).

2.3.2 TYPES OF STORED PACKETS

The packet store may or may not be aware of the contents of a packet. A stored packet is, at a minimum level of complexity, just a delimited data structure.

Examples of such packets are the CCSDS Space Packet (reference [D4]), the CCSDS Encapsulation Packet (reference [D5]), IPv4 (reference [D6]) packets, and IPv6 (reference [D7]) packets. Packet stores are tagged as containing either CCSDS packets or non-CCSDS packets as searches on structural elements of a packet can only be performed on the primary header of CCSDS Space and CCSDS Encapsulation Packets (see 2.3.3.2.2).

Any storage technology that is suitable for spacecraft missions may be used to implement a packet store. This approach allows complete independence from the technology used to implement the packet store. The way in which this mapping is performed is implementation-specific.

The FPSS should provide for multiple client processes on multiple machines accessing the same packet store. Hence access from one client should not interfere with accesses from other clients. It is assumed that *concurrency control* or *locking* is handled by the packet store itself.

2.3.3 TYPES OF PACKET STORES

Packet stores onboard a spacecraft can be organized in different ways, with the most common being:

a) **First-in, First-Out** (FIFO):

   1) bounded;

   2) circular;

b) **random-access**.

Other packet store types exist, but this specification covers only the above mentioned types. The following subsections provide more detail on these common types.

2.3.3.1 FIFO Packet Store

The FIFO Packet Store is used to store packets in a FIFO queue; i.e., packets are written to the tail of the queue and read from the head of the queue.

As the packet store is a FIFO, only the packet sizes have to be known, not their structure.

Writing to and reading from the packet store is always performed in sequential order, writing new packets to the tail, reading the oldest unread packets from the head, and freeing the oldest packets from the head of the queue.

Although this Recommended Practice does not exclude multiple user entities' reading, dumping, and freeing from the packet store, it is strongly advised that only a single application should read, dump, and free from the packet store. FIFO Packet Stores can be of two forms:

- **Bounded**. A bounded FIFO Packet Store has a defined maximum number of packets that can be stored in its queue. When the queue is full, no more packets can be added to the tail of the queue: packets in the Packet Store are NEVER removed/discarded by the service in order to store new packets; instead, the user must explicitly free them. When freeing from the head of the queue, the packet is removed and a space freed up at the tail for a new packet.
- **Circular**. A circular FIFO Packet Store also has a defined maximum number of packets that can be stored in its queue. However, unlike the bounded FIFO Packet Store, when attempting to write to the tail of a full queue, the packet at the head of the queue, i.e., the oldest, is deleted. This kind of packet store can never run out of space to store new packets but results in the possibility of packets being discarded by the service.

2.3.3.2 Random-Access Packet Store

In addition to the FIFO queue of a FIFO packet store, a random-access Packet Store allows selective read and delete access to stored packets. This may for example be used for payload-specific telemetry.

The logical structure of a random-access packet store is illustrated in figure 2-2. The packets in the packet store are stored in a time-stamped (see 2.3.3.2.1) FIFO queue, with a write pointer at the tail of the queue (i.e., the newest packet), a read pointer in the queue pointing to the next packet to be read, and a free pointer at the head of the queue (i.e., the oldest packet). When a packet is written to the packet store, the packet is added to the tail of the queue and the write pointer modified to point to the packet. When a packet is read from the packet store, the packet pointed to by the read pointer is returned to the user and the read

pointer is moved to the next packet in the queue. When a packet is freed from the packet store, any storage associated with the packet pointed to by the free pointer is released and the free pointer is moved to the next packet in time order.

A selection of packets can be made, based on selection criteria (see 2.3.3.2.2). This is also held as a time-stamped FIFO queue (the 'selection queue') of references to the packets in the main FIFO queue. A selective read pointer points to the next packet to be read from the selection. A selective free pointer points to the head of the selection queue. When a packet is selectively read from the packet store, the packet pointed to by selective read pointer is returned and the selective read pointer moved to the next reference in the selective queue. When a packet is selectively freed from the packet store, any storage associated with the packet referenced by the selective free pointer is released and the selective free pointer is moved to the next reference in the selection queue (potentially updating any pointers associated with the packet store, e.g., where the freed packet was also the oldest packet in the packet store).

A request to free packets from the packet store may free packets referenced in the selection queue. Therefore, the selection will be marked invalid and any resources associated with the selection queue released.

Although this Recommended Practice does not exclude multiple user entities' using packet selections, it is strongly advised that only a single application at any one time should use packet selections.

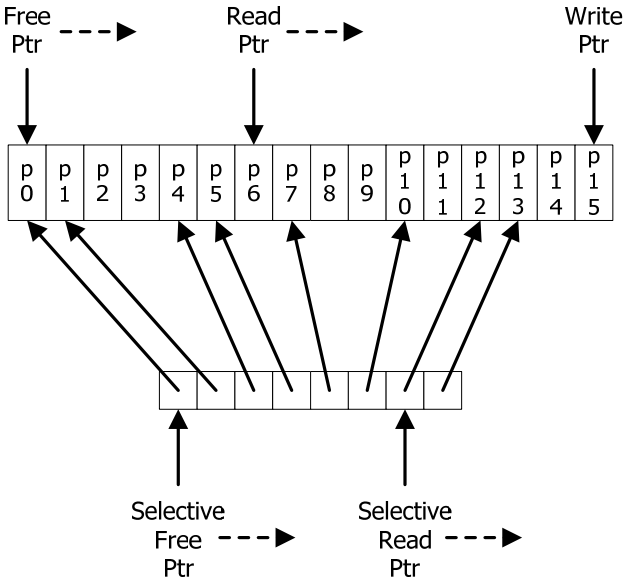

Figure 2-2: Random-Access Packet Store Logical Structure

NOTE – Having selective delete raises the problem of the **packet store fragmentation**. The implementation of the service should therefore take into account a cleaning policy to compact the live data in order to avoid any performance degradation due to progressive fragmentation.

2.3.3.2.1 Time Stamping of Packets

The time stamp associated with a packet will be the time when the packet was written to the packet store. This is only applied to either FIFO or random access packet stores.

NOTE – A time stamp may also be present in a CCSDS packet's secondary header. Packet Store implementations may have knowledge of this; however, this is not covered by this CCSDS Recommended Practice.

2.3.3.2.2 Packet Selection Criteria

The applicable selection criteria (only applied to random access packet stores) are as follows:

a) packet store '*write time*';

b) one or more structural elements within the primary header of the CCSDS packet (valid only for packet stores containing CCSDS Packets).

NOTES

1 The selection criteria used for a single field is dependent upon the particular field and is implementation-dependent, e.g., an exact match to the APID field.

2 A Packet Store implementation may have knowledge of the contents of a secondary header of the CCSDS packet; however, this is not covered by this CCSDS Recommended Practice.

2.3.4 DUMPING OF TELEMETRY PACKETS

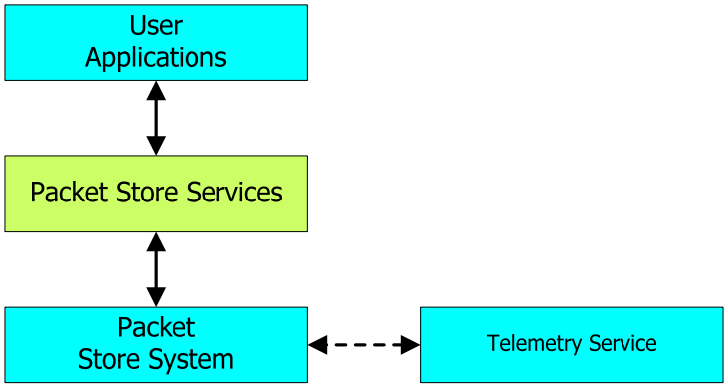

Figure 2-3: Dumping Packets from the Packet Store System to the Spacecraft Telemetry Chain

A number of packet store systems support the direct dumping of packets to the spacecraft telemetry chain, this being more efficient than a packet store service user reading packets and then sending them to the spacecraft telemetry system.

The packet store services provide interfaces for dumping and selectively dumping packets directly from the packet store system to the spacecraft telemetry system.

NOTE – The implementation of the interface from the packet store system to the spacecraft telemetry chain is not specified here. It may or may not make use of other SOIS services.

2.4 PURPOSE AND OPERATION OF THE FILE AND PACKET STORE SERVICES

Each service provides a consistent, standard interface to onboard software; the interfaces are described by sets of primitives and related parameters.

From the user's perspective, use of the FPSS will result in onboard software that is more portable, easier to develop, and more tolerant of changes in the spacecraft hardware configuration.

From the spacecraft platform implementer's perspective, use of the FPSS will make it easier to control access and management of shared hardware resources (i.e., mass memories).

The FPSS are operated using service requests and service indications passed between the service user and the service provider.

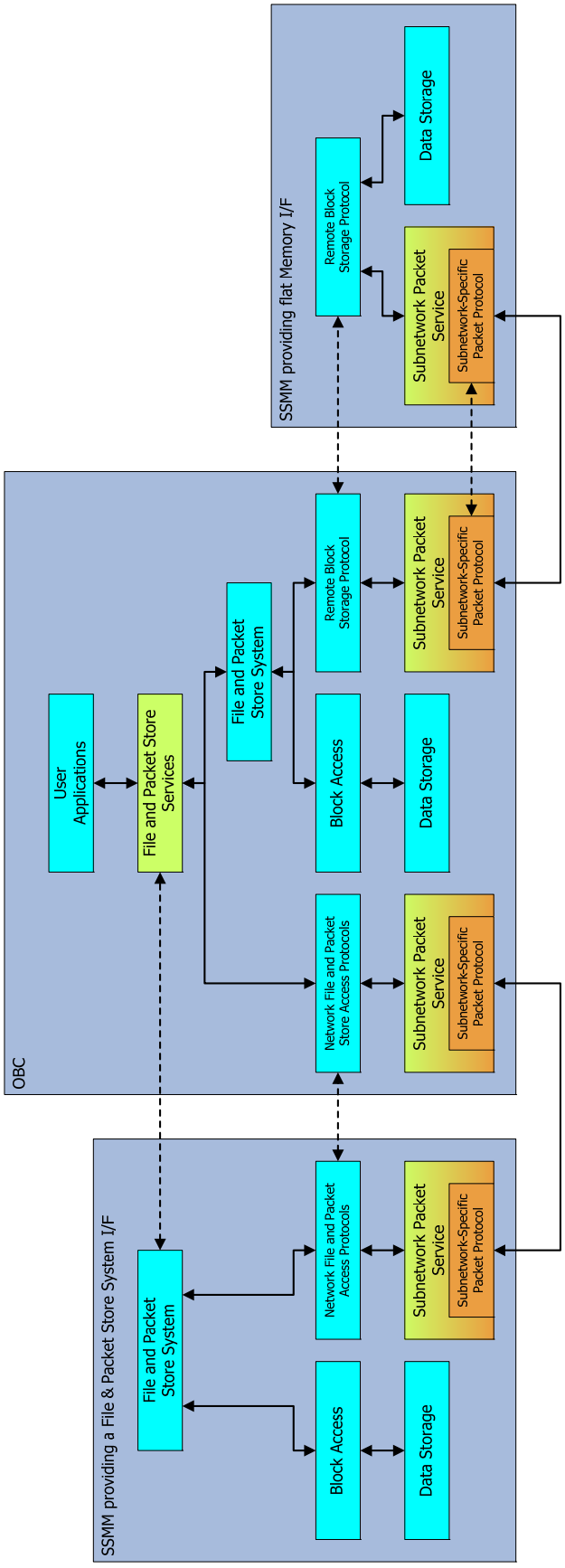

Figure 2-4: Example Deployment of FPSS and File and Packet Store Systems

2.5 EXAMPLE DEPLOYMENT

There are a variety of possible provisions of file and packet store systems and the associated access from the FPSS running on an OnBoard Computer (OBC), illustrated in figure 2-4:

a) a local file and packet store system using:

   1) local data storage, directly accessed;

   2) remote data storage on a simple mass memory ('SSMM[1] providing flat Memory I/F'), accessed using a Remote Block Storage Protocol;

b) a remote file and packet stores on a complex mass memory ('SSMM providing a File System I/F'), accessed using Network File and Packet Access Protocols.

It should be noted that, in addition, a file and/or packet store can be composed of more than one mass memory device interfaced by a single file or packet store system. Multiple, distributed mass memories may be accessed through the single file or packet store system over the same subnetwork or different subnetworks (see figure 2-5, below). This requires the FPSS to use protocols providing network file access and packet store access (labelled generically as Network File and Packet Access Protocols). However, any protocol used specifically by the file or packet store system (labelled generically as Remote Block Storage Protocol, not specified here) is expected to be distinct from the protocols used by the FPSS. The application is isolated from any use of such multiple, distributed mass memories; e.g., file and directory names need to be unique across a complete file store, not just within an individual mass memory.

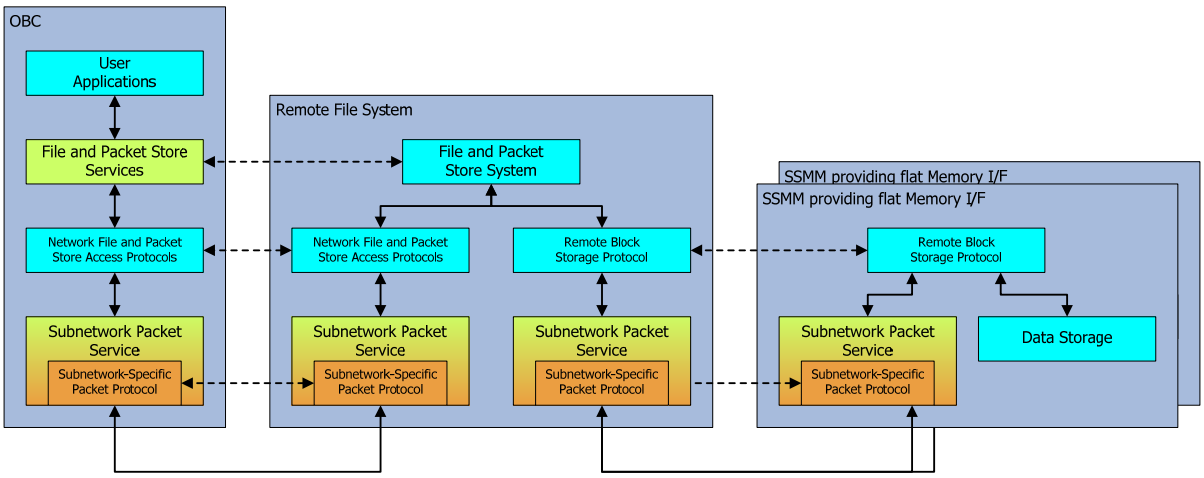

Figure 2-5: Multiple Distributed Mass Memories in a File Store

[1] SSMM – Solid State Mass Memory

With regard to the dumping of telemetry packets, as described in 2.3.4, the interface between a Packet Store and the Spacecraft Telemetry System required for this function is private and outside the scope of the Packet Store Services. Below are illustrations providing two possible implementations, figure 2-6 for a mass memory providing a Packet Store Service interface, and figure 2-7 for a Packet Store System implemented on an onboard computer.

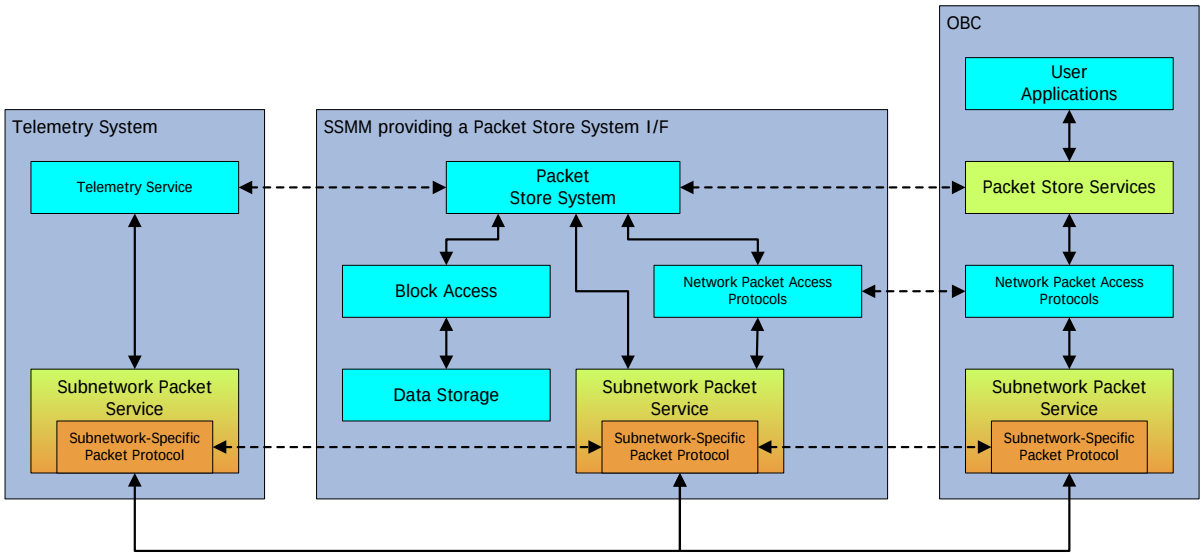

Figure 2-6: Example Interfacing from SSMM-based Packet Store System to Telemetry System

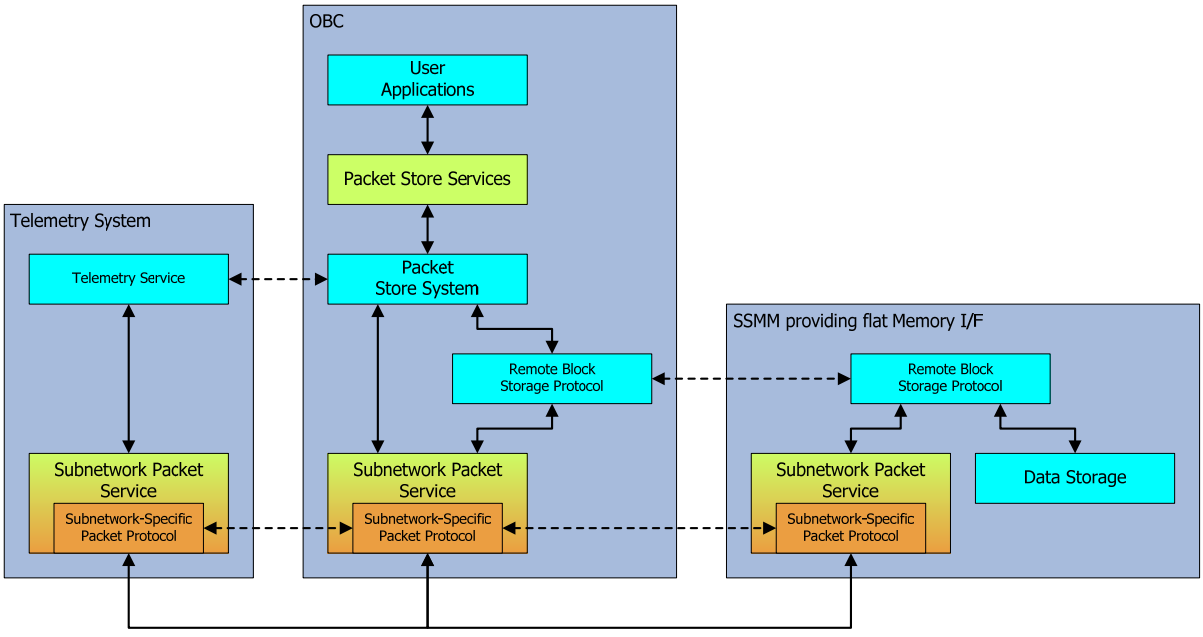

Figure 2-7: Example Interfacing from OBC-based Packet Store System to Telemetry System

3 SERVICE DEFINITION

3.1 PROVIDED SERVICE

3.1.1 GENERAL

NOTE – The FPSS are divided into the following four services:

– File Access Service;

– File Management Service;

– Packet Store Access Service;

– Packet Store Management Service.

3.1.1.1 To provide a file system onboard a spacecraft, both the File Access Service and the File Management Service shall be provided.

3.1.1.2 To provide a packet store onboard a spacecraft, both the Packet Store Access Service and the Packet Store Management Service shall be provided.

3.1.1.3 A packet store may be provided where only a file system is required.

3.1.1.4 A file system may be provided where only a packet store is required.

NOTE – The following subsections detail the provisions of each of these services.

3.1.2 FILE ACCESS SERVICE

3.1.2.1 The *File Access Service* (FAS) shall enable a user entity to access 'existing' files in a file store, regardless of that file store's location; i.e., the accessed files can reside on local or remote file stores with respect to the user entity.

3.1.2.2 The service shall, as a minimum, give a user entity the ability to:

– open and close a file;

– read from a file;

– write to a file.

3.1.2.3 Optionally, the service may give a user entity the ability to 'seek', i.e., move, the user entity's current position within a file.

3.1.2.4 Each FAS implementation shall support either a local file store or a remote file store and may support both.

3.1.2.5 If a local file store is implemented, each FAS implementation shall be mapped to a single, local file store's file system.

NOTE – For a local file store, whilst the file system itself is local, the associated storage may be remote, in which case the storage is accessed by a Remote Block Storage Protocol (not defined here).

3.1.2.6 If a remote file store is implemented, each FAS implementation shall be mapped onto a Network File Access Protocol (see 3.2.4).

3.1.2.7 Such mappings shall be transparent to a user entity.

3.1.3 FILE MANAGEMENT SERVICE

3.1.3.1 General

3.1.3.1.1 The *File Management Service* (FMS) shall enable a user entity to create and manipulate files in a file store, regardless of that file store's location; i.e., the accessed files can reside on local or remote file stores with respect to the user entity.

3.1.3.1.2 The service shall, at a minimum, give a user entity the ability to:

- list the contents of a directory;
- copy and move files;
- create and delete files.

3.1.3.1.3 Optionally, the service may give a user entity the ability to:

- create, rename, and delete directories;
- get and change the user entity's current directory;
- rename files;
- lock or unlock files;
- list the currently locked files;
- find a file within a file store;
- get the status of a file.

3.1.3.1.4 To manage a file store, each FMS implementation must be mapped to a file store's file system.

3.1.3.1.5 Such mappings shall be transparent to a user entity.

3.1.3.2 Locking Files

3.1.3.2.1 If file locking is implemented, a user entity shall be able to lock files in order to prevent other users from reading from or writing to a particular file.

3.1.3.2.2 The user entity that applied a lock to a file shall be known as the lock-owner.

3.1.3.2.3 A file shall stay locked until it is released by a user entity (not necessarily the lock owner).

NOTE – Locks provide control of access to files.

3.1.3.2.4 The access control options associated with locks are: no access, read-only access, and read-write access. Different access control associated with locks can be applied to lock-owners compared with all other user entities. The following permutations are supported:

- lock-owner read-only access, all other user entities no access;
- lock-owner read-only access, all other user entities read-only access;
- lock-owner read-write access, all other user entities no access.

3.1.3.2.5 In case of successful locking, an identifier shall be associated with the lock, known as the lock identifier, and the lock shall remain until unlocked.

3.1.3.2.6 Any user entity providing the correct lock identifier shall be able to unlock the locked file.

3.1.3.2.7 A user entity shall be able to obtain a list of current lock identifiers.

NOTE – This, for example, may be used to obtain the locks held by a failed application so that they may be unlocked.

3.1.4 PACKET STORE ACCESS SERVICE

3.1.4.1 The *Packet Store Access Service* (PSAS) shall enable a user entity to access packets in a packet store, regardless of that packet store's location; i.e., the accessed packets can reside on local or remote packet stores with respect to the user entity.

3.1.4.2 The packet store may provide a private interface to the spacecraft telemetry system to permit packets to be dumped (i.e., sent) directly from the packet store to that system for transmission without being transferred over the interface to the PSAS implementation.

3.1.4.3 The service shall, at a minimum, give a user entity the ability to:

- get list of all packet stores;
- clear all packets from a packet store;

– write packets to a packet store;

– read packets from a packet store;

– set the read pointer position in a packet store;

– delete packets from a packet store;

– get the status of a packet store.

3.1.4.4 Optionally, the service may give a user entity the ability to read and free selected packets in a packet store (random-access packet stores only).

3.1.4.5 Optionally, if the packet store implements a private interface to the spacecraft telemetry system, the service may give a user entity the ability to:

– dump packets to the telemetry system;

– dump to the telemetry system selected packets in a packet store (random-access packet stores only).

3.1.4.5.1 To access a local packet store, each PSAS implementation must be mapped onto a single local packet store's storage system.

NOTE – For a local packet store, whilst the storage system itself is local, the associated storage may be remote, in which case the storage is accessed by a Remote Block Storage Protocol (not defined here).

3.1.4.5.2 To access a remote packet store, each PSAS implementation must be mapped onto a Network Packet Access Protocol (see 3.2.5). Such mappings shall be transparent to the user entity.

3.1.5 PACKET STORE MANAGEMENT SERVICE

3.1.5.1 The *Packet Store Management Service* (PSMS) shall enable a user entity to manage packet stores, regardless of that packet store's location; i.e., it can be local or remote with respect to the user entity on the same spacecraft.

3.1.5.2 The service shall at a minimum give a user entity the ability to create and delete packet stores.

3.2 SERVICES EXPECTED FROM THE UNDERLYING LAYERS

3.2.1 OVERVIEW

The services that the FPSS expect from the underlying layers are divided into the following capabilities:

– local file store services;

- local packet store services;
- network file access protocol (optional);
- network packet access protocol (optional).

The requirements on each are detailed in the following subsections.

3.2.2 LOCAL FILE STORE SERVICES

NOTE – As the way in which the storage capability is provided may vary, the FPSS is built on the premise that any file or organized set of files (i.e., a file store) can be accessed and managed through a set of file system services.

3.2.2.1 The file store shall provide either of the following file system types:

- flat file system;
- hierarchical file system.

3.2.2.2 The file store shall provide the following set of file system services:

- list contents of directory;
- create file;
- open a file;
- close a file;
- read from file;
- write to file;
- delete file;
- move file;
- copy file.

3.2.2.3 Optionally, the file store may provide the following set of file system services:

- create directory;
- remove directory;
- rename directory;
- get user entity’s current directory;
- change user entity’s current directory;
- lock and unlock files;

– move a user entity's current position within file;

– get file status.

3.2.2.4 The file store shall maintain the following attributes to a file:

– name;

– lock status (only where the lock and unlock files capabilities are provided);

– file size.

3.2.2.5 Optionally, the file store may maintain the following attributes to a file:

– creation time;

– last write time.

3.2.2.6 In an implementation of the FPSS, the file system services must be mapped to and from actual hardware and software that constitute the real file store.

3.2.2.7 These file system services shall be used by the FPSS to access, manage, and transfer files in a local file store or via a network file access protocol (see 3.2.4) for a remote file store.

3.2.3 LOCAL PACKET STORE SERVICES

NOTE – As the way in which the storage capability is provided may vary, the FPSS is built on the premise that any packet or organized set of packets (i.e., a packet store) can be accessed and managed through packet storage system services.

3.2.3.1 The packet store shall provide the following packet store types:

– bounded FIFO packet store.

3.2.3.2 The packet store may provide the following packet store types:

– circular FIFO packet store.

– hierarchical file system;

3.2.3.3 The packet store shall provide the following set of packet storage system services:

– reset—delete all packets in a packet store;

– write—write packets to the packet store;

– read—read the next *n* packets from the packet store;

– move by—move the user's next packet pointer backwards or forwards by *n* packets in the packet store;

– free—free the oldest *n* packets in the packet store;

– status report—report the status of the packet store;

3.2.3.4 Optionally, the packet store may provide the following set of packet storage system services:

– dump—dump packets to the ground using the spacecraft telemetry system;

– selective seek—create a new time-ordered set of packets in the packet store (random-access packet stores only) meeting the specified selection criteria;

– selective read—read the next *n* packets from the time-ordered set in the packet store (random-access packet stores only) selected by the selective seek capability;

– selective free—free the oldest *n* packets in the time-ordered set in the packet store (random-access packet stores only) selected by the selective seek capability;

– selective dump—dump the next *n* packets from the time-ordered set in the packet store (random-access packet stores only) selected by the selective seek capability.

3.2.3.5 The Packet Store shall support (where provided) concurrently the following operations: reading, selective reading, writing, dumping, freeing, and selective freeing of disjoint packets in the same packet store.

3.2.3.6 In an implementation of the FPSS, the packet storage system services must be mapped to and from actual hardware and software that constitute the real packet store.

3.2.3.7 These packet storage system services shall be used by the FPSS to access packets in a local packet store or via a network packet access protocol for a remote packet store.

3.2.4 NETWORK FILE ACCESS PROTOCOL

Where support is provided for the set of capabilities defined in 3.2.2 for accessing and managing files residing in a *remote* file store, an underlying protocol providing network file access is required to be used. This specification does not include a definition of the functions or design of such a protocol, this being a matter of the FPSS implementation.

NOTE – As the way in which the storage capability is provided may vary, the FPSS is built on the premise that any file or organized set of files (i.e., a file store) can be accessed and managed remotely through a network file system interface protocol.

3.2.5 NETWORK PACKET ACCESS PROTOCOL

Where support is provided for the set of capabilities defined in 3.2.3 for accessing and managing packets residing in a *remote* packet store, an underlying protocol providing

network packet access is required to be used. This specification does not include a definition of the functions or design of such a protocol, this being a matter of the FPSS implementation.

NOTE – As the way in which the storage capability is provided may vary, the FPSS is built on the premise that any packet or organized set of packets (i.e., a packet store) can be accessed and managed remotely through a network packet access protocol.

3.3 SERVICE PARAMETERS

3.3.1 GENERAL

The File and Packet Store Services shall use the parameters specified in 3.3.2 to 3.3.4.

3.3.2 GENERAL PARAMETERS

3.3.2.1 Transaction Identifier

3.3.2.1.1 The Transaction Identifier parameter shall be a value, assigned by the invoking user entity, which is subsequently used to associate indication primitives with the causal request primitives.

NOTE – The user entity is thus able to correlate all indications and confirmations with the originating service request.

3.3.2.1.2 The Transaction Identifier shall be unique within the user application entity.

3.3.2.1.3 Uniqueness in the service provider shall be achieved through the concatenation of Transaction Identifier and either Packet Store Access Service Access Point (PSASAP) Address or Packet Store Management Service Access Point (PSMSAP) Address depending upon the service.

3.3.2.2 Result Metadata

The Result Metadata parameter shall be used to provide information generated by the service provider to the user entity to provide information related to the successful or failed result of a device access operation.

NOTE – The parameter can also include other information indicating failure conditions; e.g., the specified request could not be serviced within the managed timeout period or the Device Access Service is not functioning correctly.

3.3.3 FILE ACCESS AND MANAGEMENT SERVICE PARAMETERS

3.3.3.1 File Identifier

The File Identifier parameter shall be used to logically identify an opened file in a file store.

3.3.3.2 File Name

The File Name parameter shall be a string used to uniquely identify a file in a directory and may or may not contain the full file path from the ‘root’ directory (see 3.3.3.6). If no path is provided, it shall be assumed to refer to the user entity’s current directory.

3.3.3.3 File Attributes

The File Attributes parameter shall be used to list the attributes of a file, containing the following information:

- **File Name** identifies the file (see 3.3.3.2).
- **Creation Time** (optionally) is the time at which the file was created.
- **Last Write Time** (optionally) is the time at which the file was last written to.
- **Lock Identifier** (optionally) identifies a lock on the file (if locked) (see 3.3.3.12).
- **Lock Type** identifies the type of lock on the file (if locked) (see 3.3.3.11).
- **Lock Owner** identifies the user entity owning the lock on the file (if locked).
- **Size** is the size of the file in octets.

3.3.3.4 Directory Listing

The Directory Listing shall be used to list the attributes (see 3.3.3.3) of all files in a directory.

3.3.3.5 File Selection Pattern

The File Selection Pattern parameter shall be used to specify the pattern of any single file attribute (see 3.3.3.3) on which the files are to be found. This may include wildcards and string pattern matching expressions.

NOTE – The format and implementation of wildcards and string pattern matching expressions is implementation-specific.

3.3.3.6 Directory Name

3.3.3.6.1 The Directory Name parameter shall be used to uniquely identify a directory and may or may not contain the full path from the ‘root’ directory.

3.3.3.6.2 If no path is provided, it shall be assumed to be in the user entity’s current directory.

3.3.3.6.3 A special Directory Name parameter value shall identify the ‘root’ directory.

3.3.3.6.4 A special Directory Name parameter value may identify the directory above the user entity’s current directory in the hierarchy.

3.3.3.7 File Opening Criteria

3.3.3.7.1 The File Opening Criteria parameter shall indicate the criteria to be applied to a file when opened. It should be noted that the access rights granted to the file may differ from the ones requested by the File Opening Criteria parameter. Possible attributes associated with opening a file are:

- **Access Type***:*
  - *Read-only*—the user entity can only read from the file;
  - *Read-write*—the user entity can read from or write to the file.
- **Create**—if specified, the file does not exist it will be created.
- **Lock Type**—(optionally) if specified, the file is locked atomically when opened (see 3.3.3.11).
- **Append**—if specified, the user entity's current position is initialized to the end of the file.

3.3.3.7.2 In the above list of attribute types only the Access-type is mandatory.

3.3.3.8 File Offset

The File Offset parameter shall indicate the desired current position within a file to which file data is to be read from or written to. Offset 'zero' indicates the first octet in the file.

3.3.3.9 File Data Length

The File Data Length parameter shall indicate the length in octets of the data to be read or written (including the octet located at the current position).

3.3.3.10 File Data

The File Data parameter shall be the data read from or to be written to a file.

3.3.3.11 Lock Type

The Lock Type parameter shall specify a lock action on a file. Possible values are:

- *Exclusive_Read_Only*. The lock-owner can only read from the file. Other user entities cannot read from or modify the file.
- *Read_Only*. The lock-owner can only read from the file. Other user entities can only read from the file.

– *Exclusive_Access*. The lock-owner can read from and modify the file. Other user entity cannot read from or modify the file.

3.3.3.12 File Lock Identifier

The File Lock Identifier parameter shall be used to logically identify an open lock on a file.

3.3.3.13 File Lock List

The File Lock List parameter shall be used to list currently locked files and shall consist of entries, each containing the following information:

– **File Lock Identifier** identifies a lock on a file (see 3.3.3.12).

– **File Name** identifies the associated file upon which the lock is applied (see 3.3.3.2).

– **Lock Type** identifies the associated lock action (see 3.3.3.11).

– **Lock Owner** identifies the associated user entity owning the lock.

3.3.4 PACKET STORE ACCESS AND MANAGEMENT SERVICE PARAMETERS

3.3.4.1 Packet Store Description List

The Packet Store Description List parameter shall be used to list descriptions of packet stores. Each packet store description consists of the following information:

– **Packet Store Identifier** identifies the packet store. This is unique within the spacecraft domain.

– **Packet Store Type** is the type of packet store; either Bounded FIFO, Circular FIFO or Random-Access.

– **Packet Type** is a flag indicating the type of packets in the packet store: either CCSDS or non-CCSDS packets.

– **Number of Stored Packets** is the number of packets currently stored in a packet store.

– **Time Range** is the time of the earliest packet and the time of the latest packet in the packet store.

– **Allocated Space** is the amount of octets allocated to storage of packets in the packet store.

– **Free Space** is the number of octets currently not used to store packets in the packet store.

3.3.4.2 Packet Store Identifier

The Packet Store Identifier parameter shall be used to identify the packet store.

3.3.4.3 Packet Store Type

The Packet Store Type parameter shall be used to identify the type of packet store; either Bounded FIFO, Circular FIFO, or Random-Access.

3.3.4.4 Packet Type

The Packet Type parameter shall be used to indicate the type of packets: either CCSDS or non-CCSDS packets.

3.3.4.5 Number of Stored Packets

The Number of Stored Packets parameter shall be used to indicate the number of packets currently stored in a packet store.

3.3.4.6 Time Range

The Time Range parameter shall be used to indicate the time of the earliest packet and the time of the latest packet in a packet store.

3.3.4.7 Allocated Space

The Allocated Space parameter shall be used to indicate the number of octets allocated to store packets in a packet store.

3.3.4.8 Free Space

The Free Space parameter shall be used to indicate the number of octets currently not used to store packets in the packet store.

3.3.4.9 Packet List

The Packet List parameter shall contain a list of packets read from or to be written to a packet store.

NOTE – The mechanism by which the number of packets in the list is specified is implementation-dependent.

3.3.4.10 Timestamp

The Timestamp parameter shall be used to indicate the time at which a packet was written to a packet store.

3.3.4.11 Start Time

The Start Time parameter shall be used to indicate the lower limit (i.e., earliest time) of a time range.

3.3.4.12 End Time

The End Time parameter shall be used to indicate the upper limit (i.e., latest time) of a time range.

3.3.4.13 Relative Offset

The Relative Offset parameter shall be use to indicate the number of packets to move the read pointer into a packet store. A positive number moves the read pointer forward in time; a negative number moves the read pointer backwards in time.

3.3.4.14 Packet Selection Criteria

3.3.4.14.1 The Packet Selection Criteria parameter shall be used to indicate the selection criteria by which packets are to be marked as selected.

3.3.4.14.2 This shall be the packet write timestamp (for all packet stores) or any single field in the primary header of the CCSDS packet (only for packet stores containing CCSDS packets).

NOTE – The selection criteria used for a single field is dependent upon the particular field and is implementation-dependent, e.g., an exact match to the APID field.

3.3.4.15 Requested Packet Count

The Requested Packet Count parameter shall be used to indicate the number of packets to be read.

3.4 SERVICE INTERFACE

3.4.1 OVERVIEW

The following subsections define the service interface for the entire FPSS service suite (FAS, FMS, PSAS, and PSMS). The Service interface is defined in terms of the availability and use of primitives and of parameters for each primitive.

3.4.2 FILE ACCESS SERVICE PRIMITIVES

3.4.2.1 General

The File Access Service interface shall implement the following primitives:

a) OPEN_FILE.request, as specified in 3.4.2.2;

b) OPEN_FILE.indication, as specified in 3.4.2.3;

c) CLOSE_FILE.request, as specified in 3.4.2.4;

d) CLOSE_FILE.indication, as specified in 3.4.2.5;

e) READ_FROM_FILE.request, as specified in 3.4.2.6;

f) READ_FROM_FILE.indication, as specified in 3.4.2.7;

g) WRITE_TO_FILE.request, as specified in 3.4.2.8;

h) WRITE_TO_FILE.indication, as specified in 3.4.2.9.

Optionally, the File Access Service may implement the following primitives:

a) FILE_SEEK.request as specified in 3.4.2.10;

b) FILE_SEEK.indication as specified in 3.4.2.11.

3.4.2.2 OPEN_FILE.request

3.4.2.2.1 Function

3.4.2.2.1.1 The **OPEN_FILE.request** shall be passed to the SOIS File Access Service provider to request the opening of an existing file, or (optionally) the creation and then opening of a new file, for access in the file store with the specified opening criteria.

3.4.2.2.1.2 Optionally, the user entity may request as an atomic action the application of a file lock while opening the file.

3.4.2.2.2 Semantics

The **OPEN_FILE.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

OPEN_FILE.request (Transaction Identifier, File Name, File Opening Criteria)

3.4.2.2.3 When Generated

The **OPEN_FILE.request** primitive shall be passed to the SOIS File Access Service provider to request the file to be opened or created and then opened. Optionally, the file may be requested to be locked.

3.4.2.2.4 Effect on Receipt

Receipt of the **OPEN_FILE.request** primitive shall cause the SOIS File Access Service provider to open or create and open the specified file. The user entity's current position within the file is initialized to the start of the file, unless the Append file opening criteria is specified, in which case the user entity's current position within the file is initialized to the end of the file. Optionally, the file may be locked.

3.4.2.2.5 Additional Comments

Although this Recommended Practice does not exclude a file's being open for simultaneous writing by multiple user entities, it is strongly advised that only a single user entity at any one time should have write access.

3.4.2.3 OPEN_FILE.indication

3.4.2.3.1 Function

The **OPEN_FILE.indication** primitive shall be used to indicate the outcome of opening a file to the user entity.

3.4.2.3.2 Semantics

The **OPEN_FILE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

OPEN_FILE.indication (Transaction Identifier, File Identifier, Result Metadata)

3.4.2.3.3 When Generated

The **OPEN_FILE.indication** primitive shall be passed by the service provider to the receiving user entity in response to an OPEN_FILE.request.

NOTE – This primitive:

- contains the File Identifier allocated to the open file (if successfully opened); and
- provides metadata indicating if the request was executed successfully or not.

3.4.2.3.4 Effect on Receipt

The response of the user entity to an **OPEN_FILE.indication** is unspecified.

3.4.2.3.5 Additional Comments

None.

3.4.2.4 CLOSE_FILE.request

3.4.2.4.1 Function

The **CLOSE_FILE.request** primitive shall be passed to the SOIS File Access Service provider to request the closing of the specified file in the specified file store.

3.4.2.4.2 Semantics

The **CLOSE_FILE.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CLOSE_FILE.request (Transaction Identifier, File Identifier)

3.4.2.4.3 When Generated

The **CLOSE_FILE.request** primitive shall be passed to the SOIS File Access Service provider to request the file to be closed.

3.4.2.4.4 Effect on Receipt

Receipt of the **CLOSE_FILE.request** primitive shall cause the SOIS File Access Service provider to close the specified file.

3.4.2.4.5 Additional Comments

None.

3.4.2.5 CLOSE_FILE.indication

3.4.2.5.1 Function

The **CLOSE_FILE.indication** primitive shall be used to indicate the outcome of closing a file to the user entity.

3.4.2.5.2 Semantics

The **CLOSE_FILE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CLOSE_FILE.indication (Transaction Identifier, Result Metadata)

3.4.2.5.3 When Generated

The **CLOSE_FILE.indication** primitive shall be passed by the service provider to the receiving user entity in response to a CLOSE_FILE.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.2.5.4 Effect on Receipt

The response of the user entity to a **CLOSE_FILE.indication** is unspecified.

3.4.2.5.5 Additional Comments

None.

3.4.2.6 READ_FROM_FILE.request

3.4.2.6.1 Function

The **READ_FROM_FILE.request** primitive shall be passed to the SOIS File Access Service provider to request the reading of a segment of specified length starting at the specified offset from the specified file, in the specified file store.

3.4.2.6.2 Semantics

The **READ_FROM_FILE.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

READ_FROM_FILE.request (Transaction Identifier, File Identifier,
File Data Length)

3.4.2.6.3 When Generated

The **READ_FROM_FILE.request** primitive shall be passed to the SOIS File Access Service provider to request that data be read from an open file.

3.4.2.6.4 Effect on Receipt

Receipt of the **READ_FROM_FILE.request** primitive shall cause the SOIS File Access Service provider to read data from the specified file at the user entity's current position. If the user entity's current position is at the end of the file, no data is read. If there is less data than requested from the user entity's current position to the end of the file, only the available data is read. The user entity's current position is subsequently advanced through the file by the length of data read.

3.4.2.6.5 Additional Comments

None.

3.4.2.7 READ_FROM_FILE.indication

3.4.2.7.1 Function

The **READ_FROM_FILE.indication** primitive shall be used to pass the data read from a file to the user entity.

3.4.2.7.2 Semantics

The **READ_FROM_FILE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

READ_FROM_FILE.indication (Transaction Identifier, File Data, File Data Length, Result Metadata)

3.4.2.7.3 When Generated

The **READ_FROM_FILE.indication** primitive shall be passed by the service provider to the receiving user entity in response to a READ_FROM_FILE.request.

NOTE – This primitive:

- contains the data read from the file;
- contains the length of data read from the file; and
- provides metadata indicating if the request was executed successfully or not.

3.4.2.7.4 Effect on Receipt

The response of the user entity to a **READ_FROM_FILE.indication** is unspecified.

3.4.2.7.5 Additional Comments

The File Data Length parameter shall be used to indicate the length of the data actually read from the file (in octets).

3.4.2.8 WRITE_TO_FILE.request

3.4.2.8.1 Function

The **WRITE_TO_FILE.request** primitive shall be passed to the SOIS File Access Service provider to request the writing of a data segment of specified length starting at the specified offset to the specified file in the specified file store.

3.4.2.8.2 Semantics

The **WRITE_TO_FILE.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

WRITE_TO_FILE.request (Transaction Identifier, File Identifier, File Data, File Data Length)

3.4.2.8.3 When Generated

The **WRITE_TO_FILE.request** primitive shall be passed to the SOIS File Access Service provider to request that data be written to an open file at the user entity's current position.

3.4.2.8.4 Effect on Receipt

Receipt of the **WRITE_TO_FILE.request** primitive shall cause the SOIS File Access Service provider to write data to the specified file.

If the user entity's current position is not at the end of the file, the file's original data shall be overwritten with the new data; otherwise, the new data shall be appended to the end of the file. The user entity's current position shall be subsequently advanced through the file by the length of data written.

3.4.2.8.5 Additional Comments

None.

3.4.2.9 WRITE_TO_FILE.indication

3.4.2.9.1 Function

The **WRITE_TO_FILE.indication** primitive shall be used to pass the outcome of writing to a file to the user entity.

3.4.2.9.2 Semantics

The **WRITE_TO_FILE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

WRITE_TO_FILE.indication (Transaction Identifier, File Data Length, Result Metadata)

3.4.2.9.3 When Generated

The **WRITE_TO_FILE.indication** primitive shall be passed by the service provider to the receiving user entity in response to a WRITE_TO_FILE.request.

NOTE – This primitive provides:

- the length of data actually written to the file; and
- metadata indicating if the request was executed successfully or not.

3.4.2.9.4 Effect on Receipt

The response of the user entity to a **WRITE_TO_FILE.indication** is unspecified.

3.4.2.9.5 Additional Comments

The File Data Length parameter shall be used to indicate the length of the data actually written to the file (in octets).

3.4.2.10 FILE_SEEK.request

3.4.2.10.1 Function

The **FILE_SEEK.request** primitive shall be passed to the SOIS File Access Service provider to request repositioning of the user entity's current position in a specified open file to the requested offset from the start of the file.

3.4.2.10.2 Semantics

The **FILE_SEEK.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

FILE_SEEK.request (Transaction Identifier, File Identifier, Requested File Offset)

3.4.2.10.3 When Generated

The **FILE_SEEK.request** primitive shall be passed to the SOIS File Access Service provider to request the reposition the offset of a specified opened file in the file store.

3.4.2.10.4 Effect on Receipt

Receipt of the **FILE_SEEK.request** primitive shall cause the SOIS File Access Service provider to reposition the user entity's current position in a specified open file to the requested offset from the start of the file. If the requested offset is beyond the end of the file, the user entity's current position shall be set to the end of the file and an error shall be returned.

3.4.2.10.5 Additional Comments

None.

3.4.2.11 FILE_SEEK.indication

3.4.2.11.1 Function

The **FILE_SEEK.indication** primitive shall be used to pass the outcome of repositioning the user entity's current position in the specified file to the user entity.

3.4.2.11.2 Semantics

The **FILE_SEEK.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

> **FILE_SEEK.indication** (Transaction Identifier, Resulting File Offset, Result Metadata)

3.4.2.11.3 When Generated

The **FILE_SEEK.indication** primitive shall be passed by the service provider to the receiving user entity in response to a FILE_SEEK.request.

NOTE – This primitive provides:

- the resulting offset of the user entity's current position in the file (this may be different to the requested offset—see 3.4.2.10.4);
- metadata indicating if the request was executed successfully or not.

3.4.2.11.4 Effect on Receipt

The response of the user entity to a **FILE_SEEK.indication** is unspecified.

3.4.2.11.5 Additional Comments

None.

3.4.3 FILE MANAGEMENT SERVICE PRIMITIVES

3.4.3.1 General

3.4.3.1.1 The File Management Service shall implement the following primitives:

a) LIST_DIR.request, as specified in 3.4.3.12;

b) LIST_DIR.indication, as specified in 3.4.3.13;

c) CREATE_FILE.request, as specified in 3.4.3.14;

d) CREATE_FILE.indication, as specified in 3.4.3.15;

e) DELETE_FILE.request, as specified in 3.4.3.16;

f) DELETE_FILE.indication, as specified in 3.4.3.17;

g) COPY_FILE.request, as specified in 3.4.3.18;

h) COPY_FILE.indication, as specified in 3.4.3.19;

i) MOVE_FILE.request, as specified in 3.4.3.20;

j) MOVE_FILE.indication, as specified in 3.4.3.21.

3.4.3.1.2 Optionally, the File Management Service may implement the following primitives:

a) CREATE_DIR.request, as specified in 3.4.3.2;

b) CREATE_DIR.indication, as specified in 3.4.3.3;

c) GET_CURRENT_DIR.request, as specified in 3.4.3.4;

d) GET_CURRENT_DIR.indication, as specified in 3.4.3.5;

e) CHANGE_DIR.request, as specified in 3.4.3.6;

f) CHANGE_DIR.indication, as specified in 3.4.3.7;

g) DELETE_DIR.request, as specified in 3.4.3.8;

h) DELETE_DIR.indication, as specified in 3.4.3.9;

i) RENAME_DIR.request, as specified in 3.4.3.10;

j) RENAME_DIR.indication, as specified in 3.4.3.11;

k) LOCK_FILE.request, as specified in 3.4.3.22;

l) LOCK_FILE.indication, as specified in 3.4.3.23;

m) UNLOCK_FILE.request, as specified in 3.4.3.24;

n) UNLOCK_FILE.indication, as specified in 3.4.3.25;

o) LIST_LOCKED_FILES.request, as specified in 3.4.3.26;

p) LIST_LOCKED_FILES.indication, as specified in 3.4.3.27;

q) FIND_FILE.request, as specified in 3.4.3.28;

r) FIND_FILE.indication, as specified in 3.4.3.29;

s) FILE_STATUS.request as specified in 3.4.3.30;

t) FILE_STATUS.indication as specified in 3.4.3.31.

3.4.3.2 CREATE_DIR.request

3.4.3.2.1 Function

The **CREATE_DIR.request** primitive shall be passed to the SOIS File Management Service provider to request the creation of the specified directory in the file store.

3.4.3.2.2 Semantics

The **CREATE_DIR.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CREATE_DIR.request (Transaction Identifier, Directory Name)

3.4.3.2.3 When Generated

The **CREATE_DIR.request** primitive shall be passed to the SOIS File Management Service provider to request the creation of a directory.

3.4.3.2.4 Effect on Receipt

Receipt of the **CREATE_DIR.request** primitive shall cause the SOIS File Management Service provider to create the specified directory.

3.4.3.2.5 Additional Comments

None.

3.4.3.3 CREATE_DIR.indication

3.4.3.3.1 Function

The **CREATE_DIR.indication** primitive shall be used to pass the outcome of creating a directory to the user entity.

3.4.3.3.2 Semantics

The **CREATE_DIR.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CREATE_DIR.indication (Transaction Identifier, Result Metadata)

3.4.3.3.3 When Generated

The **CREATE_DIR.indication** primitive shall be issued by the service provider to the receiving user entity in response to a CREATE_DIR.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.3.3.4 Effect on Receipt

The response of the user entity to a **CREATE_DIR.indication** is unspecified.

3.4.3.3.5 Additional Comments

None.

3.4.3.4 GET_CURRENT_DIR.request

3.4.3.4.1 Function

The **GET_CURRENT_DIR.request** primitive shall be passed to the SOIS File Management Service provider to get the name (including full path) of the user entity's current directory in the file store.

3.4.3.4.2 Semantics

The **GET_CURRENT_DIR.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

GET_CURRENT_DIR.request (Transaction Identifier)

3.4.3.4.3 When Generated

The **GET_CURRENT_DIR.request** primitive shall be passed to the SOIS File Management Service provider to request the name (including full path) of the user entity's current directory be notified.

3.4.3.4.4 Effect on Receipt

Receipt of the **GET_CURRENT_DIR.request** primitive shall cause the SOIS File Management Service provider to provide the user entity with the name (including full path) of the user entity's current directory.

3.4.3.4.5 Additional Comments

None.

3.4.3.5 GET_CURRENT_DIR.indication

3.4.3.5.1 Function

The **GET_CURRENT_DIR.indication** primitive shall be used to pass the name (including full path) of the user entity's current directory to the user entity.

3.4.3.5.2 Semantics

The **GET_CURRENT_DIR.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

GET_CURRENT_DIR.indication (Transaction Identifier, Directory Name, Result Metadata)

3.4.3.5.3 When Generated

The **GET_CURRENT_DIR.indication** primitive shall be issued by the service provider to the receiving user entity in response to a GET_CURRENT_DIR.request.

NOTE – This primitive:

- contains the name and full path of the user entity's current directory; and
- provides metadata indicating if the request was executed successfully or not.

3.4.3.5.4 Effect on Receipt

The response of the user entity to a **GET_CURRENT_DIR.indication** is unspecified.

3.4.3.5.5 Additional Comments

None.

3.4.3.6 CHANGE_DIR.request

3.4.3.6.1 Function

The **CHANGE_DIR.request** primitive shall be passed to the SOIS File Management Service provider to request the changing of the user entity's current directory in the file store.

3.4.3.6.2 Semantics

The **CHANGE_DIR.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CHANGE_DIR.request (Transaction Identifier, New Directory Name)

3.4.3.6.3 When Generated

The **CHANGE_DIR.request** primitive shall be passed to the SOIS File Management Service provider to request the change of the user entity's current directory.

3.4.3.6.4 Effect on Receipt

Receipt of the **CHANGE_DIR.request** primitive shall cause the SOIS File Management Service provider to change the user entity's current directory.

3.4.3.6.5 Additional Comments

3.4.3.6.5.1 To change directory 'down' the hierarchy, the user entity shall use a New Directory Name parameter of the name of the new directory (implying its existence in the user entity's current directory) or the full path of the new directory (i.e., being equivalent to the full path of the user entity's current directory appended with the new directory's name).

3.4.3.6.5.2 To change directory 'up' the hierarchy, the user entity shall use a New Directory Name parameter of the full path of the new directory (i.e., being equivalent to the full path to the directory 'above' the user entity's current directory) or (optionally) a special character to indicate the directory above the user entity's current directory.

3.4.3.7 CHANGE_DIR.indication

3.4.3.7.1 Function

The **CHANGE_DIR.indication** primitive shall be used to indicate the outcome of changing the user entity's current directory to the user entity.

3.4.3.7.2 Semantics

The **CHANGE_DIR.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CHANGE_DIR.indication (Transaction Identifier, Result Metadata)

3.4.3.7.3 When Generated

The **CHANGE_DIR.indication** primitive shall be passed by the service provider to the receiving user entity in response to a CHANGE_DIR.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.3.7.4 Effect on Receipt

The response of the user entity to a **CHANGE_DIR.indication** is unspecified.

3.4.3.7.5 Additional Comments

None.

3.4.3.8 DELETE_DIR.request

3.4.3.8.1 Function

The **DELETE_DIR.request** primitive shall be passed to the SOIS File Management Service provider to request the removal of a specified directory in the file store.

3.4.3.8.2 Semantics

The **DELETE_DIR.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

DELETE_DIR.request (Transaction Identifier, Directory Name)

3.4.3.8.3 When Generated

The **DELETE_DIR.request** primitive shall be passed to the SOIS File Management Service provider to request the deletion of the specified directory.

3.4.3.8.4 Effect on Receipt

Receipt of the **DELETE_DIR.request** primitive shall cause the SOIS File Management Service provider to delete the specified directory and all the files contained in the directory.

3.4.3.8.5 Additional Comments

3.4.3.8.5.1 The Directory Name parameter shall be unique within a directory and may or may not contain the full path. If no path is provided, the user entity's current directory shall be assumed.

3.4.3.8.5.2 The directory must be completely empty; otherwise the request shall result in an error indication.

3.4.3.9 DELETE_DIR.indication

3.4.3.9.1 Function

The **DELETE_DIR.indication** shall be used to pass the outcome of a remove directory request to the user entity.

3.4.3.9.2 Semantics

The **DELETE_DIR.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

DELETE_DIR.indication (Transaction Identifier, Result Metadata)

3.4.3.9.3 When Generated

The **DELETE_DIR.indication** primitive shall be passed by the service provider to the receiving user entity in response to a DELETE_DIR.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.3.9.4 Effect on Receipt

The response of the user entity to a **DELETE_DIR.indication** is unspecified.

3.4.3.9.5 Additional Comments

None.

3.4.3.10 RENAME_DIR.request

3.4.3.10.1 Function

The **RENAME_DIR.request** primitive shall be passed to the SOIS File Management Service provider to request the renaming of a specified directory in the file store.

NOTE – This primitive does not support moving directories within the file system hierarchy, only renaming a directory within its current directory.

3.4.3.10.2 Semantics

The **RENAME_DIR.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

RENAME_DIR.request (Transaction Identifier,
New Directory Name, Old Directory Name)

3.4.3.10.3 When Generated

The **RENAME_DIR.request** primitive shall be passed to the SOIS File Management Service provider to request the renaming of the specified directory.

3.4.3.10.4 Effect on Receipt

Receipt of the **RENAME_DIR.request** primitive shall cause the SOIS File Management Service provider to rename the specified directory.

3.4.3.10.5 Additional Comments

3.4.3.10.5.1 The New Directory Name parameter shall contain the new name of the directory. It must be unique within the directory in which the directory to be named resides and must NOT contain a path.

3.4.3.10.5.2 The Old Directory Name parameter shall contain the name of the directory to be renamed. It may contain the full path. If no path is provided, it shall be assumed to be a directory in the user entity's current directory.

3.4.3.11 RENAME_DIR.indication

3.4.3.11.1 Function

The **RENAME_DIR.indication** primitive shall be used to pass the outcome of renaming a director to the user entity.

3.4.3.11.2 Semantics

The **RENAME_DIR.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

RENAME_DIR.indication (Transaction Identifier, Result Metadata)

3.4.3.11.3 When Generated

The **RENAME_DIR.indication** primitive shall be passed by the service provider to the receiving user entity in response to a RENAME_DIR.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.3.11.4 Effect on Receipt

The response of the user entity to a **RENAME_DIR.indication** is unspecified.

3.4.3.11.5 Additional Comments

None.

3.4.3.12 LIST_DIR.request

3.4.3.12.1 Function

The **LIST_DIR.request** primitive shall be passed to the SOIS File Management Service provider to request the listing of a content of the user entity's current director or (optionally) a specified directory.

3.4.3.12.2 Semantics

The **LIST_DIR.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

LIST_DIR.request (Transaction Identifier, Directory Name (optional))

3.4.3.12.3 When Generated

The **LIST_DIR.request** primitive shall be passed to the SOIS File Management Service provider to request the list of the specified directory.

3.4.3.12.4 Effect on Receipt

Receipt of the **LIST_DIR.request** primitive shall cause the SOIS File Management Service provider to list the specified directory.

3.4.3.12.5 Additional Comments

3.4.3.12.5.1 If no Directory Name parameter is supplied, the content of the user entity's current directory shall be listed.

3.4.3.12.5.2 The Directory Name parameter shall, if supplied, contain the name of directory to be listed. It may contain the full path. If no path is provided, it shall be assumed to be a directory in the user entity's current directory.

3.4.3.13 LIST_DIR.indication

3.4.3.13.1 Function

The **LIST_DIR.indication** primitive shall be used to pass the directory listing to the user entity.

3.4.3.13.2 Semantics

The **LIST_DIR.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

LIST_DIR.indication (Transaction Identifier, Directory Listing, Result Metadata)

3.4.3.13.3 When Generated

The **LIST_DIR.indication** primitive shall be passed by the service provider to the receiving user entity in response to a LIST_DIR.request.

NOTE – This primitive:

- contains the list of files in a directory; and
- provides metadata indicating if the request was executed successfully or not.

3.4.3.13.4 Effect on Receipt

The response of the user entity to a **LIST_DIR.indication** is unspecified.

3.4.3.13.5 Additional Comments

None.

3.4.3.14 CREATE_FILE.request

3.4.3.14.1 Function

The **CREATE_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request the creation of a new file in the file store.

3.4.3.14.2 Semantics

The **CREATE_DIR.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CREATE_FILE.request (Transaction Identifier, File Name)

3.4.3.14.3 When Generated

The **CREATE_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request the creation of the specified file.

3.4.3.14.4 Effect on Receipt

Receipt of the **CREATE_FILE.request** primitive shall cause the SOIS File Management Service provider to create the specified file.

3.4.3.14.5 Additional Comments

The File Name parameter shall be unique within a directory. It may contain the full file path. If no path is provided, the user entity's current directory shall be assumed.

3.4.3.15 CREATE_FILE.indication

3.4.3.15.1 Function

The **CREATE_FILE.indication** primitive shall be used to pass the outcome of creating a file to the user entity.

3.4.3.15.2 Semantics

The **CREATE_FILE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CREATE_FILE.indication (Transaction Identifier, Result Metadata)

3.4.3.15.3 When Generated

The **CREATE_FILE.indication** primitive shall be passed by the service provider to the receiving user entity in response to a CREATE_FILE.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.3.15.4 Effect on Receipt

The response of the user entity to a **CREATE_FILE.indication** is unspecified.

3.4.3.15.5 Additional Comments

None.

3.4.3.16 DELETE_FILE.request

3.4.3.16.1 Function

The **DELETE_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request deletion of an existing, unopened file in the file store.

3.4.3.16.2 Semantics

The **DELETE_FILE.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

DELETE_FILE.request (Transaction Identifier, File Name)

3.4.3.16.3 When Generated

The **DELETE_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request the deletion of the specified file.

3.4.3.16.4 Effect on Receipt

Receipt of the **DELETE_FILE.request** primitive shall cause the SOIS File Management Service provider to delete the specified file.

3.4.3.16.5 Additional Comments

3.4.3.16.5.1 The File Name parameter shall be unique within a directory. It may contain the full file path. If no path is provided, it shall be assumed to be the user entity's current directory.

3.4.3.16.5.2 The requested delete shall be rejected if the file is currently open.

3.4.3.17 DELETE_FILE.indication

3.4.3.17.1 Function

The **DELETE_FILE.indication** primitive shall be used to pass the outcome of deleting a file to the user entity.

3.4.3.17.2 Semantics

The **DELETE_FILE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

DELETE_FILE.indication (Transaction Identifier, Result Metadata)

3.4.3.17.3 When Generated

The **DELETE_FILE.indication** primitive shall be passed by the service provider to the receiving user entity in response to a DELETE_FILE.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.3.17.4 Effect on Receipt

The response of the user entity to a **DELETE_FILE.indication** is unspecified.

3.4.3.17.5 Additional Comments

None.

3.4.3.18 COPY_FILE.request

3.4.3.18.1 Function

The **COPY_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request the copying of an existing specified file from one specified directory to another, within the same file store.

3.4.3.18.2 Semantics

The **COPY_FILE.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

COPY_FILE.request (Transaction Identifier, Source File Name, Destination File Name)

3.4.3.18.3 When Generated

The **COPY_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request the copy of the specified file.

3.4.3.18.4 Effect on Receipt

Receipt of the **COPY_FILE.request** primitive shall cause the SOIS File Management Service provider to copy the specified file.

3.4.3.18.5 Additional Comments

3.4.3.18.5.1 The Source File Name parameter shall specify the name of the file to be copied. It shall be unique within a directory. If no path is provided, the user entity's current directory shall be assumed.

3.4.3.18.5.2 The Destination File Name shall specify the name of the copied file. It shall be unique within a directory. If no path is provided, the user entity's current directory shall be assumed. Otherwise, the path shall identify an existing directory.

3.4.3.18.5.3 The requested copy shall be rejected if the Destination File Name contains a path to a directory that does not exist.

3.4.3.18.5.4 The requested copy shall be rejected if the Source File Name and the Destination File Name are the same.

3.4.3.19 COPY_FILE.indication

3.4.3.19.1 Function

The **COPY_FILE.indication** primitive shall be used to pass the outcome of copying a file to the user entity.

3.4.3.19.2 Semantics

The **COPY_FILE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

COPY_FILE.indication (Transaction Identifier, Result Metadata)

3.4.3.19.3 When Generated

The **COPY_FILE.indication** primitive shall be passed by the service provider to the receiving user entity in response to a COPY_FILE.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.3.19.4 Effect on Receipt

The response of the user entity to a **COPY_FILE.indication** is unspecified.

3.4.3.19.5 Additional Comments

None.

3.4.3.20 MOVE_FILE.request

3.4.3.20.1 Function

The **MOVE_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request the moving of an existing specified file within the same file store.

NOTE – The process of moving a file implies that, in case of success, the original file is deleted from its original location.

3.4.3.20.2 Semantics

The **MOVE_FILE.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

MOVE_FILE.request (Transaction Identifier, Source File Name, Destination File Name)

3.4.3.20.3 When Generated

The **MOVE_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request the move of the specified file.

3.4.3.20.4 Effect on Receipt

Receipt of the **MOVE_FILE.request** primitive shall cause the SOIS File Management Service provider to move the specified file.

3.4.3.20.5 Additional Comments

3.4.3.20.5.1 The Source File Name shall specify the name of the file to be moved. It shall be unique within a directory. If no path is provided, the user entity's current directory shall be assumed.

3.4.3.20.5.2 The Destination File Path shall specify the name of the moved file. It shall be unique within a directory. If no path is provided, the user entity's current directory shall be assumed. Otherwise, the path must identify an existing directory.

3.4.3.20.5.3 The requested move shall fail if the Source File Name and the Destination File Name are the same.

3.4.3.21 MOVE_FILE.indication

3.4.3.21.1 Function

The **MOVE_FILE.indication** primitive shall be used to pass the outcome of moving a file to the user entity.

3.4.3.21.2 Semantics

The **MOVE_FILE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

MOVE_FILE.indication (Transaction Identifier, Result Metadata)

3.4.3.21.3 When Generated

The **MOVE_FILE.indication** primitive shall be passed by the service provider to the receiving user entity in response to a MOVE_FILE.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.3.21.4 Effect on Receipt

The response of the user entity to a **MOVE_FILE.indication** is unspecified.

3.4.3.21.5 Additional Comments

None.

3.4.3.22 LOCK_FILE.request

3.4.3.22.1 Function

The **LOCK_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request the locking of an existing, unopened file in the file store.

3.4.3.22.2 Semantics

The **LOCK_FILE.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

LOCK_FILE.request (Transaction Identifier, File Name, Lock Type)

3.4.3.22.3 When Generated

The **LOCK_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request the lock of the specified file.

3.4.3.22.4 Effect on Receipt

Receipt of the **LOCK_FILE.request** primitive shall cause the SOIS File Management Service provider to lock the specified file (if not already open).

3.4.3.22.5 Additional Comments

The requested locking shall be rejected if the file is currently open.

3.4.3.23 LOCK_FILE.indication

3.4.3.23.1 Function

The **LOCK_FILE.indication** primitive shall be used to pass the outcome of locking a file to the user entity.

3.4.3.23.2 Semantics

The **LOCK_FILE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

LOCK_FILE.indication (Transaction Identifier, File Lock Identifier, Result Metadata)

3.4.3.23.3 When Generated

The **LOCK_FILE.indication** primitive shall be passed by the service provider to the receiving user entity in response to a LOCK_FILE.request.

NOTE – This primitive:

- contains the identifier of the lock on the file; and
- provides metadata indicating if the request was executed successfully or not.

3.4.3.23.4 Effect on Receipt

The response of the user entity to a **LOCK_FILE.indication** is unspecified.

3.4.3.23.5 Additional Comments

None.

3.4.3.24 UNLOCK_FILE.request

3.4.3.24.1 Function

The **UNLOCK_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request the unlocking of a locked file in the file store.

3.4.3.24.2 Semantics

The **UNLOCK_FILE.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

UNLOCK_FILE.request (Transaction Identifier, File Lock Identifier)

3.4.3.24.3 When Generated

The **UNLOCK_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request the unlocking of the specified file.

3.4.3.24.4 Effect on Receipt

Receipt of the **UNLOCK_FILE.request** primitive shall cause the SOIS File Management Service provider to unlock the specified file.

3.4.3.24.5 Additional Comments

No restrictions are placed on what user entities may unlock a file, e.g., other than the lock-owner.

3.4.3.25 UNLOCK_FILE.indication

3.4.3.25.1 Function

The **UNLOCK_FILE.indication** primitive shall be used to pass the outcome of unlocking a file to the user entity.

3.4.3.25.2 Semantics

The **UNLOCK_FILE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

UNLOCK_FILE.indication (Transaction Identifier, Result Metadata)

3.4.3.25.3 When Generated

The **UNLOCK_FILE.indication** primitive shall be passed by the service provider to the receiving user entity in response to an UNLOCK_FILE.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.3.25.4 Effect on Receipt

The response of the user entity to an **UNLOCK_FILE.indication** is unspecified.

3.4.3.25.5 Additional Comments

None.

3.4.3.26 LIST_LOCKED_FILES.request

3.4.3.26.1 Function

The **LIST_LOCKED_FILES.request** primitive shall be passed to the SOIS File Management Service provider to request the listing of all locked files in the file store.

3.4.3.26.2 Semantics

The **LIST_LOCKED_FILES.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

LIST_LOCKED_FILES.request (Transaction Identifier)

3.4.3.26.3 When Generated

The **LIST_LOCKED_FILES.request** primitive shall be passed to the SOIS File Management Service provider to request the listing of all locked files in the file store.

3.4.3.26.4 Effect on Receipt

Receipt of the **LIST_LOCKED_FILES.request** primitive shall cause the SOIS File Management Service provider to list all locked files in the file store.

3.4.3.26.5 Additional Comments

None.

3.4.3.27 LIST_LOCKED_FILES.indication

3.4.3.27.1 Function

The **LIST_LOCKED_FILES.indication** shall be used to pass the list of all locked files in the file store to the user entity.

3.4.3.27.2 Semantics

The **LIST_LOCKED_FILES.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

LIST_LOCKED_FILES.indication (Transaction Identifier, File Lock List, Result Metadata)

3.4.3.27.3 When Generated

The **LIST_LOCKED_FILES.indication** primitive shall be passed by the service provider to the receiving user entity in response to a LIST_LOCKED_FILES.request.

NOTE – This primitive:

- contains the list of locked files; and
- provides metadata indicating if the request was executed successfully or not.

3.4.3.27.4 Effect on Receipt

The response of the user entity to a **LIST_LOCKED_FILES.indication** is unspecified.

3.4.3.27.5 Additional Comments

None.

3.4.3.28 FIND_FILE.request

3.4.3.28.1 Function

The **FIND_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request the finding of a specified file in the file store matching a selection pattern on one of its attributes.

3.4.3.28.2 Semantics

The **FIND_FILE.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

FIND_FILE.request (Transaction Identifier, File Selection Pattern)

3.4.3.28.3 When Generated

The **FIND_FILE.request** primitive shall be passed to the SOIS File Management Service provider to request the find operation of the specified file(s) matching the selection pattern.

3.4.3.28.4 Effect on Receipt

Receipt of the **FIND_FILE.request** primitive shall cause the SOIS File Management Service provider to find the specified file(s) matching the selection pattern.

3.4.3.28.5 Additional Comments

Finding a file in a hierarchical file store shall result in a search through all ‘branches’ of the ‘tree-like’ hierarchy.

3.4.3.29 FIND_FILE.indication

3.4.3.29.1 Function

The **FIND_FILE.indication** primitive shall be used to pass the list of file names matching the file selection pattern to the user entity.

3.4.3.29.2 Semantics

The **FIND_FILE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

FIND_FILE.indication (Transaction Identifier, File Names List, Result Metadata)

3.4.3.29.3 When Generated

The **FIND_FILE.indication** primitive shall be passed by the service provider to the receiving user entity in response to a FIND_FILE.request.

NOTE – This primitive:

- contains the list of file names found to be matching the selection pattern; and
- provides metadata indicating if the request was executed successfully or not.

3.4.3.29.4 Effect on Receipt

The response of the user entity to a **FIND_FILE.indication** is unspecified.

3.4.3.29.5 Additional Comments

The File Names List shall contain the list of the found files. Each file name shall contain the full file path.

3.4.3.30 FILE_STATUS.request

3.4.3.30.1 Function

The **FILE_STATUS.request** primitive shall be passed to the SOIS File Management Service provider to request the status of a specified file in the file store.

3.4.3.30.2 Semantics

The **FILE_STATUS.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

FILE_STATUS.request (Transaction Identifier, File Name)

3.4.3.30.3 When Generated

The **FILE_STATUS.request** primitive shall be passed to the SOIS File Management Service provider to request status of a specified file in the file store.

3.4.3.30.4 Effect on Receipt

Receipt of the **FILE_STATUS.request** primitive shall cause the SOIS File Management Service provider to get the status of a specified file in the file store.

3.4.3.30.5 Additional Comments

None.

3.4.3.31 FILE_STATUS.indication

3.4.3.31.1 Function

The **FILE_STATUS.indication** primitive shall be used to pass the status of the specified file to the user entity.

3.4.3.31.2 Semantics

The **FILE_STATUS.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

FILE_STATUS.indication (Transaction Identifier, File Attributes, Result Metadata)

3.4.3.31.3 When Generated

The **FILE_STATUS.indication** primitive shall be passed by the service provider to the receiving user entity in response to a FILE_STATUS.request.

NOTE – This primitive:

- provides a parameter that contain the attributes of the file;
- provides metadata indicating if the request was executed successfully or not.

3.4.3.31.4 Effect on Receipt

The response of the user entity to a **FILE_STATUS.indication** is unspecified.

3.4.3.31.5 Additional Comments

None.

3.4.4 PACKET STORE ACCESS SERVICE PRIMITIVES

3.4.4.1 General

3.4.4.1.1 The Packet Store Access Service interface shall implement the following primitives:

a) GET_PACKET_STORES_INFO.request, as specified in 3.4.4.2;

b) GET_PACKET_STORES_INFO.indication, as specified in 3.4.4.3;

c) CLEAR_PACKET_STORE.request, as specified in 3.4.4.4;

d) CLEAR_PACKET_STORE.indication, as specified in 3.4.4.5;

e) WRITE_PACKETS.request, as specified in 3.4.4.6;

f) WRITE_PACKETS.indication, as specified in 3.4.4.7;

g) READ_PACKETS.request, as specified in 3.4.4.8;

h) READ_PACKETS.indication, as specified in 3.4.4.9;

i) SET_PACKET_STORE_POSITION.request, as specified in 3.4.4.10;

j) SET_PACKET_STORE_POSITION.indication, as specified in 3.4.4.11;

k) FREE_PACKETS.request, as specified in 3.4.4.12;

l) FREE_PACKETS.indication, as specified in 3.4.4.13;

m) PACKET_STORE_STATUS.request, as specified in 3.4.4.14;

n) PACKET_STORE_STATUS.indication, as specified in 3.4.4.15.

3.4.4.1.2 Optionally, the Packet Store Access Service interface may implement the following primitives:

a) DUMP_PACKETS.request, as specified in 3.4.4.16;

b) DUMP_PACKETS.indication, as specified in 3.4.4.17;

c) DUMP_PACKETS_COMPLETED.indication, as specified in 3.4.4.18;

d) CREATE_SELECTION.request, as specified in 3.4.4.19;

e) CREATE_SELECTION.indication, as specified in 3.4.4.20.;

f) SELECTIVE_READ_PACKETS.request, as specified in 3.4.4.21;

g) SELECTIVE_READ_PACKETS.indication, as specified in 3.4.4.22.;

h) SELECTIVE_FREE_PACKETS.request, as specified in 3.4.4.23;

i) SELECTIVE_FREE_PACKETS.indication, as specified in 3.4.4.24;

j) SELECTIVE_DUMP_PACKETS.request, as specified in 3.4.4.25;

k) SELECTIVE_DUMP_PACKETS.indication, as specified in 3.4.4.26.

l) SELECTIVE_DUMP_PACKETS_COMPLETED.indication, as specified in 3.4.4.27.

3.4.4.2 GET_PACKET_STORES_INFO.request

3.4.4.2.1 Function

The **GET_PACKET_STORES_INFO.request** primitive shall be passed to the SOIS Packet Store Access Service to request a description of all of the available packet stores, with each description consisting of:

a) store identifier;

b) type of packet store.

3.4.4.2.2 Semantics

The **GET_PACKET_STORES_INFO.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

GET_PACKET_STORES_INFO.request (Transaction Identifier)

3.4.4.2.3 When Generated

The **GET_PACKET_STORES_INFO.request** primitive shall be passed to the SOIS Packet Store Access Service provider to request the description of all of the available packet stores.

3.4.4.2.4 Effect on Receipt

Receipt of the **GET_PACKET_STORES_INFO.request** primitive shall cause the SOIS Packet Store Access Service provider to return the description of all of the available packet stores.

3.4.4.2.5 Additional Comments

None.

3.4.4.3 GET_PACKET_STORES_INFO.indication

3.4.4.3.1 Function

The **GET_PACKET_STORES_INFO.indication** primitive shall be used to pass a description of all available packet stores to the user entity.

3.4.4.3.2 Semantics

The **GET_PACKET_STORES_INFO.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

GET_PACKET_STORES_INFO.indication (Transaction Identifier,
Packet Store Description List,
Result Metadata)

3.4.4.3.3 When Generated

The **GET_PACKET_STORES_INFO.indication** primitive shall be passed by the service provider to the receiving user entity in response to a GET_PACKET_STORES_INFO.request.

NOTE – This primitive:

- contains the list of packet store descriptions; and
- provides metadata indicating if the request was executed successfully or not.

3.4.4.3.4 Effect on Receipt

The response of the user entity to a **GET_PACKET_STORES_INFO.indication** primitive is unspecified.

3.4.4.3.5 Additional Comments

None.

3.4.4.4 CLEAR_PACKET_STORE.request

3.4.4.4.1 Function

The **CLEAR_PACKET_STORE.request** primitive shall be passed to the SOIS Packet Store Access Service to request the resetting of the specified packet store to its initial configuration with no packets stored.

3.4.4.4.2 Semantics

The **CLEAR_PACKET_STORE.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CLEAR_PACKET_STORE.request (Transaction Identifier, Packet Store Identifier)

3.4.4.4.3 When Generated

The **CLEAR_PACKET_STORE.request** primitive shall be passed to the SOIS Packet Store Access Service provider to request the reset of the specified Packet Store.

3.4.4.4.4 Effect on Receipt

Receipt of the **CLEAR_PACKET_STORE.request** primitive shall cause the SOIS Packet Store Access Service provider to reset the specified Packet Store.

3.4.4.4.5 Additional Comments

All packets in the packet store shall be deleted.

3.4.4.5 CLEAR_PACKET_STORE.indication

3.4.4.5.1 Function

The **CLEAR_PACKET_STORE.indication** primitive shall be used to pass the outcome of resetting a packet store to the user entity.

3.4.4.5.2 Semantics

The **CLEAR_PACKET_STORE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CLEAR_PACKET_STORE.indication (Transaction Identifier, Result Metadata)

3.4.4.5.3 When Generated

The **CLEAR_PACKET_STORE.indication** primitive shall be passed by the service provider to the receiving user entity in response to a CLEAR_PACKET_STORE.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.4.5.4 Effect on Receipt

The response of the user entity to a **CLEAR_PACKET_STORE.indication** primitive is unspecified.

3.4.4.5.5 Additional Comments

None.

3.4.4.6 WRITE_PACKETS.request

3.4.4.6.1 Function

The **WRITE_PACKETS.request** primitive shall be passed to the SOIS Packet Store Access Service to request the writing of a packet into the specified packet store.

3.4.4.6.2 Semantics

The **WRITE_PACKETS.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

WRITE_PACKETS.request (Transaction Identifier, Packet Store Identifier, Packet List)

3.4.4.6.3 When Generated

The **WRITE_PACKETS.request** primitive shall be passed to the SOIS Packet Store Access Service provider to request the writing of a packet to the specified Packet Store.

3.4.4.6.4 Effect on Receipt

Receipt of the **WRITE_PACKETS.request** primitive shall cause the SOIS Packet Store Access Service provider to write the provided packet to the Packet Store.

3.4.4.6.5 Additional Comments

3.4.4.6.5.1 For bounded packet stores, if the store contains its maximum number of packets, the request shall fail.

3.4.4.6.5.2 For circular packet stores, if the store contains its maximum number of packets, the oldest packets shall be freed to make room for the written packets.

3.4.4.7 WRITE_PACKETS.indication

3.4.4.7.1 Function

The **WRITE_PACKETS.indication** primitive shall be used to pass the outcome of writing a packet to a packet store to the user entity.

3.4.4.7.2 Semantics

The **WRITE_PACKETS.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

WRITE_PACKETS.indication (Transaction Identifier, Result Metadata)

3.4.4.7.3 When Generated

The **WRITE_PACKETS.indication** primitive shall be passed by the service provider to the receiving user entity in response to a WRITE_PACKETS.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.4.7.4 Effect on Receipt

The response of the user entity to a **WRITE_PACKETS.indication** primitive is unspecified.

3.4.4.7.5 Additional Comments

None.

3.4.4.8 READ_PACKETS.request

The **READ_PACKETS.request** primitive shall be used to request the reading of the requested number of packets from the specified packet store after and including the read pointer.

3.4.4.8.1 Semantics

The **READ_PACKETS.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

READ_PACKETS.request (Transaction Identifier, Packet Store Identifier,
Requested Packet Count)

3.4.4.8.2 When Generated

The **READ_PACKETS.request** primitive shall be passed to the SOIS Packet Store Access Service provider to request the reading of the requested number of packets from the specified Packet Store after and including the read pointer.

3.4.4.8.3 Effect on Receipt

Receipt of the **READ_PACKETS.request** primitive shall cause the SOIS Packet Store Access Service provider to read the requested number of packets after and including the read pointer. The read pointer is moved forward by the requested number of packets.

3.4.4.8.4 Additional Comments

Although this Recommended Practice does not exclude multiple user entities' reading from the packet store, it is strongly advised that only a single user entity should read from the packet store.

3.4.4.9 READ_PACKETS.indication

3.4.4.9.1 Function

The **READ_PACKETS.indication** primitive shall be used to pass packets read from the packet store to the user entity.

3.4.4.9.2 Semantics

The **READ_PACKETS.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

READ_PACKETS.indication (Transaction Identifier, Packet List, Result Metadata)

3.4.4.9.3 When Generated

The **READ_PACKETS.indication** primitive shall be passed by the service provider to the receiving user entity in response to a READ_PACKETS.request.

NOTE – This primitive:

- contains the list of packets read from the packet store; and
- provides metadata indicating if the request was executed successfully or not.

3.4.4.9.4 Effect on Receipt

The response of the user entity to a **READ_PACKETS.indication** primitive is unspecified.

3.4.4.9.5 Additional Comments

Where there are fewer packets in the packet store than requested to be read, the Result Metadata parameter shall indicate this and the Packet List parameter shall contain all the packets in the store after and including the read pointer.

3.4.4.10 SET_PACKET_STORE_POSITION.request

3.4.4.10.1 Function

The **SET_PACKET_STORE_POSITION.request** primitive shall be passed to the SOIS Packet Store Access Service to request the setting of the read pointer of the packet store forward or backwards by a specified number of packets. The read pointer shall stay within the limits of the write pointer and the last freed packet.

3.4.4.10.2 Semantics

The **SET_PACKET_STORE_POSITION.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

SET_PACKET_STORE_POSITION.request (Transaction Identifier,
Packet Store Identifier, Relative
Offset)

3.4.4.10.3 When Generated

The **SET_PACKET_STORE_POSITION.request** primitive shall be passed to the SOIS Packet Store Access Service provider to request the setting of the read pointer of the specified Packet Store.

3.4.4.10.4 Effect on Receipt

Receipt of the **SET_PACKET_STORE_POSITION.request** primitive shall cause the SOIS Packet Store Access Service provider to set the read pointer of the Packet Store.

3.4.4.10.5 Additional Comments

None.

3.4.4.11 SET_PACKET_STORE_POSITION.indication

3.4.4.11.1 Function

The **SET_PACKET_STORE_POSITION.indication** primitive shall be used to pass the outcome of setting the read pointer in a packet store to the user entity.

3.4.4.11.2 Semantics

The **SET_PACKET_STORE_POSITION.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

SET_PACKET_STORE_POSITION.indication (Transaction Identifier,
Result Metadata)

3.4.4.11.3 When Generated

The **SET_PACKET_STORE_POSITION.indication** primitive shall be passed by the service provider to the receiving user entity in response to a **SET_PACKET_STORE_POSITION.request**.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.4.11.4 Effect on Receipt

The response of the user entity to a **SET_PACKET_STORE_POSITION.indication** primitive is unspecified.

3.4.4.11.5 Additional Comments

None.

3.4.4.12 FREE_PACKETS.request

3.4.4.12.1 Function

The **FREE_PACKETS.request** primitive shall be passed to the SOIS Packet Store Access Service to request the freeing, i.e., deleting, of the oldest packets from the specified packet store.

3.4.4.12.2 Semantics

The **FREE_PACKETS.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

FREE_PACKETS.request (PSASAP, Transaction Identifier,
Packet Store Identifier, Requested Packet Count)

3.4.4.12.3 When Generated

The **FREE_PACKETS.request** primitive shall be passed to the SOIS Packet Store Access Service provider to request to delete the oldest packets from the specified Packet Store.

3.4.4.12.4 Effect on Receipt

Receipt of the **FREE_PACKETS.request** primitive shall cause the SOIS Packet Store Access Service provider to free the oldest packets from the specified Packet Store.

3.4.4.12.5 Additional Comments

None.

3.4.4.13 FREE_PACKETS.indication

3.4.4.13.1 Function

The **FREE_PACKETS.indication** primitive shall be used to pass the outcome of freeing packets from a packet store to the user entity.

3.4.4.13.2 Semantics

The **FREE_PACKETS.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

FREE_PACKETS.indication (PSASAP, Transaction Identifier, Result Metadata)

3.4.4.13.3 When Generated

The **FREE_PACKETS.indication** primitive shall be passed by the service provider to the receiving user entity in response to a FREE_PACKETS.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.4.13.4 Effect on Receipt

The response of the user entity to a **FREE_PACKETS.indication** primitive is unspecified.

3.4.4.13.5 Additional Comments

None.

3.4.4.14 PACKET_STORE_STATUS.request

3.4.4.14.1 Function

The **PACKET_STORE_STATUS.request** primitive shall be passed to the SOIS Packet Store Access Service to request the status of the specified packet store.

3.4.4.14.2 Semantics

The **PACKET_STORE_STATUS.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

PACKET_STORE_STATUS.request (Transaction Identifier, Packet Store Identifier)

3.4.4.14.3 When Generated

The **PACKET_STORE_STATUS.request** primitive shall be passed to the SOIS Packet Store Access Service provider to request the status report of the specified Packet Store.

3.4.4.14.4 Effect on Receipt

Receipt of the **PACKET_STORE_STATUS.request** primitive shall cause the SOIS Packet Store Access Service provider to provide the status report of the specified Packet Store.

3.4.4.14.5 Additional Comments

None.

3.4.4.15 PACKET_STORE_STATUS.indication

3.4.4.15.1 Function

The **PACKET_STORE_STATUS.indication** shall be used to pass the status report for a packet store to the user entity.

3.4.4.15.2 Semantics

The **PACKET_STORE_STATUS.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

PACKET_STORE_STATUS.indication (Transaction Identifier, Packet Store Type,
Number of Stored Packets,
Time Range (optional), Allocated Space,
Free Space, Result Metadata)

3.4.4.15.3 When Generated

The **PACKET_STORE_STATUS.indication** primitive shall be passed by the service provider to the receiving user entity in response to a PACKET_STORE_STATUS.request.

NOTE – This primitive:

- contains the status report for the packet store; and
- provides metadata indicating if the request was executed successfully or not.

3.4.4.15.4 Effect on Receipt

The response of the user entity to a **PACKET_STORE_STATUS.indication** primitive is unspecified.

3.4.4.15.5 Additional Comments

The Time Range parameter is used in conjunction with the selective seek, read, and free functionality.

3.4.4.16 DUMP_PACKETS.request

3.4.4.16.1 Function

The **DUMP_PACKETS.request** primitive shall be passed to the SOIS Packet Store Access Service to request the reading of the first unread packet or set of packets from the packet store and the direct transmission of the read packet(s) to the spacecraft telemetry chain over a private interface, for transfer to the ground.

3.4.4.16.2 Semantics

The **DUMP_PACKETS.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

DUMP_PACKETS.request (Transaction Identifier, Packet Store Identifier,
Requested Packet Count)

3.4.4.16.3 When Generated

The **DUMP_PACKETS.request** primitive shall be passed to the SOIS Packet Store Access Service provider to request the reading of the first unread packet or set of packets in the specified packet store and direct transmission of the read packet(s) to the spacecraft telemetry chain.

3.4.4.16.4 Effect on Receipt

Receipt of the **DUMP_PACKETS.request** primitive shall cause the SOIS Packet Store Access Service provider to read the first unread packet or set of packets in the specified packet store and directly transmit the read packet(s) to the spacecraft telemetry chain. The number of packets read and transmitted shall be in accordance with the Requested Packet Count parameter.

3.4.4.16.5 Additional Comments

3.4.4.16.5.1 This primitive is intended to provide dumping of packets to ground via the spacecraft telemetry chain with minimum user entity involvement.

3.4.4.16.5.2 The service requires access to the spacecraft telemetry chain to enable the transmission of packets to the ground. The manner in which this access is provided is implementation-specific.

3.4.4.16.5.3 The mechanisms by which the spacecraft telemetry chain transmits the packet to ground are not specified by this Recommended Practice. They may or may not provide guaranteed delivery.

3.4.4.17 DUMP_PACKETS.indication

3.4.4.17.1 Function

The **DUMP_PACKETS.indication** shall be used to indicate to the user entity the result of initiating a packet dump as a result of the DUMP_PACKETS.request.

3.4.4.17.2 Semantics

The **DUMP_PACKETS.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

DUMP_PACKETS.indication (Transaction Identifier, Result Metadata)

3.4.4.17.3 When Generated

The **DUMP_PACKETS.indication** primitive shall be passed by the service provider to the receiving user entity in response to a dump's being initiated by a DUMP_PACKETS.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.4.17.4 Effect on Receipt

The response of the user entity to a **DUMP_PACKETS.indication** primitive is unspecified.

3.4.4.17.5 Additional Comments

None.

3.4.4.18 DUMP_PACKETS_COMPLETED.indication

3.4.4.18.1 Function

The **DUMP_PACKETS_COMPLETED.indication** shall be used to indicate to the user entity the completion a packet dump initiated by a DUMP_PACKETS.request.

3.4.4.18.2 Semantics

The **DUMP_PACKETS_COMPLETED.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

DUMP_PACKETS_COMPLETED.indication (Transaction Identifier, Dumped Packet Count, Result Metadata)

3.4.4.18.3 When Generated

The **DUMP_PACKETS_COMPLETED.indication** primitive shall be passed by the service provider to the receiving user entity upon completion of a dump initiated by a DUMP_PACKETS.request.

NOTE – This primitive provides:

- the number of packets actually dumped;
- metadata indicating if the dump was executed successfully or not.

3.4.4.18.4 Effect on Receipt

The response of the user entity to a **DUMP_PACKETS_COMPLETED.indication** primitive is unspecified.

3.4.4.18.5 Additional Comments

None.

3.4.4.19 CREATE_SELECTION.request

3.4.4.19.1 Function

The **CREATE_SELECTION.request** primitive shall be passed to the SOIS Packet Store Access Service to request the selection of a set of packets in the specified packet store with a specified selection criterion.

3.4.4.19.2 Semantics

The **CREATE_SELECTION.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CREATE_SELECTION.request (Transaction Identifier, Packet Store Identifier, Packet Selection Criteria)

3.4.4.19.3 When Generated

The **CREATE_SELECTION.request** primitive shall be passed to the SOIS Packet Store Access Service provider to request the selection of a set of packets in the specified packet store with a specified selection criterion.

3.4.4.19.4 Effect on Receipt

Receipt of the **CREATE_SELECTION.request** primitive shall cause the SOIS Packet Store Access Service provider to create a selection of a set of packets in the specified packet store with the specified selection criterion.

3.4.4.19.5 Additional Comments

3.4.4.19.5.1 This primitive shall only be used on packet stores of Random Access type.

3.4.4.19.5.2 This primitive shall not be used with a Packet Selection Criteria using a single field in the primary header of the CCSDS packet on packet stores flagged as containing a packet type of non-CCSDS packets.

3.4.4.19.5.3 Although this Recommended Practice does not exclude multiple user entities' using packet selections, it is strongly advised that only a single user entity at any one time should use packet selections.

3.4.4.20 CREATE_SELECTION.indication

3.4.4.20.1 Function

The **CREATE_SELECTION.indication** shall be used to pass the outcome of a selective seek on a packet store to the user entity.

3.4.4.20.2 Semantics

The **CREATE_SELECTION.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CREATE_SELECTION.indication (Transaction Identifier, Selected Packet Count, Result Metadata)

3.4.4.20.3 When Generated

The **CREATE_SELECTION.indication** primitive shall be passed by the service provider to the receiving user entity in response to a CREATE_SELECTION.request.

NOTE – This primitive

- contains the number of packets selected;
- provides metadata indicating if the request was executed successfully or not.

3.4.4.20.4 Effect on Receipt

The response of the user entity to a **CREATE_SELECTION.indication** primitive is unspecified.

3.4.4.20.5 Additional Comments

None.

3.4.4.21 SELECTIVE_READ_PACKETS.request

3.4.4.21.1 Function

The **SELECTIVE_READ_PACKETS.request** primitive shall be passed to the SOIS Packet Store Access Service to request the reading of the first unread packet or list of packets in the current selection for the specified packet store.

3.4.4.21.2 Semantics

The **SELECTIVE_READ_PACKETS.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

SELECTIVE_READ_PACKETS.request (Transaction Identifier,
Packet Store Identifier,
Requested Packet Count)

3.4.4.21.3 When Generated

The **SELECTIVE_READ_PACKETS.request** primitive shall be passed to the SOIS Packet Store Access Service provider to request the read of the first unread packet or list of packets in the current selection criteria for the specified packet store.

3.4.4.21.4 Effect on Receipt

Receipt of the **SELECTIVE_READ_PACKETS.request** primitive shall cause the SOIS Packet Store Access Service provider to read the first unread packet or list of packets in the current selection criteria for the specified packet store.

3.4.4.21.5 Additional Comments

3.4.4.21.5.1 This primitive shall only be used on packet stores of Random Access type.

3.4.4.21.5.2 This primitive differs from the READ_PACKETS.request in that the packets to be read are first preselected using the CREATE_SELECTION primitive.

3.4.4.22 SELECTIVE_READ_PACKETS.indication

3.4.4.22.1 Function

The **SELECTIVE_READ_PACKETS.indication** shall be used to pass the packets selectively read from a packet store to the user entity.

3.4.4.22.2 Semantics

The **SELECTIVE_READ_PACKETS.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

SELECTIVE_READ_PACKETS.indication (Transaction Identifier, Packet List, Result Metadata)

3.4.4.22.3 When Generated

The **SELECTIVE_READ_PACKETS.indication** primitive shall be passed by the service provider to the receiving user entity in response to a SELECTIVE_READ_PACKETS.request.

NOTE – This primitive:

- contains the list of packets selectively read from the packet store; and
- provides metadata indicating if the request was executed successfully or not.

3.4.4.22.4 Effect on Receipt

The response of the user entity to a **SELECTIVE_READ_PACKETS.indication** primitive is unspecified.

3.4.4.22.5 Additional Comments

None.

3.4.4.23 SELECTIVE_FREE_PACKETS.request

3.4.4.23.1 Function

The **SELECTIVE_FREE_PACKETS.request** primitive shall be passed to the SOIS Packet Store Access Service to request the freeing of the oldest packet or list of packets for the current selection criteria from the specified packet store.

3.4.4.23.2 Semantics

The **SELECTIVE_FREE_PACKETS.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

SELECTIVE_FREE_PACKETS.request (Transaction Identifier,
Packet Store Identifier,
Requested Packet Count)

3.4.4.23.3 When Generated

The **SELECTIVE_FREE_PACKETS.request** primitive shall be passed to the SOIS Packet Store Access Service provider to request the freeing of the oldest packet or list of packets for the current selection criteria from the specified packet store.

3.4.4.23.4 Effect on Receipt

Receipt of the **SELECTIVE_FREE_PACKETS.request** primitive shall cause the SOIS Packet Store Access Service provider to free the oldest packet or list of packets for the current selection criteria from the specified packet store.

3.4.4.23.5 Additional Comments

3.4.4.23.5.1 This primitive shall only be used on packet stores of Random Access type.

3.4.4.23.5.2 This primitive differs from the FREE_PACKETS.request in that the packets to be freed are first preselected using the CREATE_SELECTION primitive.

3.4.4.24 SELECTIVE_FREE_PACKETS.indication

3.4.4.24.1 Function

The **SELECTIVE_FREE_PACKETS.indication** primitive shall be used to pass the outcome of selectively freeing packets from a packet store to the user entity.

3.4.4.24.2 Semantics

The **SELECTIVE_FREE_PACKETS.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

SELECTIVE_FREE_PACKETS.indication (Transaction Identifier,
Result Metadata)

3.4.4.24.3 When Generated

The **SELECTIVE_FREE_PACKETS.indication** primitive shall be passed by the service provider to the receiving user entity in response to a SELECTIVE_FREE_PACKETS.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.4.24.4 Effect on Receipt

The response of the user entity to a **SELECTIVE_FREE_PACKETS.indication** primitive is unspecified.

3.4.4.24.5 Additional Comments

None.

3.4.4.25 SELECTIVE_DUMP_PACKETS.request

3.4.4.25.1 Function

The **SELECTIVE_DUMP_PACKETS.request** primitive shall be passed to the SOIS Packet Store Access Service to request the reading of the first unread packet or list of packets in the current selection for the packet store and the transmission of the read packet(s) to the spacecraft telemetry chain over a private interface, for transfer to the ground.

3.4.4.25.2 Semantics

The **SELECTIVE_DUMP_PACKETS.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

SELECTIVE_DUMP_PACKETS.request (Transaction Identifier,
Packet Store Identifier,
Requested Packet Count)

3.4.4.25.3 When Generated

The **SELECTIVE_DUMP_PACKETS.request** primitive shall be passed to the SOIS Packet Store Access Service provider to request the reading of the first unread packet or list of packets in the current selection criteria for the packet store and directly transmitting the read packet(s) to the spacecraft telemetry chain.

3.4.4.25.4 Effect on Receipt

Receipt of the **SELECTIVE_DUMP_PACKETS.request** primitive shall cause the SOIS Packet Store Access Service provider to read the first unread packet or list of packets in the current selection criteria for the specified packet store and dump them to the spacecraft telemetry chain. The number of packets read and transmitted shall be in accordance with the Requested Packet Count parameter.

3.4.4.25.5 Additional Comments

3.4.4.25.5.1 The service differs from the DUMP service in that the packets to be dumped are first preselected using the CREATE_SELECTION primitive.

3.4.4.25.5.2 This primitive is intended to provide dumping of packets to ground via the spacecraft telemetry chain with minimum user entity involvement.

3.4.4.25.5.3 The service requires access to the spacecraft telemetry chain to enable the transmission of packets to the ground. The manner in which this access is provided is implementation-specific.

3.4.4.25.5.4 The mechanisms by which the spacecraft telemetry chain transmits the packet to ground are not specified by this Recommended Practice. They may or may not provide guaranteed delivery.

3.4.4.26 SELECTIVE_DUMP_PACKETS.indication

3.4.4.26.1 Function

The **SELECTIVE_DUMP_PACKETS.indication** shall be used to indicate to the user entity the result of initiating a packet dump as a result of the SELECTIVE_DUMP_PACKETS.request.

3.4.4.26.2 Semantics

The **SELECTIVE_DUMP_PACKETS.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

SELECTIVE_DUMP_PACKETS.indication (Transaction Identifier,
Result Metadata)

3.4.4.26.3 When Generated

The **SELECTIVE_DUMP_PACKETS.indication** primitive shall be passed by the service provider to the receiving user entity in response to a dump being initiated by a SELECTIVE_DUMP_PACKETS.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.4.26.4 Effect on Receipt

The response of the user entity to a **SELECTIVE_DUMP_PACKETS.indication** primitive is unspecified.

3.4.4.26.5 Additional Comments

None.

3.4.4.27 SELECTIVE_DUMP_PACKETS_COMPLETED.indication

3.4.4.27.1 Function

The **SELECTIVE_DUMP_PACKETS_COMPLETED.indication** shall be used to indicate to the user entity the result of completing a packet dump initiated by a SELECTIVE_DUMP_PACKETS.request.

3.4.4.27.2 Semantics

The **SELECTIVE_DUMP_PACKETS_COMPLETED.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

SELECTIVE_DUMP_PACKETS_COMPLETED.indication
(Transaction Identifier, Dumped Packet Count, Result Metadata)

3.4.4.27.3 When Generated

The **SELECTIVE_DUMP_PACKETS_COMPLETED.indication** primitive shall be passed by the service provider to the receiving user entity to indicate the completion of a dump being initiated by a SELECTIVE_DUMP_PACKETS.request.

NOTE – This primitive provides:

- the number of packets actually dumped;
- metadata indicating if the dump was executed successfully or not.

3.4.4.27.4 Effect on Receipt

The response of the user entity to a **SELECTIVE_DUMP_PACKETS_COMPLETED.indication** primitive is unspecified.

3.4.4.27.5 Additional Comments

None.

3.4.5 PACKET STORE MANAGEMENT SERVICE PRIMITIVES

3.4.5.1 General

The Packet Store Management Service interface shall implement the following primitives:

a) CREATE_PACKET_STORE.request, as requested in 3.4.5.2.

b) CREATE_PACKET_STORE.indication, as requested in 3.4.5.3.

c) DELETE_PACKET_STORE.request, as requested in 3.4.5.4.

d) DELETE_PACKET_STORE.indication, as requested in 3.4.5.5.

3.4.5.2 CREATE_PACKET_STORE.request

3.4.5.2.1 Function

The **CREATE_PACKET_STORE.request** primitive shall be passed to the SOIS Packet Store Management Service provider to request the creation of a new packet store.

3.4.5.2.2 Semantics

The **CREATE_PACKET_STORE.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CREATE_PACKET_STORE.request (Transaction Identifier, Packet Store Type, Packet Type, Allocated Space)

3.4.5.2.3 When Generated

The **CREATE_PACKET_STORE.request** primitive shall be passed to the SOIS Packet Store Management Service provider to request the creation of a new Packet Store according to the specified parameters.

3.4.5.2.4 Effect on Receipt

Receipt of the **CREATE_PACKET_STORE.request** primitive shall cause the SOIS Packet Store Management Service provider to create a new Packet Store according to the specified parameters.

3.4.5.2.5 Additional Comments

None.

3.4.5.3 CREATE_PACKET_STORE.indication

3.4.5.3.1 Function

The **CREATE_PACKET_STORE.indication** primitive shall be used to pass the outcome of making a packet store to the user entity.

3.4.5.3.2 Semantics

The **CREATE_PACKET_STORE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

CREATE_PACKET_STORE.indication (Transaction Identifier,
Packet Store Identifier, Result Metadata)

3.4.5.3.3 When Generated

The **CREATE_PACKET_STORE.indication** primitive shall be passed by the service provider to the receiving user entity in response to a CREATE_PACKET_STORE.request.

NOTE – This primitive:

- contains the identifier allocated to the new packet store; and
- provides metadata indicating if the request was executed successfully or not.

3.4.5.3.4 Effect on Receipt

The response of the user entity to a **CREATE_PACKET_STORE.indication** primitive is unspecified.

3.4.5.3.5 Additional Comments

None.

3.4.5.4 DELETE_PACKET_STORE.request

3.4.5.4.1 Function

The **DELETE_PACKET_STORE.request** primitive shall be passed to the SOIS Packet Store Management Service provider to request the removal of a specified packet store.

3.4.5.4.2 Semantics

The **DELETE_PACKET_STORE.request** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

DELETE_PACKET_STORE.request (Transaction Identifier, Packet Store Identifier)

3.4.5.4.3 When Generated

The **DELETE_PACKET_STORE.request** primitive shall be passed to the SOIS Packet Store Management Service provider to request the removal of the specified Packet Store.

3.4.5.4.4 Effect on Receipt

Receipt of the **DELETE_PACKET_STORE.request** primitive shall cause the SOIS Packet Store Management Service provider to remove the specified Packet Store.

3.4.5.4.5 Additional Comments

None.

3.4.5.5 DELETE_PACKET_STORE.indication

3.4.5.5.1 Function

The **DELETE_PACKET_STORE.indication** primitive shall be used to pass the outcome of removing a packet store to the user entity.

3.4.5.5.2 Semantics

The **DELETE_PACKET_STORE.indication** primitive shall use the following semantics, with the meaning of the parameters specified in 3.3:

DELETE_PACKET_STORE.indication (Transaction Identifier, Result Metadata)

3.4.5.5.3 When Generated

The **DELETE_PACKET_STORE.indication** primitive shall be passed by the service provider to the receiving user entity in response to a DELETE_PACKET_STORE.request.

NOTE – This primitive provides metadata indicating if the request was executed successfully or not.

3.4.5.5.4 Effect on Receipt

The response of the user entity to a **DELETE_PACKET_STORE.indication** primitive is unspecified.

3.4.5.5.5 Additional Comments

None.

4 MANAGEMENT INFORMATION BASE

4.1 DISCUSSION

There is currently no Management Information Base associated with this service. All management items are associated with the protocol providing the service. However, guidance is provided as to MIB contents in 4.3.

4.2 SPECIFICATIONS

Any implementation claiming to provide this service in a SOIS-compliant manner shall publish its Management Information Base as part of the protocol specification.

4.3 MIB GUIDANCE

The MIB of the protocol providing the File and Packet Store Services should consider the following aspects:

a) Maximum File Size, as specified in 4.4;

b) Maximum Packet Store Size, as specified in 4.5.

NOTE – These aspects are not in any way an indication of the complete contents of a MIB for an implementation providing the File and Packet Store Services but are offered as guidance as to those aspects of the MIB which may relate to File and Packet Store Services interface.

4.4 MAXIMUM FILE SIZE

The **Maximum File Size** shall indicate the maximum permissible size in octets of a file in the file store.

4.5 MAXIMUM PACKET STORE SIZE

The **Maximum Packet Store Size** shall indicate the maximum permissible size in octets of a packet store.

ANNEX A

FILE AND PACKET STORE SERVICES PROTOCOL IMPLEMENTATION CONFORMANCE STATEMENT PROFORMA

(NORMATIVE)

A.1 INTRODUCTION

This annex provides the Protocol Implementation Conformance Statement (PICS) Requirements List (PRL) for implementation of the File and Packet Store Services, CCSDS 873.0-M-1, September 2012. The PICS for an implementation is generated by completing the PRL in accordance with the instructions below. An implementation shall satisfy the mandatory conformance requirements of the base standards referenced in the PRL.

The PRL in this annex is blank. An implementation's complete PRL is called a PICS. The PICS states which capabilities and options of the services have been implemented. The following can be use the PICS:

- The service 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 interoperability with another implementation;
- A service tester, as a basis for selecting appropriate tests against which to assess the claim for conformance of the implementation.

A.2 NOTATION

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

Status Symbols

M	mandatory
O	optional

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

A.3 REFERENCED BASE STANDARDS

The base standards references in the PRL are:

– File and Packet Store Services – this document.

A.4 GENERATION INFORMATION

A4.1 IDENTIFICATION OF PICS

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

A4.2 IDENTIFICATION OF IMPLEMENTATION UNDER TEST (IUT)

Ref	Question	Response
1	Implementation name	
2	Implementation version	
3	Special configuration	
4	Other information	

A.5 IDENTIFICATION

Ref	Question	Response
1	Supplier	
2	Contact Point for Queries	
3	Implementation name(s) and Versions	
4	Other information necessary for full identification, e.g., name(s) and version(s) for machines and/or operating systems: System Name(s)	

A.6 SERVICE SUMMARY

Ref	Question	Response
1	Service Version	
2	Addenda implemented	
3	Amendments implemented	
4	Have any exceptions been required? (Note: a YES answer means that the implementation does not conform to the service. Non-supported mandatory capabilities are to be identified in the PICS, with an explanation of why the implementation is non-conforming.	Yes _____ No

A.7 INSTRUCTIONS FOR COMPLETING THE PRL

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

A.8 GENERAL/MAJOR CAPABILITIES

Item	Service Feature	Reference (Magenta Book)	Status	Support
File Access Service				
	OPEN_FILE.request	3.4.2.2	M	
	File Opening Criteria parameter—Lock Type attribute	3.3.3.7	O	
	OPEN_FILE.indication	3.4.2.3	M	
	CLOSE_FILE.request	3.4.2.4	M	
	CLOSE_FILE.indication	3.4.2.5	M	
	READ_FROM_FILE.request	3.4.2.6	M	
	READ_FROM_FILE.indication	3.4.2.7	M	
	WRITE_TO_FILE.request	3.4.2.8	M	
	WRITE_TO_FILE.indication	3.4.2.9	M	
	FILE_SEEK.request	3.4.2.10	O	
	FILE_SEEK.indication	3.4.2.11	O	
File Management Service				
	LIST_DIR.request	3.4.3.2	M	
	LIST_DIR.request—Directory Name parameter	3.4.3.2	O	
	Directory Name parameter—special name representing the directory above the user entity's current directory	3.3.3.6	O	
	LIST_DIR.indication	3.4.3.3	M	
	CREATE_FILE.request	3.4.3.14	M	
	CREATE_FILE.indication	3.4.3.15	M	
	DELETE_FILE.request	3.4.3.16	M	
	DELETE_FILE.indication	3.4.3.17	M	
	COPY_FILE.request	3.4.3.18	M	
	COPY_FILE.indication	3.4.3.19	M	
	MOVE_FILE.request	3.4.3.20	M	
	MOVE_FILE.indication	3.4.3.21	M	
	CREATE_DIR.request	3.4.3.2	O	
	CREATE_DIR.indication	3.4.3.3	O	
	GET_CURRENT_DIR.request	3.4.3.4	O	
	GET_CURRENT_DIR.indication	3.4.3.5	O	
	CHANGE_DIR.request	3.4.3.6	O	
	CHANGE_DIR.indication	3.4.3.7	O	
	DELETE_DIR.request	3.4.3.8	O	
	DELETE_DIR.indication	3.4.3.9	O	
	RENAME_DIR.request	3.4.3.10	O	
	RENAME_DIR.indication	3.4.3.11	O	
	LOCK_FILE.request	3.4.3.22	O	

Item	Service Feature	Reference (Magenta Book)	Status	Support
	LOCK_FILE.indication	3.4.3.23	O	
	UNLOCK_FILE.request	3.4.3.24	O	
	UNLOCK_FILE.indication	3.4.3.25	O	
	LIST_LOCKED_FILES.request	3.4.3.26	O	
	LIST_LOCKED_FILES.indication	3.4.3.27	O	
	FIND_FILE.request	3.4.3.28	O	
	FIND_FILE.indication	3.4.3.29	O	
	FILE_STATUS.request	3.4.3.30	O	
	FILE_STATUS.indication	3.4.3.31	O	
	File Attributes parameter—Creation Time attribute	3.3.3.3	O	
	File Attributes parameter—Last Write Time attribute	3.3.3.3	O	
	File Attributes parameter—Lock Identifier attribute	3.3.3.3	O	
Packet Store Access Service				
	GET_PACKET_STORES_INFO.request	3.4.4.2	M	
	GET_PACKET_STORES_INFO.indication	3.4.4.3	M	
	CLEAR_PACKET_STORE.request	3.4.4.4	M	
	CLEAR_PACKET_STORE.indication	3.4.4.5	M	
	WRITE_PACKETS.request	3.4.4.6	M	
	WRITE_PACKETS.indication	3.4.4.7	M	
	READ_PACKETS.request	3.4.4.8	M	
	READ_PACKETS.indication	3.4.4.9	M	
	SET_PACKET_STORE_POSITION.request	3.4.4.10	M	
	SET_PACKET_STORE_POSITION.indication	3.4.4.11	M	
	FREE_PACKETS.request	3.4.4.12	M	
	FREE_PACKETS.indication	3.4.4.13	M	
	PACKET_STORE_STATUS.request	3.4.4.14	M	
	PACKET_STORE_STATUS.indication	3.4.4.15	M	
	DUMP_PACKETS.request	3.4.4.16	O	
	DUMP_PACKETS.indication	3.4.4.17	O	
	DUMP_PACKETS_COMPLETED.indication	3.4.4.18	O	
	CREATE_SELECTION.request	3.4.4.19	O	
	CREATE_SELECTION.indication	3.4.4.20	O	
	SELECTIVE_READ_PACKETS.request	3.4.4.21	O	
	SELECTIVE_READ_PACKETS.indication	3.4.4.22	O	
	SELECTIVE_FREE_PACKETS.request	3.4.4.23	O	
	SELECTIVE_FREE_PACKETS.indication	3.4.4.24	O	
	SELECTIVE_DUMP_PACKETS.request	3.4.4.25	O	
	SELECTIVE_DUMP_PACKETS.indication	3.4.4.26	O	
	SELECTIVE_DUMP_PACKETS_COMPLETED.indication	3.4.4.27	O	

Item	Service Feature	Reference (Magenta Book)	Status	Support
Packet Store Management Service				
	CREATE_PACKET_STORE.request	3.4.5.2	O	
	CREATE_PACKET_STORE.indication	3.4.5.3	O	
	DELETE_PACKET_STORE.request	3.4.5.4	O	
	DELETE_PACKET_STORE.indication	3.4.5.5	O	

A.9 UNDERLYING LAYERS PROVIDING SERVICES TO IMPLEMENTATION

This annex subsection provides identification of the Underlying Layers providing Services to the implementation.

Service Feature	Reference	Status	Support
Local File Store		O	
A Network File Access Protocol		O	
Local Packet Store		O	
A Network Packet Access Protocol		O	

ANNEX B

SECURITY CONSIDERATIONS

(INFORMATIVE)

B.1 SECURITY BACKGROUND

The SOIS services are intended for use with protocols that operate solely within the confines of an onboard subnetwork on a single spacecraft. It is therefore assumed that SOIS services operate in a closed onboard environment which is protected from external threats. Security issues introduced by equipment connected to these onboard networks must be managed as a part of the design, integration, and Validation and Verification (V&V) activities. Any external communication is assumed to be protected by services associated with the relevant space-link protocols. The specification of such security services is out of scope of this document.

B.2 SECURITY CONCERNS

At the time of writing there are no identified security concerns. If confidentiality of data is required within a spacecraft it is assumed it is applied at the application layer. For more information regarding the choice of service and where it can be implemented, see reference [D3].

B.3 POTENTIAL THREATS AND ATTACK SCENARIOS

Potential threats and attack scenario typically derive from external communication and are therefore not the direct concern of the SOIS services which makes the assumption that the services operate within a safe and secure environment. It is assumed that all onboard software executing within the spacecraft has been thoroughly tested and cleared for use by the mission implementer. Confidentiality of onboard software may be provided by application layer mechanisms or by specific implementation methods such as time and space partitioning. Such methods are outside the scope of SOIS.

B.4 CONSEQUENCES OF NOT APPLYING SECURITY

The security services are out of scope of this document and should be applied at layers above or below those specified in this document. If confidentiality is not implemented, science data or other parameters transmitted within the spacecraft might be visible to other onboard software resident within the spacecraft resulting in disclosure of sensitive or private information.

B.5 RELIABILITY

While it is assumed that the underlying mechanisms used to implement the file and packet stores operate correctly, the FPSS make no assumptions as to their reliability.

ANNEX C

ACRONYMS

(INFORMATIVE)

CCSDS	Consultative Committee for Space Data Standards
FAS	File Access Service
FIFO	First In, First Out
FMS	File Management Service
FPSS	File and Packet Store Services
IP	Internet Protocol
MIB	Management Information Base
OBC	Onboard Computer
OSI	Open Systems Interconnection
PSAS	Packet Store Access Service
PSASAP	Packet Store Access Service Access Point
PSMS	Packet Store Management Service
PSMSAP	Packet Store Management Service Access Point
SAP	Service Access Point
SOIS	Spacecraft Onboard Interface Services
SSMM	Solid State Mass Memory

ANNEX D

INFORMATIVE REFERENCES

(INFORMATIVE)

[D1] *Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model*. International Standard, ISO/IEC 7498-1:1994. 2nd ed. Geneva: ISO, 1994.

[D2] *Spacecraft Onboard Interface Services*. Report Concerning Space Data System Standards, CCSDS 850.0-G-1. Green Book. Issue 1. Washington, D.C.: CCSDS, June 2007.

[D3] *The Application of CCSDS Protocols to Secure Systems*. Report Concerning Space Data System Standards, CCSDS 350.0-G-2. Green Book. Issue 2. Washington, D.C.: CCSDS, January 2006.

[D4] *Space Packet Protocol*. Recommendation for Space Data System Standards, CCSDS 133.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, September 2003.

[D5] *Encapsulation Service*. Recommendation for Space Data System Standards, CCSDS 133.1-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, October 2009.

[D6] J. Postel. *Internet Protocol*. STD 5. Reston, Virginia: ISOC, September 1981.

[D7] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460. Reston, Virginia: ISOC, December 1998.

[D8] *IEEE Standard for Information Technology—Portable Operating System Interface (POSIX) Base Specifications*. Issue 7. IEEE Std 1003.1. San Francisco and Piscataway, NJ: The Open Group and IEEE, 2008.

ANNEX E

TYPICAL API—POSIX

(INFORMATIVE)

This annex proposes a mapping of the FPSS service interface onto the POSIX API as defined in [D8], in particular the manual pages associated with the <fcntl.h>, <sys/stat.h> and <unistd.h> headers.

E.1 MAPPING OF FILE ACCESS SERVICE ONTO POSIX

FAS Primitives	**POSIX API**	**Comments**
File Open/Close Capability		
OPEN_FILE.request and OPEN_FILE.indication	int open()	*oflag* argument provides File Opening Criteria and File Lock Type parameters using O_RDONLY, O_RDWR, O_APPEND, O_CREAT flags.
CLOSE_FILE.request and CLOSE_FILE.indication	int close()	
File Read/Write Capability		
READ_FROM_FILE.request and READ_FROM_FILE.indication	ssize_t read()	
WRITE_TO_FILE.request and WRITE_TO_FILE.indication	ssize_t write()	
File Seek Capability		
FILE_SEEK.request and FILE_SEEK.indication	off_t lseek()	*whence* argument set to SEEK_SET.

E.2 MAPPING OF FILE MANAGEMENT SERVICE ONTO POSIX

FMS Primitives	**POSIX API**	**Comments**
File Management Capability		
CREATE_FILE.request and CREATE_FILE.indication	int open()	*oflag* set to O_CREAT
DELETE_FILE.request and DELETE_FILE.indication	int unlink()	
MOVE_FILE.request and MOVE_FILE.indication	int rename()	
COPY_FILE.request and COPY_FILE.indication	Not supported directly.	Can be built using *open(), read()* and *write()* APIs in a similar way as Unix shell does for '*cp*' command.
LOCK_FILE.request and LOCK_FILE.indication	int chmod()	Lock/unlock mechanism provided by assigning adequate access rights to the file. However, no blocking/pending behaviour is provided by this.
UNLOCK_FILE.request and UNLOCK_FILE.indication	int chmod()	Lock/unlock mechanism provided by assigning adequate access rights to the file.
FILE_STATUS.request and FILE_STATUS.indication	int stat()	

FMS Primitives	**POSIX API**	**Comments**
Directory Management Capability		
LIST_DIR.request and LIST_DIR.indication	Not supported directly.	Can be built using *open()* and *stat()* APIs in a similar way as Unix shell does for '*ls*' command. '.' is the filename of the current directory.
Directory Name parameter—special name representing the directory above the user entity's current directory	'..'	
CREATE_DIR.request and CREATE_DIR.indication	int mkdir()	
GET_CURRENT_DIR.request and GET_CURRENT_DIR.indication	char *getcwd()	
CHANGE_DIR.request and CHANGE_DIR.indication	int chdir()	
DELETE_DIR.request and DELETE_DIR.indication	int rmdir()	
RENAME_DIR.request and RENAME_DIR.indication	int rename()	
LIST_LOCKED_FILES.request and LIST_LOCKED_FILES.indication	Not supported directly.	Can be built using *open()* and *stat()* APIs.
FIND_FILE.request and FIND_FILE.indication	Not supported directly.	Can be built using *open()* and *stat()* APIs.

E.3 MAPPING OF PACKET STORE ACCESS SERVICE ONTO POSIX

Packet Stores are not natively supported in POSIX; therefore there is no recommended mapping of the Packet Store Access Service onto the POSIX API.

E.4 MAPPING OF PACKET STORE MANAGEMENT SERVICE ONTO POSIX

Packet Stores are not natively supported in POSIX; therefore there is no recommended mapping of the Packet Store Management Service onto the POSIX API.