The Consultative Committee for Space Data Systems

Recommendation for Space Data System Standards

CROSS SUPPORT TRANSFER SERVICE—SPECIFICATION FRAMEWORK

RECOMMENDED STANDARD

CCSDS 921.1-B-2

BLUE BOOK

February 2021

The Consultative Committee for Space Data Systems

Recommendation for Space Data System Standards

CROSS SUPPORT TRANSFER SERVICE—SPECIFICATION FRAMEWORK

RECOMMENDED STANDARD

CCSDS 921.1-B-2

BLUE BOOK

February 2021

AUTHORITY

Issue:	Recommended Standard, Issue 2
Date:	February 2021
Location:	Washington, DC, USA

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

This document is published and maintained by:

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

STATEMENT OF INTENT

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 **Recommended Standards** and are not considered binding on any Agency.

This **Recommended Standard** is issued by, and represents the consensus of, the CCSDS members. Endorsement of this **Recommendation** is entirely voluntary. Endorsement, however, indicates the following understandings:

- Whenever a member establishes a CCSDS-related **standard**, this **standard** will be in accord with the relevant **Recommended Standard**. Establishing such a **standard** does not preclude other provisions which a member may develop.
- Whenever a member establishes a CCSDS-related **standard**, that member will provide other CCSDS members with the following information:
  - -- The **standard** itself.
  - -- The anticipated date of initial operational capability.
  - -- The anticipated duration of operational service.
- Specific service arrangements shall be made via memoranda of agreement. Neither this **Recommended Standard** nor any ensuing **standard** is a substitute for a memorandum of agreement.

No later than five years from its date of issuance, this **Recommended Standard** 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 Standard** is issued, existing CCSDS-related member standards and implementations are not negated or deemed to be non-CCSDS compatible. It is the responsibility of each member to determine when such standards or implementations are to be modified. Each member is, however, strongly encouraged to direct planning for its new standards and implementations towards the later version of the Recommended Standard.

FOREWORD

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

http://www.ccsds.org/

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

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 (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 Science Policy Office (BELSPO)/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.
- China Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- Danish National Space Center (DNSC)/Denmark.
- Departamento de Ciência e Tecnologia Aeroespacial (DCTA)/Brazil.
- Electronics and Telecommunications Research Institute (ETRI)/Korea.
- 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.
- Hellenic Space Agency (HSA)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Institute of Space Research (IKI)/Russian Federation.
- Korea Aerospace Research Institute (KARI)/Korea.
- Ministry of Communications (MOC)/Israel.
- Mohammed Bin Rashid Space Centre (MBRSC)/United Arab Emirates.
- 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.
- Netherlands Space Office (NSO)/The Netherlands.
- Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- Scientific and Technological Research Council of Turkey (TUBITAK)/Turkey.
- South African National Space Agency (SANSA)/Republic of South Africa.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.
- Swiss Space Office (SSO)/Switzerland.
- United States Geological Survey (USGS)/USA.

DOCUMENT CONTROL

Document	Title	Date	Status
CCSDS 921.1-B-1	Cross Support Transfer Service—Specification Framework, Recommended Standard, Issue 1	April 2017	Original issue, superseded
CCSDS 921.1-B-2	Cross Support Transfer Service—Specification Framework, Recommended Standard, Issue 2	February 2021	Current issue
EC 1	Editorial Correction 1	February 2024	Corrects annex M page numbering.

NOTE – Changes from the original issue are too numerous to permit meaningful markup.

CONTENTS

CONTENTS (continued)

Section Page

Figure

CONTENTS (continued)

Figure Page

Table

CONTENTS (continued)

CONTENTS (continued)

Table Page

CONTENTS (continued)

1 INTRODUCTION

1.1 PURPOSE

The purpose of this Recommended Standard is to define the various logical components, also known within this Recommended Standard as procedures that are required for specifying Cross Support Transfer Services (CSTSes).

1.2 SCOPE

1.2.1 This Recommended Standard defines, in an abstract manner, a CSTS in terms of

a) the procedures necessary to provide the service;

b) the states of the service;

c) the behavior of each procedure;

d) the states of the procedures;

e) the operations necessary to constitute the procedures; and

f) the parameters associated with each operation.

1.2.2 It does not specify

a) individual application services, implementations, or products;

b) the implementation of entities or interfaces within real systems;

c) the methods or technologies required to acquire data;

d) the methods or technologies required to provide a suitable environment for communications; or

e) the management activities required to schedule and configure services.

1.3 APPLICABILITY

1.3.1 APPLICABILITY OF THIS RECOMMENDED STANDARD

This Recommended Standard provides a basis for the specification and development of Cross Support Services that are intended to be used for developing real systems that implement such services.

Implementation of a service based on the CSTS procedures defined in this Recommended Standard in a real system additionally requires the availability of a communications service to convey invocations and responses of the CSTS operations between the service user and the service provider.

This Recommended Standard requires that such a communications service provides a reliable protocol, that is, that it ensures that invocations and responses of operations are transferred

a) in sequence;

b) completely and with integrity;

c) without duplication;

d) with flow control that notifies the application layer in the event of congestion or backpressure; and

e) with notification to the application layer in the event that communications between the service user and the service provider are disrupted, possibly resulting in a loss of data.

It is the specific intent of this Recommended Standard to define the CSTS independently of any particular communications services, protocols, technologies, or formatting of the data content.

1.3.2 LIMITS OF APPLICABILITY

This Recommended Standard specifies the CSTS procedures that may be used for the definition of Cross Support Transfer Services. It does not intend to specify a Cross Support Transfer Service.

1.4 RATIONALE

The goal of this Recommended Standard is to create a standard for interoperability between various Agencies' tracking stations or ground data handling systems and the consumers or producers of spacecraft data and related monitor and/or control information.

1.5 DOCUMENT STRUCTURE

1.5.1 ORGANIZATION OF THIS DOCUMENT

This document is organized as follows:

a) section 1 presents the purpose, scope, applicability, and rationale of this Recommended Standard and lists the definitions, conventions, and references;

b) section 2 provides an overview of the CSTS Specification Framework;

c) section 3 specifies the common operations to be used by Cross Support Transfer Services;

d) section 4 specifies the procedures to be used by the Cross Support Transfer Services;

e) annex A contains the proforma of the Protocol Implementation Conformance Statement;

f) annex B provides a formal specification of the `production-status` parameter and the 'production status change' and 'production configuration change' events;

g) annex C provides a formal specification of what a qualified parameter is;

h) annex D provides a formal specification of the Object Identifiers and the management of their allocation;

i) annex E defines the composition of Functional Resource Names, Parameter Names, Event Names, Parameter Lists, and Event Lists using Published Identifiers of the appropriate types;

j) annex F provides a formal specification of data types for Protocol Data Units (PDUs) for common operations using Abstract Syntax Notation One (ASN.1);

k) annex G provides a description of the service provider states;

l) annex H contains considerations related to security, SANA, and patents;

m) annex I provides a list of informative references;

n) annex J lists acronyms used in this document;

o) annex K provides an informative list of Object Identifiers used by this Recommended Standard;

p) annex L illustrates by means of examples the concept of Published Identifiers;

q) annex M illustrates how the ASN.1 representations of qualified parameters are constructed for example CSTS procedure parameters and functional resource parameters.

1.5.2 CROSS SUPPORT TRANSFER SERVICES DOCUMENTATION

The basic organization of the Cross Support Services documentation and the relationship to CSTS documentation is shown in figure 1-1.

The Cross Support Architecture is documented in:

a) *Space Communications Cross Support—Architecture Description Document* (reference [I8]): An Informational Report describing an architecture in terms of CCSDS-recommended configurations for secure space communications cross support. This architecture is intended to be used as a common framework when CCSDS Agencies a) provide and use space communications cross support services, and b) develop systems that provide interoperable space communications cross support.

b) *Space Communications Cross Support—Architecture Requirements Document* (reference [I9]): A Recommended Practice defining a set of requirements for CCSDS-recommended configurations for secure Space Communications Cross Support architectures.

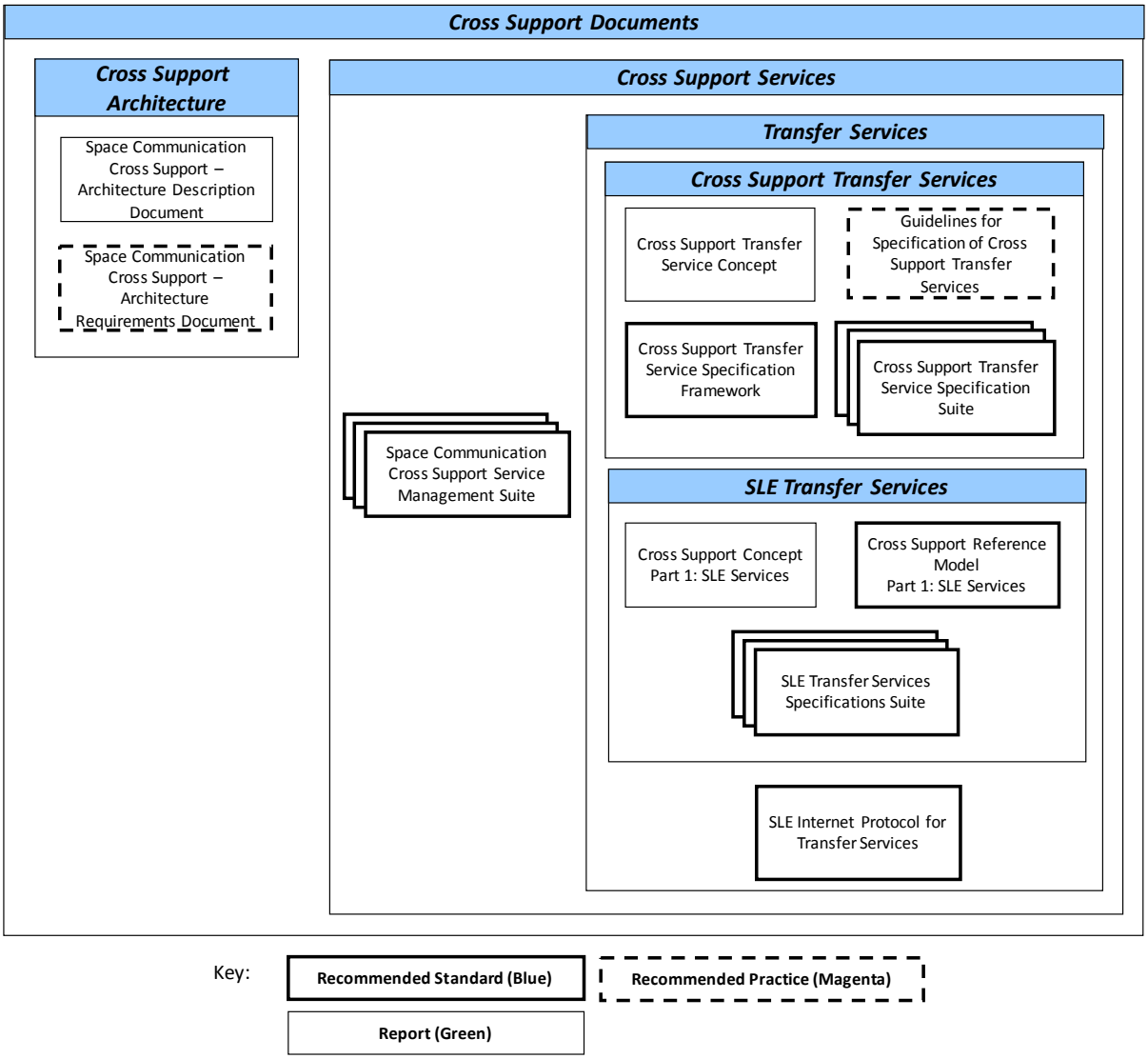

Figure 1-1: Cross Support Service Documentation

Common to all Cross Support Services:

c) *Space Communication Cross Support Service Management suite* (references [I4], [I6], [I7], and [I10]). Data format Recommended Standards that specify the Service Management Information Entities that are used to configure and schedule cross support services, which include transfer services.

Common to the Transfer Services, that is, SLE Transfer Services and Cross Support Transfer Services:

d) *Space Link Extension—Internet Protocol for Transfer Services* (reference [2]): A Recommended Standard that defines a protocol for transfer of PDUs defined in the Cross Support Transfer Services. This Recommended Standard was originally developed to support SLE transfer services (hence the title), but it is also applicable to use by Cross Support Transfer Services.

The concept, the reference model, and the SLE Transfer Services as such are documented in:

e) *Cross Support Concept—Part 1: Space Link Extension Services* (reference [I2]): A report introducing the concepts of cross support and the SLE services. Many of the concepts for the SLE transfer services have been adopted for the CSTSes (see k) below).

f) *Cross Support Reference Model—Part 1: Space Link Extension Services* (reference [1]): A Recommended Standard that defines the framework and terminology for the specification of SLE services. Much of the framework and terminology of this reference model has been adopted or adapted for CSTSes (see 1.6.1.3 and 2.2).

g) The *SLE Transfer Services suite*: The SLE Transfer Services are a suite of Cross Support Services that are used to transfer specific telecommand and telemetry PDUs. The SLE Transfer Services are closely related to the CSTS suite in that they collectively define the set of operations that are the basis for the CSTS Specification Framework. However, because of history (the SLE Transfer Services were already specified and implemented prior to development of the CSTS Specification Framework) the SLE Transfer Services are separated from CSTSes.

The documents specific to Cross Support Transfer Services are:

h) *Cross Support Transfer Services Specification Framework* (this Recommended Standard): A Recommended Standard that defines the specification of the Cross Support Transfer Service procedures;

i) *Guidelines for Specification of Cross Support Transfer Services* (reference [I3]): A Recommended Practice that defines the guidelines for construction of a Cross Support Transfer Service based on the CSTS Specification Framework;

j) *Cross Support Transfer Services Concept* (reference [I5]): A Report that provides tutorial material on the objectives and concepts of the CSTS specification;

k) *Cross Support Transfer Services Suite*: The set of specifications for actual CSTSes built from the procedures in the CSTS Specification Framework and in accordance with the CSTS Guidelines.

1.6 DEFINITIONS

1.6.1 TERMS

1.6.1.1 Terms from Open Systems Interconnection (OSI) Basic Reference Model, Reference [3]

This Recommended Standard makes use of a number of terms defined in reference [3]. The use of those terms in this Recommended Standard shall be understood in a generic sense, that is, in the sense that those terms are generally applicable to technologies that provide for the exchange of information between real systems. Those terms are

a) abstract syntax;

b) application entity;

c) application layer;

d) flow control; and

e) real system.

1.6.1.2 Terms from Abstract Syntax Notation One, Reference [4]

This Recommended Standard makes use of the following terms defined in reference [4]:

a) ASN.1;

b) Object Identifier (OID);

c) (data) type;

d) (data) value.

NOTE – In annex F of this Recommended Standard, ASN.1 is used for specifying the abstract syntax of service operation invocations and responses.

1.6.1.3 Terms from Cross Support Reference Model, Reference [1]

This Recommended Standard makes use of the following terms defined in reference [1]:

a) binding;

b) initiator;

c) invoker;

d) operation;

e) performer;

f) physical channel;

g) responder;
h) service agreement;
i) service provider;
j) service user;
k) space link.

1.6.1.4 Terms from the Space Communications Cross Support Architecture Description Document, Reference [I8]

This Recommended Standard makes use of the following terms defined in reference [I8]:

a) Cross Support Service System (CSSS);
b) Earth Space Link Terminal (ESLT);
c) Earth User CSSS;
d) Element Management (EM);
e) Provider CSSS;
f) Provision Management (PM) of the Provider CSSS;
g) Utilization Management (UM) of the Earth User CSSS.

1.6.1.5 Terms from CCSDS SANA Registry Management Policy, Reference [6]

This Recommended Standard makes use of the following terms defined in reference [6]:

a) OID Registry;
b) owned by Agencies;
c) owned by xxx Area;
d) Review Authority;
e) Service Site and Aperture Registry;
f) Spacecraft Registry.

1.6.1.6 Terms from SANA Role, Responsibilities, Policies, and Procedures, Reference [7]

This Recommended Standard makes use of the following terms defined in reference [7]:

a) delegation;

b) registration rules;

c) registry;

d) SANA Steering Group.

1.6.1.7 Terms Defined in this Specification

1.6.1.7.1 abstract service: The specification of the common operations and behavior of CSTSes that is not directly implementable because service-specific specifications are still missing. Real CSTSes can be derived from such abstract service.

1.6.1.7.2 acknowledgement: A confirmation that the invocation sent by the invoker was accepted by the performer and will now be acted upon.

a) An acknowledgement with a negative result is also called a negative acknowledgement.

b) An acknowledgement with a positive result is also called a positive acknowledgement.

1.6.1.7.3 association: A cooperative relationship between a service-providing application entity and a service-using application entity. An association is formed by exchanging service PDUs using an underlying communications service.

1.6.1.7.4 communications service: A capability that enables a CSTS-providing application entity and a CSTS-using application entity to exchange information.

1.6.1.7.5 confirmed operation: An operation that requires the performer to return a report of the operation outcome to the invoker. Confirmed operations are further classified as two-phase operations and three-phase operations.

1.6.1.7.6 Cross Support Service: A set of capabilities that an object that belongs to one Space Agency provides to objects that belong to other Space Agencies by means of one or more ports, in support of spacecraft operations. A Cross Support Service can also be used within a given Space Agency.

1.6.1.7.7 Cross Support Transfer Service, CSTS: A subclass of Cross Support Service that provides reliable, access-controlled transfer of spaceflight mission-related data between ground element entities, realized through the invocation and performance of defined operations in accordance with defined procedures. A CSTS is qualified by the kind of data it transfers (e.g., telemetry frames, tracking data). A CSTS may optionally have capabilities to coordinate and observe the behavior of the production with which this service is associated.

1.6.1.7.8 Cross Support Transfer Service instance: An instance of a specific CSTS type by means of which an ESLT provides the capability to the service user to transfer the service-type specific spaceflight mission-related data. There may be CSTSes that operate in other Provider CSSS nodes than an ESLT. In such cases, "Element Management" would refer to the management functions that have local purview over the provision and production of those CSTSes.

1.6.1.7.9 Cross Support Service production: Performance of the data acquisition process and/or the data transformation as necessary for the given type of CSTS.

1.6.1.7.10 Cross Support Utilization Management: The abstraction of the entities within a real Earth User CSSS that on behalf of the cross supported mission interacts with the PM of the Provider CSSS to arrange for the required Cross Support Services.

1.6.1.7.11 Cross Support Service provision: Exposing the operations necessary so that the service user can obtain the service. Provision involves the interface between the service user and the service provider.

1.6.1.7.12 data acquisition process: The means by which service production provides the capability to a service provider to access data required for service provisioning or to access information on service production processes (e.g., physical channel of the space link).

1.6.1.7.13 derivation: A mechanism that allows extending (see 1.6.1.7.19) or refining (see 1.6.1.7.45) an operation or the behavior of a procedure.

NOTE – Derivation can also be applied to a CSTS.

1.6.1.7.14 Directive Identifier: In the context of a CSTS, the unique identifier of a directive defined for service provision or service production. A Directive Identifier is defined as a Published Identifier (see D6).

1.6.1.7.15 Directive Name: A data structure consisting of a Directive Identifier that represents an individual directive type, and a Functional Resource Name that represents the Functional Resource with which the specific instance of that directive type is associated.

1.6.1.7.16 Event Identifier: In the context of a CSTS, the unique identifier of an event defined for service provision or service production. An Event Identifier is defined as a Published Identifier (see D6).

1.6.1.7.17 Event Label: (See E8 for the detailed definition.)

1.6.1.7.18 Event Name: (See E5 for the detailed definition.)

1.6.1.7.19 extension: The act of extending operations or procedures. Operations are extended by a) adding new parameters to the invocation or response message, or b) extending the range of values for already existing parameters, or c) changing an unconfirmed operation into a confirmed operation. Procedures are extended by a) adding new operations or b) extending the used operations.

1.6.1.7.20 Framework: A set of generic behaviors (called common procedures) and the constituent common operations used to simplify the specification of systems providing or using CSTSes.

1.6.1.7.21 Functional Resource Instance: An instance of a Functional Resource Type.

1.6.1.7.22 Functional Resource Instance Number: An integer index used to identify an instance of a Functional Resource Type.

1.6.1.7.23 Functional Resource Name: The unique identifier of an instance of a Functional Resource within the scope of a given service package. A Functional Resource Name is made of a Functional Resource Type and a Functional Resource Instance Number.

1.6.1.7.24 Functional Resource Type: A logical function or related set of functions that characterizes a unique instance of service provider or production capability. A Functional Resource Type is defined as a Published Identifier (see D6).

1.6.1.7.25 invocation: The making of a request by an entity (the invoker) to another entity (the performer) to carry out the invoked operation.

1.6.1.7.26 Label List: A data structure that specifies the name of a list of Parameter Labels or Event Labels, indicates if the given list is the default list, and contains all Parameter Labels or Event Labels represented by that Label List name.

1.6.1.7.27 Label List Set: The set of Label Lists accessible by the user of the given service instance.

1.6.1.7.28 object: A functional entity that interacts with other objects. Objects are of different types, which determine their function and behavior. Objects are characterized by their interfaces, which are called ports. One object may provide multiple ports.

1.6.1.7.29 operation: A task that the invoker requests the performer to execute. Depending on the type of operation, the performer may or may not report the result of the operation to the invoker. Service user and service provider interact by invoking and performing operations.

1.6.1.7.30 parameter: In the context of an operation, data that may accompany the operation's invocation, acknowledgement, or return.

NOTE – The term parameter is also used to refer to mission-dependent configuration information used in the production or provision of the service.

1.6.1.7.31 Parameter Identifier: In the context of a CSTS, the unique identifier of a parameter type defined for service provision or service production. A Parameter Identifier is defined as a Published Identifier (see D6).

1.6.1.7.32 Parameter Label: (See E7 for the detailed definition.)

1.6.1.7.33 Parameter Name: (See E4 for the detailed definition.)

1.6.1.7.34 performance: In the context of an operation, the carrying out of the operation by an object (the performer).

1.6.1.7.35 port identifier: Identifier of a source or a destination in a communications system.

NOTE – For purposes of the communications mapping, the endpoints of a CSTS association are identified by port identifiers, namely, an 'initiator port identifier' and a 'responder port identifier'. The port identifiers represent all the technology-specific addressing information needed to establish communications between service user and service provider and to route CSTS PDUs between them.

1.6.1.7.36 procedure: A specified series of actions performed using operations that have to be executed in order to implement a specified behavior.

1.6.1.7.37 procedure configuration parameter: A configuration parameter for a procedure type. If a service permits multiple instances of a procedure type, each such procedure instance will have its own instances of (and values for) the procedure configuration parameters for that procedure type. Unless explicitly specified otherwise in the Framework definition of a procedure type, the method(s) by which the values of the procedure configuration parameters are set is (are) delegated to the service using the procedure or to a derived procedure that may delegate setting of procedure configuration to service management (see 1.6.1.7.51 and 2.3). Some procedure configuration parameters may be dynamically modifiable (see 1.6.3.2.6). Dynamic modification of procedure configuration parameter values is performed by the user of the service that uses that procedure. Unless explicitly specified otherwise in the Framework definition of a procedure type, the method(s) by which the values of dynamically modifiable procedure configuration parameters are modified is (are) delegated to the service using the procedure or to a derived procedure.

NOTE – A method a service may use to set the value of dynamically modifiable procedure configuration parameters is the THROW-EVENT procedure (see 4.12). A derived procedure may comprise the EXECUTE-DIRECTIVE operation (see 3.13) as another method to set dynamically modifiable procedure configuration parameter values. Other methods can be used by services to set dynamically modifiable configuration parameter values.

1.6.1.7.38 procedure name: The unique identifier of a procedure instance within the scope of a given service instance. A procedure name is made of the procedure type and the procedure role. If the procedure role is 'association control' or 'prime procedure', no instance number is defined, because only a single procedure instance exists. If the role is 'secondary procedure', the procedure role is the instance number of the procedure instance.

1.6.1.7.39 production status: The aggregate status of the production processes that generate or process the data that is transferred by the transfer service instance.

1.6.1.7.40 protocol abort: A communications failure event.

NOTE – In the case of a communications disruption, the underlying communications service will report a communications failure event to the application. Occurrence of the communications failure event is referred to as a 'protocol abort'.

1.6.1.7.41 Provision Management: The authority of a Provider CSSS that negotiates the provision of service packages with UM of an Earth User CSSS and controls and monitors the production and provision of the CSTS instances by the Functional Resources belonging to the ESLT via ESLT EM.

1.6.1.7.42 Published Identifier: A unique identifier that allows identification of a spacecraft, a facility where a CSTS provider is located, a Functional Resource Type, a procedure type, a parameter, a directive, or an event. Published Identifiers are allocated by the Space Assigned Numbers Authority (SANA) and registered in standard SANA registries for use in CCSDS services and protocols, including but not limited to CSTSes.

1.6.1.7.43 real-time data: Data that can be accessed by the service user as soon as it is collected or generated by the service provider.

1.6.1.7.44 recorded data: Data that has been collected and stored by the service provider for access by the service user at some later time.

1.6.1.7.45 refinement: The act of refining operations or procedures. Operations are refined by constraining the values of parameters or by detailing the parameter semantics. Procedures are refined by modifying (e.g., narrowing) their semantics or defining their behavior or states in more detail.

1.6.1.7.46 registered parameter: A parameter that has been assigned a Published Identifier that is registered with SANA as specified in more detail in annex E.

1.6.1.7.47 response: A report, from the performer to the invoker

a) of the acceptance of the invocation sent by the invoker; such response is referred to as acknowledgement; or

b) of the outcome of the performance of the invoked operation; such report is referred to as return.

1.6.1.7.48 return: A report, from the performer to the invoker, of the outcome of the performance of the operation.

a) A return with a negative result is also called a negative return.

b) A return with a positive result is also called a positive return.

1.6.1.7.49 service-original procedure: A procedure that is defined specifically for a particular service on the basis of one or more of the operations specified in this Recommended Standard. A service-original procedure is not derived from any other procedure.

1.6.1.7.50 service instance provision period: The time during which a service is scheduled to be provided by this instance.

1.6.1.7.51 service management parameter: A configuration parameter of a CSTS for which the initial value is set by Service Management methods, for example, through a Utilization Request (see reference [I6]). Some CSTS configuration parameters are explicitly defined to be service management parameters by this Recommended Standard. In addition, a service using a procedure or a derived procedure may specify some or all of the procedure configuration parameters (see 1.6.3.2.6) of that (derived) procedure to be service management parameters. For each procedure configuration parameter that the service using the procedure/derived procedure specifies as being a service management parameter, Service Management will set one value for that procedure configuration parameter, which will apply to every instance of that (derived) procedure.

NOTE – All procedure configuration parameters specified to be service management parameters also have to be part of the parameters of the functional resource types forming the service production of the given CSTS type or the functional resource type representing the service instance of the given CSTS type. However, in the functional resource specification such parameter can be a component of a complex functional resource parameter. The CSTS specification has to state how the functional resource parameters are related to procedure configuration parameters being specified to be service management parameters.

1.6.1.7.52 service package: The set of CSTS instances, which may be of different service types, together with the specification of the characteristics of the production of those CSTS instances that are provided by one ESLT to one or more Earth User CSSS as agreed between the UM function of the Earth User CSSS and the PM function of the Provider CSSS.

1.6.1.7.53 started procedure: A procedure placed in the ‘active’ substate by a successful START operation.

NOTE – ‘started’ applies to a stateful CSTS procedure that has START and STOP operations. A started procedure is placed in the ‘active’ substate by a successful START operation and remains active until the procedure is (a) placed in the ‘inactive’ substate by a successful STOP operation, (b) terminated in response to a protocol error, or (c) terminated by the Association Control procedure.

1.6.1.7.54 stateful procedure: A procedure with two substates: ‘inactive’ and ‘active’.

1.6.1.7.55 stateless procedure: A procedure without substates.

1.6.1.7.56 three-phase operation: An operation that requires the performer to return to the invoker an initial acknowledgement of the acceptance of the invocation in addition to the report of the execution of the operation.

1.6.1.7.57 two-phase operation: An operation that requires the performer to return to the invoker the report of the execution of the operation.

1.6.1.7.58 unconfirmed operation: An operation for which the performer does not return a report of its outcome to the invoker.

1.6.2 NOMENCLATURE

1.6.2.1 Normative Text

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

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.

1.6.2.2 Informative Text

In the normative sections of this document, informative text is set off from the normative specifications either in notes or under one of the following subsection headings:

– Overview;

– Background;

– Rationale;

– Discussion.

1.6.3 CONVENTIONS

1.6.3.1 Specification of Operations

1.6.3.1.1 General

Section 3 of this Recommended Standard specifies the common operations that can be used by CSTSes. The specification of each operation is divided into subsections as described in 1.6.3.1.2 and 1.6.3.1.3.

1.6.3.1.2 Behavior Subsection

The Behavior subsection provides a brief description of the operation. Additionally, it indicates whether the operation may be invoked by the service user, service provider, or both; and whether there are any constraints on when the operation may be invoked.

NOTE – Whether the operation is confirmed, unconfirmed, or acknowledged is specified in the title of the section specifying the operation.

1.6.3.1.3 Invocation, Response, and Parameters Subsection

The Invocation, Response, and Parameters subsection describes the parameters associated with each operation, including their semantics. A table accompanying the description of each operation lists all parameters associated with the operation and, for the invocation, the acknowledgement (if applicable) and the return, and whether the parameter is always present, always absent, or conditionally present.

For parameters that are conditionally present, the parameter description specifies the conditions for the presence or absence of the parameter. The condition is generally based on the value of another parameter in the same invocation, acknowledgement, or return; for example, in the return of an operation, the `diagnostic` parameter is present if and only if the value of the `result` parameter is 'negative'. For a conditional parameter in a return, the condition may be based on the value of a parameter in the corresponding invocation.

In the table, the following convention is used to indicate whether a parameter is always present, always absent, or conditionally present:

a) M Mandatory, that is, always present;

b) C Conditionally present;

c) Blank Always absent.

NOTE – Even though a parameter is characterized as always present, its description may specify that its value may be left unspecified. Given that in ASN.1 a parameter value not set to a specific value is mapped to the NULL type (see annex F), this by convention in this document is expressed as this parameter having the value 'null'.

1.6.3.2 Specification of Procedures

1.6.3.2.1 General

Section 4 of this Recommended Standard specifies the procedures that can be used by CSTSes. The specification of each procedure is divided into subsections as described in 1.6.3.2.2 through 1.6.3.2.6.

Each procedure section follows a common template covering a descriptive and a prescriptive part.

1.6.3.2.2 Version Number Subsection

The version number of the procedure is identified in the Version Number subsection and is used by the service using that procedure.

1.6.3.2.3 Discussion Subsection

The descriptive part covers the purpose and the concept of the procedure. This part provides the reader with an overview of the procedure.

1.6.3.2.4 Behavior Subsection

The prescriptive part introduces the sequence of activities that describes the behavior of the procedure, for example, starting, running, and ending the activities. The operations required by the procedure are listed, and for each of them, the procedure identifies those operations that are extended and/or refined by the procedure. For those operations that require extension or refinement, the details are provided. For those operations that do not require extension or refinement, the operations are not repeated as they fully conform to the common definition of the common operations (see section 3) or to the specification of the procedure from which the described procedure is derived (e.g., the Cyclic Report procedure is derived from the Unbuffered Data Delivery procedure).

The prescriptive part ends with the service provider state transition table applicable to the procedure.

1.6.3.2.5 Required Operations Subsection

This Required Operations subsection identifies the operations used by the procedure.

1.6.3.2.6 Configuration Parameters Subsection

This Configuration Parameters subsection lists the parameters that need to be configured in the context of this procedure.

This subsection also identifies those parameters that may be accessed (read) by the service user of any service that includes a procedure that contains a GET operation. For each configuration parameter, this subsection provides cross references to the use of the parameter in the specification of the procedure and also identifies the Parameter Identifier to be used in reporting the value of the parameter (see 1.6.1.7.33).

Furthermore, this subsection specifies which, if any, of the configuration parameter values may be dynamically modified. The modification of such parameters can be performed at any time while the procedure instance for the configuration to be modified exists, including while the service that is executing the procedure is bound. If one or more configuration parameters are dynamically modifiable, the subsection identifies the event that is used to notify (by means of the NOTIFY operation) when such a modification has occurred.

1.6.3.2.7 Procedure State Table Subsection

Each stateful procedure is complemented with a state transition table describing the behavior of the service provider implementing the procedure. The state tables follow the following rules:

a) The state tables specify operation interactions and state transitions for the procedures on the service provider side in its role as either invoker or performer.

b) The leftmost column simply numbers the rows of the table.

c) The second column of the state table lists all incoming events. Where these events correspond to the arrival of an incoming PDU, the ASN.1 type defined for this PDU in annex F is indicated in parentheses (( )).

d) The description of an event that is internal to the service provider is put in single quotation marks (‘ ’).

e) The following columns (one column per state) on the right side of the table specify the behavior the service provider will exhibit, which depends on the current state and the incoming event. In some cases, the behavior additionally depends on Boolean conditions, also referred to as predicates. Such conditions are put in double quotation marks (“ ”). The predicates are defined in a table following the procedure state table. Predicates that are simple Boolean variables set only by that state machine itself are referred to as Boolean flags. The dependency on a predicate is presented in the form of an IF <condition> THEN <action> [[ELSEIF <condition> THEN <action>] ELSE <action>] ENDIF clause.

f) If the action given in the table is simply to send a specific PDU, that is indicated by the appearance of the name of the ASN.1 type of the PDU to be sent in parentheses (( )). If that PDU is a return or an acknowledgement, the name may be preceded by the plus symbol (‘+’) to indicate that `result` is ‘positive’ or by the negative symbol (‘−’) to indicate that `result` is ‘negative’. If several actions are to be taken (referred to as a ‘compound action’), the name of the compound action is put in curly braces ({ }). The individual actions making up each compound action are identified in a table following the procedure state table. If a simple action is to be taken, the name of the action is put in single quotes (‘ ’).

g) ‘Not applicable’ is stated when the given event can only occur in the given state because of an implementation error on the service provider side.

h) If the consequences of an incoming event are not visible to the service user because the service provider does not send any PDU in reaction to the given event, the action is put in square brackets ([ ]).

i) State transitions are indicated by an arrow and the number of the state that will be entered; for example, ‘→ 1’ indicates the transition to state 1.

j) The actions to be taken and the state transition are considered to be one atomic action. The sequence in which the actions shown in the table are executed is irrelevant except that the sending of PDUs shall be performed in the sequence stated in the table.

k) Whenever the service user invokes a confirmed operation with `invoke-id` set to <n>, it is expected to start an associated response <n> timer with the value specified by the `response-timeout` functional resource configuration parameter (see 3.2.1.2). Should this timer expire before the response <n> is received, the service user is expected to invoke PEER-ABORT with the `diagnostic` set to 'response timeout'.

l) Whenever a procedure is derived from another procedure (also called parent procedure), the following applies:

   1) the derived procedure's state table contains at least the same incoming events as the parent procedure, but may have additional ones;

   2) whenever the behavior in a given state and for an incoming event is fully identical to that of the parent procedure, its description is copied from the parent procedure, but the text is italicized to highlight that the behavior is fully inherited from the parent procedure;

   3) whenever the behavior in a given state and for an incoming event is not fully identical to that of the parent procedure,

      – the behavior copied from the parent procedure is written in italic typeface, and

      – the behavior specific to the derived procedure is written in plain text;

   4) the same convention applies to all tables directly associated with the state table.

1.6.3.3 Typographic Conventions

1.6.3.3.1 Operation Names

Names of service operations appear in uppercase.

1.6.3.3.2 Procedure Type Naming

The strings used to identify procedure types appear with initial capital letters (e.g., Buffered Data Delivery).

1.6.3.3.3 Parameter Names

In the main text, names of parameters of service operations generally appear in lowercase and are typeset in a fixed-width font (e.g., `responder-port-identifier`). The same convention is applied to parameters that are not service operation parameters but are elements of a complex type parameter that is a service operation parameter. In annex F, the corresponding name is formed by omitting any hyphens contained in the name and using mixed-case (e.g., `responderPortIdentifier`).

1.6.3.3.4 Value Names

The values of many parameters discussed in this Recommended Standard are represented by names. In the main text, those names are shown in quotation marks (e.g., 'no such service instance'). The corresponding name in annex F is formed by omitting any hyphens or white space contained in the name and using mixed-case (e.g., `noSuchServiceInstance`). The actual value associated with the name is constrained by the type of the parameter taking on that value. Parameter types are specified in annex F of this Recommended Standard.

NOTE – The name of a value does not imply anything about its type. For example, the value 'no such service instance' has the appearance of a character string but might be assigned to a parameter whose type is 'integer'.

1.6.3.3.5 State Names

This Recommended Standard specifies the states of service providers. States may be referred to by number (e.g., state 2) or by name. State names are always shown in single quotation marks (e.g., 'active').

1.6.3.3.6 PDU Names

The names of PDUs appear in mixed-case (e.g., BindInvocation).

1.6.3.3.7 Data Type Definitions

Data type definitions are presented in annex F in the form of a set of ASN.1 modules. Regardless of the conventions used elsewhere in this Recommended Standard, the text of the ASN.1 modules is typeset entirely in a fixed-width font.

1.6.3.3.8 Normative Mechanism and Data Structure Names

The names of normative mechanisms and data structures appear with initial capital letters (e.g., CSTS, Functional Resource Name, Published Identifier). Such mechanisms and data structures have defined roles in the functioning of operations and procedures.

1.7 REFERENCES

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

NOTE – A list of informative references is provided in annex I.

[1] *Cross Support Reference Model—Part 1: Space Link Extension Services*. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 910.4-B-2. Washington, D.C.: CCSDS, October 2005.

[2] *Space Link Extension—Internet Protocol for Transfer Services*. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 913.1-B-2. Washington, D.C.: CCSDS, September 2015.

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

[4] *Information Technology—Abstract Syntax Notation One (ASN.1): Specification of Basic Notation*. 4th ed. International Standard, ISO/IEC 8824-1:2008. Geneva: ISO, 2008.

[5] *Time Code Formats*. Issue 4. Recommendation for Space Data System Standards (Blue Book), CCSDS 301.0-B-4. Washington, D.C.: CCSDS, November 2010.

[6] *CCSDS SANA Registry Management Policy*. Issue 2. CCSDS Record (Yellow Book), CCSDS 313.1-Y-2. Washington, D.C.: CCSDS, October 2020.

[7] *Space Assigned Numbers Authority (SANA)—Role, Responsibilities, Policies, and Procedures*. Issue 3. CCSDS Record (Yellow Book), CCSDS 313.0-Y-3. Washington, D.C.: CCSDS, October 2020.

2 DESCRIPTION OF CROSS SUPPORT SERVICES

2.1 OVERVIEW

Cross Support Transfer Services provide for reliable, access-controlled transfer of spaceflight mission related data between ground element entities. A Cross Support Service is characterized by the kind of data it transfers (e.g., telemetry data, tracking data, service production monitoring data), and therefore different CSTSes need to respond to specific requirements that may demand specific solutions. On the other hand, all CSTSes defined by CCSDS apply the same basic communications patterns in order to simplify specification, implementation, and operation of these services. The basic approach applied to CSTSes is that they are realized through invocation and performance of operations in accordance with well-defined procedures.

This Recommended Standard provides a Specification Framework for CSTSes with the objective to

a) maximize the commonality between CCSDS CSTSes;

b) simplify specification of new CCSDS CSTSes; and

c) enable the design and implementation of reusable software components with the potential of simplifying the implementation of CSTSes.

NOTE – While this specification strives for enabling the design and implementation of reusable software components, it does not intend to provide a specification for such components.

This Recommended Standard defines basic building blocks from which services can be constructed. The companion Recommended Practice, *Guidelines for Specifications of Cross Support Transfer Services* (reference [I3]), defines the rules to be used for the specification of CSTSes.

Building blocks defined by the CSTS Specification Framework include

a) data types for information exchanged between service user and service provider;

b) common operations that are invoked by one entity and performed by the other entity and as such implement the basic elements of interaction between service user and service provider; and

c) common procedures that define the behavior and protocol for the invocation and performance of a set of operations to achieve a well-defined objective.

A CSTS may be defined by combining a set of procedures specified in the CSTS Specification Framework in a manner that best suits the objective of the service.

To be generally applicable, parameters, operations, and procedures defined by the CSTS Specification Framework are sufficiently abstract that they may or may not be directly usable for real services. For cases in which the building blocks are not directly usable, new operations or procedures can be derived from those defined in the CSTS Specification Framework.

Taking the concept of generic definitions one step further, this Recommended Standard supports the concept of abstract services (see reference [I5]) that specify the common operations and behavior of CSTSes but are not directly implementable because service-specific specifications are still missing. Real CSTSes can be derived from such abstract services.

NOTE – An example of an abstract service could be a CSTS Return Link Service from which real services such as CSTS RAF, CSTS RCF, and CSTS ROCF could be derived. These CSTSes would be equivalent to SLE RAF, SLE RCF, and SLE ROCF.

The relationships between the elements of the CSTS Specification Framework and example CSTSes based on the Framework are illustrated in figure 2-1.

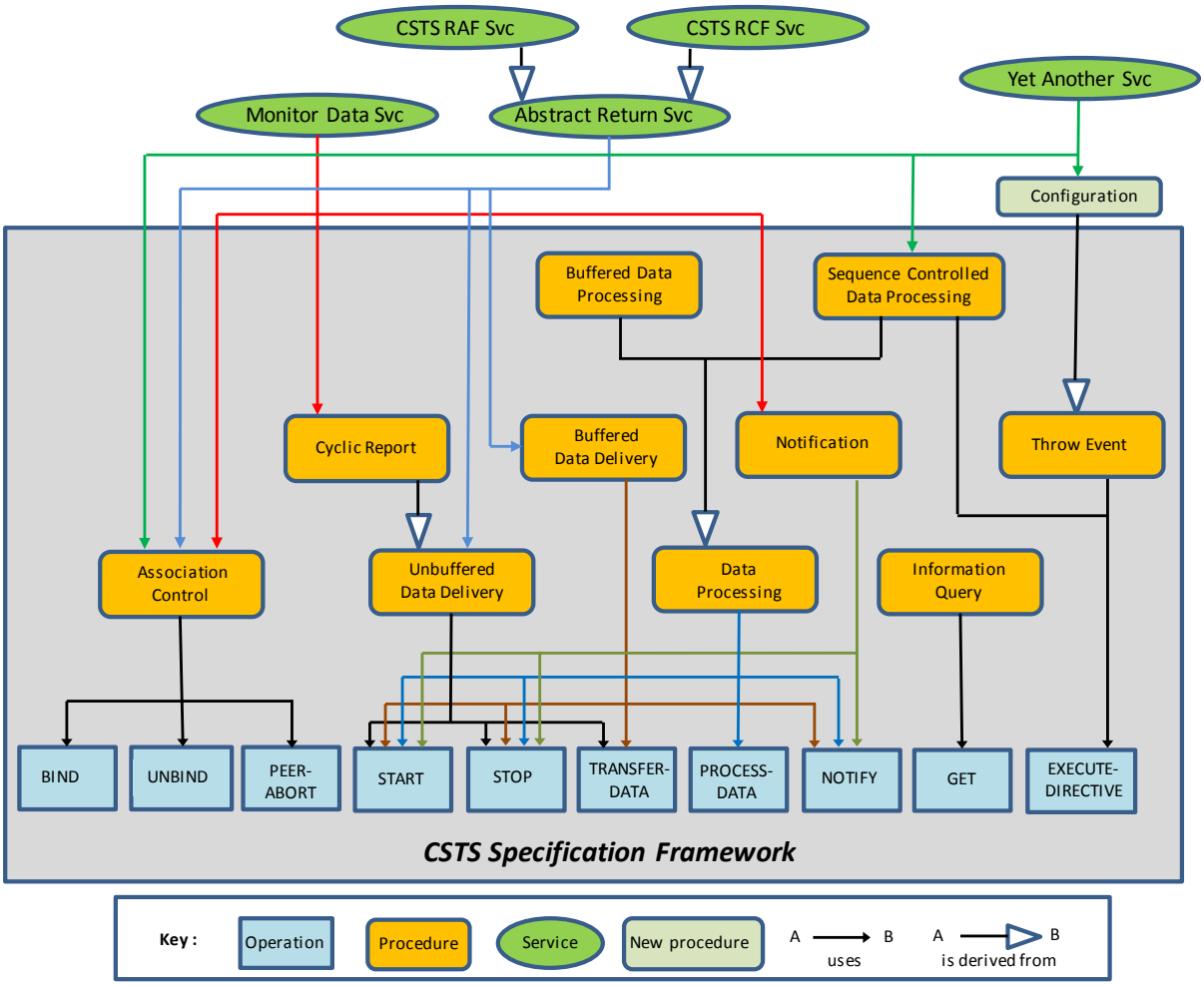

Figure 2-1: CSTS Specification Framework Concept

2.2 CROSS SUPPORT REFERENCE MODEL

2.2.1 INTRODUCTION

CCSDS Cross Support Transfer Services are defined within a framework that is a logical extension of the Cross Support Reference Model (reference [1]) that was originally defined for Space Link Extension (SLE) services. The following subsections summarize concepts from the reference model that are supported by the CSTS Specification Framework and to which later sections in this Recommended Standard refer. Formal definitions of terms are provided in 1.6.

Before a Cross Support Transfer Service can be used, an association needs to be established between service user and service provider by binding the ports associated with the specific CSTS. To that end, the initiator issues the request to bind to the responder. If the responder can complete the binding, the required association is established and service user and service provider can invoke and perform the operations of the given CSTS type. Depending on the type of operation, the performer may or may not report the result of the operation to the invoker.

2.2.2 SERVICE PRODUCTION AND PROVISION

2.2.2.1 General

The Cross Support Reference Model (reference [1]) distinguishes between service production and service provision.

Cross Support Service production refers to the common processes performed by service production of an ESLT associated with the provision of one or more Cross Support Services.

An ESLT is said to **provide** a service when it makes available to the service user the capability to obtain the service via one or more of its ports. Provision involves the interface between the service user and the service provider, and is characterized by the type(s) of data transferred and the quality of service with which they are transferred (e.g., completely, reliably, etc.).

The CSTS Specification Framework deals primarily with service provision. Service production is generally very specific for specific services or a class of services, and therefore the CSTS Specification Framework makes only very general assumptions on service production.

2.2.2.2 Production Status

Each Functional Resource that represents a CSTS instance has a `production-status` parameter. Each CSTS specifies how the `production-status` value shall be determined based on the aggregate status of the Functional Resource Instances involved in the service production, each of which maintains a `resource-status` parameter that is accessible by the CSTS instance. This Recommended Standard defines the permissible values of `production-status` and the transitions between these values. The effects of these `production-status` value changes on the behavior of the procedures are specified in 4.3 to 4.12.

If absolutely necessary, the definitions of these `production-status` values may be refined by service specifications that are based on the CSTS Specification Framework. CSTS specifications may also define substates for one or more of the `production-status` values where appropriate.

Since `production-status` is a parameter of the Functional Resource representing the CSTS instance, the formal assignment of the parameter OID and the specification of the parameter data type is part of the specification of that Functional Resource. The SANA Functional Resources Registry (https://sanaregistry.org/r/functional_resources) specifies a production status data type that conforms to the production status specification in annex B. Every Functional Resource that represents a CSTS defines a `production-status` parameter that uses either the `production-status` data type specified in the SANA Functional Resource Registry or a derivation of that data type. Use of the same data type specification for all CSTS instances representing Functional Resources is strongly recommended.

Production status changes may be notified to the service user via any procedure that includes the NOTIFY operation (3.11), for example, the Notification procedure (see 4.11). As for any other parameter, the current value of the `production-status` parameter may be obtained using (a) the Cyclic Report procedure defined in 4.10, if that is supported by the service, or (b) the GET operation, if any procedure of the service (including the Information Query procedure) uses the GET operation.

2.3 SERVICE MANAGEMENT

For all CSTSes, service management determines the number and schedule of service instances to be provided, the resources required to enable those service instances, and the initial configuration of all service instances and their supporting resources. Configuration parameters that have their initial values set through Service Management are called service management parameters (see 1.6.1.7.51). Cross Support Service Management is the subject of separate CCSDS Recommended Standards (see references [I6] and [I7]).

Service management parameters may refer to service provision or to service production. Explicit specification of service management parameters in this Recommended Standard is confined to a small set of service provision parameters, including

a) the service initiator (i.e., the service user) and the service responder (i.e., the service provider);

b) the port at which the service is made available; and

c) the service instance identifier.

For some procedures, this Recommended Standard identifies procedure configuration parameters that are known to the service user and the service provider but the definition of how their values are determined is delegated to the service specification. The service specification may specify these values, or designate them to be service management parameters. Reference [I3] provides guidelines on how service management parameters are to be handled in derived service specifications.

For service management parameters that correspond to procedure configuration parameters, Service Management will configure all instances of a given procedure type being associated with a given service instance equally.

Some procedure configuration parameters of individual procedure instances may be updated dynamically at any time while the procedure instance exists, including the time while the service instance executing the procedure instances is bound. The dynamic modification of such procedure configuration parameters is outside the scope of Service Management and must be carried out within the CSTS instance itself, for example, through a Throw Event procedure that is part of that service. The particular method(s) by which dynamic modifications of procedure configuration parameters are performed is left to the definition of the services that use such dynamically modifiable procedure configurations. For each CSTS that has dynamically modifiable configuration parameters, the service specification specifies whether and how the current values of those dynamically modifiable procedure configuration parameters are made visible through parameters of the Functional Resource that represents that CSTS.

2.4 ELEMENTS OF THE CSTS SPECIFICATION FRAMEWORK

2.4.1 COMMON OPERATIONS

An operation is a task that the invoker requests the performer to execute and as such presents the basic interaction pattern between the service user and the service provider. The invocation of an operation may include parameters that specify further details of the task to be performed.

Operations are further classified by the number of interactions required to complete the operation:

a) An **unconfirmed operation** is invoked by the invoker and performed by the performer, but there is no report on the outcome of the operation.

b) For a **confirmed operation**, the performer provides a report (the '**return**') to the invoker on success or failure of the operation. Operation returns comprise an indication of success (`result` has the value 'positive') or failure (`result` has the value 'negative') and, in case of failure, a `diagnostic` that further specifies the reason for the failure. Confirmed operations are further classified as

   1) **two-phase operations**, for which the performer returns a single report to the invoker; and

   2) **three-phase operations**, for which the performer provides an initial response in the form of an **acknowledgement** when receiving the invocation and subsequently issues the return when the operation has completed. This type of operation is typically used for operations in which the task to be performed requires non-negligible time to complete. The acknowledgement confirms that the invocation has been received and understood and that the general preconditions for the operation are fulfilled.

NOTE – In labeling confirmed operations, the designation (CONFIRMED) identifies a two-phase confirmed operation, and (ACKNOWLEDGED) identifies a three-phase confirmed operation.

Confirmed operations are further classified as **blocking** or **non-blocking** with respect to the procedure that uses the operation. When a blocking operation has been invoked, no further operation of the same procedure instance can be invoked before the return to the invocation has been received. Non-Blocking (NB) operations do not impose any constraints with respect to invocation of further operations.

The CSTS Specification Framework defines standard operation headers for unconfirmed and confirmed operations and a set of common operations for use within CSTS. To be of general use, the operations of the CSTS Specification Framework are rather abstract and might not all be directly usable within a procedure used by a specific CSTS. However, all operations used in a procedure of a CSTS that conforms to this Recommended Standard are derived directly or indirectly from the operations specified herein applying the rules described in 2.5.

The common operations defined by the CSTS Specification Framework are listed in table 2-1. Section 3 of this Recommended Standard provides the detailed specification of these operations.

Table 2-1: Common Operations Defined by the CSTS Specification Framework

Operation	Invoked By	Purpose	Confirmed	
			Ack.	Ret
BIND	service user	Establishment of an association with the peer.	No	Yes
UNBIND	service user	Release of an association previously established by a BIND operation.	No	Yes
PEER-ABORT	service user or service provider	Notification to the peer that the local application detected an error that requires the association to be terminated.	No	No
START	service user	Request that the service provider start performing activities associated with the procedure using the operation.	No	Yes
STOP	service user	Request that the service provider stop performing activities associated with the procedure using the operation.	No	Yes
TRANSFER-DATA	service provider	Transfer of a data unit to the service user.	No	No
PROCESS-DATA	service user	Request that the service provider process the data received.	No	No/ Yes (see NOTE)
NOTIFY	service provider	Sending of a notification of an event to a service user.	No	No
GET	service user	Ascertainment of the value of (a) service parameter(s).	No	Yes
EXECUTE-DIRECTIVE	service user	Request that the service provider perform a predefined action.	Yes	Yes

NOTE – For the PROCESS-DATA operation, both an unconfirmed and a confirmed variant exist. It is at the discretion of the procedure using this operation to specify if the unconfirmed or confirmed variant is used.

2.4.2 COMMON PROCEDURES

Procedures define the protocol and the expected behavior for the invocation and performance of a set of operations to achieve a well-defined objective. A CSTS may be specified by assembling a given set of procedures to handle specific aspects of the service.

For instance, a service that forwards telemetry received on the space link can be broken down into a number of procedures that handle association establishment and release, actual transfer of telemetry data units, status reporting, and querying of the service configuration parameters. All of these procedures except for telemetry data transfer can also be used for a service that supports reception and radiation of telecommands, in which only a subset of the parameters in the reports differs.

The specification of a procedure includes

a) a description of the objective of the procedure, including all assumptions that are made with respect to the type of data to be transferred and the service production process (descriptive);

b) a definition of the behavior of this procedure (prescriptive);

c) a definition of the operations that constitute this procedure (prescriptive);

d) a description of behavior expected of the service provider supported by a service provider side state matrix, as applicable (prescriptive).

As far as the protocol is concerned, procedures can be stateful or stateless. A stateful procedure supports the states ‘inactive’ and ‘active’, with transitions between these states triggered by well-defined operation invocations or responses. Typically, the transition from ‘inactive’ to ‘active’ is triggered by a START invocation, and the transition from ‘active’ to ‘inactive’ is triggered by a STOP return. However, other transitions may exist. For instance, procedures using three-phase operations might transition from ‘inactive’ to ‘active’ with the operation invocation and back to ‘inactive’ with the final return.

NOTE – The Association Control procedure is an exception to the above rules. The Association Control procedure does not belong to any class (stateful or stateless) and is required to manage the association.

A CSTS will typically use more than one procedure in addition to the Association Control procedure and may require more than one procedure of the same type to be active at the same time, for example, to transfer different data streams concurrently. Therefore this Recommended Standard distinguishes between the **procedure type** and the **procedure instance**. The procedure type corresponds to the specification of the procedure or the supporting program that implements the procedure specification. A procedure of a given type can be instantiated once or several times as part of the instantiation of the service using this procedure. Different instances of the procedure are distinguished by a ‘procedure name’ that is assigned by service management when creating the service package and included in all operation invocation and response headers.

NOTE – Further information on how a service uses predefined procedures is provided in 2.5. Normative rules for this purpose are specified in reference [I3].

The CSTS Specification Framework defines a set of common procedures that cover typical tasks of a CSTS. In order to be commonly usable, the procedure definition and in particular the purpose and semantics specification are reasonably abstract, such that the specifications as provided by this Recommended Standard may sometimes not be directly usable. For those cases, reference [I3] specifies methods to derive more specific procedures from these common procedures.

Table 2-2 identifies the common procedures included in this CSTS Specification Framework and table 2-3 shows the operations that are used by these procedures. Section 4 of this Recommended Standard provides the detailed specification of these procedures.

Table 2-2: Common Procedures Defined by the CSTS Specification Framework

Procedure	Purpose	Class
Association Control	Establishment and release of an association between a service user and a service provider.	N/A
Unbuffered Data Delivery	Best effort transfer of bulk data, structured into data units, sent from the service provider to the service user in real-time delivery mode.	SF
Buffered Data Delivery	In-sequence transfer of bulk data, structured into data units, sent from the service provider to the service user. In real-time mode, low latency is given priority over data completeness. In complete mode, data completeness is given priority over low latency.	SF
Data Processing	Processing of individual data units in the sequence as sent by the service user to the service provider and reporting of processing progress.	SF
Buffered Data Processing	Processing of data units contained in buffers in the sequence as sent by the service user to the service provider and reporting of processing progress. Processing may be in complete mode at the expense of latency or in timely mode at the expense of data loss.	SF
Sequence-Controlled Data Processing	Processing with full flow control of individual data units in the sequence as sent by the service user to the service provider and reporting of processing progress.	SF
Information Query	Query by the service user of the value of one or more parameters related to the service provider or service production behavior.	SL
Cyclic Report	Periodic reporting of parameter values from the service provider to the service user.	SF
Notification	Notification of the service user by the service provider of events of interest to the service user.	SF
Throw Event	Signaling to the service provider the occurrence of an event requiring a configuration change action and report of the result of the action back to the service user.	SF
NOTE – ‘SF’ and ‘SL’ indicate ‘stateful’ and ‘stateless’, respectively.		

Table 2-3: Use of Operations by Common Procedures

	Association Control	Unbuffered Data Delivery	Buffered Data Delivery	Data Processing	Buffered Data Processing	Sequence-controlled Data Processing	Information Query	Cyclic Report	Notification	Throw Event
BIND	B									
UNBIND	B									
PEER-ABORT	NB									
START		B	B	B	B	B		B	B	
STOP		B	B	B	B	B		B	B	
TRANSFER-DATA		NB	NB					NB		
PROCESS-DATA				NB	NB	NB				
GET							NB			
NOTIFY			NB	NB	NB	NB			NB	
EXECUTE-DIRECTIVE						NB				NB

NOTE – 'B' and 'NB' indicate that the operation is 'Blocking' or 'Non-Blocking', respectively.

2.5 PRINCIPLES OF USING THE CSTS SPECIFICATION FRAMEWORK

2.5.1 OVERVIEW

The CSTS Specification Framework provides a set of reusable building blocks for the specification of CSTSes and defines general rules by which such building blocks can be extended and refined to match the specific requirements of a CSTS.

The specification of a CSTS can be constructed by composition of procedures that either are fully predefined within the CSTS Specification Framework or are derived from those defined in the Framework. Specification of a completely new service-specific procedure using common operations or operations derived from the common operations is permissible but is not recommended.

The rules for derivation of operations and procedures have been rigorously applied to the specification of the CSTS Specification Framework itself:

a) The common procedures specified in section 4 present the most abstract and general version; several procedures defined within the CSTS Specification Framework derive extended operations, which may be further derived by services using these procedures.

b) Some of the procedures defined within the CSTS Specification Framework are derived from more basic procedures also defined in the Framework. For instance, the Cyclic Report procedure is derived from the Unbuffered Data Delivery procedure.

The derivation rules can be further applied to operations and procedures that have been derived from the operations and procedures defined within the CSTS Specification Framework. It is envisaged that such operations and procedures might eventually be included in the Framework if they prove to be useful for more than one service type.

The normative specification of the rules for derivation of operations and procedures and for composition of services can be found in reference [I3]. The following subsections provide a brief summary.

2.5.2 DERIVATION OF OPERATIONS AND PROCEDURES

Derivation of operations and procedures from the building blocks defined in the CSTS Specification Framework encompasses two dimensions:

a) extension, by which new data or behavior are added to a specification; and

b) refinement, by which the range of values may be constrained or the semantics of parameters may be specialized.

Operations may be extended in the following ways:

a) New parameters may be added (syntax and semantics specification) to an operation invocation or response; unless explicitly specified otherwise, all operations defined in the CSTS Specification Framework can be extended in this way.

b) In some cases, additional values may be defined for an existing parameter (e.g., additional `diagnostic` values or additional types of notification).

c) In some cases, the format and content of a parameter may have to be specified by the service using the corresponding operation. This is necessary because, for some operations, there are parameters whose structure is unspecified.

Operations may be refined by narrowing the value range of a parameter or by specializing the meaning of a parameter. For instance, the `generation-time` parameter of the TRANSFER-DATA operation in the Buffered Data Delivery procedure may be refined for a procedure dealing with transfer of telemetry by stating that the `generation-time` is the Earth Receive Time of the telemetry frame being transferred.

Procedures may be extended in the following ways:

a) extension of the operations used by a procedure, as described above;

b) addition of operations defined in the CSTS Specification Framework or derived from operations defined in the Framework.

Procedures may be refined by narrowing the specification of the semantics associated with the procedure or by adding more detail to the behavior description or the state table. The refinement of a procedure does not imply the definition of a new procedure.

As an example, a Telemetry Delivery procedure could be derived from the Buffered Data Delivery procedure by extension and refinement of the operations START and TRANSFER DATA and by refinement of the semantics associated with the data to be transferred and the parameters of the operations. The extension of a procedure implies the definition of a new procedure (derived procedure).

2.5.3 SPECIFICATION OF SERVICES BY COMPOSITION OF PROCEDURES

A CSTS can be specified by composition of procedures defined in the CSTS Specification Framework, derived from procedures in the Framework or derived from an existing CSTS.

Every service includes the Association Control procedure (see 4.3). Only a single instance of the Association Control procedure exists throughout the lifetime of a transfer service instance.

In addition to the Association Control procedure a service may comprise any number of further procedure types and may use one or more instances of each procedure type. However, one single procedure instance is designated the **prime procedure** of the service. All other procedure instances are referred to as secondary procedures.

NOTE – The prime procedure (instance) can be of the same type as one or more secondary procedure instances.

The prime procedure (instance) of a service should reflect the primary purpose of the service. In practical terms it determines the state of the service and as such determines the orderly association release as detailed in 2.6.

2.6 PROTOCOL DESCRIPTION

2.6.1 STATES OF THE SERVICE PROVIDER

The service user and service provider of a CSTS can interact as soon as the associated service ports are bound by establishment of an association between them. Binding is achieved by the Association Control procedure that is included in every CSTS.

Once the service instance is bound, other procedures used by the service can be operated according to their specific protocol.

NOTE – This Recommended Standard does not specify any interdependencies between the procedures that constitute a service. However, the service specification may specify dependencies that must be observed, for example, with respect to the sequence in which procedures are started.

When the service instance is bound, the service state is further determined by the state of the prime procedure (instance). If the prime procedure is stateful, the state 'bound' has two substates: 'ready' and 'active'. When the prime procedure is 'inactive', the service instance is in the state 'bound.ready', and when the prime procedure is 'active', the service instance is in the state 'bound.active'. If the prime procedure is stateless, then the service state 'bound' has no substates.

A service instance having a stateful prime procedure can be unbound by the service user in an orderly manner only while in the state 'bound.ready', that is, while the prime procedure is 'inactive'. Service instances with a stateless prime procedure can be unbound at any time. Finally, a service instance may be aborted by either entity at any time by invoking the PEER-ABORT operation.

Any stateful secondary procedure has a state machine of its own but has no effect on the overall service instance state. This implies that the UNBIND operation can be invoked while one or more stateful secondary procedures are in the state 'active'. Therefore, for a stateful secondary procedure, unbinding has the same effect as PEER-ABORT.

Any stateless secondary procedure is unaffected by the overall service instance state. The operations of any stateless secondary procedure instance may be invoked whenever the service instance is in the 'bound.ready' or 'bound.active' state if the prime procedure of that service is stateful, or when the service instance is in the 'bound' state if the prime procedure of that service is stateless.

The overall relationship of the states of the service instance, the prime procedure, and secondary procedures is illustrated in figures 2-2 and 2-3 for a stateful and a stateless prime procedure, respectively.

NOTE – The diagrams in figures 2-2 and 2-3 do not present state diagrams in a formal sense, as they present different state machines in a synoptic view. The corresponding normative statements can be found in annex G.

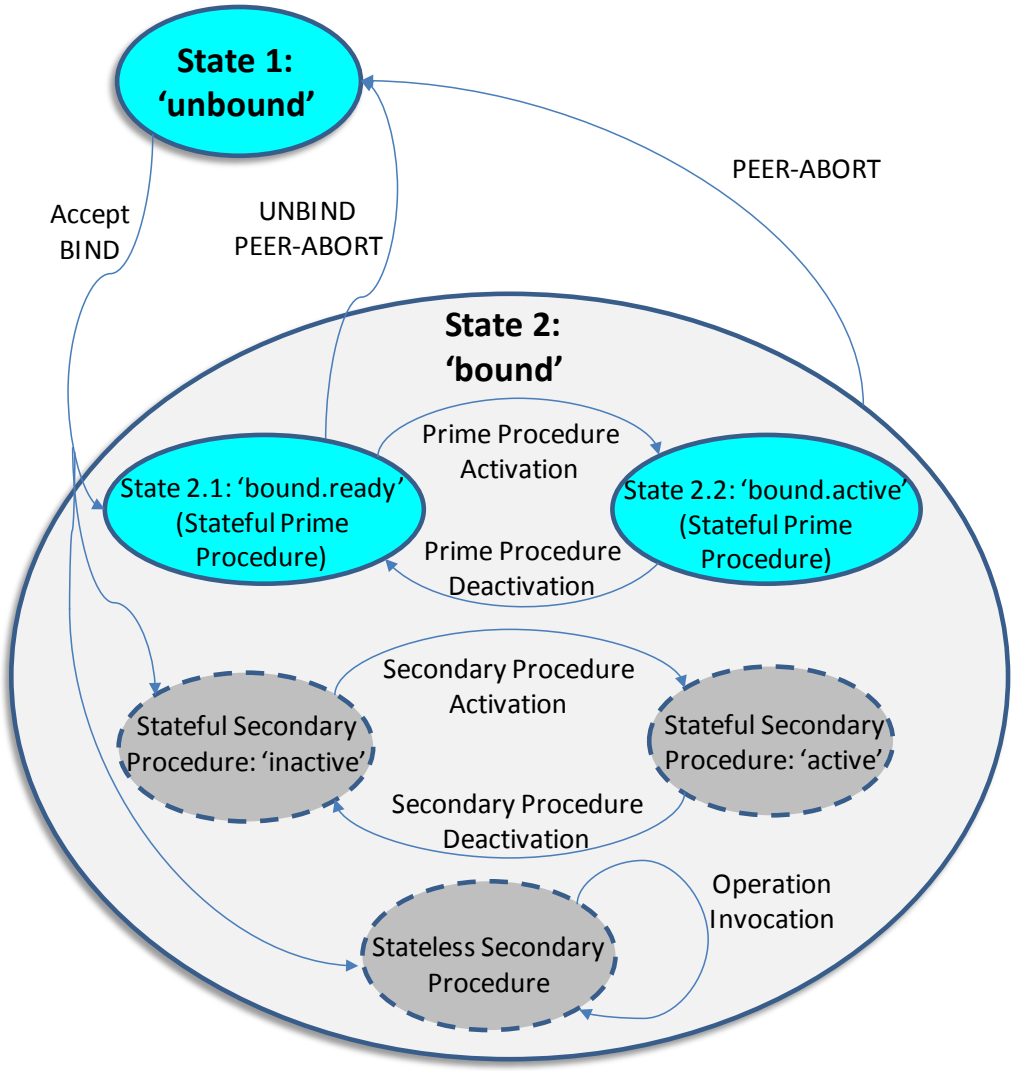

Figure 2-2: Service and Procedure States (Stateful Prime Procedure)

NOTE – The activation and deactivation of stateful procedures is further specified in 4.2.4.

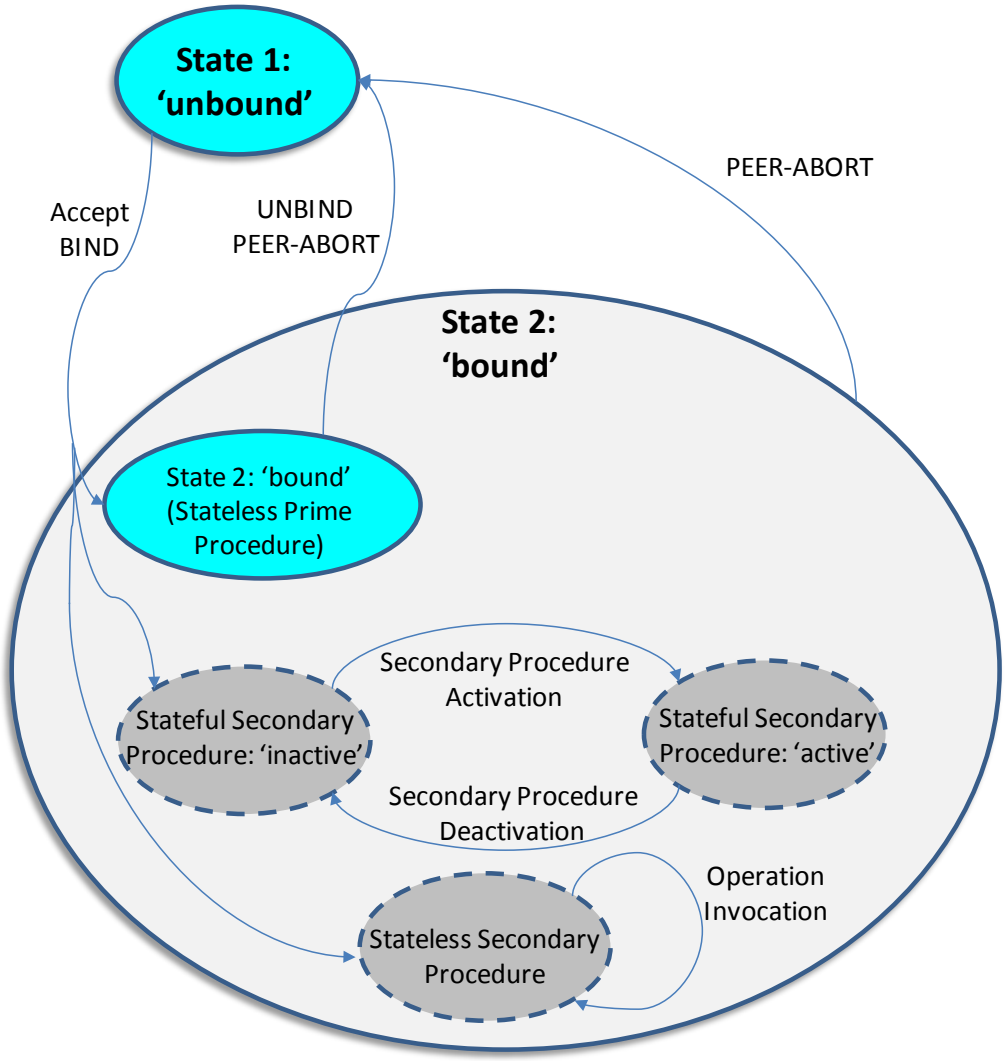

Figure 2-3: Service and Procedure States (Stateless Prime Procedure)

2.6.2 ASSOCIATION MANAGEMENT

Establishment and release of an association between the service user and the service provider is handled by the Association Control procedure specified in 4.3. All other procedures rely on the presence of such association and therefore establishment and release of the association must be coordinated between the procedure instances within a service instance.

An association is established by means of the BIND operation (see 3.4), which is always invoked by the service user. Before the BIND operation is completed, that is, the BIND return is issued by the service provider, no further operation can be invoked. The only exception to this rule is the PEER-ABORT operation, which can be invoked by the service user if the BIND return is not received quickly enough. Once the association is established, other procedures used by the service can be operated according to their defined protocol.

During the lifetime of the association, the prime procedure (instance) of the service instance and the Association Control procedure instance must cooperate to determine the applicable substate of the state 'bound' and the transitions between these states (see 2.6.1).

An association is released normally when an UNBIND operation is invoked by the service user (the initiator) and accepted by the service provider (the responder). An association may be aborted by either the service user or the service provider by means of the PEER-ABORT operation. An association may also be aborted because of a failure in the underlying communications system. Such failures are signaled to the local application entity by the 'protocol abort' event described in 4.3.3.1.11.5.

Conceptually, invocations of the operations UNBIND and PEER-ABORT will be received by the Association Control procedure, which will

a) verify the validity of the event according to the defined protocol and react accordingly;

b) inform all other procedures of the event;

c) respond to the event as defined by the protocol, for example, by issuing an UNBIND return; and

d) terminate the underlying communications connection, if applicable.

If a procedure needs to abort the association, for example, because of a protocol error, it forwards this request to the Association Control procedure, which will invoke the PEER-ABORT operation and inform all other procedures.

All procedures are expected to close down and release all resources associated with the service instance when they are notified of the termination event.

When an association is released or aborted, no further operations can be exchanged between the service user and the service provider. The systems may re-establish an association via a new BIND operation if that is consistent with the service instance provision period. However, status information from the prior association is not preserved unless specified differently by the service specification.

2.6.3 TECHNOLOGY SPECIFIC ASPECTS

This Recommended Standard defines building blocks for the specification of CSTSes. Provision of a CSTS in a real system also requires a specification of how the service is mapped to a communications service, such that all invocations and responses of the service operations can be conveyed between the service user and the service provider. In order not to restrict the applicability of service specifications based on this Recommended Standard to a specific communications technology, as few assumptions as possible have been made about the characteristics of the underlying communications service (see 1.3.1).

NOTE – While this Recommended Standard makes only a few assumptions on the underlying communications service, reference [2] specifies a communications service that is assumed to be used by default for CSTSes.

Elements of the CSTS interface between the service user and the service provider are specified in this Recommended Standard in terms of the operations that the service implements. These operations are realized by mapping the service operation invocations, acknowledgements, and returns to PDUs that can be conveyed by means of the underlying communications service. This Recommended Standard conceptualizes such mapping in two parts:

a) operation invocations, acknowledgements, and returns are mapped to CSTS PDUs defined in annex F;

b) CSTS PDUs are conveyed by means of an underlying communications service.

Typically, one CSTS PDU corresponds to the invocation, acknowledgement, or return of an operation. There is one exception in which multiple TRANSFER-DATA or PROCESS-DATA invocations are deliberately buffered and subsequently mapped to a single CSTS PDU (see 4.5.2.2.2.2 and 4.7.3.2.1). From the point of view of the service provider or service user application, the interaction between the service user and service provider is in terms of operations, but from the point of view of the application entities that implement the CSTS protocol, what is exchanged are CSTS PDUs.

The mapping of CSTS PDUs to an underlying communications service is intentionally outside the scope of this Recommended Standard. In order to achieve interoperability, the service user and service provider must conform not only to the service specification based on this Recommended Standard but also to an agreed-upon specification of the mapping of the service to the underlying communications service. The specification of a mapping of the service onto a particular communications service must address such points as

a) selection of communications network(s) to ensure connectivity;

b) compatible configuration of protocol stacks (e.g., timeout values);

c) specification of port identifiers and their translation onto the communications technology; and

d) specification of security-related information.

Figure 2-4 illustrates a communications realization of a CSTS that results from such a mapping. The specification of a suitable mapping for CSTS to the Transmission Control Protocol (TCP) and the ASN.1 is provided in reference [2].

The procedures specified in section 4 each have their individual state tables, giving the impression that the interaction between service user and service provider in the context of a given procedure is mutually independent of any concurrently executing procedures. However, this is not the case because all PDUs exchanged in the context of a given service instance share the same communications channel. The consequence is that, should one procedure create backpressure by not reading from the transport layer socket, this backpressure affects all PDUs being exchanged in the context of the given service instance, regardless of the procedure instance emitting the PDU.

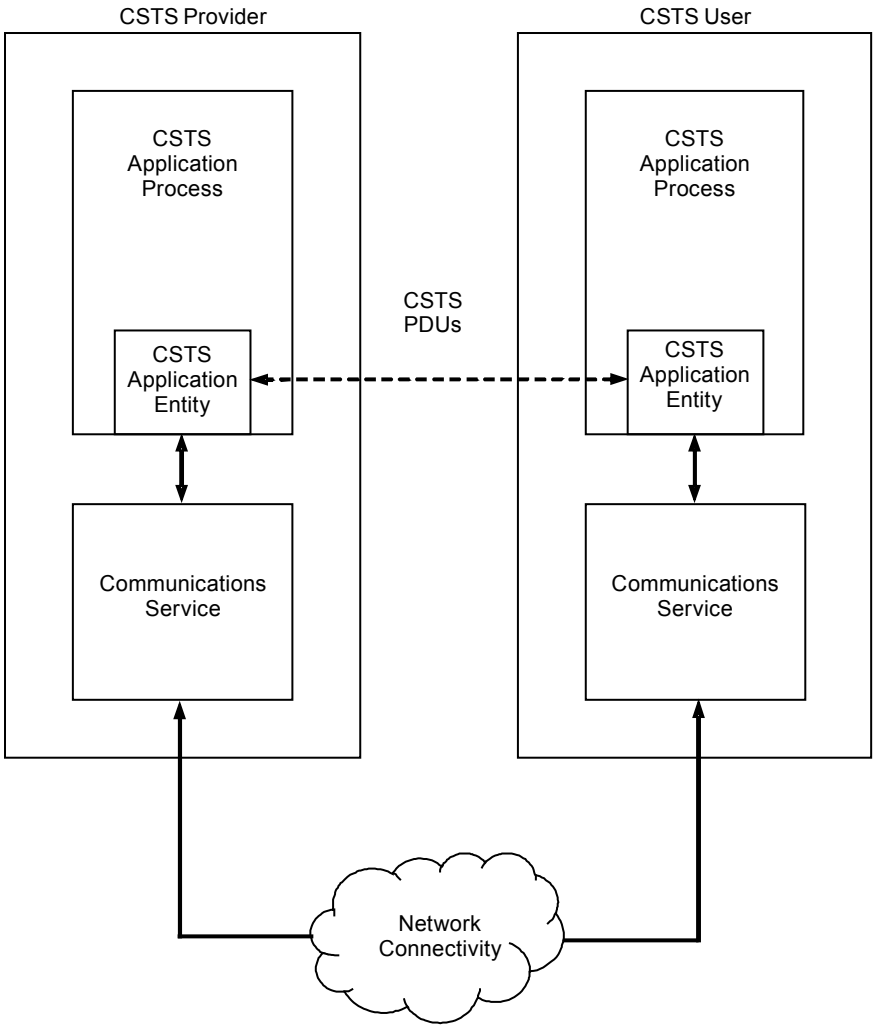

Figure 2-4: Communications Realization of a Cross Support Transfer Service

3 COMMON OPERATIONS

3.1 OVERVIEW

This section specifies the operations that are used by the procedures defined in this Recommended Standard. Subsection 3.2 and 3.3 specify behaviors that are generally applicable to all operations. Subsections 3.4 through 3.13 specify individual operations that are used by the procedures defined in this Recommended Standard.

3.2 GENERAL CONSIDERATIONS

3.2.1 COMMON OPERATION BEHAVIOR

NOTE – This Recommended Standard defines operations and procedures that are required for specifying Cross Support Transfer Services (CSTSes) when such services specify the behavior of the service providers. Regarding the service users, assumptions may be made regarding the expected behavior, but no binding specifications regarding the user behavior are provided. Subsections 3.2.1.1 to 3.2.1.3 specify invoker and responder behavior where the invoker may be either the service user or the service provider. To the extent that the user is the invoker, subsections 3.2.1.1 to 3.2.1.3 are to be understood to describe the expected user behavior, but not to be a binding specification of the user behavior.

3.2.1.1 All invokers of confirmed operations shall implement a timer for the acknowledgement, in the case of three-phase operations, or the return, in the case of two-phase operations. In the case that the timer expires before the expected acknowledgement or return is received, the invoker shall issue a PEER-ABORT with the `diagnostic` set to 'response timeout'.

3.2.1.2 Each functional resource representing a CSTS instance shall have a configurable `response-timeout` parameter the value of which determines the setting of the timers specified in 1.6.3.2.7 k), for which the same setting is applicable to all confirmed operations regardless of the operation being invoked by the service user or service provider.

3.2.1.3 On reception of the acknowledgement in the case of three-phase operations or the return in the case of two-phase operations, the invoker of the corresponding confirmed operation shall stop the timer.

NOTES

1 This issue of this Recommended Standard does not specify any confirmed operation being invoked by the service provider. Therefore such a PEER-ABORT operation will never be invoked by the service provider but it may be received from the service user. Future issues of the Recommended Standard may include provider-invoked confirmed operations. Also, certain CSTSes may add provider invoked confirmed operations by means of extension (see 1.6.1.7.19).

2 For all user-invoked confirmed operations (for example BIND and UNBIND) the service provider may use the `response-timeout` parameter value to anticipate how long it (the service provider) has to respond to the invocation of a confirmed operation to avoid causing the service user to invoke a PEER-ABORT.

3.2.1.4 All acknowledgements and returns shall include a `result` parameter that indicates whether the outcome of the operation was successful (`result` has the value 'positive') or unsuccessful (`result` has the value 'negative').

3.2.1.5 If the `result` reported in an acknowledgement is 'negative', no return shall be generated.

3.2.1.6 If `result` reported in an acknowledgement or return is 'negative', the acknowledgement or return shall also include a `diagnostic` parameter, the value of which is descriptive of the reason for the negative result.

3.2.1.7 For certain values, the `diagnostic` parameter shall be complemented by

a) a visible string whose value shall start with the diagnostic name identified in this Recommended Standard or in the service definition using this Recommended Standard; and

b) additional diagnostic information whenever required by this Recommended Standard.

NOTES

1 Possible values of the `diagnostic` parameter are listed in the specification of each operation.

2 The visible string diagnostic is for troubleshooting purposes, in order to allow implementers to add information to the diagnostic and to accommodate service-type or implementation-specific information. The value of the string, apart from the diagnostic name, shall be chosen by the implementers or defined in the CSTS specification.

3 The presence or absence of a visible string complementing the diagnostic value is specified in annex F.

3.2.1.8 The `diagnostic` parameter value may be extended by derived operations, procedures and services to support dedicated `diagnostic` values.

3.2.1.9 Unless otherwise specified, all operation invocations, acknowledgements, and returns defined in this Recommended Standard can be extended.

NOTE – The extension capability is not explicitly mentioned in the rest of the document and is to be considered implicit for all operations.

3.2.2 PARAMETER TYPES

The types of all parameters shall conform to the abstract syntax specified in annex F.

NOTES

1 Some parameter types in annex F are chosen such that an extension is possible. For example, the `diagnostic` parameter can be extended so that a service can define its own `diagnostic` values.

2 The syntax specified in annex F ensures the possibility of extension by defining fields that can be, whenever required, defined externally to the proposed syntax (e.g., operations extension).

3.2.3 PARAMETER CHECKING AND PDU VALIDATION

3.2.3.1 Validity checks shall be performed on the values of parameters associated with an operation.

NOTE – Rules governing the validity of parameter values are included in the specification of individual operations. General reasons for regarding a parameter value as invalid are specified in the following paragraphs.

3.2.3.2 A parameter value shall be treated as invalid if it is outside the range or not in the set of values permitted by the operation using the given parameter.

NOTE – A conforming implementation is capable of supporting the full range or set of values as specified in annex F and applicable to the service using this Recommended Standard.

3.2.3.3 A parameter value shall be treated as invalid if it is in conflict with the value of another parameter in the same invocation.

NOTE – For example, the value of the `start-time` parameter in the invocation of an operation is invalid if it is later than the value of the `stop-time` parameter.

3.2.3.4 If a parameter value is not valid, the operation shall not be performed and, for confirmed operations, a report of the negative result shall be returned to the invoker.

3.2.3.5 While this Recommended Standard does not prescribe the sequence of checks for the parameters, the implementer shall document the implemented sequence.

3.2.3.6 A PDU shall be treated as invalid if

a) it contains an unrecognized operation type or a parameter of the wrong type;

b) it is otherwise not decodable;

c) the procedure identifier is not one of the specified service procedures identifiers;

d) the incoming PDU violates the procedure's state table, that is, the PDU is not permitted in the present procedure state;

e) the value of the `invoke-id` parameter is the same as the `invoke-id` value of another operation still being performed; or

f) procedure-type-specific constraints apply that the incoming PDU violates.

NOTE – An example of such a constraint is the maximum number of PROCESS-DATA invocations contained in the forward buffer PDU that the Buffered Data Processing procedure is configured to accept (refer to 4.7.3.2.1.4).

3.2.3.7 In the case of an invalid PDU, the service provider shall abort the association using a PEER-ABORT with the corresponding `diagnostic` value (see 3.6.2.2.1).

3.2.4 AUTHENTICATION

NOTE – Requirements for security depend on the application and the system environment (e.g., whether closed or public networks are used or if access is only from physically restricted areas). In many environments, security may be provided by the communications service transparently to the application. This Recommended Standard does not preclude the use of security features that are provided by the communications service or the local environment, nor does it assume the availability of such features.

3.2.4.1 The service shall provide the following options with respect to the level of authentication of invocations and responses of operations:

a) 'all': all invocations, acknowledgments, and returns, except the invocation of PEER-ABORT, shall be authenticated;

b) 'bind': only the BIND invocation and return shall be authenticated;

c) 'none': neither invocations nor responses shall be authenticated.

3.2.4.2 The UM function of the Earth User CSSS and the PM function of the Provider CSSS shall agree on the level of authentication to be required for a given service instance and shall configure the service user and service provider accordingly.

3.2.4.3 The UM function of the Earth User CSSS and the PM function of the Provider CSSS shall agree on the algorithm used to generate and check credentials parameters and shall make this algorithm known to the service user and service provider together with associated parameters such as passwords or keys as necessary for the selected algorithm.

NOTES

1 The specification of the algorithms themselves is outside the scope of this Recommended Standard. However, the SLE Internet Protocol for Transfer Services (ISP1, reference [2]), which is the default underlying communication protocol for CSTSes, specifies a particular algorithm for the generation of credentials. Any implementation that uses ISP1 will use the credentials algorithm specified therein.

2 The `initiator-identifier` and `responder-identifier` parameters of the BIND operation identify the service user and service provider and therefore the applicable authentication level and algorithm necessary to generate and check credentials.

3.2.4.4 For operations for which authentication is required by the terms of the agreement between PM and UM of the CSSSes,

a) invocations shall include an `invoker-credentials` parameter to permit the performer to authenticate the invocation; and

b) responses shall include a `performer-credentials` parameter to permit the invoker to authenticate the response.

3.2.4.5 For operations for which authentication is not required, the `invoker-credentials` and `performer-credentials` parameters should be set to the value 'unused' to signify that the invocation or response does not carry credentials.

3.2.4.5.1 An incoming invocation, return, or acknowledgment shall be ignored if the credentials parameter cannot be authenticated when, by management arrangement, credentials are required.

3.2.4.5.2 If an invocation is ignored, the operation shall not be performed, and a report of the outcome shall not be returned to the invoker.

3.2.4.5.3 If a return is ignored, it shall be as if no report of the outcome of the operation has been received.

3.2.5 INVOKE IDENTIFIER

3.2.5.1 To support applications that may need to invoke several operations concurrently, the parameter `invoke-id` is specified for all operations.

NOTES

1 The `invoke-id` parameter allows the invoker to correlate a particular response to the invocation that prompted it.

2 Confirmed operations may be blocking or non-blocking. The choice is to be specified in the procedure definition. Unconfirmed operations are always non-blocking.

3.2.5.2 The value of the `invoke-id` parameter shall be an invoker-supplied arbitrary integer value that shall be returned, unchanged, by the performer.

3.2.5.3 In case the value of the `invoke-id` parameter is the same as the `invoke-id` of another operation that is still being performed within the context of the same service instance, the service provider shall issue a PEER-ABORT.

3.2.5.4 To ensure that the service behaves in a predictable manner, the effects of operations shall be as though the operations were performed in the order in which they were invoked.

3.2.6 BLOCKING AND NON-BLOCKING OPERATIONS

3.2.6.1 Invocation of an operation for the same procedure instance when a blocking operation response is outstanding constitutes a protocol error and shall lead to the service provider issuing a PEER-ABORT with the `diagnostic` parameter set to 'protocol error'.

NOTE – After invoking a blocking operation, the invoker will invoke a PEER-ABORT with the `diagnostic` parameter set to 'response timeout' if the response to the previously invoked blocking operation is not received in good time (see 3.6.2.2.1.3 g)).

3.2.6.2 After invoking an NB operation, invocation of another operation for the same procedure instance without waiting for the response from the first invocation is allowed.

3.2.6.3 Compliance with this Recommended Standard does not require the performer to process invocations concurrently; however, the performer must accept invocations from a non-blocking invoker and buffer and serialize them by local means not visible externally.

3.2.7 TIME

The time reference for all parameters containing a time value shall be based on Coordinated Universal Time (UTC).

NOTE – The type of all time parameters is specified in annex F.

3.3 STANDARD OPERATION HEADER

3.3.1 BEHAVIOR

3.3.1.1 All operation invocations, except the PEER-ABORT invocation, shall be defined with a common header containing the parameters specified in 3.3.2.

3.3.1.2 All operation responses shall be defined with a common header containing the parameters specified in 3.3.2.

3.3.1.3 A confirmed operation may be defined with an acknowledgement and a return or with a return only.

NOTE – While the acknowledgement and return PDUs may use the same syntax, the invoker of the operation can differentiate the acknowledgement and return PDUs by reading the tag at the beginning of the PDU (see annex F).

3.3.2 INVOCATION, ACKNOWLEDGEMENT AND RETURN PARAMETERS

3.3.2.1 General

Table 3-1 identifies the parameters that appear in the invocation, acknowledgement, and return of the Standard Confirmed Operation Header. Table 3-2 identifies the parameters that appear in the invocation of the Standard Unconfirmed Operation Header. The following subsections specify each of these parameters.

Table 3-1: Standard Confirmed Operation Header Parameters

Parameters	Invocation	Acknowledgement	Return
`invoker-credentials`	M		
`performer-credentials`		M	M
`invoke-id`	M	M	M
`procedure-name`	M		
`result`		M	M
`diagnostic`		C	C

Table 3-2: Standard Unconfirmed Operation Header Parameters

Parameter	Invocation
`invoker-credentials`	M
`invoke-id`	M
`procedure-name`	M

3.3.2.2 `invoker-credentials`

The `invoker-credentials` parameter shall provide information that enables the performer to authenticate the invocation (see 3.2.4). If authentication is not required, the `invoker-credentials` parameter shall be set to ‘unused’.

3.3.2.3 performer-credentials

The performer-credentials parameter shall provide information that enables the invoker to authenticate the response from the performance of the invoked operation (see 3.2.4). If authentication is not required, the performer-credentials parameter shall be set to 'unused'.

3.3.2.4 invoke-id

3.3.2.4.1 The value of the invoke-id parameter shall be an invoker-supplied arbitrary integer value (see 3.2.5).

NOTE – The presence of the invoke-id in the standard unconfirmed operation header is required to maintain commonality with the confirmed operation.

3.3.2.4.2 The performer shall insert unchanged the invoker-supplied value of the invoke-id parameter in each corresponding operation response.

3.3.2.5 procedure-name

3.3.2.5.1 The procedure-name shall consist of

a) the procedure-type (e.g., Buffered Data Delivery);

b) the procedure role:

   1) the role of the Association Control procedure is always association control; as there is only one instance of the Association Control procedure, the procedure role parameter is set to the 'associationControl' value;

   2) the role of any other procedure is either prime procedure instance or secondary procedure instance;

c) for a prime procedure, the 'primeProcedure' value;

NOTE – There is only a single instance of any prime procedure; consequently, the procedure instance number parameter in this case is set to the 'primeProcedure' value.

d) for secondary procedures, the procedure instance number parameter.

3.3.2.5.2 The procedure-type shall uniquely identify the type of the procedure.

3.3.2.5.3 The value of the procedure-name shall be the same for all operations used in the context of that procedure instance.

3.3.2.5.4 The procedure instance number in the `procedure-name` of secondary procedures shall be set to an incrementing number starting with 1. The value of the procedure instance number shall represent the count of the instances of the given secondary procedure type within one service instance.

NOTE – If there is more than one instance of the prime procedure type, only one instance is considered the prime procedure; the other instances are considered secondary procedures.

EXAMPLE – A service has three instances of the procedure type 'x' and three instances of the procedure type 'y'. One of the three procedure instances of 'x' is prime:

a) prime procedure 'x' instance number: not applicable;

b) secondary procedure 'x' instance numbers: 1 and 2;

c) secondary procedure 'y' instance numbers: 1, 2, and 3.

3.3.2.6 `result`

3.3.2.6.1 Definition

3.3.2.6.1.1 The `result` parameter of an operation acknowledgment shall specify the result of the related invocation and shall contain one of the following values:

a) 'positive'—the invocation has been accepted by the performer;

b) 'negative'—the invocation has not been accepted by the performer.

3.3.2.6.1.2 The `result` parameter of an operation return shall specify the result of an invocation and shall contain one of the following values:

a) 'positive'—the operation has been performed by the performer;

b) 'negative'—the operation has not been performed by the performer.

3.3.2.6.2 'positive'

All procedures shall have the possibility to extend positive responses with additional parameters, the names, types, and values of which shall depend on the procedure using that operation.

3.3.2.6.3 'negative'

If `result` is 'negative', a `diagnostic` parameter shall be present in the response.

3.3.2.7 `diagnostic`

3.3.2.7.1 If present (see 3.3.2.6.3), the `diagnostic` parameter value shall be one of the following:

a) 'invalid parameter value'—the value of one of the parameters provided is invalid (i.e., is not within the specified range); the `diagnostic` shall be complemented with the name of the invalid parameter (see 3.2.3.2);

b) 'conflicting values'—the value of one of the parameters is in conflict with the value of another parameter in the invocation (see 3.2.3.3);

c) 'unsupported option'—one or more of the options required by the invocation are not supported, for example, an optional procedure of a CSTS is not part of the given CSTS provider implementation;

d) 'other reason'—the reason for rejection of the operation will have to be found by other means.

3.3.2.7.2 An implementation using the `diagnostic` value 'other reason' in a negative acknowledgement or negative return shall document under which conditions this `diagnostic` value is used.

3.3.2.7.3 All procedures shall have the possibility to extend 'negative' with additional parameters, the names, types, and values of which shall depend on the procedure using that operation. In particular, additional values of the `diagnostic` parameter may be defined.

3.4 BIND (CONFIRMED)

3.4.1 BEHAVIOR

NOTE – Establishment of an association between service user and service provider can only be initiated by the service user by means of invoking the BIND operation.

3.4.1.1 The service provider shall return a report of the outcome of the performance of the BIND operation to the service user unless the BIND invocation is ignored because of invalid credentials (see 3.2.4).

3.4.1.2 The report sent by the service provider shall be a negative return if any of the conditions specified in 3.4.2.3.1 are met; otherwise a positive return shall be sent.

3.4.2 INVOCATION AND RETURN PARAMETERS

3.4.2.1 General

The parameters of the BIND operation shall be present in the invocation and return as specified in table 3-3.

Table 3-3: BIND Operation Parameters

Parameters	Invocation	Return
Standard Operation Header (confirmed)	*M*	*M*
`initiator-identifier`	M	
`responder-identifier`		M
`responder-port-identifier`	M	
`service-type`	M	
`version-number`	M	
`service-instance-identifier`	M	

3.4.2.2 Operation Parameters Definitions

3.4.2.2.1 Standard Confirmed Operation Header

This operation shall use the Standard Confirmed Operation Header (see 3.3).

3.4.2.2.2 `initiator-identifier`

3.4.2.2.2.1 The `initiator-identifier` parameter shall identify the authority on whose behalf the application entity is initiating an association.

3.4.2.2.2.2 The `initiator-identifier` parameter shall be a service management parameter.

3.4.2.2.3 `responder-identifier`

3.4.2.2.3.1 The `responder-identifier` parameter shall identify the authority on whose behalf the responding application is acting.

3.4.2.2.3.2 The `responder-identifier` parameter shall be a service management parameter.

3.4.2.2.4 `responder-port-identifier`

3.4.2.2.4.1 The `responder-port-identifier` parameter shall specify the port identifier of the responding application entity with which the initiator seeks to establish an association.

NOTE – The `responder-port-identifier` parameter also permits the use of particular kinds of gateways for which, because of different communications technologies at the peer entities, establishing a direct end-to-end communications channel is not possible or for which a direct end-to-end channel is not desired for security reasons. The `responder-port-identifier` parameter is used by such gateways to complete the association with the responding application entity, but it is not intended to be used by the responding application entity itself. Beyond this statement, the behavior of such gateways is outside the scope of this Recommended Standard.

3.4.2.2.4.2 The responding application entity shall ignore the value of the `responder-port-identifier` parameter for the purpose of determining the validity of the invocation.

3.4.2.2.4.3 The `responder-port-identifier` parameter shall be a service management parameter.

NOTES

1 The value of the `responder-port-identifier` parameter is a logical name that can be translated into the technology-specific addressing information required to establish a connection with the responder using the agreed-upon communications service. The CSTS application entity on the service user side needs to know this parameter so that it is properly populated for use by the port-designation mechanism of the underlying communications service. On the service provider side, the application entity completely ignores this parameter. However, it needs to be known on the provider side for the configuration of the underlying communications service.

2 PM and UM of the CSSSes must have previously agreed on the `responder-port-identifier` and its translation to the required communications-technology specific information that is applicable to a particular instance of service. The parameter value can conveniently be used beyond the port-designation mechanism as a pointer to a set of configuration parameters of the underlying communications service, such as the size of transmit and receive buffers.

3 The `responder-port-identifier` parameter is included in the BIND invocation for selecting the communications port on the responder side or to support its possible use by particular kinds of gateways. The responding application entity ignores its value.

4 In case the association between service user and service provider is established via a gateway and the value of the `responder-port-identifier` parameter is incorrectly set, the gateway will not be able to relay the BIND invocation to the target responding application entity. Likewise, if because of an incorrectly set `responder-port-identifier` parameter the target application entity is not listening on the correct communications port for an incoming BIND invocation, the BIND invocation will not be noticed and the application entity will not issue a BIND return. The lack of a return for the BIND operation is expected to cause the user to abort the association by invoking the PEER-ABORT operation with the `diagnostic` parameter set to ‘response timeout’ (see 3.6.2.2.1.3 g)).

3.4.2.2.5 `service-type`

The `service-type` parameter shall uniquely identify the type of service that will be provided if the BIND operation succeeds.

3.4.2.2.6 `version-number`

The `version-number` parameter shall identify the version number of the service specification that is to govern this association if the BIND operation succeeds.

3.4.2.2.7 `service-instance-identifier`

3.4.2.2.7.1 The `service-instance-identifier` parameter shall uniquely identify this service instance within the scope of the service-providing CSSS.

3.4.2.2.7.2 The `service-instance-identifier` shall consist of the following sequence:

a) an identifier of the spacecraft being supported by the given CSTS instance;

b) an identifier of the facility where the CSTS provider is located;

c) an identifier of the CSTS type;

d) the service instance number.

3.4.2.2.7.3 The ordering of the elements constituting the `service-instance-identifier` shall be significant. The order shall be: spacecraft identifier first, followed by the facility identifier, followed by the transfer service type, and ending with the service instance number.

3.4.2.2.7.4 The elements of the `service-instance-identifier`, being identifiers (spacecraft, facility, CSTS type), shall be defined as OIDs, while the service instance number shall be a positive integer.

3.4.2.2.7.5 The spacecraft supported by the given CSTS instance shall be specified by means of the spacecraft OID assigned in the CCSDS Spacecraft Registry (see H2.5).

3.4.2.2.7.6 The 'facility' where the CSTS provider is located shall be specified by means of an OID assigned in the CCSDS Service Site and Aperture Registry (see H2.5).

3.4.2.2.7.7 The CSTS type shall be identified by means of the OID assigned to the service type as specified in D5.

3.4.2.2.7.8 The `service-instance-identifier` parameter shall be a service management parameter.

NOTES

1 For all OIDs being elements of the `service-instance-identifier` parameter, SANA maintains registries in which these OIDs can be looked up.

2 For the convenience of human users, these registries may specify names in the form of strings associated with the various OIDs. However, these strings are not part of the BIND invocation PDU. They are used in the following example only to illustrate the elements of the `service-instance-identifier` parameter.

EXAMPLE:

- Spacecraft Identifier: 'XenoSat'
- Facility Identifier: 'DSNDSS5'
- CSTS type: 'monitoredData'
- Service Instance Number: 1

3.4.2.2.7.9 The service instance number shall be identical to the Functional Resource Instance Number of the Functional Resource Name assigned to the service instance.

3.4.2.2.8 BIND Invocation Syntax

The type `BindInvocation`, as defined in F3.5, shall define the syntax of the BIND invocation PDU and its parameters.

3.4.2.2.9 BIND Return Syntax

The type `BindReturn`, as defined in F3.5, shall define the syntax of the BIND return PDU and its parameters.

3.4.2.3 `diagnostic` Parameter Extension Value Definitions and Syntax

3.4.2.3.1 If a negative BIND return is sent, one of the `diagnostic` values specified in 3.3.2.7 or one of the following `diagnostic` values shall be used:

a) 'access denied'—the value of the `initiator-identifier` parameter is not recognized by the service provider (the value does not identify an authorized service user of any service instance known to the service provider);

b) 'service type not supported'—the value of the `service-type` parameter of the BIND invocation does not identify a service type supported by the service provider;

c) 'version not supported'—the service type version is not supported;

d) 'no such service instance'—the requested service instance is unknown to the service provider;

e) 'already bound'—the service instance is already bound via a different association;

f) 'service instance not accessible to this initiator'—the service user identified by the `initiator-identifier` parameter of the BIND invocation does not match the authorized initiator for the service instance identified by the `service-instance-identifier` parameter;

g) 'inconsistent service type'—the value of the `service-type` parameter of the BIND invocation is not a valid one, or the value of the `service-type` parameter does not match the service type of the service instance identified by the `service-instance-identifier` parameter;

h) 'out of service'—`production-status` is 'halted' (see annex B).

3.4.2.3.2 The type `AssocBindDiagnosticExt`, as defined in F3.5, shall specify the syntax of the `diagnostic` parameter of the BIND return, extended as listed in 3.4.2.3.1.

3.5 UNBIND (CONFIRMED)

3.5.1 BEHAVIOR

NOTE – Release of a previously established association between service user and service provider can only be initiated by the service user by means of invoking the UNBIND operation.

3.5.1.1 In case the service provider cannot accept the UNBIND invocation (i.e., 'duplicate invoke ID'), it shall abort the association by invoking PEER-ABORT.

3.5.1.2 Otherwise, the service provider shall return a positive report of the outcome of the performance of the UNBIND operation to the service user.

3.5.2 INVOCATION AND RETURN PARAMETERS

3.5.2.1 General

The parameters of the UNBIND operation shall be present in the invocation and return as specified in table 3-4.

Table 3-4: UNBIND Operation Parameters

Parameters	Invocation	Return
Standard Operation Header (confirmed)	*M*	*M*

3.5.2.2 Operation Parameters Definitions

3.5.2.2.1 Standard Confirmed Operation Header

This operation shall use the Standard Confirmed Operation Header (see 3.3) without extension.

3.5.2.2.2 UNBIND Invocation Syntax

The type `UnbindInvocation`, as defined in F3.5, shall specify the syntax of the UNBIND invocation PDU and its parameters.

3.5.2.2.3 UNBIND Return Syntax

The type `UnbindReturn`, as defined in F3.5, shall specify the syntax of the UNBIND return PDU and its parameters.

3.6 PEER-ABORT (UNCONFIRMED)

3.6.1 BEHAVIOR

NOTE – The service user can abort the association between service user and service provider by invoking the PEER-ABORT operation. The conditions under which this can happen are listed in 3.6.2.2.1.3.

3.6.1.1 The service provider shall invoke the PEER-ABORT operation to terminate unconditionally an association between a service user and a service provider if any of the conditions listed in 3.6.2.2.1.2 are met.

3.6.1.2 The PEER-ABORT operation shall not be extended.

3.6.1.3 On reception of a PEER-ABORT invocation, the service provider shall flush all queued data.

3.6.2 INVOCATION PARAMETERS

3.6.2.1 The parameters of the PEER-ABORT operation shall be present in the invocation as specified in table 3-5.

Table 3-5: PEER-ABORT Operation Parameters

Parameters	Invocation
`diagnostic`	M

3.6.2.2 Operation Parameters Definitions

3.6.2.2.1 `diagnostic`

3.6.2.2.1.1 The `diagnostic` parameter shall specify why the PEER-ABORT is being invoked.

3.6.2.2.1.2 The value of the `diagnostic` parameter in a PEER-ABORT invocation sent by the service provider shall be one of the following:

a) 'operational requirement'—the local system had to terminate the association to accommodate some other operational need;

b) 'protocol error'—the local application detected an error in the sequencing of service operations;

c) 'communications failure'—the communications service on the other side of a gateway was disrupted;

NOTE – The 'communications failure' `diagnostic` is included in the PEER-ABORT invocation to support its possible use by particular kinds of gateways. It is used by such gateways to report on a communications failure with the peer application entity. It is not intended to be used by the peer application entity itself. Beyond this statement, the behavior of such gateways is outside the scope of this Recommended Standard.

d) 'encoding error'—the local application detected an error in the encoding of one or more operation parameters or did not recognize that the operation or the data is badly formatted (e.g., one of the service instance identifier fields is missing);

e) 'response timeout'—the local application detected that the acknowledgement from a three-phase operation or the return from a two-phase operation was not received within the time period specified by the `response-timeout` configuration parameter (see 3.2.1.2). This `diagnostic` is used only when the service uses one or more confirmed operations that are invoked by the service provider;

NOTE – This issue of this Recommended Standard does not specify any confirmed operation being invoked by the service provider. Therefore a 'response timeout' PEER-ABORT operation cannot be invoked by the service provider of any service that uses only procedures that are directly adopted from this Recommended Standard. However, future issues of this Recommended Standard may include provider-invoked confirmed operations. Also, certain CSTSes may add provider-invoked confirmed operations by means of extension (see 1.6.1.7.19). The 'response timeout' `diagnostic` is applicable to any provider-invoked confirmed operation that might be defined either in a future issue of this Recommended Standard or in CSTS specifications that are derived from this Recommended Standard.

f) ‘end of service instance provision period’—the service provider detected that the service instance provision period has ended and the service user has not invoked the UNBIND operation;

g) ‘duplicate invoke ID’—the value of the `invoke-id` parameter is the same as the `invoke-id` value of another operation still being performed;

h) ‘invalid procedure name’—the `procedure-name` of the received operation contains an unknown procedure type, or the procedure role does not match the expected one, or the `procedure-name` contains an unknown procedure instance;

NOTE – If a CSTS does not implement an optional procedure, invocation of operations of such an unimplemented procedure will not result in the invocation of PEER-ABORT (refer to 3.3.2.7.1 c)).

i) ‘unrecognized operation or parameter type’—the operation type of the received operation does not match any of the defined types, or one of the parameter types does not match any of the defined types;

j) ‘other reason’—the local application detected an unspecified error during the processing of one or more operations.

3.6.2.2.1.3 The service provider shall handle PEER-ABORT invocations sent by the service user containing one of the following `diagnostic` values:

NOTE – This Recommended Standard does not specify how the service provider shall handle a PEER-ABORT `diagnostic` value. It may, for example, be displayed to a local operator and/or written to a local log file.

a) ‘access denied’—a service provider with an identity as presented in the `responder-identifier` parameter of the BIND return is not known to the service user;

b) ‘unexpected responder ID’—the value of the `responder-identifier` parameter in the BIND return does not match the identity of the authorized service provider for this service instance as specified by service management;

c) ‘operational requirement’—the user system had to terminate the association to accommodate some other operational need;

d) ‘protocol error’—the user application detected an error in the sequencing of service operations;

e) ‘communications failure’—the communications service on the other side of a gateway was disrupted;

NOTE – The 'communications failure' `diagnostic` is included in the PEER-ABORT invocation to support its possible use by particular kinds of gateways. It is used by such gateways to report on a communications failure with the peer application entity. It is not intended to be used by the peer application entity itself. Beyond this statement, the behavior of such gateways is outside the scope of this Recommended Standard.

f) 'encoding error'—the user application detected an error in the encoding of one or more operation parameters or did not recognize the operation or the data is badly formatted (e.g., one of the `service-instance-identifier` fields is missing);

g) 'response timeout'—the user application detected that the acknowledgement from a three-phase operation or the return from a two-phase operation was not received within the expected time period;

NOTE – The user is expected to use the value of the `response-timeout` configuration parameter (see 3.2.1.2) to determine when a response is overdue.

h) 'unsolicited invoke-id'—the user application received a response with an `invoke-id` that does not match the `invoke-id` of any of the operations for which a response is pending;

i) 'invalid procedure name'—the `procedure-name` of the received operation contains an unknown procedure type, or the procedure role does not match the expected one, or the `procedure-name` contains an unknown procedure instance;

j) 'unrecognized operation or parameter type'—the operation type of the received operation does not match any of the defined types, or one of the parameter types does not match any of the defined types;

k) 'other reason'—the user application detected an unspecified error during the processing of one or more operations.

NOTE – An implementation using the PEER-ABORT `diagnostic` value 'other reason' shall document under which conditions this `diagnostic` value is used (see 3.3.2.7.2).

3.6.2.2.1.4 A service-type-specific derived or service-original procedure shall have the capability to specify further `diagnostic` values within the constraints specified in F3.5. These values need to be documented in the specification of that service type.

3.6.2.2.2 PEER-ABORT Invocation Syntax

The type `PeerAbortInvocation`, as defined in F3.5, shall specify the syntax of the PEER-ABORT invocation PDU and its parameters.

3.7 START (CONFIRMED)

3.7.1 BEHAVIOR

NOTE – The service user can start the activities of a stateful procedure by invoking the START operation of that procedure.

3.7.1.1 The activity to be started shall be defined by the procedure using the START operation.

3.7.1.2 The service provider shall return a report of the outcome of the performance of the START operation to the service user.

3.7.1.3 After a successful START, the service provider shall

a) transition the procedure to the state ‘active’; and

b) perform the activities associated with the procedure using the operation.

3.7.1.4 After an unsuccessful START, the service provider shall remain in procedure state ‘inactive’.

3.7.2 INVOCATION AND RETURN PARAMETERS

3.7.2.1 General

The parameters of the START operation shall be present in the invocation and return as specified in table 3-6.

Table 3-6: START Operation Parameters

Parameters	Invocation	Return
Standard Operation Header (confirmed)	*M*	*M*

3.7.2.2 Operation Parameters Definitions

3.7.2.2.1 Standard Confirmed Operation Header

This operation shall use the Standard Confirmed Operation Header (see 3.3).

3.7.2.2.2 START Invocation Syntax

The type `StartInvocation`, as defined in F3.4, shall specify the syntax of the START invocation PDU and its parameters.

3.7.2.2.3 START Return Syntax

The type `StartReturn`, as defined in F3.4, shall specify the syntax of the START return PDU and its parameters.

3.7.2.3 `diagnostic` Parameter Extension Value Definitions and Syntax

3.7.2.3.1 If a negative START return is sent, one of the `diagnostic` values specified in 3.3.2.7 or one of the following `diagnostic` values shall be used:

a) 'unable to comply'—the service provider is unable to activate the procedure at this time because of a fault affecting the service;

b) 'out of service'—the service provider has been taken out of service for an indefinite period by management action (i.e., `production-status` is 'halted').

3.7.2.3.2 The type `StartDiagnosticExt`, as defined in F3.4 shall specify the syntax of the `diagnostic` parameter of the START return, extended as listed in 3.7.2.3.1.

3.8 STOP (CONFIRMED)

3.8.1 BEHAVIOR

NOTE – The service user can stop the activities of a stateful procedure by invoking the STOP operation of that procedure.

3.8.1.1 The activity to be stopped shall be defined by the procedure using the STOP operation.

NOTE – Within the constraints of the service instance provision period, the service user may re-enable the activity by again invoking the START operation.

3.8.1.2 The service provider shall provide a report of the outcome of the performance of the STOP operation to the service user.

3.8.1.3 After a successful STOP, the service provider shall

a) stop performing its activities; and

b) transition the procedure to the state 'inactive'.

3.8.1.4 After an unsuccessful STOP, the service provider shall

a) continue performing its activities; and

b) remain in procedure state 'active'.

3.8.2 INVOCATION, RETURN, AND PARAMETERS

3.8.2.1 General

The parameters of the STOP operation shall be present in the invocation and return as specified in table 3-7.

Table 3-7: STOP Operation Parameters

Parameters	Invocation	Return
Standard Operation Header (confirmed)	*M*	*M*

3.8.2.2 Operation Parameters Definitions

3.8.2.2.1 Standard Confirmed Operation Header

This operation shall use the Standard Confirmed Operation Header (see 3.3) without extension.

3.8.2.2.2 STOP Invocation Syntax

The type `StopInvocation`, as defined in F3.4, shall specify the syntax of the STOP invocation PDU and its parameters.

3.8.2.2.3 STOP Return Syntax

The type `StopReturn`, as defined in F3.4, shall specify the syntax of the STOP return PDU and its parameters.

3.9 TRANSFER-DATA (UNCONFIRMED)

3.9.1 BEHAVIOR

The service provider shall invoke the TRANSFER-DATA operation to deliver data units to the service user.

3.9.2 INVOCATION AND PARAMETERS

3.9.2.1 General

The parameters of the TRANSFER-DATA operation shall be present in the invocation as specified in table 3-8.

Table 3-8: TRANSFER-DATA Operation Parameters

Parameters	Invocation
Standard Operation Header (unconfirmed)	*M*
`generation-time`	M
`sequence-counter`	M
`data`	M

3.9.2.2 Operation Parameters Definitions

3.9.2.2.1 Standard Confirmed Operation Header

This operation shall use the Standard Unconfirmed Operation Header (see 3.3) without extension.

3.9.2.2.2 `generation-time`

The `generation-time` parameter shall contain the UTC time at which the data unit was generated.

NOTE – The meaning of data generation can be understood only in the context of the service defining the data to be transferred.

3.9.2.2.3 `sequence-counter`

3.9.2.2.3.1 For each instance of a procedure that uses the TRANSFER-DATA operation, the service provider shall set the `sequence-counter` parameter of the first data unit due for transmission to 0 after acceptance of the START operation for that procedure instance.

3.9.2.2.3.2 For each instance of a procedure that uses the TRANSFER-DATA operation, the `sequence-counter` parameter shall be incremented with each data unit after the first one that the service provider transmits to the service user using that procedure instance.

NOTE – The purpose of the `sequence-counter` is to give the service user a mechanism for checking the continuity of the data transmitted by the service provider.

3.9.2.2.3.3 When the maximum value of the counter is reached, the `sequence-counter` parameter shall be reset to 0.

3.9.2.2.4 `data`

3.9.2.2.4.1 The value of the `data` parameter is the data unit generated.

3.9.2.2.4.2 The `data` parameter shall be either of the type `OCTET STRING`, or the type shall be specified as an extension.

3.9.2.2.4.3 A procedure using this operation shall

a) refine (i.e., specify) the format and the semantics of the octet string in the `data` parameter; or

b) extend the operation by defining the structure and the semantics of the extension field in the `data` parameter.

3.9.2.2.5 TRANSFER-DATA Invocation Syntax

The type `TransferDataInvocation`, as defined in F3.4, shall specify the syntax of the TRANSFER-DATA invocation PDU and its parameters.

3.10 PROCESS-DATA (UNCONFIRMED / CONFIRMED)

3.10.1 BEHAVIOR

NOTE – The service user can transfer data to the service provider for further processing by invoking the PROCESS-DATA operation.

3.10.2 INVOCATION, RETURN, AND PARAMETERS

3.10.2.1 General

The parameters of the PROCESS-DATA operation shall be present in the invocation as specified in table 3-9, in which a procedure using this operation shall specify if the unconfirmed variant or the confirmed variant is applied.

Table 3-9: PROCESS-DATA Operation Parameters

Parameters	Invocation	Return (optional)
Standard Operation Header (unconfirmed or confirmed)	*M*	*M*
`data-unit-id`	M	
`data`	M	

3.10.2.2 Operation Parameters Definitions

3.10.2.2.1 Standard Unconfirmed Operation Header

If the procedure using this operation opts for the unconfirmed variant of the PROCESS-DATA operation, the operation shall use the Standard Unconfirmed Operation Header (see 3.3).

3.10.2.2.2 Standard Confirmed Operation Header

If the procedure using this operation opts for the confirmed variant of the PROCESS-DATA operation, the operation shall use the Standard Confirmed Operation Header (see 3.3).

3.10.2.2.3 `data-unit-id`

The `data-unit-id` parameter value, defined to be of the type `IntUnsigned`, can be freely chosen by the service user. The service provider shall copy this parameter into the respective notifications reporting on the outcome of the processing of the data unit.

NOTE – Notifications can only be unambiguously associated with a specific data unit if the service user ensures that the `data-unit-id` parameter values of all data units already sent to the service provider (but of which processing did not yet complete or abort) are unique. One simple way of achieving uniqueness of the `data-unit-id` values is using a counter that is incremented by one for each data unit sent to the service provider.

3.10.2.2.4 `data`

3.10.2.2.4.1 The `data` parameter shall contain the data to be transferred from the service user to the service provider and to be subsequently processed by the service provider.

3.10.2.2.4.2 The `data` parameter shall be either of the type `OCTET STRING`, or the type shall be specified as an extension.

3.10.2.2.4.3 A procedure using this operation shall

a) refine (i.e., specify) the format and the semantics of the octet string in the `data` parameter; or

b) extend the operation by defining the structure and the semantics of the extension field in the `data` parameter.

3.10.2.2.5 PROCESS-DATA Invocation Syntax

The type `ProcessDataInvocation`, as defined in F3.4, shall specify the syntax of the TRANSFER-DATA invocation PDU and its parameters.

3.10.2.2.6 PROCESS-DATA Return Syntax

The type `ProcessDataReturn`, as defined in F3.4, shall specify the syntax of the TRANSFER-DATA return PDU and its parameters. It only applies if the procedure using the PROCESS-DATA operation has opted for the confirmed variant of this operation.

3.11 NOTIFY (UNCONFIRMED)

3.11.1 BEHAVIOR

The service provider shall invoke the NOTIFY operation to notify the service user of the occurrence of an event of interest to the service user.

NOTE – Notification of events may be of value to the service user in understanding specific service provider behavior, such as an interruption in data delivery.

3.11.2 INVOCATION AND PARAMETERS

3.11.2.1 General

The parameters of the NOTIFY operation shall be present in the invocation, as specified in table 3-10.

Table 3-10: NOTIFY Operation Parameters

Parameters	Invocation
Standard Operation Header (unconfirmed)	*M*
`event-time`	M
`event-name`	M
`event-value`	M

3.11.2.2 Operation Parameters Definitions

3.11.2.2.1 Standard Confirmed Operation Header

This operation shall use the Standard Unconfirmed Operation Header (see 3.3) without extension.

3.11.2.2.2 `event-time`

The `event-time` parameter shall contain the UTC time at which the event occurred.

3.11.2.2.3 `event-name`

3.11.2.2.3.1 The `event-name` is in the form of an Event Name, as defined in E5; it consists of the Event Identifier defined for the event being notified and either the Functional Resource Name of the Functional Resource Instance triggering the event or the `procedure-name` of the procedure that issues the notification.

3.11.2.2.3.2 This operation shall define the following service-production-related published events:

a) 'production status change' (`event-name`)—the status of service production has changed (see annex B). For any CSTS type this event is specified as part of the Functional Resource that represents the CSTS instance. The associated `event-value` shall report the `production-status` parameter value of that Functional Resource that applied after the 'production status change' event had triggered.

b) 'production configuration change' (`event-name`)—at least one parameter controlling the configuration of service production has been changed. For any CSTS type this event is specified as part of the Functional Resource that represents the CSTS instance. Unless otherwise specified by the procedure using the operation, the associated `event-value` shall be set to 'empty'.

3.11.2.2.3.3 The Published Identifiers for the Event Names of the above listed events are specified by the Functional Resource that represents the affected service instance.

3.11.2.2.3.4 The events defined in 3.11.2.2.3.2 shall be transferred with the Functional Resource Name of the Functional Resource Instance triggering the events, that is, the Functional Resource Instance representing the CSTS instance of which the `production-status` parameter changed or that detected and reported that the production configuration changed.

3.11.2.2.3.5 Procedures may require that the occurrence of events related to the procedure be notified; such a procedure shall specify the Published Identifier for each of these events as well as the `event-value` associated with the given event.

3.11.2.2.3.6 Events of the kind addressed in 3.11.2.2.3.5 shall be transferred with the `procedure-name` of the procedure triggering the events.

3.11.2.2.3.7 A derived procedure that inherits one or more procedure related events from a parent procedure shall notify the event by using the Published Identifier defined for this event by the parent procedure.

3.11.2.2.4 `event-value`

3.11.2.2.4.1 The `event-value` allows the notification to optionally carry additional information in the form of the type `SequenceOfQualifiedValue` as defined in F3.3 or in the form of an extension.

3.11.2.2.4.2 In case a derived procedure inherits one or more procedure related events, the derived procedure shall specify for each of these events the associated `event-value` parameter, regardless of the `event-value` specification given in the parent procedure.

3.11.2.2.4.3 The `event-value` of the ‘production status change’ event shall report the `production-status` parameter value of the service instance that notifies the ‘production status change’ event. The first part of the path specifying the type to be used is ‘NotifyInvocation’: ‘eventValue’: ‘EventValue’: ‘qualifiedValues’: ‘SequenceOfQualifiedValue’: ‘SEQUENCE OF QualifiedValue’, where this sequence has the length 1. The second part of the path is ‘QualifiedValue’: ‘valid’: ‘TypeAndValue’: ‘Embedded’: ‘EMBEDDED PDV’, where the PDV carries either (a) the ASN.1 type ‘ProductionStatus’, when the service directly adopts the production status values listed in table B-1, or (b) a service-specific ASN.1 type that modifies the standard ‘ProductionStatus’ type with service-specific refinements and/or substates (see 2.2.2.2). The standard ‘ProductionStatus’ ASN.1 type and all service-specific production status ASN.1 types (if any) are specified in SANA registry https://sanaregistry.org/r/functional_resources.

3.11.2.2.5 NOTIFY Invocation Syntax

The type `NotifyInvocation`, as defined in F3.4, shall specify the syntax of the NOTIFY invocation PDU and its parameters.

3.12 GET (CONFIRMED)

3.12.1 BEHAVIOR

NOTE – The service user can retrieve the values of specific parameters by invoking the GET operation.

3.12.1.1 The service provider shall have the capability to deliver to the service user individual parameters or a set of parameters represented by a list of Parameter Labels (see annex E).

NOTE – The list of Parameter Labels to be retrieved may be a list explicitly named in the invocation or a default list to be specified by the procedures or services using this operation.

3.12.1.2 The GET invocation is valid if it meets any one of the following conditions:

a) if the `list-of-parameters` parameter is 'empty', signifying the selection of the default list of Parameter Labels, and if such a default list has been established;

b) if the `list-of-parameters` parameter contains one parameter list name for a list of Parameter Labels that is contained in the set of label lists that has been established for the service for use by the GET operation;

c) if the `list-of-parameters` parameter contains one Functional Resource Type that is associated with the service instance that executes the procedure that contains the GET operation;

d) if the `list-of-parameters` parameter contains one name of a Functional Resource Instance that is associated with the service instance that executes the procedure containing the GET operation;

e) if the `list-of-parameters` parameter contains one procedure type that is associated with the service instance that executes the procedure containing the GET operation;

f) if the `list-of-parameters` parameter contains one procedure name of a procedure that is associated with the service instance that executes the procedure containing the GET operation;

g) if (1) the `list-of-parameters` parameter contains one or more Functional Resource Parameter Names or Functional Resource Parameter Labels and (2) every one of these names or labels is the name or label of a parameter of a Functional Resource that is associated with the service instance that executes the procedure that contains the GET operation;

h) if (1) the `list-of-parameters` parameter contains one or more procedure configuration Parameter Labels or Parameter Names and (2) every one of these labels or names is the label or name of a configuration parameter of a procedure that is associated with the service instance that executes the procedure that contains the GET operation.

3.12.1.3 If the GET invocation is valid, the service provider shall return the qualified parameters (Parameter Name, the value, the type, and the qualifier of the parameter—see annex C) using the `qualified-parameters` parameter. More specifically, if the `list-of-parameters`

a) is left empty, then

   1) for each Functional Resource Parameter Label represented by the default list (see annex E), the service provider shall return the qualified parameter (see annex C) for that label for the Functional Resource Instances of the given type that are directly associated with the service instance that executes the procedure containing the GET operation; and

   2) for each procedure configuration Parameter Label in the default list, the service provider shall return the qualified parameter for that label for every configured instance of the procedure that is associated with the service instance that executes the procedure containing the GET operation;

b) contains the name of a list of Parameter Labels, then

   1) for each Functional Resource Parameter Label in the named list (see E), the service provider shall return the qualified parameter for that label for each Functional Resource Instance of the given type that is directly associated with the service instance that executes the procedure containing the GET operation; and

   2) for each procedure configuration Parameter Label in the named list, the service provider shall return the qualified parameter for that label for every configured instance of the procedure that is associated with the service instance that executes the procedure containing the GET operation;

c) contains one Functional Resource Type, then the service provider shall return the qualified parameters for all parameters of all Functional Resource Instances of the given type that are directly associated with the service instance invoking the GET operation;

d) contains one Functional Resource Name, then the service provider shall return the qualified parameters for all parameters of the named Functional Resource Instance;

e) contains one procedure type, then the service provider shall return the qualified parameter for all parameters of every configured instance of that procedure type that is directly associated with the service instance invoking the GET operation;

f) contains one procedure name, then the service provider shall return the qualified parameters for all configuration parameters for that procedure instance;

g) contains any Functional Resource Parameter Labels, then for each Functional Resource Parameter Label, the service provider shall return the qualified parameter for that label for each of the Functional Resource Instances of the given type that is directly associated with the service instance that executes the procedure containing the GET operation;

h) contains any labels for procedure configuration parameters, then for each procedure configuration Parameter Label, the service provider shall return the qualified parameter for every configured instance of the procedure that is associated with the service instance that executes the procedure containing the GET operation;

i) contains one or more Parameter Names, then the service provider shall return the qualified parameter for each of the listed parameters.

3.12.1.4 If the GET invocation is invalid, the service provider shall issue a negative return using one of the `diagnostic` values specified in 3.12.2.4.

3.12.1.5 Procedures using this operation shall define

a) the names of the lists of Parameter Labels;

b) the Parameter Labels contained in the named lists; and

c) which of the named lists shall serve as default lists, if any.

3.12.2 INVOCATION, RETURN, AND PARAMETERS

3.12.2.1 General

The parameters of the GET operation shall be present in the invocation and return as specified in table 3-11.

Table 3-11: GET Operation Parameters

Parameters	Invocation	Return
Standard Operation Header (confirmed)	*M*	*M*
`list-of-parameters`	M	
`qualified-parameters`		C

3.12.2.2 Operation Parameters Definitions

3.12.2.2.1 Standard Confirmed Operation Header

This operation shall use the Standard Confirmed Operation Header (see 3.3).

3.12.2.2.2 `list-of-parameters`

3.12.2.2.2.1 The `list-of-parameters` parameter shall contain one of the following:

a) 'empty' (signifying default list);

b) the name of a list;

c) one Functional Resource Type;

d) one Functional Resource Name;

e) one procedure type;

f) one procedure name;

g) a set of individual Parameter Labels; or

h) a set of individual Parameter Names.

NOTE – The ASN.1 type of the `list-of-parameters` parameter is `ListOfParametersEvents` (see F3.3).

3.12.2.2.2.2 If the `list-of-parameters` parameter is set to 'empty', the service provider shall transmit the values of the parameters represented by the default list of Parameter Labels, provided such list is known to the service provider.

NOTE – The definition of what is represented by the list of Parameter Labels can be found in annex E.

3.12.2.2.2.3 A name of a list of Parameter Labels shall be defined as a string.

NOTE – The composition of Parameter Names is discussed in annex E.

3.12.2.2.2.4 The parameters that may be contained in the `list-of-parameters` parameter shall include (but not be limited to) the parameter that reports the `production-status` of the service, as specified in B2.2.2 and B2.2.3.

3.12.2.2.3 `qualified-parameters`

If the `result` is 'positive' (i.e., positive GET return), the parameter values requested via the `list-of-parameters` shall be returned in the `qualified-parameters` parameter as specified in annex C.

3.12.2.2.4 GET Invocation Syntax

The type `GetInvocation`, as defined in F3.4, shall specify the syntax of the GET invocation PDU and its parameters.

3.12.2.2.5 GET Return Syntax

The type `GetReturn`, as defined in F3.4, shall specify the syntax of the GET return PDU and its parameters.

3.12.2.3 `positive` Parameter Extension Value Definitions

The type `GetPosReturnExt`, as defined in F3.4, shall specify the syntax of the positive return extension of the GET operation.

3.12.2.4 `diagnostic` Parameter Extension Value Definitions and Syntax

3.12.2.4.1 If a negative GET return is sent, one of the `diagnostic` values specified in 3.3.2.7 or one of the following `diagnostic` values shall be used:

a) 'default not defined'—the default list (`list-of-parameters` set to 'empty') is unknown to the service provider.

b) 'unknown list name'—the list name contained in the `list-of-parameters` is unknown to the service provider. The unknown list name shall be returned with the `diagnostic`.

c) 'unknown Functional Resource Type'—the Functional Resource Type contained in the `list-of-parameters` is unknown to the service provider (see 3.12.2.2.2), or the Functional Resource Type is not associated with the service instance that executes the procedure containing the GET operation. The unknown Functional Resource Type shall be returned with the `diagnostic`.

d) 'unknown Functional Resource Name'—while the Functional Resource Type is known, the Functional Resource Name contained in the `list-of-parameters` is unknown to the service provider (see 3.12.2.2.2), or the selected Functional Resource Instance is not associated with the service instance that executes the procedure containing the GET operation. The unknown Functional Resource Name shall be returned with the `diagnostic`.

e) 'unknown procedure type'—the procedure type contained in the `list-of-parameters` is unknown to the service provider (see 3.12.2.2.2). The unknown procedure type shall be returned with the `diagnostic`.

f) 'unknown procedure name'—while the procedure type is known, the procedure instance contained in the `list-of-parameters` is unknown to the service provider (see 3.12.2.2.2). The unknown procedure name shall be returned with the `diagnostic`.

g) 'unknown parameter identifier'—one or more Parameter Identifiers contained in the `list-of-parameters` parameter are unknown to the service provider (see 3.12.2.2.2) for one of the following reasons:

1) the Functional Resource or procedure type specified as part of the Parameter Label is not associated with the service instance executing the procedure containing the GET operation;

2) the Functional Resource or procedure instance specified as part of the Parameter Name is not associated with the service instance executing the procedure containing the GET operation;

3) a parameter with the given Published Identifier does not exist for the specified Functional Resource or procedure type.

The list of unknown Parameter Names or Parameter Labels shall be returned with the `diagnostic`. For each unknown Parameter Identifier that is contained in a Parameter Name in the `list-of-parameters`, the Parameter Name shall be returned. For each unknown Parameter Identifier that is contained in a Parameter Label in the `list-of-parameters`, the Parameter Label shall be returned.

3.12.2.4.2 The type `GetDiagnosticExt`, as defined in F3.4, shall specify the syntax of the `diagnostic` parameter of the GET return, extended as listed in 3.12.2.4.1.

3.13 EXECUTE-DIRECTIVE (ACKNOWLEDGED)

3.13.1 BEHAVIOR

NOTES

1 The service user can invoke the EXECUTE-DIRECTIVE operation to cause the EM of the ESLT to perform a specified action.

2 The EXECUTE-DIRECTIVE is an acknowledged operation, which provides intermediate feedback to the service user to acknowledge that the invocation has been received and is valid, as well as a final feedback regarding the outcome of the operation, which is returned some time after the acknowledgement.

3.13.1.1 The specified action to be performed when invoked by means of the EXECUTE-DIRECTIVE operation shall be defined by one of the following:

a) a `directive-identifier` registered for the procedure containing the invoked EXECUTE-DIRECTIVE operation;

b) a `directive-identifier` registered for a procedure type, if that type is associated with the service instance executing the procedure containing the invoked EXECUTE-DIRECTIVE operation;

c) a `directive-identifier` registered for a Functional Resource Type, if that type is associated with the service instance executing the procedure containing the invoked EXECUTE-DIRECTIVE operation.

3.13.1.2 The service provider shall provide a report (acknowledgement) on the acceptance of the EXECUTE-DIRECTIVE invocation to the service user.

3.13.1.3 If the EXECUTE-DIRECTIVE invocation is valid, the service provider shall execute the action and then provide a report (return) on the outcome of the overall performance of the EXECUTE-DIRECTIVE operation.

3.13.2 INVOCATION, RESPONSES, AND PARAMETERS

3.13.2.1 General

The parameters of the EXECUTE-DIRECTIVE operation shall be present in the invocation, acknowledgement, and return, as specified in table 3-12.

Table 3-12: EXECUTE-DIRECTIVE Operation Parameters

Parameters	Invocation	Acknowledgement	Return
Standard Operation Header (confirmed)	*M*	*M*	*M*
`directive-identifier`	M		
`directive-qualifier`	M		

3.13.2.2 Operation Parameters Definitions

3.13.2.2.1 Standard Confirmed Operation Header

This operation shall use the Standard Confirmed Operation Header (see 3.3).

3.13.2.2.2 `directive-identifier`

3.13.2.2.2.1 The `directive-identifier` shall identify the action that is to be performed by the EM of the ESLT.

3.13.2.2.2.2 The `directive-identifier` parameter shall be of the type `PublishedIdentifier` and may be registered either for a procedure type or for a Functional Resource Type (see E6.1).

3.13.2.2.3 `directive-qualifier`

3.13.2.2.3.1 The `directive-qualifier` shall contain complementary data necessary to perform the action specified by the `directive-identifier` parameter.

3.13.2.2.3.2 Depending on the scope of the `directive-identifier` (see 3.13.1.1) the `directive-qualifier` parameter shall be comprised of one of the following:

a) the directive qualifier values;

b) the type and instance number of the procedure the directive shall act on and the directive qualifier values;

c) the type and instance number of the Functional Resource the directive shall act on and the directive qualifier values.

3.13.2.2.3.3 The directive qualifier values shall be one of the following:

a) 'noQualifierValues' to indicate that, for the given `directive-identifier`, the `directive-qualifier` parameter does not carry complementary information;

b) a possibly complex structure of values not associated with formally defined parameters identified by means of Published Identifiers; the data structure shall be of the type `TypeAndValue` defined in F3.3;

c) a sequence of parameter identifiers and the associated parameter values in which the parameter identifier is the Published Identifier assigned to this parameter and the parameter value is of the type `TypeAndValue` defined in F3.3.

3.13.2.2.3.4 A procedure using this operation may extend the operation by defining the structure and the semantics of the extension field in the `directive-qualifier` parameter.

3.13.2.2.4 EXECUTE-DIRECTIVE Invocation Syntax

The type `ExecuteDirectiveInvocation`, as defined in F3.4, shall specify the syntax of the EXECUTE-DIRECTIVE invocation PDU and its parameters.

3.13.2.2.5 EXECUTE-DIRECTIVE Acknowledgement Syntax

The type `ExecuteDirectiveAcknowledge`, as defined in F3.4, shall specify the syntax of the EXECUTE-DIRECTIVE acknowledgement PDU and its parameters.

3.13.2.2.6 EXECUTE-DIRECTIVE Return Syntax

The type `ExecuteDirectiveReturn`, as defined in F3.4, shall specify the syntax of the EXECUTE-DIRECTIVE return PDU and its parameters.

3.13.2.3 `diagnostic` Parameter Extension Value Definitions and Syntax

3.13.2.3.1 If an EXECUTE-DIRECTIVE negative acknowledgement is sent, one of the `diagnostic` values specified in 3.3.2.7 or one of the following `diagnostic` values shall be used:

a) 'unknown directive'—the `directive-identifier` specified in the invocation is not valid; that is, the `directive-identifier` is not registered for the procedure or for the Functional Resource Type specified in the `directive-qualifier` parameter;

b) 'unknown qualifier'—the `directive-qualifier` specified in the invocation is not valid for the given `directive-identifier`;

c) 'invalid procedure instance'—the specified instance of the procedure type that the directive shall act on does not exist;

d) 'invalid functional resource instance'—the specified instance of the given Functional Resource Type that the directive shall act on does not exist;

e) 'invalid Functional Resource parameter'—either (1) one or more of the Parameter Identifiers in the sequence of Parameter Identifiers in the `directive-qualifier` are not published Parameter Identifiers of the Functional Resource the directive shall act on, or (2) the associated parameter value is not of the type specified for that parameter of the Functional Resource. This `diagnostic` value shall include a list of all Parameter Names contained in the `directive-qualifier` that are invalid for the given Functional Resource Type or for which the type of the parameter value is invalid;

f) 'invalid procedure parameter'—either (1) one or more of the Parameter Identifiers in the sequence of Parameter Identifiers in the `directive-qualifier` are not published Parameter Identifiers of a configuration parameter of the procedure the directive shall act on, or (2) the associated parameter value is not of the type specified for that configuration parameter of the procedure. This `diagnostic` value shall include a list of all Parameter Names contained in the `directive-qualifier` that are invalid for the given procedure or for which the type of the parameter value is invalid;

g) 'parameter value out of range'—one or more of the parameters in the parameter sequence of the `directive-qualifier` have a value that is outside the range that is defined for that parameter. This `diagnostic` value shall include a list of all Parameter Names contained in the `directive-qualifier` for which the parameter value falls outside the valid range.

3.13.2.3.2 The type `ExecDirNegAckDiagnosticExt`, as defined in F3.4 shall specify the syntax of the `diagnostic` parameter of the EXECUTE-DIRECTIVE acknowledgement, extended as listed in 3.13.2.3.1.

3.13.2.3.3 If an EXECUTE-DIRECTIVE negative return is sent, one of the `diagnostic` values specified in 3.3.2.7 or the following `diagnostic` value shall be used:

> 'action not completed'—the requested action was not completed, for example, because the guard condition of some of the parameters to be updated evaluated to FALSE. This `diagnostic` value shall include a list of all Parameter Names contained in the `directive-qualifier` for which setting of the parameter value was successful. If the given directive is not intended to change parameter values, the list of Parameter Names shall be empty.

NOTE – The only `diagnostic` value of those defined in 3.3.2.7 that might be applicable in this case is 'other reason'. An implementation using this `diagnostic` value needs to document the conditions under which this `diagnostic` value applies (see 3.3.2.7.2). In case any of the other `diagnostic` values defined in 3.3.2.7 apply, the EXECUTE-DIRECTIVE operation will fail with a negative acknowledgement, and therefore a negative return will not be sent.

3.13.2.3.4 The type `ExecDirNegReturnDiagnosticExt`, as defined in F3.4, shall specify the syntax of the `diagnostic` parameter of the EXECUTE-DIRECTIVE return extended, as listed in 3.13.2.3.3.

4 PROCEDURES

4.1 OVERVIEW

This section specifies the procedures defined in this Recommended Standard. Subsection 4.2 specifies behaviors that are generally applicable to all procedures. Subsections 4.3 through 4.12 specify common procedures defined in this Recommended Standard.

NOTE – Unless otherwise specified, all statements made in this section shall be understood to refer to a single procedure instance only.

4.2 COMMON PROCEDURES BEHAVIOR

4.2.1 PROCEDURE INSTANCES CREATION

4.2.1.1 The Association Control procedure shall be instantiated at service instance creation and shall exist for the lifetime of the service instance.

4.2.1.2 All procedures other than the Association Control procedure shall be instantiated as soon as a positive BIND return is issued by the service provider.

4.2.1.3 The service provider shall not accept and process any operations except BIND until it returns a positive BIND return. Exception to that statement is the reception of a PEER-ABORT following the reception of a BIND invocation and preceding the issue of the BIND return.

4.2.2 TERMINATION OF THE ASSOCIATION

4.2.2.1 On reception of an UNBIND invocation, the Association Control procedure shall issue a ‘terminate procedure’ event to all procedure instances of the service instance.

4.2.2.2 On ‘protocol abort’ (see 1.6.1.7.40) or reception of a PEER-ABORT invocation (see 3.6), the Association Control procedure shall communicate a ‘terminate procedure’ event to all procedure instances of the service instance.

4.2.2.3 If any procedure other than the Association Control procedure initiates an abort, it shall issue a ‘procedure to association abort ‘xxx’’ event to the Association Control procedure, where ‘xxx’ represents the `diagnostic` value for the abort event (see 4.2.2.5).

4.2.2.4 One of the conditions under which a procedure shall issue a ‘procedure to association abort ‘xxx’’ event to the Association Control procedure is that the ‘invalid PDU’ incoming event is triggered. This shall be the case whenever one of the conditions specified in 3.2.3.6 a), b), c), or e) is given.

NOTE – The condition specified in 3.2.3.6 d) is covered by the individual procedure state tables.

4.2.2.5 The `diagnostic` parameter value (see 3.6.2.2.1.2) to be used in the 'procedure to association abort 'xxx'' action is

a) 'encoding error' in case the condition specified in 3.2.3.6 b) applies;

b) 'duplicate invoke id' in case the condition specified in 3.2.3.6 e) applies;

c) 'invalid procedure name' in case the condition specified in 3.2.3.6 c) applies; and

d) 'unrecognized operation or parameter type' in case the condition specified in 3.2.3.6 a) applies.

NOTE – If an implementation chooses to use the `diagnostic` value 'other reason' (see 3.6.2.2.1.2 j)), the exact condition under which this is done needs to be specified (see 3.3.2.7.2).

4.2.2.6 On reception of the 'procedure to association abort 'xxx'' event from any other procedure, the Association Control procedure shall abort the association.

4.2.2.7 In aborting the association, the Association Control procedure shall invoke the PEER-ABORT operation with the appropriate `diagnostic` value.

4.2.2.8 When invoking the PEER-ABORT, the Association Control procedure of the service provider shall communicate a 'terminate procedure' event to all procedure instances of the service instance.

4.2.3 TERMINATING

4.2.3.1 The Association Control procedure is responsible for sending the 'terminate procedure' event to all procedures of the service instance.

4.2.3.2 On reception of a 'terminate procedure' event from the Association Control procedure, all procedures shall terminate all their activities, release their resources, and cease to exist, unless otherwise specified by the procedures.

NOTE – If a procedure is terminated prior to action completion, the service provider will not report on the result of the on-going operations after re-establishing the association.

4.2.4 PROCEDURE STATES

4.2.4.1 Once the association is established, all stateful procedures shall be in state 'inactive'.

4.2.4.2 For stateful procedures that have a START and a STOP operation,

a) the transition from 'inactive' to 'active' occurs after the procedure accepts the START invocation (i.e., when the procedure has sent a positive START return); and

b) the transition from 'active' to 'inactive' occurs after the procedure accepts the STOP invocation (i.e., when the procedure has sent a positive STOP return).

4.2.4.3 For stateful procedures that do not have a START and a STOP operation, the state transition shall be specified by the procedure.

NOTE – Annex G should also be consulted for a description of the service state tables.

4.2.5 DERIVED PROCEDURES

4.2.5.1 The specification of a derived procedure shall identify the parent procedure type from which it is derived.

4.2.5.2 If a procedure is designated as a derived procedure, it inherits the complete behavior of the parent procedure except when the specification of the derived procedure explicitly modifies the parent procedure's behavior.

4.2.5.3 The state table and the associated tables of a derived procedure shall be self contained; however, elements being inherited from the parent procedure without any modification shall be presented in italic font.

NOTE – The procedures specified in 4.7, 4.8, and 4.10 are examples of derived procedures.

4.3 ASSOCIATION CONTROL

4.3.1 VERSION NUMBER

The version number of this procedure is 2.

4.3.2 DISCUSSION

4.3.2.1 Purpose

The Association Control procedure establishes and releases an association between a service user and a service provider for a given service instance.

4.3.2.2 Concept

The service user initiates the association by sending a BIND invocation. On reception of the BIND invocation, the service provider reports to the service user whether the association is established or not.

To orderly terminate the association the service user sends an UNBIND invocation. On reception of the UNBIND invocation, the service provider reports to the service user whether the association release is accepted or aborted. The Association Control procedure forwards the ‘terminate procedure’ event to all procedure instances of the service instance.

The association can be aborted by either the service user or the service provider to inform the peer system that the local system detected an error that requires the association be terminated. In case of abort, the Association Control procedure forwards the ‘terminate procedure’ event to all procedure instances of the service instance.

The values of the `version-number` and `service-type` parameters are selected as part of the service definition.

4.3.3 BEHAVIOR

4.3.3.1 Activities

4.3.3.1.1 At the beginning of the service instance provision period, as specified by Service Management, the Association Control procedure of the service instance shall be placed in the ‘unbound’ state and made available for binding by the service user.

4.3.3.1.2 After having received a valid BIND invocation, the service provider shall establish the association for the given service instance with the service user having invoked the BIND operation, transitioning from service instance state 1 (‘unbound’) to service instance state 2 (‘bound’).

4.3.3.1.3 After having received a valid UNBIND invocation, the service provider shall release the association transitioning from service instance state 2 (‘bound’) to service instance state 1 (‘unbound’).

4.3.3.1.4 The service provider may abort the association by invoking the PEER-ABORT operation.

4.3.3.1.5 The service provider shall abort the association when receiving the PEER-ABORT invocation from the service user.

4.3.3.1.6 An association also may be aborted because of certain failures of the underlying communications service; such failures are signaled to the local application by a ‘protocol abort’ event (see 1.6.1.7.40).

4.3.3.1.7 The deletion of a service instance shall result in the release of all resources associated with that service instance.

4.3.3.1.8 Binding

4.3.3.1.8.1 Should the service user after having invoked the BIND operation invoke any further operations for this service instance before the service provider has sent the BIND return, such invocation shall be treated as an invalid PDU (see 3.2.3.6 d)).

4.3.3.1.8.2 If a BIND invocation is received from a service user for a service instance that is already bound to another service user, it shall be rejected with a BIND return with the `result` parameter set to ‘negative’ and the `diagnostic` parameter set to ‘already bound’.

NOTE – If the return from the BIND invocation is not received after a sufficiently long time, the service user may attempt to recover by invoking the PEER-ABORT operation followed by another BIND invocation. The length of the duration that constitutes ‘a sufficiently long time’ is expected to be the value of the response-timeout configuration parameter (see 3.2.1.2).

4.3.3.1.8.3 On reception of the BIND invocation, if the invocation is accepted, the service provider shall allocate all resources needed for the service instance.

4.3.3.1.8.4 Once the association is established, the Association Control procedure shall transition to state 2 (‘bound’).

NOTE – Following receipt of the return from an UNBIND invocation or following the invocation of PEER-ABORT, the service user may issue another BIND invocation if permissible at that point (e.g., if the end of the service instance provision period has not yet been reached).

4.3.3.1.9 Unbinding

4.3.3.1.9.1 On reception of a valid UNBIND invocation, the service provider shall send a positive UNBIND return and release the association previously established by a BIND operation.

4.3.3.1.9.2 Should the service user after having invoked the UNBIND operation invoke any further operations for this service instance before the service provider has sent the UNBIND return, such an invocation shall be treated as an invalid PDU (see 3.2.3.6 d)).

NOTE – If the return from the UNBIND operation is not received after a sufficiently long time, the service user may attempt to recover by invoking the PEER-ABORT operation to abort the association. The length of the duration that constitutes 'a sufficiently long time' is expected to be the value of the `response-timeout` configuration parameter (see 3.2.1.2).

4.3.3.1.9.3 The service provider shall accept the UNBIND invocation only in service instance state 2.1 ('bound.ready').

4.3.3.1.9.4 If the UNBIND invocation is accepted, then the Association Control procedure shall confirm the release of the association by issuing a positive UNBIND return.

4.3.3.1.10 Releasing

Releasing the association has the following effect: the service provider shall transition to state 1 ('unbound').

NOTE – The act of releasing the association for a particular service instance does not necessarily terminate the associated service production.

4.3.3.1.11 Aborting

4.3.3.1.11.1 The association may be aborted in one of three ways:

a) service provider initiated PEER-ABORT;

b) protocol abort signaled by the underlying communication layer;

c) service user initiated PEER-ABORT.

4.3.3.1.11.2 Regardless of the way the association is aborted, the service provider shall transition to the state 'unbound'.

4.3.3.1.11.3 If the event 'end of service instance provision period' occurs, then the Association Control procedure of the service provider shall abort the association using the PEER-ABORT operation, delete the service instance, and release its resources.

4.3.3.1.11.4 The Association Control procedure shall abort the association upon receipt of a 'procedure to association abort 'xxx'' event from any of the other procedures that constitute the service.

4.3.3.1.11.5 If the occurrence of an underlying communication problem is flagged by means of the 'protocol abort' event, then the Association Control procedure shall act as if it had invoked the PEER-ABORT operation.

4.3.3.1.11.6 In case the prime procedure is stateful, any attempt to release the association while the prime procedure is 'active' shall result in a PEER-ABORT invoked by the service provider.

4.3.3.1.12 Access Control

4.3.3.1.12.1 The Association Control procedure shall implement access control based on the identity of the initiator and responder. Access control is performed at two levels:

a) the initiator must be registered at the responder, and the responder must be registered at the initiator;

b) the initiator and responder must be authorized for the given service instance.

4.3.3.1.12.2 The initiator shall have access to a registry of authorized responders, and the responder shall have access to a registry of authorized initiators. These registries shall be maintained by the UM and the PM of the CSSSes, respectively.

4.3.3.1.12.3 The initiator and responder shall indicate their identities by setting the parameters `initiator-identifier` and `responder-identifier` in the BIND operation to the values assigned by service management.

4.3.3.1.13 Extensibility

4.3.3.1.13.1 The Association Control procedure may be extended with additional parameters.

NOTE – Extending the parameters is not recommended as use of this feature may impact the generality of the Association Control procedure.

4.3.3.1.13.2 The Association Control procedure may be extended with additional `diagnostic` values.

NOTE – Extending the `diagnostic` with additional values is not recommended, as use of this feature may impact the generality of the Association Control procedure.

4.3.3.1.13.3 A CSTS shall not derive or refine the Association Procedure through inclusion of operations other than the BIND, UNBIND, and PEER-ABORT operations.

4.3.4 REQUIRED OPERATIONS

Table 4-1: Association Control Procedure Required Operations

Operations	Source	Extended	Refined	Procedure Blocking/Non-Blocking
BIND	Common	N	N	Blocking
UNBIND	Common	N	N	Blocking
PEER-ABORT	Common	N	N	Non-Blocking

4.3.5 CONFIGURATION PARAMETERS

The Association Control procedure configuration parameters that need to be configured in the context of the procedure shall be as defined in table 4-2.

NOTE – For each configuration parameter, the table identifies the engineering unit (if applicable), a cross reference to the use of the parameter in the specification of the procedure, whether the parameter may be read, and the Parameter Identifier and type to be used in reporting the value of the parameter. None of the configuration parameters of this procedure can be dynamically changed while the service instance executing the procedure is bound.

Table 4-2: Association Control Procedure Configuration Parameters

Parameters	Cross-Reference	Readable	Configuration Parameter Identifier and Type (F3.16)
`initiator-identifier`	3.4.2.2.2, 4.3.3.1.12.3	Yes	`pACinitiatorId` `PACinitiatorIdType`
`responder-identifier`	3.4.2.2.3, 4.3.3.1.12.3	Yes	`pACresponderId` `PACresponderIdType`
`responder-port-identifier`	3.4.2.2.4	Yes	`pACresponderPortId` `PACresponderPortIdType`
`service-instance-identifier`	3.4.2.2.7	Yes	`pACserviceInstanceId` `PACserviceInstanceIdType`

4.3.6 PROCEDURE STATE TABLE

NOTE – The state transition matrix specified in table 4-3 represents one instance of the Association Control procedure. Since there is one and only one instance of the Association Control procedure for each instance of a CSTS, the state table thus represents the single association for that CSTS.

Table 4-3: Association Control Procedure State Table

No.	Incoming Event	State 1 ('unbound')	State 2 ('bound')
1	(BindInvocation)	IF "positive result" THEN (+BindReturn) → 2 ELSE (-BindReturn) → 1 ENDIF	{peer abort 'protocol error'} → 1
2	'end of service instance provision period'	'delete service instance'	{peer abort 'end-of-service-instance-provision-period'} 'delete service instance' → 1
3	(UnbindInvocation)	[ignore]	IF "prime procedure is stateful and active" THEN {peer abort 'protocol error'} ELSE (+UnbindReturn) 'terminate procedure' ENDIF → 1
4	(PeerAbortInvocation)	[ignore]	'terminate procedure' → 1
5	'procedure to association abort 'xxx''	[ignore]	{peer abort 'xxx'} → 1
6	'invalid PDU' xxx''	[ignore]	{peer abort 'xxx'} → 1
7	'protocol abort'	[ignore]	'terminate procedure' → 1

Table 4-4: Procedure State Table Incoming Event Description References

Event	Reference
'end of service instance provision period'	3.6.2.2, 4.3.3.1.11.3
'invalid PDU 'xxx''	3.2.3.6, 4.2.2.4. 'xxx' is one of the `diagnostic` values specified in 4.2.2.5.
'procedure to association abort 'xxx''	4.2.2.3, 4.2.2.5, 4.3.3.1.11.4
'protocol abort'	4.3.3.1.11.5
(PeerAbortInvocation)	4.3.3.1.11.1
(UnbindInvocation)	4.3.3.1.9.4
(BindInvocation)	4.3.3.1.8.2, 4.3.3.1.8.3

Table 4-5: Procedure State Table Predicate Descriptions

Predicate	Evaluates to TRUE if
"positive result"	No reason for sending a negative BIND return has been detected; that is, none of the conditions in 3.4.2.3.1 applies.
"prime procedure is stateful and active"	The prime procedure of the given CSTS instance is stateful and currently in the state 'active'.

Table 4-6: Procedure State Table Simple Action References

Name	References
'delete service instance'	4.3.3.1.7
'abort 'xxx''	4.2.2.7, 4.2.2.5, 'xxx' indicates the `diagnostic` value that is reported by the PEER-ABORT operation
'terminate procedure'	4.2.3, internal event from the Association Control procedure to all other procedures of the service instance in response to a 'protocol abort' event, a PEER-ABORT, or an UNBIND

Table 4-7: Procedure State Table Compound Action Definitions

Name	Actions Performed
{peer abort 'xxx'}	'terminate procedure' 'abort 'xxx''

4.4 UNBUFFERED DATA DELIVERY

4.4.1 VERSION NUMBER

The version number of this procedure is 1.

4.4.2 DISCUSSION

4.4.2.1 Purpose

This Unbuffered Data Delivery procedure can be used to accomplish the transfer of data from the service provider to the service user in a 'best effort' manner; that is, data are delivered as soon as generated, if possible, and are discarded individually in case of communication-link congestion or backpressure from the peer entity.

4.4.2.2 Concept

The Unbuffered Data Delivery procedure supports transfer of data units from the service provider to the service user. The behavior of this process is the following:

As each data unit is generated, it is either immediately transferred or immediately discarded in case the underlying communication service does not accept the data unit for transfer. Each data unit contains a sequence counter allowing the service user to detect the loss of data.

Production of data units might refer to extraction of these data units from the space link or to any other process generating data.

The operations defined in this procedure allow a service user to interact with a service provider to

a) request start of the data transfer specifying the selection criteria of the data to be transferred;

b) receive the specified data units; and

c) stop and optionally later re-start the delivery of data units applying the same or a different selection.

The service user starts the data transfer by invoking the START operation and specifying the selection criteria of the data to be transferred.

The service user can stop the data transfer at any time by invoking the STOP operation.

4.4.3 BEHAVIOR

4.4.3.1 Starting

The service provider shall send a positive START return and perform the START operation invoked by the service user except if

a) the `production-status` is 'halted', in which case the START operation shall be rejected by sending a negative START return with the `diagnostic` value 'out of service'; or

b) the procedure is in State 2 ('active'), in which case the procedure shall request the Association Control procedure to abort the association with setting the `diagnostic` value to 'protocol error'.

NOTE – The service user will initiate the transfer of data by this procedure by invoking the START operation.

4.4.3.2 Transferring Data

4.4.3.2.1 After a successful START operation, the service provider shall transfer the data by means of invoking the TRANSFER-DATA operation as the data become available from the production and pass the TRANSFER-DATA PDU to the underlying communications service.

4.4.3.2.2 The transfer shall end when one of the following occurs:

a) no more data are to be expected;

b) STOP is invoked by the service user;

c) the service instance is aborted.

4.4.3.2.3 TRANSFER-DATA is valid only in procedure state 'active' and shall be invoked only by the service provider.

4.4.3.2.4 The transfer shall start with the most recently generated data; the availability of new data generated by the production engine constitutes the 'data available' event (see table 4-10).

4.4.3.2.5 Data units shall be transmitted in the sequence in which they are generated.

4.4.3.2.6 While the underlying communications service does not accept data units for transfer because of to backpressure, the affected data units shall be discarded.

NOTE – Backpressure may be caused by a congested communication link or by a user application that does not accept the data units at the rate at which they are generated.

4.4.3.3 Stopping

4.4.3.3.1 The service provider shall perform the STOP operation when receiving a valid STOP invocation from the service user.

4.4.3.3.2 If the service provider accepts the STOP invocation,

a) it shall stop sending TRANSFER-DATA invocations; and

b) it shall send the STOP return.

4.4.3.4 Terminating

Upon receipt of a 'terminate procedure' event from the Association Control procedure, the procedure shall terminate by

a) stopping transmitting TRANSFER-DATA invocations; and

b) releasing the resources.

4.4.4 REQUIRED OPERATIONS

Table 4-8: Unbuffered Data Delivery Procedure Required Operations

Operations	Source	Extended	Refined	Procedure Blocking/ Non-Blocking
START	Common	N	N	Blocking
STOP	Common	N	N	Blocking
TRANSFER-DATA	Common	N	N	Non-Blocking

NOTE – Subsection 3.9.2.2.4.3 stipulates that a procedure using the TRANSFER-DATA operation refines or extends the `data` parameter of that operation. The Unbuffered Data Delivery procedure does not do that. The `data` syntax definition is left to a derived procedure or the service using this procedure.

4.4.5 CONFIGURATION PARAMETERS

The Unbuffered Data Delivery procedure does not have any configuration parameters.

NOTE – Consequently, the Unbuffered Data Delivery procedure has neither any configuration parameter that can be read by a service using this procedure nor any parameter that can be changed dynamically while the service instance executing the Unbuffered Data Delivery procedure instance is bound.

4.4.6 PROCEDURE STATE TABLE

Table 4-9: Unbuffered Data Delivery Procedure State Table

No.	Incoming Event	State 1 ('inactive')	State 2 ('active')
1	(StartInvocation)	IF "positive result" THEN (+StartReturn) → 2 ELSE (-StartReturn) ENDIF	'procedure to association abort 'protocol error'' → 1
2	(StopInvocation)	'procedure to association abort 'protocol error''	IF "positive result" THEN (+StopReturn) → 1 ELSE (-StopReturn) ENDIF
3	'data available'	Not applicable	IF (NOT "backpressure") THEN 'send data to underlying communications service' ELSE 'discard data' ENDIF
4	'invalid PDU 'xxx''	'procedure to association abort 'xxx''	'procedure to association abort 'xxx'' → 1
5	'terminate procedure'	'terminate itself'	'terminate itself'

Table 4-10: Procedure State Table Incoming Event Description References

Event	Reference
'data available'	4.4.3.2.4
'invalid PDU 'xxx''	3.2.3.6, 4.2.2.4. 'xxx' is one of the `diagnostic` values specified in 4.2.2.5.
'terminate procedure'	4.2.3, internal event from the Association Control procedure to all other procedures of the service instance in response to a protocol abort, a PEER-ABORT, or an UNBIND

Table 4-11: Procedure State Table Predicate Descriptions

Predicate	Evaluates to TRUE if
“positive result”	No reason for sending a negative return has been detected; that is, for the START invocation, none of the conditions in 3.7.2.3.1 applies, and for the STOP invocation, none of the conditions in 3.3.2.7.1 applies.

Table 4-12: Procedure State Table Boolean Flags

Flag	Set to TRUE if
“backpressure”	The underlying communications service does not accept the data unit to be transferred because of backpressure.

Table 4-13: Procedure State Table Simple Action References

Name	References
‘send data to underlying communications service’	4.4.3.2.1
‘discard data’	4.4.3.2.6
‘procedure to association abort ‘xxx’’	4.2.2.3, 4.2.2.5, raise ‘procedure to association abort ‘xxx’’ event with `diagnostic` set to ‘xxx’ to the Association Control procedure
‘terminate itself’	4.4.3.4

4.5 BUFFERED DATA DELIVERY

4.5.1 VERSION NUMBER

The version number of this procedure is 2.

4.5.2 DISCUSSION

4.5.2.1 Purpose

The Buffered Data Delivery procedure is intended to be used for the development of services that transfer bulk data from a service provider to a service user. The data to be delivered is structured into delimited data units by a service production process. These data units are either

a) service production data units, which contain data obtained from a service production process; or

b) service production event notifications, which contain information related to changes in the status of the production process.

The Buffered Data Delivery procedure is suitable for transfer of data under either of the following conditions, which are typical of space mission operations:

a) The service user requires delivery of the most-recent data available. Some data may be discarded and thus not sent to the service user, if necessary to maintain timeliness when communication backpressure occurs.

b) The service user requires delivery of all data requested. Delayed delivery is acceptable, if necessary to provide complete delivery of data when communication backpressure occurs.

For this purpose, the procedure allows the service instance to be configured to operate in one of the following delivery modes:

a) real-time;

b) complete.

NOTE – Strictly speaking, the delivery mode is a characteristic of the Buffered Data Delivery procedure rather than a characteristic of a service instance using this procedure. However, given that `delivery-mode` is a service management parameter that cannot be dynamically modified, and given that Service Management configures the service management parameters of all instances of a procedure type equally (see 2.3), all Buffered Data Delivery procedure instances associated with a given service instance will operate in the same delivery mode. Therefore one can also associate the service instance with that delivery mode.

4.5.2.2 Concept

4.5.2.2.1 Overview

The Buffered Data Delivery (BDD) procedure supports either of the following delivery modes:

a) **real-time**—the service provider ensures that blocks containing the desired minimum number of the most recent consecutive data units are transferred with a specified maximum latency; if that latency is exceeded, such blocks of data units are discarded and the service user is notified.

b) **complete—**the service provider transfers previously recorded data accepting potentially high latency in case backpressure on the ground link does not allow transferring the data in a timely manner or the service user requested the delivery after the data generation had happened. The service provider ensures that no data is discarded.

The complete delivery mode has the following objectives:

a) data retrieved from the recording buffer are delivered to the service user;

b) data units and event notifications are always delivered in sequence without discarding data.

The BDD procedure reacts to changes of the production status. Production status always refers to the production status of the service at the time the service instance using the BDD procedure is bound. If the procedure is used in real-time delivery mode, this also reflects the status of the production process generating the data to be delivered to the service user.

In complete mode, the production process generating the data may take place at a time when the service instance using the BDD procedure does not even exist. Nonetheless, problems during the production may arise resulting, for instance, in missing data. Depending on the local implementation of the Functional Resources involved in the production process, they may report events regarding changes of the resource status, and these events may be stored together with the to-be-delivered data in the recording buffer. The CSTS using the BDD procedure will specify how the recording buffer derives an aggregate recording buffer production status (see 4.5.7.5). During the data delivery to the user, the service may evaluate the previously recorded resource status change events and the recording buffer production status change events, which may then be reported to the user and explain, for example, why certain data are missing. Such status derived from previously stored events is different from the production status of the CSTS instance.

The service user can stop the data transfer at any time by invoking the STOP operation.

The formal specification of the delivery modes is provided in 4.5.3.

NOTES

1 Transfer of data in real-time delivery mode implies that data units are transmitted as soon as possible after their generation by an ongoing service production session.

2 Transfer of previously recorded data in complete delivery mode implies that data units are retrieved from a storage filled by an ongoing service production session or by service production sessions terminated in the meantime.

3 The delivery mode to be applied to a given service instance needs to be defined by the service using this procedure or by a derived procedure, or it may be delegated to Service Management.

The operations used by this procedure allow a service user to interact with a service provider to

a) request the start of the data transfer, specifying the selection criteria of the data to be transferred;

NOTE – Apart from the start and stop generation times that are included in the START invocation, the specification of the selection criteria of the data depends on the specific service or the derived procedure and is not defined by this procedure.

b) receive the data units that meet the service user's selection criteria;

c) receive, synchronized with the transfer of data units, notifications on events that have a direct impact on the production and/or delivery of data units; and

d) stop and optionally later re-start the delivery of data units applying the same or a different selection.

An event that may result in an event notification is

a) discardable: in case of backpressure affecting the communications service, notifications reporting such an event will be discarded; or

b) non-discardable: the notifications reporting such events are not discarded, and delivery to the service user is ensured.

NOTES

1 If a derived procedure introduces events in addition to those specified in 4.5.4.2.2.1.1, then this derived procedure will have to classify those events as discardable or non-discardable.

2 In complete delivery mode, all data units and event notifications are delivered to the service user, and therefore the distinction of discardable and non-discardable events is not relevant.

4.5.2.2.2 Buffering

4.5.2.2.2.1 General

As to achieve the characteristics of the real-time and complete delivery modes, the Buffered Data Delivery procedure uses two buffering mechanisms, the return buffer and the recording buffer. A return buffer, as shown in figure 4-1, is used by any instance of the Buffered Data Delivery procedure or a procedure derived from it to prepare data for the transfer to the service user, regardless of the delivery mode.

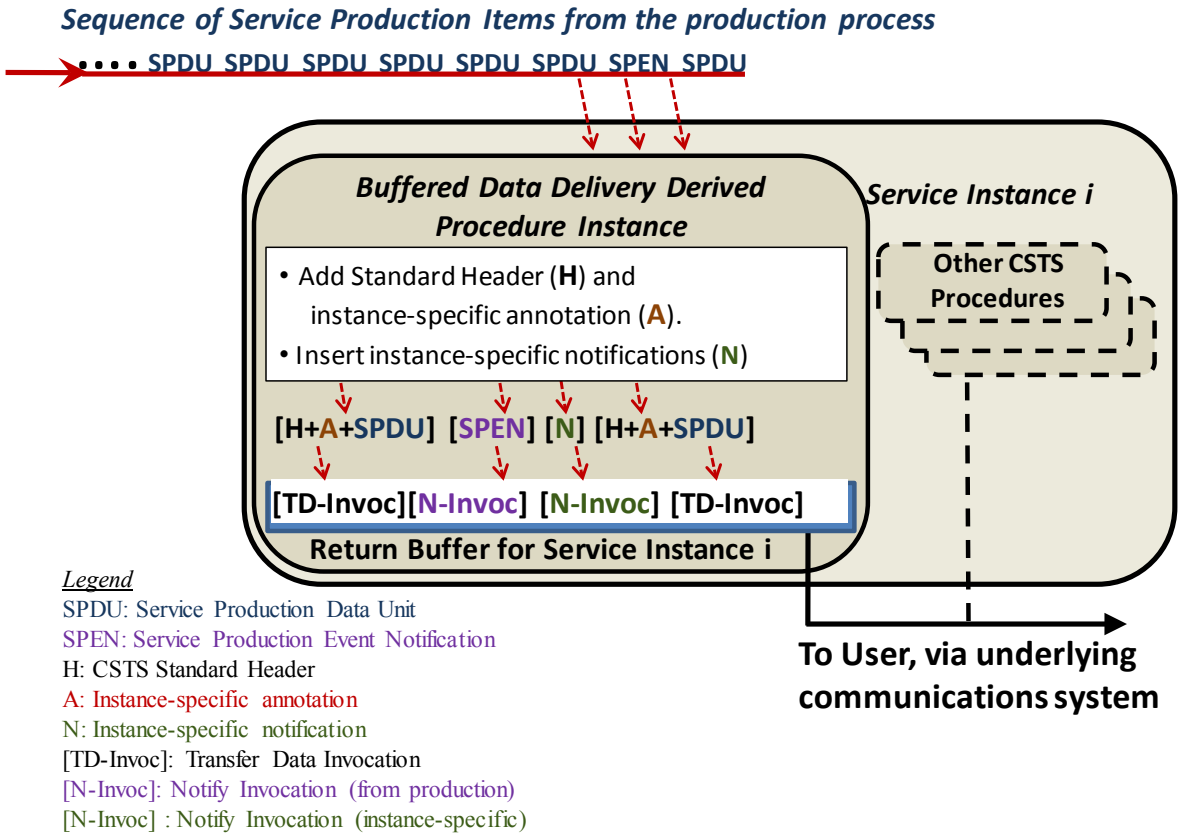

Figure 4-1: Services Using a Buffered Data Delivery Procedure

The recording buffer is used to store service production items for subsequent delivery by instances of services operating in complete delivery mode. Figure 4-2 shows both real-time and complete delivery mode service instances and the use of the recording buffer for providing data to service instances in complete delivery mode.

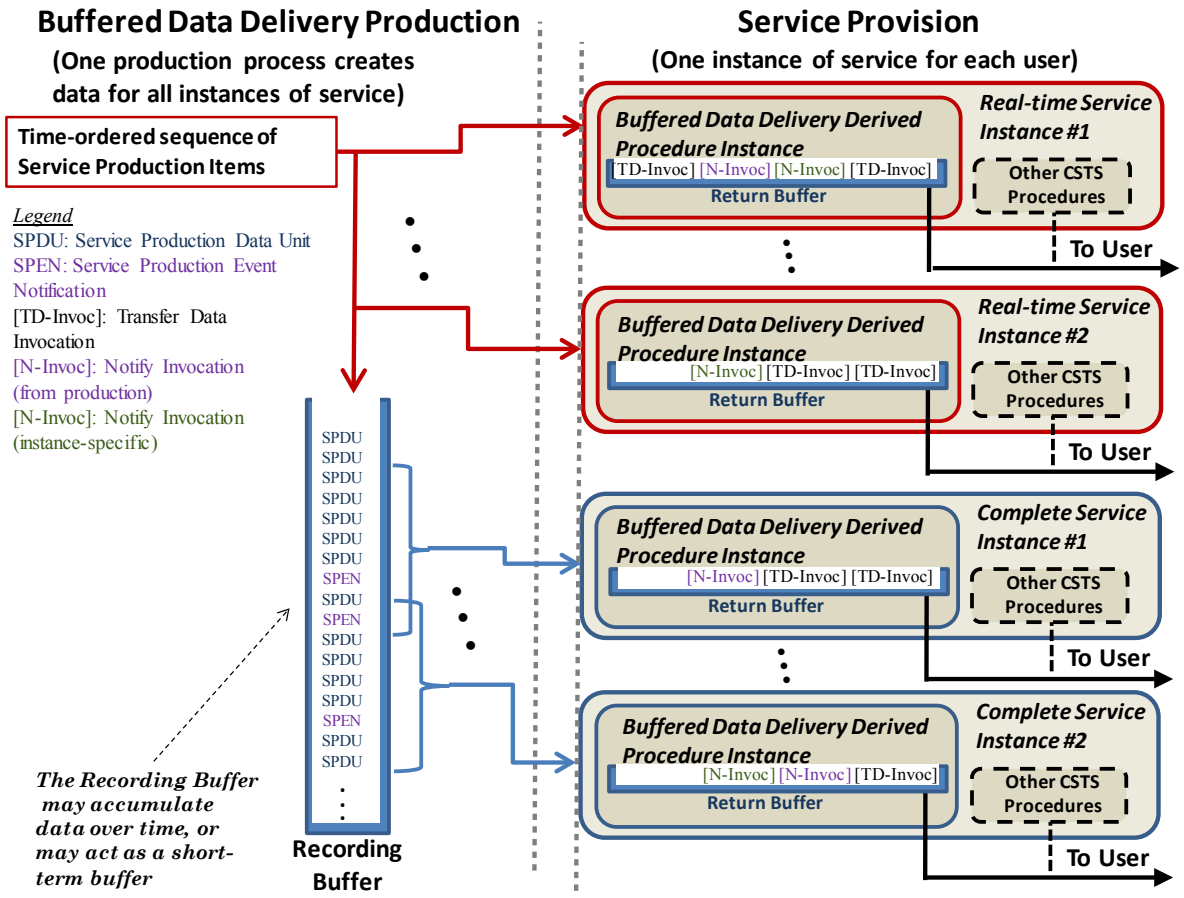

Figure 4-2: Real-Time and Complete Buffered Data Delivery Service Instances and Supporting Buffering Mechanisms

NOTE – For convenience, the following subsections are written as if

a) the contents of the recording buffer consisted of a list of data units that are either service production data units or service production event notifications; and

b) the contents of the return buffer consisted of an ordered list of data units that are either service production data units or service production event notifications.

However, this is not intended to constrain how the recording and return buffers are implemented in a real system. It is sufficient that a real system provides the externally visible behaviors that are specified herein.

4.5.2.2.2.2 Return Buffer

Each instance of a Buffered Data Delivery procedure or a procedure derived from it has a dedicated return buffer, which is used to accumulate data prior to the transfer to the service user. For each service production data unit, such a procedure adds the Standard Operation Header and other parameters, as applicable, to create a TRANSFER-DATA invocation (see 3.9). For each service production event notification, such a procedure adds the Standard Operation Header and other parameters, as applicable, to create a NOTIFY invocation. These invocations are inserted into the return buffer in the sequence in which they were generated.

The Buffered Data Delivery procedure or a procedure derived from it may also generate instance-specific notifications in the form of NOTIFY invocations to inform the service user of events or conditions that pertain only to the given instance of this procedure. A service using a Buffered Data Delivery procedure or a derived procedure, and requiring such notifications, needs to specify the events that trigger the generation of such procedure-type-specific notifications. It may also specify that multiple notifications of persistent conditions or recurring events are not to be sent in the absence of a to-be-transferred TRANSFER-DATA invocation and/or NOTIFY invocation.

The details of how the return buffer is used by a given instance of the Buffered Data Delivery procedure are specified in 4.5.3.2, 4.5.3.3, and 4.5.3.4.

4.5.2.2.2.3 Recording Buffer

Since the complete delivery mode is intended to deliver all data, even in the case of extended communications service outages or backpressure, or even several days after the service production session, extensive buffering is required. The recording buffer is intended to hold all service production data units and service production event notifications for significant periods of time.

A single instance of a recording buffer type may be shared by multiple instances of the service that share the service production resources that generate the type of data stored by that recording buffer type.

The recording buffer specification, in terms of normative behavior, queriable parameters, and notifiable events common to all recording buffer types, is provided in 4.5.7. Procedures derived from the BDD procedure may extend this recording buffer specification as required by the service using such a procedure.

4.5.3 BEHAVIOR

4.5.3.1 Starting

The service provider shall send a positive START return and perform the START operation invoked by the service user except if

a) the `production-status` is 'halted', in which case the START operation shall be rejected by sending a negative START return with the `diagnostic` value 'out of service'; or

b) the procedure is in State 2 ('active'), in which case the procedure shall request the Association Control procedure to abort the association with setting the `diagnostic` value to 'protocol error'.

NOTE – The service user will initiate the transfer of data by this procedure using the START operation including the selection criteria. The only selection criteria specified for this procedure are the start and end generation times (see 4.5.4.1.2).

4.5.3.2 Transferring Data and Notifications

4.5.3.2.1 Upon successful completion of the START operation, the service provider shall initialize the return buffer and insert the data that conform to the selection criteria in the form of TRANSFER-DATA and/or NOTIFY invocations into the return buffer as the data become available from the service production process or from the data retrieval from the recording buffer.

4.5.3.2.2 The service provider shall set the value of the `sequence-counter` parameter of the first TRANSFER-DATA invocation to be inserted into the return buffer to 0, as per 3.9.2.2.3.1.

4.5.3.2.3 The service provider shall handle the value of the `sequence-counter`, as per 3.9.2.2.3.

4.5.3.2.4 Upon successful completion of the START operation, the service provider shall inform the service user about events affecting the production or the transfer of the data by means of the NOTIFY operation.

4.5.3.2.5 All data and events shall be delivered in the order in which they were generated.

4.5.3.2.6 The service provider shall act in accordance with the applicable delivery mode.

4.5.3.2.7 The TRANSFER-DATA and NOTIFY invocations shall be inserted into the associated return buffer of the Buffered Data Delivery procedure only while the procedure state is 'active'.

4.5.3.2.7.1 At the time of insertion of a TRANSFER-DATA or NOTIFY invocation into an empty return buffer, the service provider shall start a timer called the release timer.

4.5.3.2.7.2 The duration from the time that the release timer is started until it expires is given by the `delivery-latency-limit` parameter, the value of which is configured by the service using this procedure or by a derived procedure.

4.5.3.2.7.3 For a given instance of the service, the return buffer shall accommodate a set number of TRANSFER-DATA and/or NOTIFY invocations. That number, given by the ReturnBufferSize parameter, is initially configured by the service using this procedure or by a derived procedure based on the procedure's `return-buffer-size` parameter.

4.5.3.2.7.4 The contents of the return buffer shall be passed to the communications service (in the form of one `ReturnBuffer` PDU) as soon as one of the following conditions is met:

a) the buffer becomes full; that is, the number of TRANSFER-DATA and/or NOTIFY invocations contained in the buffer is equal to the value of the ReturnBufferSize parameter;

b) the release timer expires;

c) one of the notifications 'end of data', 'production status change', 'production configuration change', or 'buffered data delivery configuration change' is inserted into the return buffer; or

d) the service user invokes the STOP operation.

4.5.3.2.7.5 The `ReturnBuffer` PDU shall contain the TRANSFER-DATA and NOTIFY invocations in the same sequence as they were inserted into the return buffer.

4.5.3.2.8 The insertion of invocations into the return buffer shall end when one of the following occurs:

a) no further data or notifications meeting the selection criteria specified in the START invocation are available;

b) data and/or notifications provided by service production have a generation time that is later than the `stop-generation-time` in the START invocation; however, in this case the 'end of data' notification is still to be inserted into the return buffer (see 4.5.3.2.9);

c) STOP is invoked by the service user;

d) the association is aborted.

4.5.3.2.9 In case the generated data contains a generation time that is later than the `stop-generation-time` in the START invocation, an 'end of data' notification shall be generated and inserted into the return buffer.

4.5.3.2.10 Further conditions triggering the 'end of data' notification may be defined by derived procedures.

NOTE – The definition of such conditions is needed, in particular, when the real-time delivery mode is used and the `stop-generation-time` may be 'undefined'.

4.5.3.3 Transferring Data and Notifications (Real-Time Delivery Mode)

4.5.3.3.1 Having built a TRANSFER-DATA invocation from the data received from service production ready for insertion into the return buffer constitutes the 'data available' event (see 4.5.6).

4.5.3.3.2 A NOTIFY invocation shall be built and inserted into the return buffer whenever one of the events 'end of data', 'production status change', 'production configuration change', or 'buffered data delivery configuration change' triggers.

4.5.3.3.3 Insertion of invocations into the return buffer shall begin with

a) the most recent data or notification with a generation time equal to or later than the time specified in the `start-generation-time` parameter in the START invocation; or

b) if the `start-generation-time` parameter in the START invocation is not set, with the most recent data or notification with a generation time equal to or later than the time at which the positive START return has been sent by the service provider.

4.5.3.3.4 The service provider shall insert notifications into the return buffer after the last data generated before the event occurrence and before the first data generated after the event.

NOTE – Such synchronous insertion of notifications is performed upon production status changes and according to further events being part of the procedure and/or service definition.

4.5.3.3.5 If the underlying communications service generates backpressure, that is, if the communications service does not accept the `ReturnBuffer` PDU and the `ReturnBuffer` PDU does not contain an 'end of data' notification, the service provider shall

a) discard this `ReturnBuffer` PDU;

b) clear the return buffer, with the exception of notifications associated with non-discardable events;

c) insert a 'data discarded due to excessive backlog' notification at the top of the return buffer; and

d) increase the size of the return buffer (ReturnBufferSize) by one plus the number of notifications of non-discardable events; that new size shall remain in effect until the contents of the return buffer are passed to the communications service, after which ReturnBufferSize shall be reverted to the original size, as configured via the `return-buffer-size` parameter.

NOTE – The temporary increase of the ReturnBufferSize ensures a minimum of data flow in case of backpressure. Otherwise, only 'data discarded due to excessive backlog' and non-discardable event notifications might be sent.

4.5.3.4 Transferring Data and Notifications (Complete Delivery Mode)

4.5.3.4.1 Having built a TRANSFER-DATA or NOTIFY invocation from the data retrieved from the recording buffer ready for insertion into the return buffer constitutes the 'data read from recording buffer' event (see 4.5.6).

NOTE – The format in which data and events are captured in the recording buffer is an implementation choice not prescribed by this Recommended Standard. Therefore it may be necessary to 'build' the corresponding invocations based on the recording buffer contents.

4.5.3.4.2 A NOTIFY invocation shall be built and inserted into the return buffer whenever one of the events 'end of data', 'production status change', 'production configuration change', or 'buffered data delivery configuration change' triggers.

NOTE – The events listed here may trigger during the delivery of the recording buffer contents to the service user. They are not related to service production event notifications previously stored in the recording buffer.

4.5.3.4.3 Insertion of invocations into the return buffer shall begin with the first data or notification generated at or after the time specified by the `start-generation-time` parameter in the START invocation and stored in the recording buffer.

4.5.3.4.4 Subsequent data units and notification records shall be retrieved from the recording buffer and inserted into the return buffer in the same order in which they were originally generated or received.

NOTE – Recording itself is outside the scope of the Buffered Data Delivery procedure. However, the key characteristics of a recording buffer suitable for this procedure are outlined in 4.5.2.2.2.3 and formally specified in 4.5.7.

4.5.3.4.5 When the underlying communications service has accepted the `ReturnBuffer` PDU, the service provider shall clear the return buffer and resume retrieving data and notifications from the recording buffer as described above.

4.5.3.5 Stopping

4.5.3.5.1 The service provider shall perform the STOP operation when receiving a valid STOP invocation from the service user.

4.5.3.5.2 If the service provider accepts the STOP invocation,

a) it shall stop inserting TRANSFER-DATA and NOTIFY invocations into the return buffer;

b) it shall immediately build from the return buffer contents a `ReturnBuffer` PDU and shall keep attempting to pass that `ReturnBuffer` PDU to the underlying communications service until it is accepted; and

c) it shall send the STOP return.

NOTE – In case the return buffer cannot be transmitted within a reasonable time, the return buffer may be discarded, as the service user is expected to invoke a PEER-ABORT as soon as the response timer expires.

4.5.3.6 Terminating

Upon receipt of a 'terminate procedure' event from the Association Control procedure, the Buffered Data Delivery procedure shall terminate by

a) stopping the insertion of invocations into the return buffer;

b) stopping the extraction of data and events from the recording buffer in case of complete delivery mode; and

c) clearing the return buffer, stopping the release timer, stopping all response timers, and releasing associated resources.

4.5.4 REQUIRED OPERATIONS

Table 4-14: Buffered Data Delivery Procedure Required Operations

Operations	Source	Extended	Refined	Procedure Blocking/Non-Blocking
START	Common	Y	N	Blocking
STOP	Common	N	N	Blocking
TRANSFER-DATA	Common	N	N	Non-Blocking
NOTIFY	Common	Y	N	Non-Blocking

NOTE – Subsection 3.9.2.2.4.3 stipulates that a procedure using the TRANSFER-DATA operation refines or extends the `data` parameter of that operation. The Buffered Data Delivery procedure does not do that. The `data` syntax definition is left to a derived procedure or the service using this procedure.

4.5.4.1 START (Confirmed)

4.5.4.1.1 General

The Buffered Data Delivery procedure shall extend the START operation defined in 3.7.2 by adding two parameters to the invocation and by adding values to the `diagnostic` parameter of the return.

4.5.4.1.2 Operation Parameters Definitions

NOTE – Table 4-15 shows the extension parameters of the START operation defined by this procedure.

Table 4-15: START Extension Parameters

Extension Parameters	Invocation	Return
`start-generation-time`	M	
`stop-generation-time`	M	

4.5.4.1.2.1 Extension Parameters Syntax

The type `BuffDataDelStartInvocExt`, as defined in F3.7, shall specify the syntax of the extension parameters of the START invocation.

4.5.4.1.2.2 `start-generation-time`

4.5.4.1.2.2.1 For the real-time delivery mode, if `start-generation-time` is 'undefined', the data transfer shall begin with the next data unit that is acquired from the data acquisition process.

4.5.4.1.2.2.2 For the real-time delivery mode, `start-generation-time`, when not 'undefined', must satisfy the following criteria:

a) `start-generation-time` must be equal to or later than the start time of the service instance provision period for this service instance;

b) `start-generation-time` must be earlier than the end time of the service instance provision period for this service instance;

c) if `stop-generation-time` and `start-generation-time` are not 'undefined', `start-generation-time` must be earlier than `stop-generation-time` (see 4.5.4.1.2.3).

4.5.4.1.2.2.3 For the complete delivery mode, the service provider shall deliver all available data that meet the delivery criteria regardless of the service production session in which they were acquired.

4.5.4.1.2.2.4 For the complete delivery mode, `start-generation-time` must not be set to 'undefined'.

4.5.4.1.2.2.5 For the complete delivery mode, `start-generation-time` must be earlier than `stop-generation-time` (see 4.5.4.1.2.3).

4.5.4.1.2.3 `stop-generation-time`

4.5.4.1.2.3.1 For the real-time delivery mode, if `stop-generation-time` is 'undefined', the service provider shall continue to transfer all data that are acquired from the service production session and satisfy the delivery criteria until either the service user invokes a STOP operation, the production terminates, or the association is released or aborted.

4.5.4.1.2.3.2 For the real-time delivery mode, `stop-generation-time` must satisfy the following criteria:

a) `stop-generation-time`, if not 'undefined', must be later than `start-generation-time`;

b) `stop-generation-time`, if not 'undefined', must be earlier than or equal to the end time of the service instance provision period for this service instance.

4.5.4.1.2.3.3 For the complete delivery mode, `stop-generation-time` must satisfy the following criteria:

a) it must not be 'undefined';

b) it must be later than `start-generation-time`;

c) it must be earlier than or equal to the end time of the service instance provision period for this service instance.

4.5.4.1.3 `diagnostic` Parameter Extension Value Definitions and Syntax

4.5.4.1.3.1 If a negative START return is sent, the `diagnostic` parameter shall use one of the `diagnostic` values specified in 3.7.2.3, or one of the following values:

a) 'missing time value'—for the complete delivery mode, the value of `start-generation-time` or `stop-generation-time` is 'undefined';

b) 'invalid start generation time'—the value of `start-generation-time` provided in the invocation is not valid;

c) 'invalid stop generation time'—the value of `stop-generation-time` provided in the invocation is not valid;

d) 'inconsistent time'—the value of `start-generation-time` is later than the value of `stop-generation-time`.

4.5.4.1.3.2 The type `BuffDataDelStartDiagnosticExt`, as defined in F3.7, shall specify the syntax of the `diagnostic` parameter of the START return, extended as listed in 4.5.4.1.3.1.

4.5.4.2 NOTIFY (Unconfirmed)

4.5.4.2.1 General

The Buffered Data Delivery procedure shall extend the NOTIFY operation defined in 3.11 by adding one permissible Event Identifier to the `event-name` parameter.

NOTE – Only the events specified in 4.5.4.2.2.1.1 a) and h) are events associated directly with the BDD procedure, and therefore the event OIDs are registered under the fwProceduresFunctionalities node of the OID tree (see figure D-1). The other events specified in 4.5.4.2.2.1.1 are emitted by Functional Resources and therefore registered under the crossSupportFunctionalities node of the OID tree. These events are forwarded to the service user by the BDD procedure by inserting them into the return buffer. Event-name and event-value remain as generated by the emitting Functional Resource. The relevant OIDs and data types are specified in the registry https://sanaregistry.org/r/functional_resources.

4.5.4.2.2 Invocation and Parameters

4.5.4.2.2.1 `event-name` Extension

4.5.4.2.2.1.1 The value of the `event-name` shall be one of the following:

a) one of the values specified for the common NOTIFY operation in 3.11.2.2.3;

NOTES

1 The notifications specified in the common NOTIFY are discardable for the Buffered Data Delivery procedure.

2 The 'production status' and 'production status change' event of a CSTS instance using the BDD procedure in 'complete' delivery mode and changes thereto depend on the 'resource status' of the associated recording buffer. Production status may also be affected by some kind of on-line procedure connected to resources, but will not be affected by the 'resource status' changes of those Functional Resources involved in the earlier filling of the recording buffer with

data and notifications. Those 'resource status change' events are stored in the recording buffer. Likewise, the 'production configuration change' event emitted by the Functional Resource representing a CSTS instance using the BDD procedure in 'complete' delivery mode will be triggered by on-line configuration changes of the recording buffer and possibly some on-line procedures, but not by configuration changes of those Functional Resources involved in the earlier filling of the recording buffer with data and notifications. Those 'configuration change' events are stored in the recording buffer.

b) 'data discarded due to excessive backlog' (`event-name`)—some data was discarded by the service provider because of timeliness considerations (real-time delivery mode);

   1) the `event-name` of this event shall contain the Functional Resource Name of the service triggering the event;

   2) unless otherwise specified by the service using that procedure or by a derived procedure, the associated `event-value` shall be empty;

   3) this notification is discardable;

c) 'recording buffer production status change' (`event-name`)—some data may have been lost because, during the service production session, a recording buffer production status change occurred, which may imply that a Functional Resource involved in the production process incurred a problem and therefore recording of service production data stopped; this service production event applies only to complete delivery mode (see 4.5.7.6):

   1) the `event-name` of this event shall contain the Functional Resource Name of the Functional Resource representing the recording buffer;

   2) the associated `event-value` shall contain the `production-status` parameter value of the recording buffer Functional Resource applicable since the 'recording buffer production status change' event triggered:

      i) the first part of the path specifying the type to be used is 'NotifyInvocation': 'eventValue': 'EventValue': 'qualifiedValues': 'SequenceOfQualifiedValue': 'SEQUENCE OF QualifiedValue', where this sequence has the length 1;

      ii) the only element of this sequence shall report the `production-status` of the recording buffer; the composition of the sequence is 'QualifiedValue': 'valid': 'TypeAndValue': 'Embedded': 'EMBEDDED PDV': 'SEQUENCE':

         – the first element of this sequence is 'identification': 'syntax': 'OBJECT IDENTIFIER', where the value of the OID is specified in the service type specific recording buffer Functional Resource;

– the second element of the sequence is ‘data-value’: ‘OCTET STRING’, where the value of this octet string is the BER encoded `ProdStat` type specified in the registry https://sanaregistry.org/r/functional_resources;

3) this notification is discardable;

NOTE – Each recording buffer generates a ‘recording buffer production status change’ event notification when the recording buffer detects the occurrence of this event and then stores the event for subsequent retrieval by BDD-using CSTSes operating in complete delivery mode. When a BDD procedure instance reads the ‘recording buffer production status change’ event notification from the recording buffer, that procedure creates a NOTIFY invocation that reports to the user of a service containing such procedure that a ‘recording buffer production production change’ had occurred during the production process.

d) ‘recording buffer production configuration change’ (`event-name`)—some expected data may have been lost and/or unexpected data may be found in the recording buffer because, during the service production session, the configuration of Functional Resources taking part in the production process was changed; this event applies only to complete delivery mode (see 4.5.7.7):

1) the `event-name` of this event shall contain the Functional Resource Name of the Functional Resource modeling the recording buffer;

2) unless otherwise specified by the service using that procedure or by a derived procedure, the associated `event-value` shall be empty;

NOTE – Each recording buffer generates a ‘recording buffer production configuration change’ event notification when the recording buffer detects the occurrence of this event and then stores the event for subsequent retrieval by BDD-using CSTSes operating in complete delivery mode. When a BDD procedure instance reads the ‘recording buffer production configuration change’ event notification from the recording buffer, that procedure creates a NOTIFY invocation that reports to the user of a service containing such procedure that a ‘recording buffer production configuration change’ had occurred during the production process.

e) ‘bdd recording buffer overflow’ (`event-name`)—some data may have been lost because, during the service production session, a recording buffer overflow occurred, and therefore recording of service production data stopped; this event applies only to complete delivery mode (see 4.5.7.8);

1) the `event-name` of this event shall contain the Functional Resource Name of the Functional Resource modeling the recording buffer;

2) unless otherwise specified by the service using that procedure or by a derived procedure, the associated `event-value` shall be empty;

3) this notification is discardable;

NOTE – Each recording buffer generates a 'recording buffer overflow' event notification when the recording buffer detects the occurrence of this event and then stores the event for subsequent retrieval by BDD-using CSTSes operating in complete delivery mode. When a BDD procedure instance reads the 'recording buffer overflow' event notification from the recording buffer, that procedure creates a NOTIFY invocation that reports to the user of a service containing such procedure that a 'recording buffer overflow' had occurred during the production process.

f) 'resource status change' (`event-name`)—the resource status of a Functional Resource involved in the production process has changed; this event applies only to complete delivery mode (see 4.5.7.4);

1) the `event-name` of this event shall contain the Functional Resource Name of the Functional Resource representing the production resource that experienced the resource status change;

2) the `event-value` of the 'resource status change' event shall report the value of the `resource-status` parameter of the Functional Resource after that status has changed:

i) the first part of the path specifying the type to be used is 'NotifyInvocation': 'eventValue': 'EventValue': 'qualifiedValues': 'SequenceOfQualifiedValue': 'SEQUENCE OF QualifiedValue', where this sequence has the length 1;

ii) the second part of the path is 'QualifiedValue': 'valid': 'TypeAndValue': 'Embedded': 'EMBEDDED PDV': 'SEQUENCE'

– the first element of this sequence is 'identification': 'syntax': 'OBJECT IDENTIFIER', where the value of the OID is specified in the specific Functional Resource;

– the second element of the sequence is 'data-value': 'OCTET STRING', where the value of this octet string is either (a) the BER encoded ASN.1 type `ResourceStat`, when the production resource directly adopts the standard `ResourceStat type`, or (b) a Functional Resource-specific BER encoded `resource-status` type with resource-specific refinements and/or substates

NOTE – The standard `ResourceStat` ASN.1 type and all Functional Resource-specific `resource-status` ASN.1 types (if any) are specified in SANA registry https://sanaregistry.org/r/functional_resources;

3) this notification is discardable;

NOTES

1 This event is used to notify resource status changes of production resources other than the recording buffer. Changes in the status of the recording buffer are reported through the 'production status change' event that is common to all CSTSes.

2 This event applies only to the complete delivery mode. In timely delivery mode, the resource status changes of individual production resources are not transferred through the return buffer; only changes to the aggregate production status are transferred through the 'production status change' event.

g) 'end of data' (`event-name`)—the service provider has no more data to send;

1) as stated in 4.5.3.2.10 the conditions triggering the 'end of data' insertion, in addition to the condition specified in 4.5.3.2.9, shall be defined by the service using that procedure or by the derived procedure;

2) the `event-name` of this event shall contain the Functional Resource Name of the service triggering the event;

3) the associated `event-value` shall be empty;

4) this notification is non-discardable;

NOTE – The end of data to be transferred is notified to the service user (see 4.5.4.2), but does not stop the procedure. The service user is expected to terminate the procedure by invoking the STOP operation when receiving this notification.

h) 'buffered data delivery configuration change' (`event-name`)—a dynamic modification of the return-buffer-size parameter or the delivery-latency-limit parameter has occurred:

1) the `event-name` of this event shall contain the `procedure-name` of the procedure instance in which the event occurs;

2) the associated event-value shall contain the current values of the return-buffer-size and delivery-latency-limit parameters;

i) the first part of the path specifying the type to be used is 'NotifyInvocation': 'eventValue': 'EventValue': 'qualifiedValues': 'SequenceOfQualifiedValue': 'SEQUENCE OF QualifiedValue', where this sequence has the length 2;

ii) the first element of this sequence shall report the `return-buffer-size`. The composition of the first element of the sequence is one of the following:

(a) 'QualifiedValue': 'valid': 'TypeAndValue': 'Embedded': 'EMBEDDED PDV': 'SEQUENCE':

– the first element of this sequence is ‘identification’: ‘syntax’: ‘OBJECT IDENTIFIER’, where the value of the OID is `pBDDreturnBufferSize`;

– the second element of the sequence is ‘data-value’: ‘OCTET STRING’, where the value of this octet string is the BER encoded `PBDDreturnBufferSizeType` type (see table 4-16);

(b) ‘QualifiedValue’: ‘unavailable’: ‘NULL’;

(c) ‘QualifiedValue’: ‘undefined’: ‘NULL’; or

(d) ‘QualifiedValue’: ‘error’: ‘NULL’;

iii) the second element of this sequence shall report the `delivery-latency-limit`. The composition of the second element of the sequence is one of the following:

(a) ‘QualifiedValue’: ‘valid’: ‘TypeAndValue’: ‘Embedded’: ‘EMBEDDED PDV’: ‘SEQUENCE’:

– the first element of this sequence is ‘identification’: ‘syntax’: ‘OBJECT IDENTIFIER’, where the value of the OID is `pBDDdeliveryLatencyLimit`;

– the second element of the sequence is ‘data-value’: ‘OCTET STRING’, where the value of this octet string is the BER encoded `PBDDdeliveryLatencyLimitType` type (see table 4-16);

(b) ‘QualifiedValue’: ‘unavailable’: ‘NULL’;

(c) ‘QualifiedValue’: ‘undefined’: ‘NULL’; or

(d) ‘QualifiedValue’: ‘error’: ‘NULL’;

NOTE – All relevant types are defined in F3.3 and F3.16.

3) this notification is discardable.

NOTES

1 A change of the configuration of a procedure can only be accomplished by invoking the associated directive by means of the EXECUTE-DIRECTIVE operation. Although the notification reporting the procedure configuration change may be discarded, the service user is nonetheless informed of the configuration change by the EXECUTE-DIRECTIVE return.

2 The Published Identifier for the above-defined new Event Identifier value for the ‘buffered data delivery configuration change’ event (4.5.4.2.2.1.1 h)) is specified in F3.16 as `pBDDconfigurationChange`.

4.5.4.2.2.1.2 The Published Identifiers for the above-defined new Event Identifier values for the 'data discarded due to excessive backlog' event (4.5.4.2.2.1.1 b)), and the 'end of data' event (4.5.4.2.2.1.1 g)) shall be registered in the SANA registry https://sanaregistry.org/r/functional_resources on the eventsId branch of the functionalResourceType Published Identifier for the CSTS's Functional Resource Type (see D6.2.3).

4.5.4.2.2.1.3 The Published Identifiers for the above-defined new Event Identifier values for the 'recording buffer production status change' event (4.5.4.2.2.1.1 c)), the 'recording buffer production configuration change' event (4.5.4.2.2.1.1 d)), and the 'recording buffer overflow' event (4.5.4.2.2.1.1 e)) shall be registered in the SANA registry https://sanaregistry.org/r/functional_resources on the eventsId branch of the functionalResourceType Published Identifier for the associated derived recording buffer Functional Resource Type (see D6.2.3).

4.5.4.2.2.1.4 The Published Identifiers for the above-defined new Event Identifiers value for the 'resource status change' events (4.5.4.2.2.1.1 f)) shall be registered in the SANA registry https://sanaregistry.org/r/functional_resources on the eventsId branch of the functionalResourceType Published Identifier for the Functional Resource Type (see D6.2.3) of each Functional Resource that comprises the production of the service..

4.5.4.2.2.2 `notification-invocation-extension` Extension

The NOTIFY invocation is not further extended, and therefore `notification-invocation-extension` shall be set to 'notUsed'.

4.5.5 CONFIGURATION PARAMETERS

The Buffered Data Delivery procedure configuration parameters that need to be configured in the context of the procedure shall be as defined in table 4-16.

NOTE – For each configuration parameter, the table identifies the engineering unit (if applicable), a cross reference to the use of the parameter in the specification of the procedure, whether the parameter may be read and/or dynamically modified, and the Parameter Identifier and type to be used in reporting the value of the parameter.

Table 4-16: Buffered Data Delivery Procedure Configuration Parameters

Parameters	Cross-Reference	Readable	Dynami-cally modify-able	Configuration Parameter Identifier and Type (F3.16)
`return-buffer-size` (in number of TRANSFER-DATA and/or NOTIFY invocations the buffer will accommodate)	4.5.3.2.7.3	Yes	Yes	`pBDDreturnBufferSize` `PBDDreturnBufferSizeType`
`delivery-latency-limit` (in seconds)	4.5.3.2.7.2	Yes	Yes	`pBDDdeliveryLatencyLimit` `PBDDdeliveryLatencyLimitType`
`delivery-mode`	4.5.2.2.2.1, 4.5.3.2.6	Yes	No	`pBDDdeliveryMode` `PBDDdeliveryModeType`

NOTE – A notification of the occurrence of the `pBDDconfigurationChange`, when the BDD procedure is operating in real time delivery mode, will be inserted into the return buffer by that procedure. A delay in the reporting of this event is kept to a minimum in that the return buffer is released to the underlying communications service as soon as such a notification is inserted into the return buffer. However, that notification may be discarded in case of backpressure on the link to the CSTS user. If the service using the Buffered Data Delivery procedure ensures that this event is notified to the service user rather than possibly being discarded, the service also needs to use a procedure that contains a NOTIFY operation and will, regardless of potential backpressure, never discard a notification. For example, the service could use the Notification procedure, through which the user could subscribe to the `pBDDconfigurationChange` event.

4.5.6 PROCEDURE STATE TABLE

Table 4-17: Buffered Data Delivery Procedure State Table

No.	Incoming Event	State 1 ('inactive')	State 2 ('active')
1	(StartInvocation)	IF "positive result" THEN (+StartReturn) → 2 'initialize return buffer' set "data ended" to FALSE ELSE (-StartReturn) ENDIF	'procedure to association abort 'protocol error'' → 1
2	(StopInvocation)	'procedure to association abort 'protocol error''	IF "positive result" THEN IF (NOT "buffer empty") THEN {pass buffer contents} ENDIF set "data ended" to FALSE 'stop all response timers' (+StopReturn) → 1 ELSE (-StopReturn) ENDIF
3	'data available'	Not applicable	IF "return buffer full" THEN {pass buffer contents} IF "backpressure" THEN 'Increment buffer size' 'notify 'data discarded' / 'empty'' 'copy non-discardable notifications' ENDIF 'insert data in return buffer' {init and start release timer} ELSE IF "buffer empty" THEN {init and start release timer} ENDIF 'insert data in return buffer' IF "return buffer full" THEN {attempt pass buffer contents} ENDIF ENDIF

No.	Incoming Event	State 1 ('inactive')	State 2 ('active')
4	'data read from recording buffer'	Not applicable	IF "buffer empty" THEN {init and start release timer} ENDIF 'insert data in return buffer' IF "return buffer full" THEN {transmit buffer} ENDIF
5	'real-time, release timer expired'	Not applicable	{pass buffer contents} IF "backpressure" THEN 'increment buffer size' 'notify 'data discarded' / 'empty'' 'copy non-discardable notifications' {init and start release timer} ENDIF
6	'complete, release timer expired'	Not applicable	{transmit buffer}
7	'end of data'	Not applicable	IF (NOT "data ended") THEN 'notify 'end of data' / 'empty'' {transmit buffer} set "data ended" to TRUE ENDIF
8	'production status change 'yyy'' [when Buffered Data Delivery procedure is in real-time delivery mode]	Not applicable	IF "return buffer full" THEN {pass buffer contents} IF "backpressure" THEN 'increment buffer size' 'notify 'data discarded' / 'empty'' 'copy non-discardable notifications' ENDIF ENDIF 'notify 'production status change' / 'yyy'' in return buffer {attempt pass buffer contents}
9	'production status change 'yyy'' [when Buffered Data Delivery procedure is in complete delivery mode]	Not applicable	'notify 'production status change' / 'yyy'' in return buffer {transmit buffer}
10	'production configuration change' [when Buffered Data Delivery procedure is in real-time delivery mode]	Not applicable	IF "return buffer full" THEN {pass buffer contents} IF "backpressure" THEN 'increment buffer size' 'notify 'data discarded' / 'empty'' 'copy non-discardable notifications' ENDIF ENDIF 'notify 'production configuration change' / 'empty'' in return buffer {attempt pass buffer contents}

No.	Incoming Event	State 1 ('inactive')	State 2 ('active')
11	'production configuration change' [when Buffered Data Delivery procedure is in complete-time delivery mode]	Not applicable	'notify 'production configuration change' / 'empty'' in return buffer {transmit buffer}
12	'buffered data delivery configuration change' [when Buffered Data Delivery procedure is in real-time delivery mode]	Not applicable	IF "return buffer full" THEN {pass buffer contents} IF "backpressure" THEN 'increment buffer size' 'notify 'data discarded' / 'empty'' 'copy non-discardable notifications' ENDIF ENDIF 'notify 'buffered data delivery configuration change' / 'empty'' in return buffer {attempt pass buffer contents}
13	'buffered data delivery configuration change' [when Buffered Data Delivery procedure is in complete delivery mode]	Not applicable	'notify 'buffered data delivery configuration change' / 'empty'' in return buffer {transmit buffer}
14	'invalid PDU 'xxx''	'procedure to association abort 'xxx''	'procedure to association abort 'xxx'' → 1
15	'terminate procedure'	'terminate itself'	'terminate itself'

Table 4-18: Procedure State Table Incoming Event Description References

Event	Reference
'data available'	4.5.3.3.1
'data read from recording buffer'	4.5.3.4.1
'end of data'	4.5.3.2.7.4, 4.5.3.2.8, 4.5.3.2.9, 4.5.3.2.10, 4.5.4.2.2.1.1
'production status change 'yyy''	B2.2.4. 'yyy' is one of the values specified in table B-1.
'production configuration change'	3.11.2.2.3.2 b)
'buffered data delivery configuration change'	4.5.4.2.2.1.1
'real-time, release timer expired'	4.5.3.2.7.2
'complete, release timer expired'	4.5.3.2.7.2
'invalid PDU 'xxx''	3.2.3.6, 4.2.2.4. 'xxx' is one of the `diagnostic` values specified in 4.2.2.5.
'terminate procedure'	4.2.3, internal event from the Association Control procedure to all other procedures of the service instance in response to a protocol abort, a PEER-ABORT, or an UNBIND

Table 4-19: Procedure State Table Predicate Descriptions

Predicate	Evaluates to TRUE if
“return buffer full”	The return buffer cannot accommodate the currently available data or notification.
“buffer empty”	The return buffer does not contain any data or notification.
“complete”	Delivery mode is complete.
“real-time”	Delivery mode is real-time.
“positive result”	No reason for sending a negative return has been detected; that is, for the START invocation, none of the conditions in 4.5.4.1.3.1 applies, and for the STOP invocation, none of the conditions in 3.3.2.7.1 applies.

Table 4-20: Procedure State Table Boolean Flags

Flag	Set to TRUE if
“backpressure”	The underlying communications service does not accept the contents of the return buffer because of backpressure.
“data ended”	The service provider has sent the ‘end of data’ notification.

Table 4-21: Procedure State Table Simple Action References

Name	References
'increment buffer size'	4.5.3.3.5
'set release timer to delivery latency limit'	4.5.3.2.7.2
'start release timer'	4.5.3.2.7.4
'stop release timer'	4.5.3.6 c)
'stop all response timers'	4.5.3.6 c)
'initialize return buffer'	4.5.3.2.1
'copy non-discardable notifications'	4.5.3.3.5
'notify 'xxx' / 'yyy''	4.5.3.2.4, (NotifyInvocation) with `event-name` set to 'xxx' and `event-value` set to 'yyy'. In case a notification does not use an `event-value`, 'yyy' shall be set to 'empty'
'submit content of return buffer to underlying communications service'	4.5.3.2.7.4
'procedure to association abort 'xxx''	4.2.2.3, 4.2.2.5, raise 'procedure to association abort 'xxx'' event with `diagnostic` set to 'xxx' to the Association Control procedure
'insert data in return buffer'	4.5.3.2.7
'reinitialize return buffer using the nominal size'	4.5.3.3.5
'terminate itself'	4.5.3.6

Table 4-22: Procedure State Table Compound Action Definitions

Name	Actions Performed
{attempt pass buffer contents}	'submit contents of return buffer to underlying communications service' IF successful THEN 'stop release timer' 'reinitialize return buffer using the normal size' ENDIF
{pass buffer contents}	'stop release timer' 'submit contents of return buffer to underlying communications service' IF successful THEN set "backpressure" to FALSE ELSE set "backpressure" to TRUE ENDIF 'reinitialize return buffer using the normal size'
{init and start release timer}	'set release timer to delivery latency limit' 'start release timer'
{transmit buffer}	'stop release timer' 'submit the contents of return buffer to underlying communications service' until accepted by that service 'reinitialize return buffer using the nominal size'

4.5.7 REQUIREMENTS FOR RECORDING BUFFERS FOR USE BY THE BUFFERED DATA DELIVERY PROCEDURE

4.5.7.1 Each recording buffer type shall be represented by an associated Functional Resource Type. A CSTS type using the BDD procedure or a procedure derived from it shall specify the associated recording buffer type and the associated Functional Resource Type.

NOTE – The Functional Resource Type shall be registered on the crossSupportFunctionalities subbranch of the OID tree structure (see figure K-6). More information regarding the registration of derived recording buffer Functional Resources and the associated events and parameters can be found in reference [I3].

4.5.7.2 The time between data being inserted into the recording buffer by the production process and availability of this data for retrieval from this buffer for transfer to a service user shall be kept to a minimum.

4.5.7.3 Multiple service instances requiring access to data generated by the same production process shall be able to share the same recording buffer.

4.5.7.4 The production process shall insert acquired service production data units and service production events as emitted by the Functional Resources taking part in the production process in the form of 'resource status change' notifications into the recording buffer.

NOTE – The formal specification of the Functional Resource type specific 'resource status change' events is provided in the Functional Resource registry https://sanaregistry.org/r/functional_resources. The format in which these events are captured in the recording buffer is a matter of local implementation and not prescribed by this Recommended Standard.

4.5.7.5 The CSTS using the BDD procedure shall specify how the recording buffer shall derive an aggregate production status from service production events such as the resource status change events reported by the Functional Resources representing the production process. Any changes of this aggregate 'recording buffer production status' shall be stored in the recording buffer as specified in 4.5.7.6.

4.5.7.6 If the recording buffer detects that the recording buffer production status has changed, the recording buffer shall

a) store a 'recording buffer production status change' event notification in the recording buffer (see 4.5.4.2.2.1.1 c));

b) emit an 'fr recording buffer production status change' event.

NOTE – Given that the 'recording buffer production status change' event is notifiable, a service including the Notify procedure could subscribe to it. However, the production process, and therefore the filling of the recording buffer, may take place well before a service instance that might subscribe to such service production event is bound. Such a service instance might not even exist at the time the 'recording buffer production status change' occurs, and, in general, it is therefore not useful if services using a recording buffer implement the capability to be notified of the 'recording buffer production status change' event in real-time.

4.5.7.7 If the recording buffer detects that the recording buffer production configuration has changed, the recording buffer shall

a) store a 'recording buffer production configuration change' event notification in the recording buffer (see 4.5.4.2.2.1.1 c));

b) emit an 'fr recording buffer production configuration change' event.

NOTE – Given that the ‘recording buffer production configuration change’ event is notifiable, a service including the Notify procedure could subscribe to it. However, the production process, and therefore the filling of the recording buffer, may take place well before a service instance that might subscribe to such a service production event is bound. Such a service instance might not even exist at the time the recording buffer production status change occurs, and, in general, it is therefore not useful if services using a recording buffer implement the capability to be notified of the ‘recording buffer production configuration change’ event in real-time.

4.5.7.8 If the recording buffer overflows, the recording buffer shall

a) store a ‘bdd recording buffer overflow’ event notification in the recording buffer (see 4.5.4.2.2.1.1 e)):

   1) the latest data unit previously stored on the recording buffer shall be deleted if that is necessary to have sufficient storage space for storing the ‘bdd recording buffer overflow’ event;

   2) following this event, no further service production data units shall be stored on the recording buffer;

   NOTE – It is normally expected that the recording buffer is sufficiently large to hold all data that might be accumulated during several service production sessions. The time span over which data is retained in the recording buffer, the policy for deleting data from the recording buffer, and the conditions under which the recording buffer begins to accept data following an overflow condition are outside the scope of this Recommended Standard. In general, this may be specified by the CSTS using a specific recording buffer type, or the service provider and service user will agree on a data custody transfer protocol.

b) emit an ‘fr recording buffer overflow’ event.

   NOTE – Given that the ‘fr recording buffer overflow’ event is notifiable, a service including the Notify procedure could subscribe to it. However, the production process, and therefore the filling of the recording buffer, may take place well before a service instance that might subscribe to the event is bound. Such a service instance might not even exist at the time the overflow occurs, and, in general, it is therefore not useful if services using a recording buffer implement the capability to be notified of the recording buffer overflow in real-time.

4.5.7.9 If the `resource-status` of the recording buffer changes, the recording buffer shall emit an ‘fr recording buffer resource status change’ event.

NOTES

1 The recording buffer is itself a Functional Resource, and as such maintains a `resource-status` of its own (see 2.2.2.2).

2 Given that the 'fr recording buffer resource status change' event is notifiable, a service including the NOTIFY procedure could subscribe to it. However, the production process, and therefore the filling of the recording buffer, may take place well before a service instance that might subscribe to the event is bound. Such a service instance might not even exist at the time the resource status changes, and, in general, it is therefore not useful if services using a recording buffer implement the capability to be notified of the recording buffer resource status change in real-time.

4.5.7.10 The recording of data and events shall be performed as specified for the given recording buffer type (see 4.5.7.1) and as determined by the controlling service agreement and service package, regardless of the state of any service instance and regardless of whether an association with any service user is established.

NOTE – The events recorded in the recording buffer relate only to the recording buffer itself and to the production process that needs to be active to produce the data to be stored in the recording buffer. Such service production events do not have any impact on the CSTS production status reported by the Functional Resource representing the CSTS instance that at some point in time will retrieve the data and events of interest from the recording buffer. The CSTS production status always refers to the current status of the CSTS instance while the recording buffer production status is the aggregate production status observed and captured during the production process that filled the recording buffer. Typically, the CSTSes that will transfer the data stored earlier in the recording buffer will not be bound or not even exist at the time the service-production-related events to be stored in the recording buffer occur. However, when a CSTS includes the BDD procedure or a procedure derived from it, and the recording buffer type is specified to record service production events and to record the 'recording buffer production status change' events, a CSTS may specify that the procedure used to read from the recording buffer shall extract such events from that buffer and report them synchronously with the data. However, a CSTS, regardless of its specification, will only be able to provide such notifications if the local implementation of the Functional Resources modeling the production process emit the required event notifications.

4.5.7.11 The 'fr recording buffer production status change', 'fr recording buffer production configuration change', 'fr recording buffer resource status change', and 'fr recording buffer overflow' events for each derived recording buffer Functional Resource Type shall be registered under the eventsId branch of the functionalResourceType Published Identifier for that derived recording buffer Functional Resource Type (see D6.2.3).

4.5.7.12 The Functional Resources that are relevant for the production process generating the data to be captured in a given recording buffer type should emit 'resource status change' event notifications, which will then be stored in the recording buffer.

4.5.7.13 The CSTS specification shall specify the recording buffer type to be used for that service type and the Functional Resources of which the 'resource status change' event notifications shall be captured in the recording buffer. Such events shall be inserted into the recording buffer after the last data acquired before the occurrence of the event and before the first data acquired following the event.

NOTE – The kind of notifications addressed in 4.5.7.12 will not be defined as events of the Functional Resource representing the recording buffer, but as events notified by the individual Functional Resources being part of the production process.

4.5.7.14 If a BDD-derived procedure extends the recording buffer Functional Resource to have buffer-operation-related events beyond the recording buffer overflow that are to be emitted in real-time, then those events shall be specified as notifiable events of the derived recording buffer Functional Resource and registered under the eventsId branch of the functionalResourceType Published Identifier for that derived recording buffer Functional Resource Type.

4.5.7.15 A Functional Resource Type representing a recording buffer shall have a queriable `recording-buffer-size` parameter that specifies the storage capacity of the recording buffer. The `recording-buffer-size` parameter for each derived recording buffer Functional Resource shall be registered under the parametersId branch of the functionalResourceType Published Identifier for that derived recording buffer Functional Resource Type (see D6.2.3).

NOTE – Given that the `recording-buffer-size` parameter is queriable, it is accessible to the GET operation as, for example, contained in the Information Query procedure. In particular, monitoring of this parameter by the service including the Buffered Data Delivery procedure, or a procedure derived from it, is possible if that service includes a procedure containing the GET operation as does, for example, the Information Query procedure.

4.6 DATA PROCESSING

4.6.1 VERSION NUMBER

The version number of this procedure is 2.

4.6.2 DISCUSSION

4.6.2.1 Purpose

The purpose of the Data Processing procedure is to provide transfer of data units from the service user to the service provider for processing of the data units by the service provider. This procedure does neither define the data units being transferred nor the processing to be performed by the service provider. It is assumed, however, that processing is part of service production and will be specified by the service using this procedure.

The Data Processing procedure is abstract and cannot be implemented directly because the specification is incomplete. To be implementable, derived procedures must provide the missing specifications.

4.6.2.2 Concept

The Data Processing procedure supports transfer of data units from the service user to the service provider by means of PROCESS-DATA operations. When the service provider receives a PROCESS-DATA invocation, it stores the invocation on an Input Queue and processes the contained data unit as soon as possible. The service provider processes the data sequentially in the order transmitted by the service user, processing only one unit at any point in time. This means that a CSTS will normally comprise a single instance of this procedure. Derived procedures may specify more complex data processing behaviors. The specification of this procedure offers the means needed in case a CSTS will comprise more than one instance of a procedure derived from the Data Processing Procedure. Such a service specification will then need to define, in particular, how concurrently executing procedure instances shall interact with the shared underlying service production in a conflict-free manner (see 4.6.7).

The ability of the service provider to process the data depends on the production status described in more detail in annex B. If `production-status` is ‘operational’, the service provider is able to perform data processing; otherwise, the service provider is unable to perform data processing. As long as `production-status` is ‘operational’, the service provider removes PROCESS-DATA invocations from the Input Queue and processes the enclosed data unit; otherwise, PROCESS-DATA invocations already queued remain in the queue until `production-status` becomes ‘operational’, until the service user stops the procedure, or until the associated service instance is aborted or unbound.

The service provider will inform the service user by means of a NOTIFY operation if `production-status` changes and the procedure instance is in the 'active' state. However, change of `production-status` to 'interrupted' does not necessarily mean that the user needs to take action to recover from this situation. If `production-status` changes to 'interrupted' while no PROCESS-DATA invocations are queued for processing, then there is a chance that `production-status` reverts back to 'operational' in time before the service user is actually affected by `production-status` presently being 'interrupted'. The notification that `production-status` is 'interrupted' has the purpose to make the service user aware that the processing of the data unit will be delayed until `production-status` becomes 'operational' and that the input queue might fill up if further PROCESS-DATA operations are invoked.

Successful completion of processing is reported to the service user by means of a NOTIFY operation only if a report has been explicitly requested for that data unit by the service user. As to enable requesting such report, this procedure has extended the PROCESS-DATA operation (see 4.6.4.1.4). Failure of processing is always notified to the service user.

In order to enable the identification of the data unit for which a report is issued, the PROCESS-DATA invocation that transfers the data unit also includes the `data-unit-id` parameter the value of which is used to refer to that data unit in notifications sent to the service user. The service user can freely choose the `data-unit-id` parameter value, and the service provider does not check if the identification of the related data unit is unique. If such uniqueness is required, it has to be ensured by the service user.

The method by which the size of the Input Queue for incoming PROCESS-DATA operations is configured is defined by a derived procedure or by the service using this procedure, possibly delegating the definition to service management. Handling of a queue overflow condition is left undefined by this procedure, and because such a condition cannot be excluded, this procedure specification remains abstract and cannot be directly implemented. In order to be implementable, derived procedures must provide the missing specification.

As this procedure neither specifies the content of the data units transferred for processing nor the type of processing to be performed, it does not specify what processing failures can occur and need to be notified to the service user, with the exception of production status changes that affect processing of data units. If the occurrence of a failure causes more than one data unit to be discarded, then the service provider sends only a single notification. All NOTIFY operations issued include information from which the service user can derive the state of processing.

The service user can terminate processing of data units at any time by invocation of the STOP operation. When PROCESS-DATA invocations are still queued for processing at the time the STOP invocation is received, the service provider discards all PROCESS-DATA invocations stored in the queue, that is, all invocations for which processing has not yet started, but the service provider completes processing of the data unit for which processing has already started, if that is possible. The user can avoid inadvertent discarding of data units by requesting a processing report for the last data unit in the sequence and not invoking the STOP operation before that processing report has been received.

4.6.3 BEHAVIOR

4.6.3.1 Starting

The service provider shall send a positive START return and perform the START operation invoked by the service user except if

a) the `production-status` is 'halted', in which case the START operation shall be rejected by sending a negative START return with the `diagnostic` value 'out of service'; or

b) the procedure is in State 2 ('active'), in which case the procedure shall request the Association Control procedure to abort the association with setting the `diagnostic` value to 'protocol error'.

NOTE – By invoking the START operation, the service user requests that the service provider prepares to receive PROCESS-DATA invocations for processing.

4.6.3.2 Transfer and Queuing of PROCESS-DATA Invocations

4.6.3.2.1 If the service provider receives a PROCESS-DATA invocation, then the invocation shall be queued until either processing of the enclosed data unit begins or the invocation is discarded.

NOTE – The service user will invoke the PROCESS-DATA operation for each data unit to be transferred to the service provider for processing.

4.6.3.2.2 The provider shall accept any value of the `data-unit-id` parameter received with a PROCESS DATA invocation and store it for later use in notifications related to the processing of the given data unit.

4.6.3.2.3 The size of the Input Queue for incoming PROCESS-DATA operations expressed as the number of PROCESS-DATA invocations the Input Queue can hold shall be defined using the `input-queue-size` procedure configuration parameter (see 4.6.5).

NOTE – The Input Queue decouples the timing of the processing of the PROCESS-DATA invocations from the transfer of those invocations.

4.6.3.2.4 The service provider shall queue received PROCESS-DATA invocations regardless of the value of `production-status`.

NOTE – If `production-status` is not 'operational', PROCESS-DATA invocations will not be processed (see 4.6.3.3), and the queue may fill up.

4.6.3.2.5 If the queue is full at the time the service provider receives a PROCESS-DATA invocation, the behavior of the provider is not defined by this procedure. Because of this missing specification, this procedure is abstract and cannot be directly implemented. To be implementable, derived procedures shall provide the missing specification.

4.6.3.3 Processing of Data Units

4.6.3.3.1 The service provider shall remove PROCESS-DATA invocations from the Input Queue and process the data units included as soon as possible, as long as `production-status` is 'operational'.

4.6.3.3.2 When the procedure has performed its processing (if any) of the data unit, the (processed) data unit shall be submitted to the associated service production functions (see 4.6.7). The data unit, as submitted to service production, shall consist of

a) the contents of the `data` and `data-unit-id` parameters of the PROCESS-DATA invocation;

b) the `procedure-name` of the Data Processing procedure; and

c) the `service-instance-identifier` of the service instance using the Data Processing procedure.

4.6.3.3.3 If `production-status` changes to a value other than 'operational', then the service provider shall suspend processing of data units until `production-status` changes to 'operational' again.

NOTES

1 Processing of data units for which processing has already started at the time `production-status` changes to 'interrupted' or 'halted' will fail and processing of such data units will not be resumed (see 4.6.3.3.6).

2 PROCESS-DATA invocations for which processing has not started remain queued and will be processed when `production-status` changes to 'operational'.

4.6.3.3.4 The service provider shall process the data units sequentially in the order in which they were received.

NOTES

1 The term sequential processing as used here excludes all concurrency and in particular means that, at most, one data unit is being processed by the procedure at any point in time. However, there may be more than one data unit being processed by the underlying service production.

2 It is assumed that a CSTS type using the Data Processing procedure comprises only a single instance of a derived procedure. Should that service type need to permit several instances of a derived procedure executing concurrently, it will need to specify how those procedure instances shall interact with the underlying shared service production in a conflict-free manner.

4.6.3.3.5 The `data-unit-id` parameter value of the PROCESS-DATA invocation shall be used as a reference to data units in any NOTIFY invocations sent by the service provider to the service user.

4.6.3.3.6 If any data unit has started but has not completed processing at the time `production-status` changes from 'operational' to 'interrupted' or 'halted', the service provider shall discard any data unit being processed by the procedure and direct the underlying service production to discard any in-process data units that are associated with the service instance that is executing the procedure.

NOTES

1 The requirement stated in 4.6.3.3.6 is valid only if a CSTS type comprising a procedure derived from the Data Processing procedure uses only a single instance of that derived procedure. If the CSTS type permits more than one instance of the derived procedure executing concurrently, it needs to specify the effect that the request to service production to discard data units emitted by an individual procedure instance shall have.

2 How service production can be requested to discard data units for which processing has started depends on the specific implementation and is therefore outside the scope of this Recommended Standard. The only assumption is that requesting the underlying service production to discard data units, although there are currently no data units being processed by service production, is not regarded an error condition.

4.6.3.4 Positive Feedback

The 'data unit processing completed' event may be notified by service production or may determined by a procedure derived from the Data Processing procedure itself if such a procedure processes the data unit without involving service production. Every CSTS therefore has to specify under which condition(s) the 'data unit processing completed' shall trigger in the context of the service.

Upon the 'data unit processing completed' event, and if the value of the `process-completion-report` parameter (see 4.6.4.1.4) in the associated PROCESS-DATA invocation was 'produce report', the service provider shall notify the service user with Event Identifier 'data unit processing completed'.

NOTE – This procedure defines only one report to be issued upon occurrence of the 'data unit processing completed' event. Derived procedures might define additional processing steps, the completion of which should be reported.

4.6.3.5 Notifications

4.6.3.5.1 Notification of Production Status Changes

4.6.3.5.1.1 The service provider shall notify the service user whenever `production-status` changes independent of the effect this change has on the processing of data units. To that end, the service provider shall invoke the NOTIFY operation with the `event-name` set to 'production status change' and the `event-value` set to the value of `production-status` of the Functional Resource representing the affected service instance after the change has occurred.

NOTES

1 When `production-status` changes to 'interrupted' or 'halted' this may imply that processing of a data unit fails; whether this is the case or not can be derived from the information provided in the notification.

2 Whether a data unit has started but not completed processing and will therefore be discarded can be derived from the information provided in the notification.

3 The notification of `production-status` having changed to 'interrupted' makes the service user aware that the processing of any data unit on the Input Queue will be delayed until `production-status` becomes 'operational' and that therefore the Input Queue might fill up if further PROCESS-DATA operations are invoked.

4.6.3.5.2 Notification of Production Configuration Changes

The service provider shall notify the service user when the production configuration is changed (see 3.11.2.2.3.2 b) while the service is bound.

4.6.3.5.3 Notification of Procedure Configuration Changes

The service provider shall notify the service user when the value of a dynamically modifiable configuration parameter is changed while the service is bound.

4.6.3.6 Stopping

NOTE – The service user will invoke the STOP operation to inform the service provider that the service user is stopping the sending of PROCESS-DATA invocations.

4.6.3.6.1 When an incoming STOP invocation is accepted, the service provider shall send a positive STOP return, and the procedure state shall change to State 1 ('inactive').

4.6.3.6.2 When receiving a STOP invocation while there are PROCESS-DATA invocations in the service provider's Input Queue, the service provider shall

a) discard any PROCESS-DATA invocations for which processing has not yet started; notifications shall not be issued for the data units enclosed in those PROCESS-DATA invocations; and

b) complete processing of the data units, if any, for which processing has already started.

NOTE – The procedure being stopped, the service provider cannot notify the service user when the remaining data unit completes processing.

4.6.3.7 Terminating

Upon receipt of a 'terminate procedure' event from the Association Control procedure, the procedure shall terminate by

a) completing processing of any data unit for which processing has already started;

b) discarding all PROCESS-DATA invocations for which processing has not yet started;

c) discarding pending notifications; and

d) releasing the resources.

4.6.3.8 Aborting

4.6.3.8.1 When detecting a condition that requires the procedure to request the Association Control procedure to abort the association, the procedure shall

a) complete processing of any data units for which processing has already started;

b) discard all PROCESS-DATA invocations for which processing has not yet started; that is, flush the service provider's Input Queue;

c) discard any pending notifications;

d) request the Association Control procedure to abort the association providing the relevant `diagnostic` value as specified in 3.6.2.2.1.2; and

e) release the resources.

4.6.4 REQUIRED OPERATIONS

Table 4-23: Data Processing Procedure Required Operations

Operations	Source	Extended	Refined	Procedure Blocking/Non-Blocking
START	Common	N	N	Blocking
STOP	Common	N	N	Blocking
PROCESS-DATA	Common	Y	N	Non-Blocking
NOTIFY	Common	Y	N	Non-Blocking

NOTE – Although there are no extensions or refinements to the START operation, this procedure defines an additional reason for returning a negative result with the `diagnostic` value 'out of service', as specified in 4.6.3.1 a).

4.6.4.1 PROCESS-DATA (Unconfirmed)

4.6.4.1.1 General

NOTE – Subsection 3.10.2.2.4 stipulates that a procedure using the PROCESS-DATA operation refines or extends the `data` parameter of that operation. The Data Processing procedure does not do that. The `data` syntax definition is left to a derived procedure or the service using this procedure.

The Data Processing procedure shall extend the PROCESS-DATA operation defined in 3.10 by adding one parameter to the invocation. The Data Processing procedure uses the unconfirmed variant of the PROCESS-DATA operation.

4.6.4.1.2 Operation Parameters Definitions

NOTE – Table 4-24 shows the extension parameters of the PROCESS-DATA operation defined by this procedure.

Table 4-24: PROCESS-DATA Extension Parameter

Extension Parameters	Invocation
`process-completion-report`	M

4.6.4.1.3 Extension Parameter Syntax

The type `DataProcProcDataInvocExt`, as defined in F3.8, shall specify the syntax of the extension parameter of the PROCESS-DATA invocation.

4.6.4.1.4 `process-completion-report`

The `process-completion-report` parameter shall specify whether the service provider shall invoke a NOTIFY operation to inform the service user that processing of the data unit enclosed in the PROCESS-DATA invocation has been completed successfully.

4.6.4.2 NOTIFY (Unconfirmed)

4.6.4.2.1 General

The Data Processing procedure shall extend the NOTIFY operation defined in 3.11 through the addition of six parameters to the invocation and through adding two permissible Event Identifier values to the `event-name` parameter.

4.6.4.2.2 Operation Parameters Definitions

NOTE – The most precise and meaningful reporting will only be obtained if the service user applies sequential numbers with a fixed increment for identifying the data units.

4.6.4.2.2.1 Overview

Table 4-25 shows the extension parameters of the NOTIFY operation defined by this procedure.

Table 4-25: NOTIFY Extension Parameters

Extension Parameters	Invocation
`data-unit-id-last-processed`	M
`data-processing-status`	C
`data-processing-start-time`	C
`data-unit-id-last-OK`	M
`data-processing-stop-time`	C
`production-status`	C

4.6.4.2.2.2 Extension Parameter Syntax

The type `DataProcNotifyInvocExt`, as defined in F3.8, shall specify the syntax of the extension parameters of the NOTIFY invocation.

4.6.4.2.2.3 `data-unit-id-last-processed`

For all notifications, the `data-unit-id-last-processed` parameter shall be present. Its value shall be set as follows:

a) if the service provider has not yet processed or attempted to process any data from the service user during the given association established by the service instance using this instance of the Data Processing procedure, the value of the `data-unit-id-last-processed` parameter shall be set to 'noDataProcessed', that is, 'null';

b) otherwise, the `data-unit-id-last-processed` parameter shall specify the identifier of the data unit, that is, the value of the `data-unit-id` parameter of that data unit that the service provider most recently processed or attempted to process, regardless of whether the data was successfully processed or an exception occurred.

4.6.4.2.2.4 `data-processing-status`

4.6.4.2.2.4.1 For all notifications, if the value of the `data-unit-id-last-processed` parameter is 'noDataProcessed', the `data-processing-status` parameter shall not be present.

4.6.4.2.2.4.2 Whenever the value of the `data-unit-id-last-processed` parameter is not 'noDataProcessed', the `data-processing-status` parameter shall be present and shall contain one of the following values representing the processing state of the data identified by `data-unit-id-last-processed`:

a) 'successfully processed'—the processing of the data completed; that is, the data is guaranteed to have been processed nominally;

b) 'processing interrupted'—the processing of the data started but did not complete because `production-status` became 'interrupted' or 'halted';

c) 'processing started' —the processing of the data started but did not yet complete.

4.6.4.2.2.4.3 Additional values of the `data-processing-status` can be introduced by using `dataProcessingStatusExtension` (see F3.8), if needed in procedures derived from the Data Processing procedure.

4.6.4.2.2.4.4 The Data Processing procedure does not extend the `data-processing-status` parameter, and therefore the CHOICE `dataProcessingStatusExtension` must not be selected (see F3.8).

4.6.4.2.2.5 data-processing-start-time

4.6.4.2.2.5.1 For all notifications, if the data-unit-id-last-processed is 'noDataProcessed', the data-processing-start-time parameter shall not be present.

4.6.4.2.2.5.2 Whenever data-unit-id-last-processed is not 'noDataProcessed', the data-processing-start-time parameter shall be present and shall contain the time at which the service provider started to process the data identified by the data-unit-last-processed parameter.

4.6.4.2.2.6 data-unit-id-last-OK

For all notifications, the data-unit-id-last-OK parameter shall be present. Its value shall be set as follows:

a) if no data have been successfully processed during the given association established by the service instance using this instance of the Data Processing procedure, the value of the data-unit-id-last-OK parameter shall be set to 'noSuccessfulProcessing';

b) otherwise, the data-unit-id-last-OK parameter shall specify the sequence number, that is, the value of the data-unit-id parameter of the most recent data unit that was successfully processed.

4.6.4.2.2.7 data-processing-stop-time

4.6.4.2.2.7.1 For all notifications, if the data-unit-id-last-OK is 'noSuccessfulProcessing', the data-processing-stop-time parameter shall not be present.

4.6.4.2.2.7.2 Whenever data-unit-id-last-OK is not 'noSuccessfulProcessing', the data-processing-stop-time parameter shall be present and shall contain the time at which processing of the data, identified by data-unit-id-last-OK, successfully completed.

4.6.4.2.2.8 production-status

4.6.4.2.2.8.1 For all notifications except 'production status change', the production-status parameter shall be present and shall contain the value of production-status at the time of the event notification (see annex B).

4.6.4.2.2.8.2 If the notified event is 'production status change', the production-status parameter shall be absent.

NOTE – The 'production status change' notification reports the post event production-status value by means of the event-value parameter (see 3.11.2.2.4.3).

4.6.4.2.3 `event-name` Extension

The value of the `event-name` shall be one of the following:

a) one of the values specified for the common NOTIFY operation in 3.11.2.2.3.2;

b) 'data processing completed' (`event-name`)—processing of the data unit identified in the parameter `data-unit-id-last-OK` completed successfully;

   1) the `event-name` of this event shall contain the procedure name of the procedure instance triggering the event;

   2) unless otherwise specified by the service using this procedure or by a derived procedure, the associated `event-value` shall be 'empty';

c) 'data processing configuration change' (`event-name`)—at least one of the dynamically modifiable configuration parameters defined in 4.6.5 has been changed;

   NOTE – The `input-queue-size` parameter defined in 4.6.5 is the only dynamically modifiable parameter of the Data Processing procedure.

   1) the `event-name` of this event shall contain the procedure name of the procedure instance in which the event occurs;

   2) the notification of the data processing configuration change event shall report the `input-queue-size` value by means of the `event-value` parameter:

      i) the first part of the path specifying the type to be used is 'NotifyInvocation': 'eventValue': 'EventValue': 'qualifiedValues': 'SequenceOfQualifiedValue': 'SEQUENCE OF QualifiedValue', where this sequence has the length 1;

      ii) if the qualifier of the to-be-reported value is not 'valid', then the second part of the path is one of the following: (a) 'QualifiedValue': 'unavailable': 'NULL'; (b) 'QualifiedValue': 'undefined': 'NULL'; or (c) 'QualifiedValue': 'error': 'NULL';

      iii) if the qualifier of the to-be-reported value is 'valid', then the second part of the path is 'QualifiedValue': 'valid': 'TypeAndValue': 'Embedded': 'EMBEDDED PDV', where the Object Identifier and type of the `input-queue-size` parameter are `pDPinputQueueSize` and `PDPinputQueueSizeType` (see table 4-26).

   NOTE – All relevant types are defined in F3.3 and F3.16.

4.6.4.2.4 The Published Identifier, that is, the Event Identifier, for the `event-name` of the 'data processing complete' event is specified in F3.16 as `pDPdataProcessingCompleted`.

4.6.4.2.5 The Published Identifier, that is, the Event Identifier, for the `event-name` of the 'data processing configuration change' event is specified in F3.16 as `pDPconfigurationChange`.

4.6.5 CONFIGURATION PARAMETERS

The Data Processing procedure configuration parameter that needs to be configured in the context of the procedure shall be as defined in table 4-26.

NOTE – For the configuration parameter, the table identifies the engineering unit, a cross reference to the use of the parameter in the specification of the procedure, whether the parameter may be read and/or dynamically modified, and the Parameter Identifier and type to be used in reporting the value of the parameter.

Table 4-26: Data Processing Procedure Configuration Parameters

Parameters	Cross-Reference	Readable	Dynamically modifiable	Configuration Parameter Identifier and Type (F3.16)
`input-queue-size` (in number of PROCESS-DATA invocations the queue will store)	4.6.3.2.3	Yes	Yes	`pDPinputQueueSize` `PDPinputQueueSizeType`

4.6.6 PROCEDURE STATE TABLE

Table 4-27: Data Processing Procedure State Table

No.	Incoming Event	State 1 ('inactive')	State 2 ('active')
1	(StartInvocation)	IF "positive result" THEN (+StartReturn) → 2 ELSE (-StartReturn) ENDIF	{procedure to association abort 'protocol error'} → 1
2	(StopInvocation)	{procedure to association abort 'protocol error'}	IF "positive result" THEN {initiate stop} → 1 ELSE (-StopReturn) ENDIF
3	(ProcessDataInvocation)	{procedure to association abort 'protocol error'}	'queue data unit'
4	'data unit ready'	Not applicable	'process data unit'
5	'data unit processing completed'	[ignore]	IF "report" THEN 'notify 'data processing completed' / 'empty'' ENDIF
6	'production status change to 'interrupted''	[ignore]	'discard data units in processing' 'notify 'production status change' / 'interrupted''
7	'production status change to 'halted''	[ignore]	'discard data units in processing' 'notify 'production status change' / 'halted''
8	'production status change to 'operational''	[ignore]	'notify 'production status change' / 'operational''
9	'production status change to 'configured''	[ignore]	'notify 'production status change' / 'configured''
10	'production configuration change'	[ignore]	'notify 'production configuration change' / 'empty''
11	'data processing configuration change'	[ignore]	'notify 'data processing configuration change' / 'procedure configuration parameter values'' (see 4.6.4.2.3 c) 2))
12	'invalid PDU' 'xxx'	{procedure to association abort 'xxx'}	{procedure to association abort 'xxx'} → 1
13	'terminate procedure'	'terminate itself'	'terminate itself'

Table 4-28: Procedure State Table Incoming Event Description References

Event	Reference
'data unit processing completed'	4.6.3.4
'data unit ready'	4.6.3.3.1, a PROCESS-DATA invocation is available at the head of the Input Queue, the production engine is ready to process the enclosed data unit, and `production-status` is 'operational'
'terminate procedure'	4.2.3, internal event from the Association Control procedure to all other procedures of the service instance in response to a protocol abort, a PEER-ABORT, or an UNBIND
'production status change to 'xxx''	B2.2.4
'production configuration change'	3.11.2.2.3.2 b)
'data processing configuration change'	4.6.4.2.3 c)
'invalid PDU 'xxx''	3.2.3.6, 4.2.2.4. 'xxx' is one of the `diagnostic` values specified in 4.2.2.5

Table 4-29: Procedure State Table Predicate Descriptions

Predicate	Evaluates to TRUE if
"positive result"	No reason for sending a negative return has been detected; that is, for the START invocation, none of the conditions in 3.7.2.3.1 or 4.6.3.1 a) (see NOTE below) applies, and for the STOP invocation, none of the conditions in 3.3.2.7.1 applies.
"report"	The `process-completion-report` parameter value in the associated (ProcessDataInvocation) is 'produce report'.

Table 4-30: Procedure State Table Simple Action References

Name	References
'queue data unit'	4.6.3.2.1, 4.6.3.2.4
'process data unit'	4.6.3.3
'complete data processing'	4.6.3.6.2 b)
'discard data units in processing'	4.6.3.3.6
'notify 'xxx' / 'yyy''	4.6.3.5.1.1, 4.6.3.5.2, 4.6.3.5.3, 4.6.4.2.3 a), 4.6.4.2.3 b), 4.6.4.2.3 c), (NotifyInvocation) with `event-name` set to 'xxx' and `event-value` set to 'yyy'. In case a notification does not use an `event-value`, 'yyy' shall be set to 'empty'
'procedure to association abort 'xxx''	4.2.2.3, 4.2.2.5, raise 'procedure to association abort 'xxx'' event with `diagnostic` set to 'xxx' to the Association Control procedure
'clear the input queue'	4.6.3.6.2 a), 4.6.3.8.1 b), remove and discard all PROCESS-DATA invocations from the Input Queue
'terminate itself'	4.6.3.7

Table 4-31: Procedure State Table Compound Action Definitions

Name	Actions Performed
{initiate stop}	'clear the input queue' 'complete data processing' (+StopReturn)
{procedure to association abort 'xxx'}	'clear the input queue' 'discard data units in processing' 'procedure to association abort 'xxx''

4.6.7 REQUIREMENTS FOR PRODUCTION PROCESSING IN SUPPORT OF THE DATA PROCESSING PROCEDURE

4.6.7.1 General

Because the Data Processing procedure can be used in a CSTS to process almost any kind of data, the nature of the production processing that supports such a CSTS is highly dependent on the nature of the data itself and the kind of processing that is to be performed.

While the essential functions performed by production processing are specific to the nature of the individual CSTS, the Data Processing procedure levies certain data management and data accounting requirements on all production processes, regardless of the nature of the data processing being performed. Specifically, production processing must (a) carry the identification of each data unit being processed, (b) discard in-process data units at the request of the Data Processing procedure, and (c) report back to the Data Processing procedure upon the completion of processing of each data unit.

4.6.7.2 Data unit identification

Production processing shall tag each data unit being processed with the `data-unit-id` of the source PROCESS-DATA invocation, the `procedure-name` of the source Data Processing procedure instance, and the `service-instance-identifier` of the source CSTS instance.

NOTE – The identification information for each data unit is supplied by the Data Processing procedure, as defined in 4.6.3.3.2.

4.6.7.3 Discarding of data units

Upon receipt of a 'discard data units' request from an instance of the Data Processing procedure executing on a CSTS instance (see 4.6.3.3.6), production processing shall discard all data units tagged with the `service-instance-identifier` specified in that request.

NOTES

1 The extent into the production processing to which the discarding of data units applies is dependent on the specific nature of the CSTS and the ability of production processing to identify each data unit and extract it. For example, if the data processing associated with a given CSTS type at some stage transforms data units from multiple CSTS instances into a product from which the contributions of individual data units cannot be removed, then the discarding of data units must be limited to a processing step before the generation of that product. Each CSTS should specify the extent into production processing that discarding of data units is meaningful.

2 As described in 4.6.3.3.6, Note 1, nominally a CSTS will use only a single instance of the Data Processing procedure. It is possible for a CSTS to use multiple instances of the Data Processing procedure concurrently, but the behavior and the interactions among those multiple instances is undefined in this Recommended Standard and must be addressed by the CSTS specification. However, if the nature of the CSTS is such that discarding of data units on the basis of the source Data Processing procedure instance is appropriate to the service, the `procedure-name` component of the data unit tag can be used to further identify the data units to be discarded.

4.6.7.4 Reporting completion of data processing

Production processing shall notify the source Data Processing procedure instance upon completion of processing of each data unit submitted by that procedure instance, using the `data-unit-id` of that data unit.

NOTES

1 As described in 4.6.3.4, the Data Processing procedure uses the data unit processing completed information to confirm processing completion to the CSTS user.

2 The meaning of completion of processing of a data unit is specific to each CSTS type. For example, for a given service it could be defined as the estimated time for the beginning of radiation of the data unit. Each CSTS specification should specify how data unit processing completion is defined in the context of that service and specify the basis upon which completion is determined (e.g., some combination of measured and estimated times).

4.7 BUFFERED DATA PROCESSING

4.7.1 VERSION NUMBER

The version number of this procedure is 2.

4.7.2 DISCUSSION

4.7.2.1 Purpose

The purpose of the Buffered Data Processing procedure is to support transfer of large volumes of data at high data rates from the service user to the service provider, for processing by the service provider for which the maximum latency of data units may have to be limited. For this purpose, the procedure allows a service to select one of the following transfer modes:

a) complete transfer mode, in which data are transmitted to the service provider completely based on the flow control capabilities of the underlying data communications service, and any latency that might be implied is accepted;

b) timely transfer mode, in which the service provider ensures that data units will not be queued longer than a configurable latency limit and will discard the oldest unprocessed data units if the capacity of the configurable Input Queue is exceeded.

The Buffered Data Processing procedure is abstract and cannot be implemented directly because the specification is incomplete. To be implementable, derived procedures must provide the missing specifications. The missing information is explicitly identified in the applicable subsection(s) of this section.

4.7.2.2 Concept

The Buffered Data Processing procedure extends the Data Processing procedure (see 4.6) and in particular provides specifications for the handling of Input Queue overflow conditions that are left unspecified by the Data Processing procedure. The specification of this behavior makes the Buffered Data Processing procedure implementable.

In order to support transfer of large data volumes at high data rates, the service user blocks PROCESS-DATA invocations into a data unit called forward buffer. The forward buffer is used exclusively for the purpose of transferring the PROCESS-DATA invocations from the service user to the service provider. When the service provider receives a forward buffer, it extracts the PROCESS-DATA invocations and places them on the Input Queue in the order they have been inserted into the forward buffer by the service user. The maximum size of the forward buffer is expressed as the number of PROCESS-DATA invocations that may be stored in the buffer. It is a configuration parameter that is specified by the service using the procedure or by a derived procedure.

The service user may also send individual PROCESS-DATA invocations that, as specified for the parent procedure, are not enclosed in a forward buffer. The service provider will handle such unbuffered PROCESS-DATA invocations in the same manner as a forward buffer enclosing a single PROCESS-DATA invocation.

The Buffered Data Processing procedure defines two modes for the transfer of PROCESS-DATA invocations from the service user to the service provider; these modes differ in the method of handling or preventing overflow of the Input Queue:

a) In complete transfer mode, the service provider stops reading data from the underlying data communications service when the available space on the Input Queue drops below the maximum size of a forward buffer and resumes reading data when there is sufficient room on the Input Queue to store all PROCESS-DATA invocations that might be included in a maximum-sized forward buffer. This approach has the effect that backpressure is built up on the transport layer, eventually preventing the service user from transmitting further data. In effect, data will always be transferred completely, accepting any delay that might be implied.

b) In timely transfer mode, when there is insufficient room on the Input Queue to store all PROCESS-DATA invocations received within a forward buffer, the service provider discards as many of the oldest PROCESS-DATA invocations, for which processing has not started, as needed to store all PROCESS-DATA invocations contained in the most recently received forward buffer. The objective of this approach is to minimize the overall processing latency at the expense of dropping data with the highest accumulated latency.

The service specification may fix the transfer mode to be used as part of the service specification or may require the selection of the transfer mode by means of a service management parameter.

When adding a PROCESS-DATA invocation to the Input Queue in timely transfer mode, the service provider starts a latency timer for that PROCESS-DATA invocation and sets its initial value to the value defined by the parameter `processing-latency-limit`. The service provider discards the PROCESS-DATA invocation if the latency timer expires before processing of the data unit enclosed in the PROCESS-DATA invocation starts.

When data are discarded in timely transfer mode because of queue overflow or because of expiry of the processing latency timer (see 4.7.3.2.2.5), the service user is not notified. This procedure assumes that detection and handling of data loss is performed by higher processing layers. Where this assumption does not apply, notification of the service user will have to be added by derived procedures.

4.7.2.3 Derivation

The Buffered Data Processing procedure extends the Data Processing procedure specified in 4.6 by the following features:

a) buffering of PROCESS-DATA invocations for the purpose of transfer from the service user to the service provider;

b) support for timely and complete transfer modes for PROCESS-DATA invocations including specifications how Input Queue overflow shall be prevented or handled;

c) support of a processing latency limit in timely transfer mode to constrain the time a PROCESS-DATA invocation may be queued by the service provider before processing of the enclosed data unit starts.

4.7.3 BEHAVIOR

4.7.3.1 Starting

The Buffered Data Processing procedure shall be started as defined in the parent procedure in 4.6.3.1.

4.7.3.2 Transfer and Queuing of PROCESS-DATA Invocations

NOTE – The following behavior is specified in addition to the specifications for the parent procedure in 4.6.3.2.

4.7.3.2.1 Forward Buffer

4.7.3.2.1.1 PROCESS-DATA invocations may be grouped into a forward buffer, which shall be handled by the service provider in its entirety.

NOTES

1 The forward buffer is used exclusively for the purpose of transferring the PROCESS-DATA invocations from the service user to the service provider and is not an operation invocation or response.

2 The service user may also send individual PROCESS-DATA invocations as specified for the parent procedure in 4.6.3.2.

4.7.3.2.1.2 The maximum forward buffer size, expressed as the maximum number of PROCESS-DATA invocations that can be stored in a forward buffer, shall be specified by the service using this procedure or by a derived procedure, when setting of this parameter may be delegated to Service Management.

NOTES

1 The number of PROCESS-DATA invocations included in a forward buffer is at the discretion of the service user as long as it is less than or equal to the specified maximum forward buffer size.

2 Even if, for a given service type, the forward buffer will not be used as only individual PROCESS-DATA invocations will be sent, the size of the forward buffer needs to be specified. In case only individual PROCESS-DATA invocations shall be handled, the size will be limited to a single invocation.

4.7.3.2.1.3 The maximum forward buffer size shall be less than the configured Input Queue size.

4.7.3.2.1.4 Upon receipt of a forward buffer containing more PROCESS-DATA invocations than the specified maximum, this forward buffer shall be considered an invalid PDU as per 3.2.3.6 f). The procedure shall issue the 'procedure to association abort' event (refer to 4.2.2.3) setting the `diagnostic` value to 'forward buffer too large'.

4.7.3.2.1.5 When receiving a forward buffer, the service provider shall extract the PROCESS-DATA invocations and place them on the Input Queue in the same order as they have been inserted into the forward buffer by the service user.

4.7.3.2.1.6 When receiving an individual unbuffered PROCESS-DATA invocation as specified in the parent procedure (see 4.6), the service provider shall handle it in the same way as a forward buffer containing a single PROCESS-DATA invocation.

4.7.3.2.2 Data Transfer Modes

4.7.3.2.2.1 The service provider shall support the following transfer modes for the reception of forward buffers:

a) complete transfer mode;

b) timely transfer mode.

NOTES

1 The complete transfer mode ensures that all data transmitted by the service user are received, queued, and processed by the service provider, accepting any delay that might be implied.

2 In timely transfer mode, the service provider will discard PROCESS-DATA invocations that are queued longer than given by the value of the `processing-latency-limit` parameter (see 4.7.2.2). In addition, the service provider will discard PROCESS-DATA invocations with the highest accumulated latency when the Input Queue overflows.

4.7.3.2.2.2 The transfer mode to be applied shall be defined by the service using the procedure or by a derived procedure, or it may be delegated to Service Management.

4.7.3.2.2.3 In complete transfer mode, the service provider shall stop reading data from the data communications service when the available space on the Input Queue drops below the maximum forward buffer size and shall resume reading data from the data communications service when the available space on the Input Queue enables accommodating at least the number of PROCESS-DATA invocations that can be included in a maximum-sized forward buffer.

NOTES

1 When reading from the data communications service is suspended, backpressure will build up that may eventually prevent the service user from sending further PROCESS-DATA invocations.

2 In this situation, the service user will not be able to terminate the procedure nominally by invocation of the STOP operation until all transmitted data have been read and queued by the service provider. The only way to terminate the procedure earlier is by invocation of PEER-ABORT.

3 This specification assumes that a service will only use a single instance of the Buffered Data Processing procedure if that is operated in complete transfer mode. If more than one instance is used then all instances of the Buffered Data Processing procedure as well as any other procedures communicating via the same connection may be blocked if one of the instances stops reading data from the communications service.

4 Most of the points addressed in the notes above will apply regardless of the specifics of the underlying communications technology, as long as such technology has the characteristics as specified in 1.3.1. The behavior, as described in the notes above, applies fully in case the underlying communication is as specified in reference [2].

4.7.3.2.2.4 In timely transfer mode, when the available space on the Input Queue does not allow storing all PROCESS-DATA invocations extracted from a forward buffer, the service provider shall discard as many of the oldest PROCESS-DATA invocations for which processing has not started as needed to queue all PROCESS-DATA invocations received.

NOTE – The service user is not notified that PROCESS-DATA invocations have been discarded.

4.7.3.2.2.5 The parameter `processing-latency-limit` shall specify the maximum time that a PROCESS-DATA invocation may be queued in timely transfer mode before processing of the enclosed data unit has to start.

4.7.3.2.2.6 A value of zero for the `processing-latency-limit` parameter shall specify that the processing latency shall not be controlled by the service provider; that is, the 'processing latency timer expired' event (see table 4-35) shall not occur.

NOTE – As in complete data transfer mode, the service provider does not control the processing latency; that is, except as provided in 4.7.3.2.2.5, data units will be discarded only in case the Input Queue overflows. The `processing-latency-limit` parameter is set to zero as long as complete transfer mode applies.

4.7.3.2.2.7 The value of the parameter `processing-latency-limit` shall be specified by the service using the procedure or by a derived procedure, or it may be delegated to Service Management.

4.7.3.2.2.8 When placing a PROCESS-DATA invocation on the Input Queue, the service provider shall start a specific latency timer for that PROCESS-DATA invocation if the value of the `processing-latency-limit` parameter is not zero.

4.7.3.2.2.9 The initial value of latency timer shall be set to the value specified by the parameter `processing-latency-limit`.

4.7.3.2.2.10 When processing of the data unit enclosed in the PROCESS-DATA invocation starts or the PROCESS-DATA invocation is discarded, the associated latency timer shall be canceled.

4.7.3.2.2.11 If the latency timer expires before processing of the enclosed data unit starts, the PROCESS-DATA invocation shall be discarded.

NOTE – The service user is not notified that a PROCESS-DATA invocation has been discarded because of expiry of the latency timer.

4.7.3.3 Processing of Data Units

Data units shall be processed as specified for the parent procedure in 4.6.3.3.

4.7.3.4 Positive Feedback

Positive feedback shall be provided as specified for the parent procedure in 4.6.3.4.

4.7.3.5 Notifications

Events shall be notified as specified for the parent procedure in 4.6.3.5 with the exception specified in 4.7.4.2.

4.7.3.6 Stopping

The procedure shall be stopped as specified for the parent procedure in 4.6.3.6.

4.7.3.7 Terminating

The procedure shall terminate as specified for the parent procedure in 4.6.3.7.

4.7.3.8 Aborting

The procedure shall handle the need to request the Association Control procedure to abort the association as specified for the parent procedure in 4.6.3.8.

4.7.4 REQUIRED OPERATIONS

Table 4-32: Buffered Data Processing Procedure Required Operations

Operations	Source	Extended	Refined	Procedure Blocking/Non-Blocking
START	Data Processing	N	N	Blocking
STOP	Data Processing	N	N	Blocking
PROCESS-DATA	Data Processing	N	N	Non-Blocking
NOTIFY	Data Processing	Y	N	Non-Blocking

4.7.4.1 PROCESS-DATA (Unconfirmed)

4.7.4.1.1 The Buffered Data Processing procedure uses the unconfirmed variant of the PROCESS-DATA operation.

4.7.4.1.2 The `data` syntax definition is left to the service using this procedure.

NOTE – Subsection 3.10.2.2.4 stipulates that a procedure using the PROCESS-DATA operation refines or extends the `data` parameter of that operation. The Buffered Data Processing procedure does not do that.

4.7.4.2 NOTIFY (Unconfirmed)

4.7.4.2.1 General

The Buffered Data Processing procedure shall inherit the notifications defined for the parent procedure in 4.6.4.2.3.

Notification shall be performed as defined in the parent procedure in 4.6.3.5.

4.7.4.2.2 `event-value`

4.7.4.2.2.1 Except for the 'data processing configuration change' (`event-name`) event the `event-value` specifications of the parent procedure shall apply.

4.7.4.2.2.2 For the 'buffered data processing configuration change' (`event-name`) event, the `event-value` shall report the values of the dynamically modifiable parameters `maximum-forward-buffer-size`, `input-queue-size`, and `processing-latency-limit`, defined in 4.7.5. The first part of the path specifying the type to be used is 'NotifyInvocation': 'eventValue': 'EventValue': 'qualifiedValues': 'SequenceOfQualifiedValue': 'SEQUENCE OF QualifiedValue', where this sequence has the length 3. The first element of this sequence shall report the `maximum-forward-buffer-size`, the second the `input-queue-size`, and the third the `processing-latency-limit` parameter value. For each of the three parameters, the following shall apply: If the qualifier of the to-be-reported value is not 'valid', then the second part of the path is one of the following: (a) 'QualifiedValue': 'unavailable': 'NULL'; (b) 'QualifiedValue': 'undefined': 'NULL'; or (c) 'QualifiedValue': 'error': 'NULL'. If the qualifier of the to-be-reported value is 'valid', then the second part of the path is 'QualifiedValue': 'valid': 'TypeAndValue': 'Embedded': 'EMBEDDED PDV', where the OID and type of the `maximum-forward-buffer-size` parameter are `pBDPmaxForwardBufferSize` and `pBDPmaxForwardBufferSizeType`, the OID and type of the `input-queue-size` parameter are `pDPinputQueueSize` and `PDPinputQueueSizeType`, and the OID and type of the `processing-latency-limit` parameter are `pBDPprocessingLatencyLimit` and `PBDPprocessingLatencyType` (see table 4-33). All relevant types are defined in F3.3 and F3.16.

4.7.5 CONFIGURATION PARAMETERS

4.7.5.1 The Buffered Data Processing procedure configuration parameters that need to be configured in the context of the procedure shall be as defined in table 4-33.

NOTE – For each configuration parameter, the table identifies the engineering unit (if applicable), a cross reference to the use of the parameter in the specification of the procedure, whether the parameter may be read and/or dynamically modified, and the Parameter Identifier and type to be used in reporting the value of the parameter.

Table 4-33: Buffered Data Processing Procedure Configuration Parameters

Parameters	Cross-Reference	Read-able	Dynami-cally modifiable	Configuration Parameter Identifier and Type (F3.16)
`data-transfer-mode`	4.7.3.2.2	Yes	No	`pBDPdataTransferMode` `PBDPdataTransferModeType`
`maximum-forward-buffer-size` (in number of PROCESS-DATA invocations the buffer will store)	4.7.3.2.1.2	Yes	Yes	`pBDPmaxForwardBufferSize` `pBDPmaxForwardBufferSizeType`

`input-queue-size` (in number of PROCESS-DATA invocations the queue will store)	4.6.3.2.3	Yes	Yes	`pDPinputQueueSize`
				`PDPinputQueueSizeType`
				(inherited from the parent Data Processing procedure)

`processing-latency-limit` (in milliseconds)	4.7.3.2.2.5	Yes	Yes	`pBDPprocessingLatencyLimit` `PBDPprocessingLatencyLimitType`

4.7.6 PROCEDURE STATE TABLE

Table 4-34: Buffered Data Processing Procedure State Table

No.	Incoming Event	State 1 ('inactive')	State 2 ('active')
1	(StartInvocation)	*IF* *"positive result"* *THEN* set "reading suspended" to FALSE *(+StartReturn)* *→ 2* *ELSE* *(-StartReturn)* *ENDIF*	*{procedure to association abort 'protocol error'}* *→ 1*
2	(StopInvocation)	*{procedure to association abort 'protocol error'}*	*IF* *"positive result"* *THEN* *{initiate stop}* *→ 1* *ELSE* *(-StopReturn)* *ENDIF*
3	(ProcessDataInvocation)[1]	*{procedure to association abort 'protocol error'}*	IF "timely mode" THEN IF "queue overflow" THEN 'discard oldest data units' ENDIF ENDIF *'queue data unit'* IF "complete mode" THEN IF "queue full" THEN 'suspend reading' set "reading suspended" to TRUE ENDIF ENDIF

[1] In terms of Service Provider behavior, handling of an incoming PROCESS-DATA invocation and handling of an incoming forward buffer containing only one PROCESS-DATA invocation are identical.

No.	Incoming Event	State 1 ('inactive')	State 2 ('active')
4	(ForwardBuffer)	{procedure to association abort 'protocol error'}	FOR_EACH (ProcessDataInvocation) IN (ForwardBuffer) IF "timely mode" THEN IF "queue overflow" THEN 'discard oldest data units' ENDIF ENDIF 'queue data unit' IF "complete mode" THEN IF "queue full" THEN 'suspend reading' set "reading suspended" to TRUE ENDIF ENDIF ENDFOR_EACH
5	'data unit ready'	*Not applicable*	*'process data unit'*
6	'data unit processing completed'	*[Ignore]*	*IF* *"report"* *THEN* *'notify 'data processing completed' / 'empty''* *ENDIF* IF "reading suspended" THEN IF (NOT "queue full") THEN 'resume reading' set "reading suspended" to FALSE ENDIF ENDIF
7	'processing latency timer expired'	Not applicable	'discard data unit'
8	'production status change to 'interrupted''	*[Ignore]*	*'discard data units in processing'* *'notify 'production status change' / 'interrupted''*
9	'production status change to 'halted''	*[Ignore]*	*'discard data units in processing'* *'notify 'production status change' / 'halted''*
10	'production status change to 'operational''	*[Ignore]*	*'notify 'production status change' / 'operational''*
11	'production status change to 'configured''	*[Ignore]*	*'notify 'production status change' / 'configured''*
12	'production configuration change'	*[Ignore]*	*'notify 'production configuration change' / 'empty''*
13	'data processing configuration change'	*[Ignore]*	'notify 'data processing configuration change' / 'procedure configuration parameter values'' (see 4.7.4.2.2.2)
14	'invalid PDU 'xxx''	*{procedure to association abort 'xxx'}*	*{procedure to association abort 'xxx'}* *→ 1*
15	'terminate procedure'	*'terminate itself'*	*'terminate itself'*

Table 4-35: Procedure State Table Incoming Event Description References

Event	Reference
'data unit processing completed'	*4.6.3.4*
'data unit ready'	*4.6.3.3.1*
'processing latency timer expired'	4.7.3.2.2.11
'terminate procedure'	*4.2.3*
'production status change to 'xxx''	*B2.2.4*
'production configuration change'	*3.11.2.2.3.2 b)*
'data processing configuration change'	4.7.4.2.2.2
'invalid PDU 'xxx''	3.2.3.6, 4.2.2.4, 4.7.3.2.1.4. 'xxx' is one of the `diagnostic` values specified in 4.2.2.5 or 4.7.3.2.1.4

Table 4-36: Procedure State Table Predicate Descriptions

Predicate	Evaluates to TRUE if
"queue overflow"	There is not sufficient space on the Input Queue to store the PROCESS-DATA invocations received (see 4.7.3.2.2.4); the transfer mode is 'timely'.
"queue full"	There is not enough space on the Input Queue to store all PROCESS-DATA invocations that might be contained in a maximum-sized forward buffer (see 4.7.3.2.2.3); the transfer mode is 'complete'.
"complete mode"	Complete transfer mode is in effect (see 4.7.3.2.2).
"timely mode"	Timely transfer mode is in effect (see 4.7.3.2.2).
"positive result"	*No reason for sending a negative return has been detected; that is, for the START invocation, none of the conditions in 3.7.2.3.1 or 4.6.3.1 a) applies, and for the STOP invocation, none of the conditions in 3.3.2.7.1 applies.*
"report"	*The `process-completion-report` parameter value in the associated (ProcessDataInvocation) is 'produce report' (see 4.6.4.1.4).*

Table 4-37: Procedure State Table Boolean Flags

Predicate Flag	Set to TRUE if
"reading suspended"	In complete transfer mode, reading data from the data communications service has been suspended.

Table 4-38: Procedure State Table Simple Action References

Name	References
'queue data unit'	4.7.3.2
'suspend reading'	4.7.3.2
'resume reading'	4.7.3.2
'discard oldest data units'	4.7.3.2
'discard data unit'	4.7.3.2.2.11
'discard data units in processing'	*4.6.3.3.6*
'clear the Input Queue'	*4.6.3.6.2 a)*
'complete data processing'	*4.6.3.6.2 b)*
'notify 'xxx' / 'yyy''	*4.6.3.5.1.1, 4.6.3.5.2, 4.6.3.5.3, 4.6.4.2.3 a), 4.6.4.2.3 b), 4.6.4.2.3 c),* 4.7.4.2.2, *(NotifyInvocation) with* `event-name` *set to 'xxx' and* `event-value` *set to 'yyy'. In case a notification does not use an* `event-value`*, 'yyy' shall be set to 'empty'*
'procedure to association abort 'xxx''	4.2.2.3, 4.2.2.5, raise 'procedure to association abort 'xxx'' event with `diagnostic` set to 'xxx' to the Association Control procedure, where the `diagnostic` value may be one of those specified in 3.6.2.2.1.2 or the value specified in 4.7.3.2.1.4
'terminate itself'	*4.6.3.7*

Table 4-39: Procedure State Table Compound Action Definitions

Name	Actions Performed
{initiate stop}	*'clear the input queue'* *'complete data processing'* *(+StopReturn)*
{procedure to association abort 'xxx'}	*'clear the input queue'* *'discard data units in processing'* *'procedure to association abort 'xxx''*

4.8 SEQUENCE-CONTROLLED DATA PROCESSING

4.8.1 VERSION NUMBER

The version number of this procedure is 2.

4.8.2 DISCUSSION

4.8.2.1 Purpose

The purpose of the Sequence-Controlled Data Processing procedure is

a) to provide strict sequential transfer and processing of data units in the sequence defined by the service user; and

b) to enable the service user to resynchronize transfer and processing of data units in case a problem is detected during processing of a data unit.

The procedure is intended for use in Cross Support Transfer Services that involve sequence-controlled data transfer and processing and that need confirmation of data transfer and status reports on the ongoing production process.

The Sequence-Controlled Data Processing procedure is abstract and cannot be implemented directly because the specification is incomplete. To be implementable, derived procedures must provide the missing specifications. The missing information is explicitly identified in the applicable subsection(s) of this section.

4.8.2.2 Concept

The Sequence-Controlled Data Processing procedure extends the Data Processing procedure (see 4.6). Before data units can be transferred for processing, the service user invokes a START operation, which contains the `data-unit-id` value the service provider shall accept in the first PROCESS-DATA invocation after the successful completion of the START operation.

When the service provider receives the first PROCESS-DATA invocation, it verifies that the contained `data-unit-id` value is equal to the one defined in the START operation; otherwise, the service provider rejects the PROCESS-DATA invocation.

NOTE – The Sequence-Controlled Data Processing procedure uses the confirmed variant of the PROCESS-DATA operation.

For all following PROCESS-DATA invocations, the service provider verifies that the `data-unit-id` is one greater than the one received in the previous accepted PROCESS-DATA invocation; if that is not the case, the PROCESS-DATA invocation is rejected.

Before a PROCESS-DATA invocation can finally be accepted, the service provider checks

a) whether `production-status` is not 'halted';

b) whether the included earliest and latest production times are consistent;

c) whether the Input Queue is not full; and

d) whether the service provider is in the 'active.processing' state (see table 4-44).

If any of these checks fails, the PROCESS-DATA invocation is rejected.

NOTE – Further details on production status changes are defined in annex B.

After acceptance of the PROCESS-DATA invocation, the service provider sends a positive return to the service user and buffers the PROCESS-DATA invocation on the Input Queue. The service provider starts processing not earlier than the `earliest-data-process-start-time` and not later than the `latest-data-process-start-time` included in the PROCESS-DATA invocation. In case no `earliest-data-process-start-time` is given for the data unit, processing starts as soon as possible, regardless if the `latest-data-process-start-time` is given or not; otherwise, the service provider starts data unit processing within the time frame defined by the earliest and latest processing times.

NOTE – Regardless of the service user having defined `earliest-data-process-start-time` and `latest-data-process-start-time`, the service provider processes the data units in the sequence defined by the `data-unit-id` in the PROCESS-DATA operation, which in this procedure serves as a sequence counter. This means that the PROCESS-DATA operations are always processed in the order they have been received and buffered.

Reporting of the completion of processing steps is performed as defined by the parent Data Processing procedure in 4.6.3.4.

If processing of a data unit fails, the service provider enters a 'locked' state and notifies the service user of the problem. While in the 'locked' state, the service provider does not perform any data processing; nor does it accept any new PROCESS-DATA invocations from the service user. To recover from the 'locked' state, the service user may issue a STOP invocation; the service provider then clears the Input Queue, exits the 'locked' state, and terminates the procedure. After the completion of the STOP operation, the service user issues a START invocation (as soon as `production-status` becomes 'operational' again—see NOTE) with the next `data-unit-id` the service provider accepts.

NOTE – Because the STOP operation puts the procedure in the 'inactive' state, the service user in this case must use other means to obtain the `production-status` value if such means are provided by the service.

As an alternative to invoking a STOP operation followed by a START operation, the service user may invoke an EXECUTE-DIRECTIVE operation with the `directive-identifier` 'reset' to clear the Input Queue and exit the 'locked' state, where the `directive-qualifier` parameter defines the next `data-unit-id` the service provider accepts.

NOTE – The 'locked' state is a substate of the 'active' state that does not affect the state of the service instance.

4.8.2.3 Derivation

The Sequence-Controlled Data Processing procedure extends the parent Data Processing procedure specified in 4.6 by the following features:

a) use of the confirmed variant of the PROCESS-DATA operation;

b) start of processing of the data units not earlier than an earliest and not later than a latest processing start time defined by the service user;

c) transition to a 'locked' state and notification of the service user in case processing of a data unit fails;

NOTE – In such a case, the service provider is blocked and does not accept any further PROCESS-DATA invocations, as the strict sequential processing cannot be guaranteed anymore.

d) resynchronization of data unit processing by the service user in case of a problem detected during data unit processing.

4.8.3 BEHAVIOR

4.8.3.1 Starting

4.8.3.1.1 The behavior of the Sequence-Controlled Data Processing procedure regarding starting shall be as specified for the Data Processing procedure in 4.6.3.1, with the additional requirements specified in the following subsection.

NOTE – The service user will invoke the START operation with the `first-data-unit-id` parameter to request that the service provider prepare to receive data units for processing, and to specify the permissible `data-unit-id` value in the first PROCESS-DATA invocation that will be sent.

4.8.3.1.2 The service provider shall record the `first-data-unit-id` parameter value that shall be accepted with the first PROCESS-DATA invocation.

4.8.3.1.3 If `production-status` is 'interrupted' at the time the START invocation is received, the service provider shall issue a negative return with the `diagnostic` value set to 'unable to comply'.

4.8.3.2 Transfer and Queuing of PROCESS-DATA invocations

4.8.3.2.1 The behavior of the Sequence-Controlled Data Processing procedure regarding the transfer and queuing of PROCESS-DATA invocations shall be as specified for the parent Data Processing procedure in 4.6.3.2, with the additional requirements specified in this subsection.

4.8.3.2.2 The service provider shall accept the first PROCESS-DATA invocation after a successful START operation only if the value of the `data-unit-id` parameter is that of the `first-data-unit-id` parameter of the previously accepted START invocation.

4.8.3.2.3 The service provider shall accept a subsequent PROCESS-DATA invocation only if the `data-unit-id` parameter value of the previous accepted PROCESS-DATA invocation is one less than the current value.

4.8.3.2.4 If the `data-unit-id` value has reached the maximum value given by the range of the `data-unit-id` parameter and has to be incremented by one, the parameter value shall wrap around to zero.

4.8.3.2.5 If the service provider detects an unexpected value of the `data-unit-id` parameter of the PROCESS-DATA operation, it shall issue a negative PROCESS-DATA return with the `diagnostic` value set to 'out of sequence'.

4.8.3.2.6 If the Input Queue used for buffering of incoming PROCESS-DATA operations is full at the time the service provider receives a PROCESS-DATA invocation, the service provider shall issue a negative PROCESS-DATA return with the `diagnostic` value set to 'unable to store'.

4.8.3.2.7 If `production-status` is 'halted' at the time of reception of a PROCESS-DATA invocation, the service provider shall issue a negative PROCESS-DATA return with the `diagnostic` value set to 'unable to process'.

4.8.3.2.8 If `production-status` is not 'halted' at the time of reception of a PROCESS-DATA operation invocation but the service provider is in the 'locked' substate, the service provider shall issue a negative PROCESS-DATA return with the `diagnostic` value set to 'service instance locked'.

4.8.3.2.9 If the service provider does not accept a PROCESS-DATA invocation, it shall not queue the data unit by appending it to the Input Queue, but discard it.

4.8.3.3 Processing of Data Units

4.8.3.3.1 The behavior of the Sequence-Controlled Data Processing procedure regarding the processing of data units while in State 2.1 (‘active.processing’) shall be as specified for the Data Processing procedure in 4.6.3.3, with the additional requirements specified in the following subsection.

NOTE – While the Sequence-Controlled Data Processing procedure is in State 2.2 (‘processing.locked’), no data units are processed. The conditions for entering and leaving the ‘processing.locked’ state are specified in 4.8.3.7. Table 4-44 specifies all behavioral differences of the procedure when in state 2.1 and 2.2, respectively.

4.8.3.3.2 Data Processing Start Time

4.8.3.3.2.1 The service provider shall start processing of the data unit not earlier than the time defined by the `earliest-data-process-start-time` parameter and not later than the time defined by the `latest-data-process-start-time` parameter, both defined in the PROCESS-DATA operation.

4.8.3.3.2.2 If the value of the `earliest-data-process-start-time` parameter is ‘undefined’ in the PROCESS-DATA invocation, the service provider shall process the data unit included as soon as possible, as long as `production-status` is ‘operational’.

NOTE – This implies that, while `production-status` is ‘configured’ or ‘interrupted’, the service provider waits until `production-status` changes to ‘operational’.

4.8.3.3.2.3 If the value of the `latest-data-process-start-time` parameter is ‘undefined’ in the PROCESS-DATA operation, then the service provider shall process the included data unit as long as `production-status` is ‘operational’.

4.8.3.3.2.4 If the `latest-data-process-start-time` parameter in the PROCESS-DATA operation is not ‘undefined’, the service provider shall defer processing of the data unit if the current `production-status` is ‘interrupted’. Processing shall be deferred until `production-status` changes to ‘operational’ before `latest-data-process-start-time` expires.

4.8.3.3.2.5 If processing of a data unit has not begun at or before `latest-data-process-start-time`, a NOTIFY operation shall be issued with the Event Identifier ‘expired’ (`event-name`) and the data unit shall be discarded.

4.8.3.3.2.6 If `production-status` changes to ‘interrupted’, and there is a data unit at the head of the queue, and its `latest-data-process-start-time` is ‘undefined’, and no other data unit is being processed, the service provider shall not start processing but wait until `production-status` changes back to ‘operational’.

4.8.3.4 Positive Feedback

Status reporting shall be performed as defined in the parent procedure in 4.6.3.4.

4.8.3.5 Notification of Production Status Changes

Production status changes shall be reported as defined in the parent procedure in 4.6.3.5.1.

4.8.3.6 Notification of Production Configuration Changes

Production configuration changes shall be reported as defined in the parent procedure in 4.6.3.5.2.

4.8.3.7 Locking

4.8.3.7.1 The state 'active.locked' shall be a substate of the 'active' state.

4.8.3.7.2 The service provider shall enter the 'active.locked' substate when

a) the data unit has already expired at the time when processing shall start; or

b) `production-status` changes to 'interrupted', and a data unit is currently being processed; or

c) `production-status` changes to 'halted'.

4.8.3.7.3 In the 'active.locked' substate, the service provider shall reject PROCESS-DATA invocations from the service user with the `diagnostic` value set to 'service instance locked'; it shall not perform any processing of data units in its Input Queue.

4.8.3.7.4 The Sequence-Controlled Data Processing procedure shall return to State 2.1 ('active.processing') after having accepted a STOP and a subsequent START invocation or an EXECUTE-DIRECTIVE invocation with the `directive-identifier` equal to 'reset'.

NOTE – Both options clear the Input Queue.

4.8.3.8 Resetting

When receiving an EXECUTE-DIRECTIVE invocation with the `directive-identifier` equal to 'reset', the service provider shall

a) send an EXECUTE-DIRECTIVE positive acknowledgement to the service user;

b) clear the Input Queue;

c) wait for processing to complete and provide feedback as described in 4.6.3.4, if any data unit is currently being processed;

d) if in 'active.locked' substate, wait for `production-status` to become 'operational' and then transition to the 'active.processing' substate;

e) set the `first-data-unit-id` to the value requested in the EXECUTE-DIRECTIVE invocation; and

f) send an EXECUTE-DIRECTIVE positive return to the service user.

4.8.3.9 Stopping

The procedure shall be stopped as specified for the parent procedure in 4.6.3.6.

4.8.3.10 Terminating

The procedure shall terminate as specified for the parent procedure in 4.6.3.7. In addition, any pending acknowledgements shall be deleted.

4.8.3.11 Aborting

The procedure shall handle the need to request the Association Control procedure to abort the association as specified for the parent procedure in 4.6.3.8.

4.8.4 REQUIRED OPERATIONS

Table 4-40: Sequence-Controlled Data Processing Procedure Required Operations

Operations	Source	Extended	Refined	Procedure Blocking/Non-Blocking
START	Data Processing	Y	N	Blocking
STOP	Common	N	N	Blocking
PROCESS-DATA	Data Processing	Y	N	Non-Blocking
NOTIFY	Data Processing	Y	N	Non-Blocking
EXECUTE-DIRECTIVE	Common	Y	N	Blocking

4.8.4.1 START (Confirmed)

4.8.4.1.1 General

The Sequence-Controlled Data Processing procedure shall extend the START operation defined in 3.7 through the addition of one parameter to the invocation.

4.8.4.1.2 Operation Parameters Definitions

The Sequence-Controlled Data Processing procedure shall extend the START operation defined in 3.7 through the addition of the `first-data-unit-id` parameter.

Table 4-41: START Extension Parameters

Extension Parameters	Invocation	Return
`first-data-unit-id`	M	

4.8.4.1.3 Extension Parameter Syntax

4.8.4.1.3.1 The type `SequContrDataProcStartInvocExt`, as defined in F3.10, shall specify the syntax of the extension parameter of the START invocation.

4.8.4.1.3.2 `first-data-unit-id`

The `first-data-unit-id` parameter shall contain the value of the `data-unit-id` parameter that will be present in the first PROCESS-DATA invocation after the preceding successful START operation.

NOTE – Following a data processing failure, processing of a data unit for which processing started before the failure condition occurred may still continue. Also, reports on the progress of this processing may still be generated. The service user (when restarting the data transfer) shall therefore choose the value of the `first-data-unit-id` parameter such that all data units can still be unambiguously identified.

4.8.4.2 PROCESS-DATA (Confirmed)

4.8.4.2.1 General

NOTE – Subsection 3.10.2.2.4 stipulates that a procedure using the PROCESS-DATA operation refines or extends the `data` parameter of that operation. The Sequence-Controlled Data Processing procedure does not do that. The `data` syntax definition is left to the service using this procedure.

4.8.4.2.1.1 The Sequence-Controlled Data Processing procedure shall extend the PROCESS-DATA invocation defined in 3.10 through the addition of the `earliest-data-process-start-time` and `latest-data-process-start-time` parameters, and additional `diagnostic` values.

4.8.4.2.1.2 The Sequence-Controlled Data Processing procedure shall use the confirmed variant of the PROCESS-DATA operation and as a consequence the confirmed variant of the Standard Operation Header.

4.8.4.2.1.3 The Sequence-Controlled Data Processing procedure shall extend the PROCESS-DATA return defined in 3.10 through the addition of the `data-unit-id` parameter.

Table 4-42: PROCESS-DATA Extension Parameters

Parameters	Invocation	Return
Standard Operation Header (confirmed)	*M*	M
`earliest-data-process-start-time`	M	
`latest-data-process-start-time`	M	
`data-unit-id`		M

4.8.4.2.2 Standard Confirmed Operation Header

This operation shall use the Standard Confirmed Operation Header (see 3.3).

4.8.4.2.3 Extension Parameters Syntax

4.8.4.2.3.1 The type `SequContrDataProcProcDataInvocExt`, as defined in F3.10, shall specify the syntax of the extension parameters of the invocation of the PROCESS-DATA operation.

4.8.4.2.3.2 The type `SequContrDataProcProcDataPosReturnExt`, as defined in F3.10, shall define the syntax of the extended positive return of the PROCESS-DATA operation.

4.8.4.2.3.3 The type `SequContrDataProcProcDataNegReturnExt`, as defined in F3.10, shall define the syntax of the extended negative return of the PROCESS-DATA operation.

4.8.4.2.4 `data-unit-id`

4.8.4.2.4.1 The service provider shall set the `data-unit-id` in the PROCESS-DATA return to the value expected in the next PROCESS-DATA invocation.

4.8.4.2.4.2 If the invocation is accepted, the `data-unit-id` in the PROCESS-DATA return contains the value of the `data-unit-id` in the PROCESS-DATA invocation incremented by one.

4.8.4.2.4.3 If the invocation is rejected, the `data-unit-id` in the PROCESS-DATA return shall contain the value expected by the service provider:

a) in case of the first PROCESS-DATA operation following a successful START operation, it is the value specified in `first-data-unit-id` parameter of the START invocation;

b) otherwise, the value is one greater than the value of the `data-unit-id` of the last accepted PROCESS-DATA invocation.

4.8.4.2.4.4 If the `data-unit-id` value has reached the maximum value given by the range of the `data-unit-id` parameter and has to be incremented by one, the parameter value shall wrap around to zero.

4.8.4.2.5 `earliest-data-processing-start-time`

4.8.4.2.5.1 The `earliest-data-process-start-time` parameter shall either be ‘undefined’ or shall contain the earliest time at which processing of the data unit may begin.

4.8.4.2.5.2 If the `earliest-data-process-start-time` parameter is ‘undefined’, the service provider shall begin processing as soon as

a) `production-status` becomes or is ‘operational’; and

b) no other data unit that was transferred earlier than this data unit has not yet completed processing.

4.8.4.2.6 `latest-data-processing-start-time`

4.8.4.2.6.1 The `latest-data-process-start-time` parameter shall either be ‘undefined’ or shall contain the latest time at which processing of the data unit shall begin.

4.8.4.2.6.2 If the `latest-data-process-start-time` is equal to the `earliest-data-process-start-time`, the processing of the data shall start at this time.

4.8.4.2.7 `diagnostic` Parameter Extension Value Definitions and Syntax

4.8.4.2.7.1 If a negative PROCESS-DATA return is sent, the `diagnostic` parameter shall use one of the `diagnostic` values specified in 3.3.2.7.1 or one of the following values:

a) 'unable to process'—the service provider cannot process data because the service provider has been taken out of service for an indefinite period by management action; that is, `production-status` is 'halted';

b) 'service instance locked'—`production-status` is not 'halted', but the service provider is in the 'active.locked' substate and therefore cannot process the data;

NOTE – The service provider has reported the fault condition causing the 'active.locked' substate to the service user via a NOTIFY operation. (For possible reasons, see 4.8.3.7.2).

c) 'out of sequence'—the value of the `data-unit-id` parameter is not equal to the value expected by the service provider; the expected value is one of the following:

1) in the case of the first PROCESS-DATA operation following a successful START, the value of the `first-data-unit-id` parameter of that START invocation;

2) otherwise, the value of the `data-unit-id` parameter of the last positive PROCESS-DATA return;

d) 'inconsistent time range'—the time specified in the `earliest-data-process-start-time` parameter is later than the time specified in the `latest-data-process-start-time` parameter;

e) 'invalid time'—the production time window is invalid for one of the following reasons:

1) the period from `earliest-data-process-start-time` to `latest-data-process-start-time` does not overlap with the range of times for which service production is scheduled;

2) the period from `earliest-data-process-start-time` to `latest-data-process-start-time` does not overlap with the service instance provision period;

NOTE – The production may be scheduled to terminate earlier than the service instance provision period ends. An ESLT may do so to have the production engine available for support of a different mission as soon as possible, but permitting the service users of the previous production period some extra time to retrieve, for example, a status report reflecting the final accounting information.

f) ‘late data’—`latest-data-process-start-time` is earlier than the time the PROCESS -DATA operation is received by the service provider;

g) ‘data error’—the service provider has performed error checks as provided in the service agreement and has determined that this data is in error; for example, the data exceeds the maximum size allowed for this service instance;

h) ‘unable to store’—the service provider has not enough buffer space available to store this PROCESS-DATA invocation.

4.8.4.2.7.2 The type `SequContrDataProcProcDataDiagnosticExt`, as defined in F3.10, shall specify the syntax of the `diagnostic` parameter of the PROCESS-DATA return, extended as listed in 4.8.4.2.7.1.

4.8.4.3 NOTIFY (Unconfirmed)

4.8.4.3.1 General

The Sequence-Controlled Data Processing procedure shall inherit the notifications defined for the parent procedure in 4.6.4.2.3, but shall extend the NOTIFY operation by introducing additional values for the `data-processing-status` parameter and by adding two permissible Event Identifier values to the `event-name` parameter, as specified in the following subsections.

4.8.4.3.2 `data-processing-status` Parameter Extension

4.8.4.3.2.1 The type `DataProcNotifyInvocExt`, as defined in F3.8, is inherited from the parent procedure, but is extended for the Sequence-Controlled Data Processing procedure by the type `SequContrDataProcStatus` as defined in F3.10.

4.8.4.3.2.2 By means of this extension, the `data-processing-status` parameter can also have one of the following values:

a) ‘expired’—at the time processing of the data unit identified by the `data-unit-last-processed` parameter was attempted, the `latest-data-process-start-time` was already in the past;

b) ‘processing not started’—processing of the data unit identified by the `data-unit-last-processed` parameter was attempted, but could not be started because the `production-status` was ‘interrupted’.

4.8.4.3.3 `event-name` Extension

4.8.4.3.3.1 The value of the `event-name` shall be one of the following:

a) one of the events specified by the Data Processing procedure NOTIFY operation in 4.6.4.2.3;

b) 'expired' (`event-name`)—at the time when processing is being started, the `latest-data-process-start-time` is already in the past; the `event-name` of this event shall contain the procedure name of the procedure instance triggering the event;

NOTE – When the 'expired' event occurs, the procedure enters the 'active.locked' substate (see 4.8.3.7.2), which in turn means that the 'locked' event is to be reported. Therefore, whenever the 'expired' event occurs, two notifications will be sent by this procedure.

c) 'locked' (`event-name`)—at the time when processing is supposed to be started or while a data unit is being processed, one of the conditions specified in 4.8.3.7.2 occurred: the `event-name` of this event shall contain the procedure name of the procedure instance triggering the event.

4.8.4.3.3.2 The Published Identifiers for the values (`event-name`) 'expired' and 'locked' are specified in F3.16 as `pSCDPexpired` and `pSCPDlocked`, respectively.

4.8.4.3.4 `event-value`

4.8.4.3.4.1 For the 'data processing configuration change' (`event-name`) event, the `event-value` shall report the value of the dynamically modifiable parameter `input-queue-size` defined in 4.8.5. The first part of the path specifying the type to be used is 'NotifyInvocation': 'eventValue': 'EventValue': 'qualifiedValues': 'SequenceOfQualifiedValue': 'SEQUENCE OF QualifiedValue', where this sequence has the length 1. If the qualifier of the to-be-reported value is not 'valid', then the second part of the path is one of the following: (a) 'QualifiedValue': 'unavailable': 'NULL'; (b) 'QualifiedValue': 'undefined': 'NULL'; or (c) 'QualifiedValue': 'error': 'NULL'. If the qualifier of the to-be-reported value is 'valid', then the second part of the path is 'QualifiedValue': 'valid': 'TypeAndValue': 'Embedded': 'EMBEDDED PDV', where the OID and type of the `input-queue-size` parameter are `pDPinputQueueSize` and `PDPinputQueueSizeType`, respectively (see table 4-43). All relevant types are defined in F3.3 and F3.16.

4.8.4.3.4.2 For the 'expired' (`event-name`) event, the `event-value` shall be 'empty' unless otherwise specified by the service using this procedure, or by a derived procedure.

4.8.4.3.4.3 For the 'locked' (`event-name`) event, the `event-value` shall be 'empty' unless otherwise specified by the service using this procedure, or by a derived procedure.

4.8.4.3.4.4 Except for the 'data processing configuration change', the 'expired', and the 'locked' (`event-name`) events, the `event-value` specifications of the parent procedure apply.

4.8.4.4 EXECUTE-DIRECTIVE (Acknowledged)

4.8.4.4.1 General

The Sequence-Controlled Data Processing procedure shall extend the EXECUTE-DIRECTIVE operation defined in 3.13 through the definition of one value possible for the `directive-identifier` parameter.

4.8.4.4.2 `directive-identifier` Extension

The Sequence-Controlled Data Processing procedure shall add 'reset' to the set of Directive Identifiers possible for the `directive-identifier` parameter (see the pSCDPdirectivesId branch of the OID tree). This Directive Identifier requests that the service provider clears the Input Queue.

4.8.4.4.3 `directive-qualifier` Value

4.8.4.4.3.1 For the 'reset' (directive-identifier) directive, the `directive-qualifier` shall identify the `next-data-unit-id` parameter value; that is, the `data-unit-id` parameter value the service provider shall accept in the next PROCESS-DATA invocation.

4.8.4.4.3.2 The `directive-qualifier` parameter is defined by 'ExecuteDirectiveInvocation': 'directiveQualifier': 'localProcDirQualifier': 'DirectiveQualifierValues': 'TypeAndValue': 'Embedded': 'EMBEDDED PDV'. The OID of this parameter shall be `pSCDPresetDirectiveDirQual`, and the directive-qualifier value type shall be `PSCDPresetDirectiveDirQualType` (see F3.16).

4.8.5 CONFIGURATION PARAMETERS

4.8.5.1 The Sequence-Controlled Data Processing procedure configuration parameters that need to be configured in the context of the procedure shall be as defined in table 4-43.

NOTE – For each configuration parameter, the table identifies the engineering unit (if applicable), a cross reference to the use of the parameter in the specification of the procedure, whether the parameter may be read and/or dynamically modified, and the Parameter Identifier and type to be used in reporting the value of the parameter.

Table 4-43: Sequence-Controlled Data Processing Procedure Configuration Parameters

Parameters	Cross-Reference	Readable	Dynamically modifiable	Configuration Parameter Identifier and Type (F3.16)
`input-queue-size` (in number of PROCESS-DATA invocations the queue will store)	4.6.3.2.3	Yes	Yes	`pDPinputQueueSize` `PDPinputQueueSizeType` (inherited from the parent Data Processing procedure)

4.8.6 PROCEDURE STATE TABLE

Table 4-44: Sequence-Controlled Data Processing Procedure State Table

No.	Incoming Event	State 1 ('inactive')	State 2.1 ('active.processing')	State 2.2 ('active.locked')
1	*(StartInvocation)*	*IF "positive result" THEN (+StartReturn)* → 2.1 *ELSE (-StartReturn) ENDIF*	{procedure to association abort 'protocol error'} → 1	{procedure to association abort 'protocol error'} → 1
2	(StopInvocation)	*{procedure to association abort 'protocol error'}*	IF "positive result" THEN {initiate stop} → 1 ELSE (-StopReturn) ENDIF	IF "positive result" THEN {initiate stop} → 1 ELSE (-StopReturn) ENDIF
3	(ProcessDataInvocation)	*{procedure to association abort 'protocol error'}*	IF "positive result" THEN 'queue data unit' (+ProcessDataReturn) ELSE (-ProcessDataReturn) ENDIF	(-ProcessDataReturn)
4	'data unit ready'	*Not applicable*	IF "production status = 'interrupted'" THEN → 2.2 ELSE 'process data unit' ENDIF	Not applicable
5	'data unit processing completed'	*[ignore]*	IF "report" THEN 'notify 'data processing completed' / 'empty'' ENDIF	Not applicable
6	'expired'	Not applicable	'notify 'expired' / 'empty'' 'discard data unit' → 2.2	Not applicable
7	'production status change to 'interrupted''	*[ignore]*	'notify 'production status change' / 'interrupted'' IF "processing data unit" THEN 'discard data units in processing' → 2.2 ENDIF	[ignore]
8	'production status change to 'halted''	*[ignore]*	'discard data units in processing' 'notify 'production status change' / 'halted'' → 2.2	'notify 'production status change' / 'halted''

No.	Incoming Event	State 1 ('inactive')	State 2.1 ('active.processing')	State 2.2 ('active.locked')
9	'production status change to 'operational''	*[ignore]*	Not applicable	'notify 'production status change' / 'operational''
10	'production status change to 'configured''	*[ignore]*	Not applicable	'notify 'production status change' / 'configured''
11	'production configuration change'	*[ignore]*	'notify 'production configuration change' / 'empty''	'notify 'production configuration change' / 'empty''
12	'data processing configuration change'	*[ignore]*	'notify 'data processing configuration change' / 'procedure configuration parameter values'' (see 4.8.4.3.4.1)	'notify 'data processing configuration change' / 'procedure configuration parameter values'' (see 4.8.4.3.4.1)
13	'invalid PDU 'xxx''	*{procedure to association abort 'protocol error'}*	{procedure to association abort 'xxx'} → 1	{procedure to association abort 'xxx'} → 1
14	'terminate procedure'	*'terminate itself'*	'terminate itself'	'terminate itself'
15	(ExecuteDirectiveInvocation)	{procedure to association abort 'protocol error'}	If directive-identifier = 'reset' THEN (+ExecuteDirectiveAcknowledge) {reset} (+ExecuteDirectiveReturn) ELSE (-ExecuteDirectiveAcknowledge) with `diagnostic` 'unknown directive' ENDIF	If directive-identifier = 'reset' THEN (+ExecuteDirectiveAcknowledge) {reset} (+ExecuteDirectiveReturn) → 2.1 ELSE (-ExecuteDirectiveAcknowledge) with `diagnostic` 'unknown directive' ENDIF

Table 4-45: Procedure State Table Incoming Event Description References

Event	Reference
'expired'	4.8.3.3.2.5, 4.8.4.3.2, a data unit is available at the head of the Input Queue, but its `latest-data-process-start-time` has expired.
'data unit processing completed'	*4.6.3.4*
'data unit ready'	4.8.3.3.2, *a data unit is available at the head of the Input Queue, and `production-status` is 'operational'*; neither `earliest-data-process-start-time` nor `latest-data-process-start-time` is specified, or the current time is between `earliest-data-process-start-time` and `latest-data-process-start-time`, and `production-status` is 'operational'.
'terminate procedure'	*4.2.3*
'production status change to 'xxx''	*B2.2.4*
'production configuration change'	*3.11.2.2.3.2 b)*

Event	Reference
'data processing configuration change'	4.8.4.3.4.1
'invalid PDU 'xxx''	*3.2.3.6, 4.2.2.4. 'xxx' is one of the* `diagnostic` *values specified in 4.2.2.5*

Table 4-46: Procedure State Table Predicate Descriptions

Predicate	Evaluates to TRUE if
"positive result"	No reason for sending a negative return has been detected; that is, for the START invocation, none of the conditions in 3.7.2.3.1 or 4.6.3.1 a) applies, for the STOP invocation none of the conditions in 3.3.2.7.1 applies, and for the PROCESS-DATA invocation none of the conditions in 4.8.4.2.7.1 applies.
"report"	*The* `process-completion-report` *parameter value in the associated (ProcessDataInvocation) is 'produce report'.*
"processing data unit"	*A data unit has been read from the top of the Input Queue and processing of this data unit has started but not completed.*
"production status = 'interrupted'"	The current value of `production-status` is 'interrupted'.

Table 4-47: Procedure State Table Simple Action References

Name	References
'queue data unit'	4.8.3.2
'process data unit'	4.8.3.3
'complete data processing'	*4.6.3.6.2 b)*
'clear the Input Queue'	*4.6.3.6.2 a)*, 4.8.3.8 b)
'notify 'xxx' / 'yyy''	4.6.3.5.1.1, 4.6.3.5.2, 4.6.3.5.3, 4.6.4.2.3 a), 4.6.4.2.3 b), 4.6.4.2.3 c), 4.8.4.3.3.1 (NotifyInvocation) with `event-name` set to 'xxx' and `event-value` set to 'yyy'; in case a notification does not use an `event-value`, 'yyy' shall be set to 'empty'
'discard data unit'	4.8.3.3.2.5
'discard data units in processing'	4.6.3.3.6
'procedure to association abort 'xxx''	*4.2.2.3, 4.2.2.5, raise 'procedure to association abort 'xxx'' event with* `diagnostic` *set to 'xxx' to the Association Control procedure*
'terminate itself'	*4.6.3.7*
'set the `data-unit-id` parameter'	4.8.3.8 e), set the `data-unit-id` parameter as per the `directive-qualifier` parameter value of the received 'reset' directive
'wait' <event>	4.8.3.8 c), 4.8.3.8 d), wait until the event <event> occurs

Table 4-48: Procedure State Table Compound Action Definitions

Name	Actions Performed
{initiate stop}	*'clear the input queue'* *'complete data processing'* *(+StopReturn)*
{procedure to association abort 'xxx'}	*'clear the input queue'* *'discard data units in processing'* *'procedure to association abort 'xxx"*
{reset}	'clear the Input Queue' 'wait 'data unit processing completed'' 'wait 'production status operational'' 'set the `data-unit-id` parameter'

4.9 INFORMATION QUERY

4.9.1 VERSION NUMBER

The version number of this procedure is 1.

4.9.2 DISCUSSION

4.9.2.1 Purpose

The Information Query procedure enables the service user to request from the service provider the provision of a standard set of parameters reflecting either the configuration of the transfer service instance using this procedure or the status of other service provider parameters.

4.9.2.2 Concept

The Information Query procedure provides the CSTS user with the capability to request the current values of registered parameters. A CSTS that incorporates the Information Query procedure is hereinafter referred to as a *queriable* CSTS.

The set of queriable parameters for a CSTS is identified by the Parameter Names of individual queriable parameters and/or parameter list names, in which such a list contains a set of Parameter Labels for that service. The `list-of-parameters` includes one of the following: (a) 'empty', signifying the default list is selected; (b) a parameter list name of a list of Parameter Labels; (c) a Functional Resource Type; (d) a Functional Resource Name; (e) a procedure type; (f) a procedure name; or (g) Parameter Names or Parameter Labels of individual queriable parameters.

Each parameter list name consists of a string that represents multiple individual Parameter Labels. One special case of a parameter list may also exist for each queriable CSTS: the default list. The default list is named as any other list that may exist for the CSTS, but it is automatically applied when the `list-of-parameters` parameter is set to 'empty'. The definition of the default list (if any) is controlled by the specification of the CSTS using the Information Query procedure or by a derived procedure, or it may be delegated to Service Management.

For a given queriable CSTS, the set of Parameter Labels included in the default list (if any) and/or the set of Parameter Labels represented by named label lists and the set of individual queriable Parameter Labels or Parameter Names are defined by the specification of that CSTS or a derived Information Query procedure. A CSTS using this procedure will have to:

a) select the Functional Resource Types and associated gettable parameters from the Published Identifiers used for cross support or Agencies functionalities (see D6), and select the procedure types and the associated gettable parameters from the Published Identifiers used for the framework or services branches;

b) define the grouping of Parameter Labels into lists and allocate a list name for each list; and

c) identify which of the defined lists will be the default list, if any.

While a queriable CSTS instance is bound, the service user invokes the GET operation of the Information Query procedure to query parameter values in one of the following ways:

a) by leaving the selection of parameters unspecified, thereby selecting the default list of Parameter Labels;

b) by specifying a list name, where the list represents a predefined set of Parameter Labels;

c) by specifying a Functional Resource Type, thereby identifying all parameters belonging to the instances of that Functional Resource Type that are directly associated with the transfer service executing the Information Query procedure;

d) by specifying a Functional Resource Name (Functional Resource Type and Instance Number), thereby identifying all parameters belonging to that Functional Resource Instance;

e) by specifying a procedure type thereby selecting all configuration parameters of all active instances of the given procedure type that are associated with the service instance that executes the Information Query procedure;

f) by specifying a procedure name thereby selecting the configuration parameters of that procedure instance; or

g) by specifying Parameter Names or Parameter Labels of the individual queriable parameters.

The service provider sends to the service user the requested qualified parameter values using the GET operation return. As specified in annex C2.4, the qualified value for each parameter value indicates whether the value is valid, unavailable, undefined, or in error.

4.9.3 BEHAVIOR

4.9.3.1 Getting Parameters

The service provider shall respond to GET invocations sent by the service user (see 3.12).

4.9.3.2 Terminating

Upon receipt of a ‘terminate procedure’ event from the Association Control procedure, the procedure shall terminate by releasing all pending GET operations without answering them.

4.9.4 REQUIRED OPERATIONS

Table 4-49: Information Query Procedure Required Operations

Operations	Source	Extended	Refined	Procedure Blocking/Non-Blocking
GET	Common	N	N	Non-Blocking

4.9.5 CONFIGURATION PARAMETERS

The Information Query procedure configuration parameters that need to be configured in the context of the procedure shall be as defined in table 4-50.

NOTE – For each configuration parameter, the table provides a cross reference to the use of the parameter in the specification of the procedure, identifies whether the parameter may be read, and also identifies the Parameter Identifier and type to be used in reporting the value of the parameter. None of the configuration parameters of this procedure can be dynamically changed while the service instance executing the procedure is bound.

Table 4-50: Information Query Procedure Configuration Parameters

Parameters	Cross-Reference	Readable	Configuration Parameter Identifier and Type (F3.16)
default list of parameters	4.9.2.2	No	N/A
named label lists	3.12.1.2 b)	Yes	`pIQnamedLabelLists` `PIQnamedLabelListsType`

NOTE – The default list of parameters is shown in table 4-50 as not readable. This is because one cannot directly query the name of the default list. However, one can retrieve the full set of list names, and for each list it is stated whether this list is specified to be the default list.

4.9.6 PROCEDURE STATE TABLE

Table 4-51: Information Query Procedure State Table

No.	Incoming Event	Stateless
1	(GetInvocation)	IF “positive result” THEN (+GetReturn) ELSE (-GetReturn) ENDIF
2	‘terminate procedure’	‘terminate itself’

Table 4-52: Procedure State Table Incoming Event Description References

Event	Reference
‘terminate procedure’	4.2.3, internal event from the Association Control procedure to all other procedures of the service instance in response to a protocol abort, a PEER-ABORT, or an UNBIND

Table 4-53: Procedure State Table Predicate Descriptions

Predicate	Evaluates to TRUE if
“positive result”	No reason for sending a negative return has been detected; that is, for the GET invocation, none of the conditions in 3.12.2.4.1 applies.

Table 4-54: Procedure State Table Simple Action References

Name	References
‘terminate itself’	4.9.3.2

4.10 CYCLIC REPORT

4.10.1 VERSION NUMBER

The version number of this procedure is 1.

4.10.2 DISCUSSION

4.10.2.1 Purpose of the Procedure

The Cyclic Report procedure enables a service user to periodically receive parameter values from a service provider.

4.10.2.2 Concept

The Cyclic Report procedure extends the Unbuffered Data Delivery procedure with the following capabilities:

a) the procedure defines the structure of the data as a set of parameter values;

b) the selected parameter values are delivered periodically.

Each instance of a CSTS that incorporates the Cyclic Report procedure (hereinafter referred to as a *reporting CSTS*) cyclically (periodically) reports on the current value of a set of predefined and selected parameters.

The set of reportable parameters for a CSTS is identified by parameter list names of lists of Parameter Labels for that service and/or the Parameter Names of individual reportable parameters. The `list-of-parameters` parameter includes one of the following: (a) 'empty' (i.e., the `list-of-parameters` parameter is unspecified), (b) a parameter list name where the list contains Parameter Labels, (c) a Functional Resource Type, (d) a Functional Resource Name, (e) a procedure type, (f) a procedure name, or the Parameter Names or Parameter Labels of individual reportable parameters.

Each parameter list name consists of a string that represents multiple individual Parameter Labels. One special case of a parameter list may also exist for each reporting CSTS: the default list. The default list is named as any other list that may exist for the given CSTS, but it is automatically applied when the `list-of-parameters` parameter is set to 'empty'. The definition of the default list is controlled by the specification of the CSTS using the Cyclic Report procedure or by a derived procedure or may be delegated to Service Management.

For a given reporting CSTS, the Parameter Labels included in the default list (if any), and/or the Parameter Labels represented by named Parameter Label lists, the set of Parameter Labels or the set of Parameter Names is defined by the specification of that CSTS or by a derived procedure. A CSTS using this procedure or a derived procedure will have to:

a) select the Functional Resource Types and associated reportable parameters from the Published Identifiers used for cross support or Agencies functionalities (see D6) and select the procedure types and the associated reportable parameters from the Published Identifiers used for the framework or services branches;

b) define the grouping of Parameter Labels into lists and allocate a list name for each list; and

c) identify which of the defined lists will be the default list.

In starting the Cyclic Report procedure, the service user subscribes to the particular parameters that are to be reported by that procedure in one of the following ways:

a) by leaving the `list-of-parameters` parameter unspecified (i.e., set to 'empty'), thus selecting the parameters represented by the default list of Parameter Labels to be reported;

b) by specifying a list name, which represents a predefined set of Parameter Labels;

c) by specifying a Functional Resource Type, thereby identifying all parameters belonging to the instances of that Functional Resource Type that are directly associated with the CSTS executing the Cyclic Report procedure;

d) by specifying a Functional Resource Name, thereby identifying all parameters belonging to that Functional Resource Instance;

e) by specifying a procedure type, thus selecting the configuration parameters of all active instances of that procedure type associated with the service instance executing the Cyclic Report procedure;

f) by specifying a procedure name, thereby selecting the configuration parameters of that procedure instance; or

g) by identifying the Parameter Names or Parameter Labels of the individual parameters that are reportable by that service instance.

The service user requests periodic reporting by invoking the START operation and specifying the following criteria:

a) the delivery cycle to be used for periodic delivery;

b) the default list of parameters to be delivered, the name of a list defining the parameters to be delivered, a Functional Resource Type, a Functional Resource Name, a procedure type, a procedure instance, or the set of individual parameters to be delivered.

The service provider delivers the qualified parameters the service user has subscribed to using the TRANSFER-DATA operation until the service user invokes a STOP operation. A qualified parameter consists of the Parameter Name, value, type, and qualifier of that parameter (see annex C).

After stopping the procedure, the service user may optionally re-start the cyclic delivery of parameter values, applying the same or a different selection of parameters.

4.10.3 BEHAVIOR

4.10.3.1 Starting

NOTE – The service user invokes the START operation (a) to subscribe to the set of parameters that are to be cyclically reported either by selecting the default list (1) by leaving the selection of parameters unspecified; or by selecting one of the named Parameter Label lists; or (2) by using a Functional Resource Type, a Functional Resource Name, a procedure type, or a procedure name; or (3) by listing the parameters individually; and (b) to configure the cyclic timer by setting the value of the `delivery-cycle` parameter.

4.10.3.1.1 Upon receipt of the START invocation, the service provider shall confirm that the invocation is valid. A START invocation for the Cyclic Report procedure is valid if it meets any one of the following conditions:

a) the `list-of-parameters` parameter is set to 'empty' signifying subscription to the default list of Parameter Labels, provided a default list has been established;

b) the `list-of-parameters` parameter contains one parameter list name for a list of Parameter Labels that is contained in the set of label lists that has been established for the CSTS for use for use by the Cyclic Report procedure;

c) the `list-of-parameters` parameter contains one Functional Resource Type that is associated with the service instance that executes the Cyclic Report procedure;

d) the `list-of-parameters` parameter contains the name of one Functional Resource Instance that is associated with the service instance that executes the Cyclic Report procedure;

e) the `list-of-parameters` parameter contains one procedure type that is associated with the service instance that executes the Cyclic Report procedure;

f) the `list-of-parameters` parameter contains one procedure name of a procedure instance that is associated with the service instance that executes the Cyclic Report procedure;

g) (1) the `list-of-parameters` parameter contains one or more Functional Resource Parameter Names or Functional Resource Parameter Labels, and (2) every one of these names or labels is the name or label of a parameter of a Functional Resource that is associated with the service instance that executes the Cyclic Report procedure; or

h) (1) the `list-of-parameters` parameter contains one or more procedure configuration Parameter Names or Parameter Labels, and (2) every one of these names or labels is the name or label of a parameter of a configured procedure that is associated with the service instance that executes the Cyclic Report procedure.

4.10.3.1.2 The service provider shall send a positive START return and perform the START operation invoked by the service user except if

a) the `production-status` parameter value is 'halted', in which case the START operation shall be rejected by sending a negative START return with the `diagnostic` value set to 'out of service';

b) the `delivery-cycle` parameter is less than the limit set by the configuration parameter `PCRminimumAllowedDeliveryCycleType` (see 4.10.5), in which case the START operation shall be rejected by sending a negative START return with the `diagnostic` value set to 'out of range';

c) none of the conditions specified in 4.10.3.1.1 is met, in which case the START operation shall be rejected by sending a negative START return with the `diagnostic` value set to the applicable value, as specified in 4.10.4.1.3.1; or

d) the procedure is in State 2 ('active'), in which case the procedure shall request the Association Control procedure to abort the association with setting the `diagnostic` value to 'protocol error'.

4.10.3.1.3 The service provider shall set the cyclic timer to the value of the `delivery-cycle` parameter in the START invocation and start it.

4.10.3.2 Transferring Data

4.10.3.2.1 After a successful START operation, the service provider shall transfer the qualified parameters cyclically by means of invoking the TRANSFER-DATA operation.

NOTE – The availability of a TRANSFER-DATA invocation for delivery to the communications service constitutes the 'data available' event.

4.10.3.2.2 Qualified parameter delivery shall be governed by the cyclic timer. The cyclic timer expiration constitutes the 'cyclic timer expired' event.

4.10.3.2.3 Upon expiration of the cyclic timer, the service provider shall send the up-to-date qualified parameters selected by means of the START invocation parameter `list-of-parameters` to the service user and restart the cyclic timer.

4.10.3.2.4 The service provider shall deliver the qualified parameters (Parameter Name, the value, the type, and the qualifier of the parameters (see annex C) using the `qualified-parameters` parameter. If `list-of-parameters`

a) is left empty, then

   1) for each Functional Resource Parameter Label in the default list (see E), the service provider shall deliver the qualified parameter (see annex C) for that label for each

of the Functional Resource Instances of the given type that are directly associated with the service instance that executes the Cyclic Report procedure; and

2) for each procedure configuration Parameter Label in the default list, the service provider shall deliver the qualified parameter for that label for every configured instance of the procedure that is directly associated with the service instance that executes the Cyclic Report procedure;

b) contains the name of a list of Parameter Labels, then;

1) for each Functional Resource Parameter Label in the named list, the service provider shall deliver the qualified parameter for that label for each instance of the given Functional Resource Type that is directly associated with the service instance that executes the Cyclic Report procedure; and

2) for each procedure configuration Parameter Label in the named list, the service provider shall deliver the qualified parameter for that label for every configured instance of the procedure that is directly associated with the service instance that executes the Cyclic Report procedure;

c) contains a Functional Resource Type, the service provider shall deliver for each Parameter Label associated with that Functional Resource Type the qualified parameter for that label of each Functional Resource Instance of the given type that is directly associated with the service instance that executes the Cyclic Report procedure;

d) contains a Functional Resource Name, the service provider shall deliver the qualified parameters for all the parameters of the named Functional Resource Instance;

e) contains a procedure type, then the service provider shall deliver the qualified parameters for all configuration parameters of every configured instance of that procedure type that is associated with the service instance executing the Cyclic Report procedure;

f) contains a procedure name, then the service provider shall deliver the qualified parameters for all configuration parameters for that procedure instance;

g) contains any Parameter Labels for Functional Resource parameters, the service provider shall deliver the qualified parameter for that label for each instance of the given Functional Resource Type that is associated with the service instance that executes the Cyclic Report procedure;

h) contains any Parameter Labels for procedure configuration parameters, the service provider shall deliver for each label the qualified parameters for every configured instance of the procedure that is associated with the service instance that executes the Cyclic Report procedure; or

i) contains one or more Parameter Names, the service provider shall deliver the qualified parameter for each of the listed parameters.

4.10.3.3 Stopping

Upon reception of a STOP invocation, the service provider shall

a) stop the cyclic timer; and

b) stop transferring the qualified parameters.

4.10.3.4 Terminating

Upon receipt of a 'terminate procedure' event from the Association Control procedure, the procedure shall terminate by

a) stopping transmitting `TransferDataInvocation` PDUs; and

b) releasing the resources.

4.10.4 REQUIRED OPERATIONS

Table 4-55: Cyclic Report Procedure Required Operations

Operations	Source	Extended	Refined	Procedure Blocking/Non-Blocking
START	Unbuffered Data Delivery	Y	N	Blocking
STOP	Unbuffered Data Delivery	N	N	Blocking
TRANSFER-DATA	Unbuffered Data Delivery	N	Y	Non-Blocking

4.10.4.1 START (Confirmed)

4.10.4.1.1 General

The Cyclic Report procedure shall extend the START operation defined by the Unbuffered Data Delivery procedure (see 4.10.3.1) through the addition of two parameters to the invocation and through the addition of one value for the `diagnostic` parameter of the return.

4.10.4.1.2 Operation Parameters Definitions

NOTE – Table 4-56 shows the extension parameters of the START operation defined by this procedure.

Table 4-56: START Extension Parameters

Extension Parameters	Invocation	Return
`delivery-cycle`	M	
`list-of-parameters`	M	

4.10.4.1.2.1 Extension Parameters Syntax

The type `CyclicReportStartInvocExt`, as defined in F3.12, shall specify the syntax of the extension parameters of the START invocation.

4.10.4.1.2.2 `delivery-cycle`

The `delivery-cycle` parameter shall be present in the invocation and shall specify the requested interval between subsequent TRANSFER-DATA invocations.

4.10.4.1.2.3 `list-of-parameters`

4.10.4.1.2.3.1 The list of Parameter Names/Parameter Labels or the names of the lists that can be requested shall be defined by the service using this procedure or a procedure derived from this procedure.

4.10.4.1.2.3.2 The `list-of-parameters` parameter shall contain one of the following:

a) 'empty' value (i.e., left unspecified);

b) the name of a list;

c) a Functional Resource Type;

d) a Functional Resource Name;

e) a procedure type;

f) a procedure name;

g) a list of individual Parameter Labels; or

h) a list of individual Parameter Names.

4.10.4.1.2.3.3 The Parameter Names and parameter list names shall comply with the definition in annex E.

4.10.4.1.2.3.4 The parameters that may be contained in the `list-of-parameters` parameter shall include (but not be limited to) the parameter that reports the `production-status` of the service, as specified in B2.2.2 and B2.2.3.

4.10.4.1.3 `diagnostic` Parameter Extension Value Definitions and Syntax

4.10.4.1.3.1 If a negative START return is sent, the `diagnostic` parameter shall use one of the `diagnostic` values specified by the START operation in the Unbuffered Data Delivery procedure (see 4.5.4.1.3.1) or one of the following values:

a) 'default not defined'—the default list (`list-of-parameters` set to 'empty') is unknown to the service provider.

b) 'unknown list name'—the list name contained in the `list-of-parameters` is unknown to the service provider. The unknown list name shall be returned with the `diagnostic`.

c) 'unknown Functional Resource Type'—the Functional Resource Type contained in the `list-of-parameters` parameter is unknown to the service provider (see 4.10.4.1.2.3), or the Functional Resource Type is not associated with the service instance that executes the Cyclic Report procedure. The unknown Functional Resource Type shall be returned with the `diagnostic`.

d) 'unknown Functional Resource Name'—while the Functional Resource Type is known, the Functional Resource Name contained in the `list-of-parameters` parameter is unknown to the service provider (see 4.10.4.1.2.3), or the Functional Resource Name is not associated with the service instance that executes the Cyclic Report procedure. The unknown Functional Resource Name shall be returned with the `diagnostic`.

e) 'unknown procedure type'—the procedure type contained in the `list-of-parameters` parameter is unknown to the service provider (see 4.10.4.1.2.3). The unknown procedure type shall be returned with the `diagnostic`.

f) 'unknown procedure name'—while the procedure type is known, the procedure name contained in the `list-of-parameters` parameter is unknown to the service provider (see 4.10.4.1.2.3). The unknown procedure name shall be returned with the `diagnostic`.

g) 'unknown parameter identifier'—one or more Parameter Identifiers contained in the `list-of-parameters` parameter are unknown to the service provider (see 4.10.4.1.2.3) for one of the following reasons:

   1) the Functional Resource or procedure type specified as part of the Parameter Label is not associated with the service instance executing the given Cyclic Report procedure instance;

   2) a parameter with the given Published Identifier does not exist for the specified Functional Resource or procedure type or instance.

   The list of unknown Parameter Names or Parameter Labels shall be returned with the `diagnostic`. For each unknown Parameter Identifier that is contained in a Parameter Name in the `list-of-parameters`, the Parameter Name shall be returned. For each unknown Parameter Identifier that is contained in a Parameter Label in the `list-of-parameters`, the Parameter Label shall be returned.

h) 'out of range'—the service user requested a delivery cycle that is shorter than the limit set by the configuration parameter `PCRminimumAllowedDeliveryCycleType` (see 4.10.5).

4.10.4.1.3.2 The type `CyclicReportStartDiagnosticExt`, as defined in F3.12, shall specify the syntax of the `diagnostic` parameter of the START return, extended as listed in 4.10.4.1.3.1.

4.10.4.2 TRANSFER-DATA (Unconfirmed)

4.10.4.2.1 General

This procedure refines the TRANSFER-DATA operation defined in 4.4.3.2 defining the syntax of one parameter of the invocation.

4.10.4.2.2 Operation Parameters Definitions

NOTE – The common parameters of the TRANSFER-DATA operation are defined in 3.9.2. This procedure refines the `data` parameter of the TRANSFER-DATA operation.

4.10.4.2.2.1 `data` Parameter Syntax

The type `CyclicReportTransferDataInvocDataRef` defined in F3.12 shall specify the syntax of the `data` parameter of the TRANSFER-DATA invocation using the `qualified-parameters` definition.

NOTE – The `data` parameter syntax is structured as a sequence of qualified parameters, each of which carries the name, the value, the type, and the qualifier of the parameter (see annex C).

4.10.4.2.2.2 `qualified-parameters`

The `qualified-parameters` parameter shall carry the name, the value, the type, and the qualifier of the parameters (see annex C).

4.10.5 CONFIGURATION PARAMETERS

The Cyclic Report procedure configuration parameters that need to be configured in the context of the procedure shall be as defined in table 4-57.

NOTE – For each configuration parameter, the table identifies the engineering unit (if applicable), a cross reference to the use of the parameter in the specification of the procedure, whether the parameter may be read, and the Parameter Identifier and type to be used in reporting the value of the parameter. None of the configuration parameters of this procedure can be dynamically changed while the service instance executing the procedure is bound.

Table 4-57: Cyclic Report Procedure Configuration Parameters

Parameters	Cross-Reference	Readable	Configuration Parameter Identifier and Type (F3.16)
`named-label-lists`	4.10.3.1.1 b)	Yes	`pCRnamedLabelLists` `PCRnamedLabelListsType`
`minimum-allowed-delivery-cycle` (in milliseconds)	4.10.4.1.2.2	Yes	`pCRminimumAllowedDeliveryCycle` `PCRminimumAllowedDeliveryCycleType`
default list of parameters	4.10.3.1.1 b)	No	N/A

NOTE – The default list of parameters is shown in table 4-57 as not readable. This is because one cannot query directly the name of the default list. However, one can retrieve the full set of list names, and for each list, it is stated whether this list is specified to be the default list (see `LabelList` in F3.16).

4.10.6 PROCEDURE STATE TABLE

Table 4-58: Cyclic Report Procedure State Table

No.	Incoming Event	State 1 ('inactive')	State 2 ('active')
1	*(StartInvocation)*	*IF* *"positive result"* *THEN* *(+StartReturn)* 'start cyclic timer' *→ 2* *ELSE* *(-StartReturn)* *ENDIF*	*'procedure to association abort 'protocol error"* *→ 1*
2	*(StopInvocation)*	*'procedure to association abort 'protocol error"*	*IF* *"positive result"* *THEN* *(+StopReturn)* 'stop cyclic timer' *→ 1* *ELSE* *(-StopReturn)* *ENDIF*
3	'cyclic timer expired'	Not applicable	{periodic delivery}
4	*'data available'*	*Not applicable*	*IF* *(NOT "backpressure")* *THEN* *'send data to underlying communications service'* *ELSE* *'discard data'* *ENDIF*
5	*'invalid PDU 'xxx"*	*'procedure to association abort 'xxx"*	*'procedure to association abort 'xxx"* *→ 1*
6	*'terminate procedure'*	*'terminate itself'*	*'terminate itself'*

Table 4-59: Procedure State Table Incoming Event Description References

Event	Reference
'cyclic timer expired'	4.10.3.2.1
'invalid PDU 'xxx"	*3.2.3.6, 4.2.2.4. 'xxx*' is one of the `diagnostic` values specified in 4.2.2.5
'terminate procedure'	*4.2.3; internal event from the Association Control procedure to all other procedures of the service instance in response to a protocol abort, a PEER-ABORT, or an UNBIND*

Table 4-60: Procedure State Table Predicate Descriptions

Predicate	Evaluates to TRUE if
"positive result"	No reason for sending a negative return has been detected; that is, for the START invocation, none of the conditions in 4.10.4.1.3.1 applies, and for the STOP invocation, none of the conditions in 3.3.2.7.1 applies.

Table 4-61: Procedure State Table Simple Action References

Name	References
'set cyclic timer to `delivery-cycle` value'	4.10.3.1.3
'start cyclic timer'	4.10.3.1.3
'data available'	4.10.3.2.1
'restart cyclic timer'	4.10.3.2.3
'stop cyclic timer'	4.10.3.3
'terminate itself'	4.10.3.4

Table 4-62: Procedure State Table Compound Action Definitions

Name	Actions Performed
{periodic delivery}	IF data is available THEN trigger the 'data available' incoming event ENDIF 'restart cyclic timer'
{start cyclic timer}	'set cyclic timer to `delivery-cycle` value' 'start cyclic timer'

4.11 NOTIFICATION

4.11.1 VERSION NUMBER

The version number of this procedure is 2.

4.11.2 DISCUSSION

4.11.2.1 Purpose

The Notification procedure provides a means by which a service user is able to select from a set of pre-identified events and subsequently receive notification of the occurrence of those selected events while the Notification procedure is active.

NOTE – A CSTS can issue notifications without using the Notification procedure by using a procedure that directly includes the NOTIFY operation. For each such NOTIFY-extended procedure, the definition of the events that are to be reported via the NOTIFY operation is specified for that procedure and/or for the service that incorporates the procedure.

4.11.2.2 Concept

Each instance of a CSTS that incorporates the Notification procedure (hereinafter referred to as a *notification-enabled CSTS*) reports on each occurrence of any event belonging to a set of predefined and selected events.

The set of notifiable events for a CSTS is identified by named event lists, that is, named lists of Event Labels for that service or the Event Labels or Event Names of individual notifiable events. The `list-of-events` parameter includes one of the following: (a) 'empty', signifying that the default list shall be applied; (b) a named list of Event Labels; (c) a Functional Resource Type; (d) a Functional Resource Name; (e) a procedure type; (f) a procedure name; or (g) the Event Labels or Names of individual notifiable events.

Each named event list has a string naming that list that represents multiple individual Event Labels. One special case of a notifiable named event list exists for each notification-enabled CSTS: the default list. The default list is named as any other list that may exist for the given CSTS, but it is automatically subscribed to when the `list-of-events` parameter is left empty. The definition of the default list is controlled by the specification of the CSTS using the Notification procedure or by a derived procedure, or it may be delegated to Service Management.

For a given notification-enabled CSTS, the event list names, the Event Labels represented by the list names, the Event Labels included in the default list (if any), and the set of Event Labels or Event Names are defined by the specification of that CSTS. A CSTS using this procedure will have to:

a) select the Functional Resource Types and procedure types, as well as the associated events from the Published Identifiers (maintained by SANA);

b) define the grouping of Event Labels into lists and allocate a list name for each list; and

c) identify which of the defined lists will be the default list.

In starting the Notification procedure as part of a CSTS, the service user subscribes to the particular events that are to be reported by that procedure in one of the following ways:

a) by leaving the `list-of-events` unspecified (i.e., set to 'empty'); then for each Event Label represented by the default list of Event Labels, the reporting of the associated events is enabled;

b) by specifying the name of a list of Event Labels that represents a predefined set of Event Labels;

c) by selecting a Functional Resource Type, thereby enabling the reporting of all events belonging to the instances of that Functional Resource Type that is directly associated with the CSTS executing the Notification procedure;

d) by selecting a Functional Resource Name, thereby enabling the reporting of all events belonging to that Functional Resource Instance;

e) by selecting a procedure type, thereby enabling the reporting of all configuration change events of all procedure instances of the given procedure type associated with the service instance executing the Notification procedure;

f) by selecting a procedure name, thereby enabling the reporting of all configuration change events belonging to that procedure instance; or

g) by listing the Event Labels or Event Names of the individual events that are reportable by that service instance.

The operations defined in this procedure allow a service user to interact with a service provider to

a) request the start of reporting on the occurrence of any of the set of pre-identified notifiable events;

b) receive notification of the occurrence of the specified events; and

c) stop, and optionally later re-start, the delivery of event notifications, applying the same or a different selection of notifiable events.

4.11.3 BEHAVIOR

4.11.3.1 Starting

NOTE – The service user selects the subset of notifiable events to which he wishes to subscribe by means of the START invocation. The selected events will be notified on their occurrence.

4.11.3.1.1 The service provider shall send a positive START return and perform the START operation invoked by the service user except if

a) the `production-status` parameter value is 'halted', in which case the START operation shall be rejected by sending a negative START return with the `diagnostic` value set to 'out of service';

b) none of the conditions specified in 4.11.3.1.2 is met, in which case the START operation shall be rejected by sending a negative START return with the `diagnostic` set to the applicable value, as specified in 4.11.4.1.3.1; or

c) the procedure is in State 2 ('active'), in which case the procedure shall request the Association Control procedure to abort the association with setting the `diagnostic` value to 'protocol error'.

4.11.3.1.2 A START invocation for the Notification procedure is valid if it meets any one of the following conditions:

a) if the `list-of-events` parameter is 'empty', signifying subscription to the default list of Event Labels, provided such a default list has been established;

b) if the `list-of-events` parameter contains one event list name for a list of Event Labels that is contained in the set of label lists that has been established for the CSTS for use by the Notification procedure;

c) if the `list-of-events` parameter contains one Functional Resource Type that is associated with the service instance that executes the Notification procedure;

d) if the `list-of-events` parameter contains one name of a Functional Resource Instance that is associated with the service instance that executes the Notification procedure;

e) if the `list-of-events` parameter contains one procedure type that is associated with the service instance that executes the Notification procedure;

f) if the `list-of-events` parameter contains one procedure name of a procedure instance that is associated with the service instance that executes the Notification procedure;

g) if (1) the `list-of-events` parameter contains one or more Functional Resource Event Names or Event Labels, and (2) every one of those labels or names is the label or name of an event of a Functional Resource that is associated with the service instance that executes the Notification procedure;

h) if (1) the `list-of-events` parameter contains one or more procedure configuration change Event Labels or Event Names, and (2) every one of these labels or names is the label or name of an event of a configured procedure that is associated with the service instance that executes the Notification procedure.

4.11.3.1.3 Upon success of the START operation, the Notification procedure instance shall be subscribed to the published events identified in the START invocation.

4.11.3.2 Notifying Occurrences of Events

4.11.3.2.1 The NOTIFY invocation is valid only in the procedure state 'active' and shall be invoked only by the service provider.

4.11.3.2.2 Upon the occurrence of any of the notifiable events to which the procedure instance has been subscribed (see 4.11.3.1), the service provider shall invoke the NOTIFY operation to inform the service user of the occurrence of the event. If the `list-of-events` parameter

a) is left empty, then

   1) for each Functional Resource Event Label in the default list (see annex E), the service provider shall notify the occurrence of the event (see annex C) for that label for each Functional Resource Instance of the given type that is directly associated with the service instance that executes the Notification procedure; and

   2) for each procedure configuration change Event Label in the default list, the service provider shall notify the occurrence of the event for that label for every configured instance of the procedure that is associated with the service instance that executes the Notification procedure;

b) contains the name of a list of Event Labels, then

   1) for each Functional Resource Event Label in the named list, the service provider shall notify the occurrence of the event for that label for the Functional Resource Instances of the given type that are directly associated with the service instance that executes the Notification procedure; and

   2) for each procedure configuration change Event Label in the named list, the service provider shall notify the occurrence of the event for that label for every configured instance of the procedure that is associated with the service instance that executes the Notification procedure;

c) contains a Functional Resource Type, then the service provider shall notify the occurrence of all events for all instances of that Functional Resource Type that are directly associated with the service instance that executes the Notification procedure;

d) contains a Functional Resource Name, then the service provider shall notify the occurrence of all events of the named Functional Resource Instance;

e) contains a procedure type, then the service provider shall notify the occurrence of any configuration parameter change event for all configured instances of the procedure type that are associated with the service instance executing the Notification procedure;

f) contains a procedure name, then the service provider shall notify any occurrence of a configuration parameter change event for that procedure instance;

g) contains any Functional Resource Event Labels, then for each label the service provider shall notify the occurrence of the event for that label for each Functional Resource Instance of the given type that is directly associated with the service instance that executes the Notification procedure;

h) contains any procedure configuration change Event Labels, then for each label the service provider shall notify the occurrence of the event for that label for every configured instance of the procedure that is associated with the service instance that executes the Notification procedure;

i) contains one or more Event Names, then the service provider shall notify the occurrence of any of the listed events.

4.11.3.2.3 If an event is subscribed in terms of an individual Event Name, the corresponding `event-name` parameter (see 3.11.2.2.3.1) shall contain that Event Name.

4.11.3.2.4 The service specification or derived procedure shall specify the conditions under which it is permissible or required to use the Event Label or Event Name in an `event-name` parameter that corresponds to an event that is subscribed in terms of an Event Label.

4.11.3.3 Stopping

When receiving a valid STOP invocation, the procedure shall disable the generation of notifications of the occurrence of previously subscribed events.

4.11.3.4 Terminating

Upon receipt of a ‘terminate procedure’ event from the Association Control procedure, the procedure shall terminate by

a) discarding the pending notification; and

b) releasing the resources.

4.11.4 REQUIRED OPERATIONS

Table 4-63: Notification Procedure Required Operations

Operations	Source	Extended	Refined	Procedure Blocking/Non-Blocking
START	Common	Y	N	Blocking
STOP	Common	N	N	Blocking
NOTIFY	Common	N	N	Non-Blocking

4.11.4.1 START (Confirmed)

4.11.4.1.1 General

The Notification procedure shall extend the START operation defined in 3.7.2 through the addition of one parameter to the invocation and through the addition of values for the `diagnostic` parameter of the return.

4.11.4.1.2 Operation Parameters Definitions

NOTE – Table 4-64 shows the extension parameters of the START operation defined by this procedure.

Table 4-64: START Extension Parameters

Extension Parameters	Invocation	Return
`list-of-events`	M	

4.11.4.1.2.1 Extension Parameters Syntax

The type `NotificationStartInvocExt`, as defined in F3.13, shall specify the syntax of the extension parameter of the START invocation.

4.11.4.1.2.2 `list-of-events`

4.11.4.1.2.2.1 The list of events or the named event lists that can be requested shall be defined by the service using this procedure or by a procedure derived from this procedure or may be delegated to Service Management.

4.11.4.1.2.2.2 The `list-of-events` parameter shall contain one of the following:

a) ‘empty’, which signifies that the service provider shall notify the events defined in the default list of events;

b) one named event list that defines a predefined set of Event Labels that represent the notifiable events that are to be reported to the service user upon their occurrence;

c) one Functional Resource Type, in which case the occurrence of events related to any of the instances of the Functional Resource Type associated with the service instance in which the Notification procedure executes shall be notified to the service user;

d) one Functional Resource Name, for which all associated events shall be notified to the service user upon their occurrence;

e) one procedure type, in which case the occurrence of any change of the configuration parameters of all configured procedure instances of that procedure type associated with the service instance in which the Notification procedure executes shall be notified to the service user;

f) one procedure name, in which case the occurrence of any change of the configuration parameters of that procedure instance shall be notified to the service user; or

g) one or more individual Event Labels or Event Names of the notifiable events that are to be reported to the service user upon their occurrence.

4.11.4.1.2.2.3 A service using this procedure or a procedure derived from this procedure may define additional individual notifiable events. For each additional individual notifiable event, an Event Label and (optionally) an event value (along with its type, the associated OID and range) shall be defined.

4.11.4.1.2.2.4 The procedure shall notify the occurrence of any of those additional individual notifiable events if the `list-of-events` parameter in the START invocation contains the Event Label or Event Name of each notifiable event to be reported.

NOTE – Notifiable Event Labels that are added by services using this procedure or by procedures derived from this procedure may also be included in the list of Event Labels represented by a named event list.

4.11.4.1.2.2.5 A service using this procedure or a procedure derived from this procedure may define one or more lists of Event Labels. Each list shall have an event list name defined for it.

4.11.4.1.2.2.6 If the `list-of-events` parameter in the START invocation contains the name of a notifiable named event list, the procedure shall notify the occurrence of the events identified by that named event list.

4.11.4.1.2.2.7 A service using this procedure or a procedure derived from this procedure may define a single list of notifiable events as the default list.

NOTE – The default list is a named event list; that is, a name is assigned to the default list. However, it is also flagged to serve as the default list and can therefore be selected by setting the `list-of-events` parameter to ‘empty’. (See `ListOfParametersEvents` in F3.3.)

4.11.4.1.2.2.8 Each named event list shall contain the Event Identifiers and associated Functional Resource Types of individually published events (see annex E).

4.11.4.1.2.2.9 The Event Identifiers shall be defined using Published Identifiers.

4.11.4.1.2.2.10 The named event list name shall be defined using a string.

4.11.4.1.3 `diagnostic` Parameter Extension Value Definitions and Syntax

4.11.4.1.3.1 If a negative START return is sent, the `diagnostic` parameter shall use one of the `diagnostic` values specified in 3.7.2.3, or one of the following values:

a) 'default not defined'—the default list (`list-of-events` set to 'empty') is unknown to the service provider.

b) 'unknown list name'— the list name contained in the `list-of-parameters` is unknown to the service provider. The unknown list name shall be returned with the `diagnostic`.

c) 'unknown Functional Resource Type'—the Functional Resource Type contained in the `list-of-events` parameter is unknown to the service provider (see 4.11.4.1.2.2), or the Functional Resource Type is not associated with the service instance that executes the Notification procedure. The unknown Functional Resource Type shall be returned with the `diagnostic`.

d) 'unknown Functional Resource Name'—while the Functional Resource Type is known, the Functional Resource Name contained in the `list-of-events` parameter is unknown to the service provider (see 4.11.4.1.2.2), or the Functional Resource Name is not associated with the service instance that executes the Notification procedure. The unknown Functional Resource Name shall be returned with the `diagnostic`.

e) 'unknown procedure type'—the procedure type contained in the `list-of-events` parameter is unknown to the service provider (see 4.11.4.1.2.2). The unknown procedure type shall be returned with the `diagnostic`.

f) 'unknown procedure name'—while the procedure type is known, the procedure name contained in the `list-of-events` parameter is unknown to the service provider (see 4.11.4.1.2.2). The unknown procedure name shall be returned with the `diagnostic`.

g) 'unknown event identifier'—one or more Event Identifiers contained in the `list-of-events` parameter are unknown to the service provider (see 4.11.4.1.2.2) for one of the following reasons:

   1) the Functional Resource specified as part of the Event Name is not associated with the service instance executing the given Notification procedure instance;

2) the Functional Resource Type or procedure type specified as part of the Event Label is not associated with the service instance executing the given Notification procedure instance;

3) an event with the given Published Identifier does not exist for the specified Functional Resource Type;

4) an event with the given Published Identifier does not exist for the specified procedure type.

The list of unknown Event Names or Event Labels shall be returned with the `diagnostic`. For each unknown Event Identifier that is contained in an Event Name in the `list-of-events`, the Event Name shall be returned. For each unknown Event Identifier that is contained in an Event Label in the `list-of-events`, the Event Label shall be returned.

4.11.4.1.3.2 The type `NotificationStartDiagnosticExt`, as defined in F3.13, shall specify the syntax of the `diagnostic` parameter of the START return, extended as listed in 4.11.4.1.3.1.

4.11.5 CONFIGURATION PARAMETERS

The Notification procedure configuration parameters that need to be configured in the context of the procedure shall be as defined in table 4-65.

NOTE – For each configuration parameter, the table provides a cross reference to the use of the parameter in the specification of the procedure, identifies whether the parameter may be read, and also identifies the Parameter Identifier and type to be used in reporting the value of the parameter. None of the configuration parameters of this procedure can be dynamically changed while the service instance executing the procedure is bound.

Table 4-65: Notification Procedure Configuration Parameters

Parameters	Cross-Reference	Readable	Configuration Parameter Identifier and Type (F3.16)
named label lists	4.11.3.2.2 b)	Yes	`pNnamedLabelLists` `PNnamedLabelListsType`
default list of events	4.11.4.1.2.2.7	No	N/A

NOTE – The default list of events is shown in table 4-65 as not readable. This is because one cannot query directly the name of the default list. However, one can retrieve the full set of list names, and for each named event list, it is stated whether this list is specified to be the default list.

4.11.6 PROCEDURE STATE TABLE

Table 4-66: Notification Procedure State Table

No.	Incoming Event	State 1 ('inactive')	State 2 ('active')
1	(StartInvocation)	IF "positive result" THEN (+StartReturn) 'enable event notification for selected events' → 2 ELSE (-StartReturn) ENDIF	'procedure to association abort 'protocol error'' → 1
2	(StopInvocation)	'procedure to association abort 'protocol error''	IF "positive result" THEN (+StopReturn) ELSE (-StopReturn) ENDIF
3	'notifiable event occurred'	Not applicable	'notify 'notifiable event' / 'event value''
4	'invalid PDU 'xxx''	'procedure to association abort 'xxx''	'procedure to association abort 'xxx'' → 1
5	'terminate procedure'	'terminate itself'	'terminate itself'

Table 4-67: Procedure State Table Event Description References

Event	Reference
'notifiable event occurred'	4.11.3.2
'invalid PDU 'xxx''	3.2.3.6, 4.2.2.4. 'xxx' is one of the `diagnostic` values specified in 4.2.2.5.
'terminate procedure'	4.2.3, internal event from the Association Control procedure to all other procedures of the service instance in response to a protocol abort, a PEER-ABORT, or an UNBIND.

Table 4-68: Procedure State Table Predicate Descriptions

Predicate	Evaluates to TRUE if
"positive result"	No reason for sending a negative return has been detected; that is, for the START invocation none of the conditions in 4.11.4.1.3.1 applies, and for the STOP invocation none of the conditions in 3.3.2.7.1 applies.

Table 4-69: Procedure State Table Simple Action References

Name	References
'create a notification'	4.11.3.2.2
'notify 'xxx' / 'yyy''	4.11.3.2.2, (NotifyInvocation) with `event-name` set to 'xxx' and `event-value` set to 'yyy'. In case a notification does not use an `event-value`, 'yyy' shall be set to 'empty'
'procedure to association abort 'xxx''	4.2.2.3, 4.2.2.5, raise 'procedure to association abort 'xxx'' event with `diagnostic` set to 'xxx' to the Association Control procedure
'terminate itself'	4.11.3.4

4.12 THROW EVENT

4.12.1 VERSION NUMBER

The version number of this procedure is 2.

4.12.2 DISCUSSION

4.12.2.1 Purpose

The Throw Event procedure provides the capability for a service user to request the service provider to initiate predefined actions to be performed by the EM of the ESLT, to receive acknowledgements of successful receipt of the requests, and to receive reports of the final outcomes of the actions from the service provider.

The service provider executes the actions and checks the guard conditions applicable to the actions. The guard conditions are the conditions under which the service provider can properly and safely execute the actions. The guard conditions are defined by the service using this procedure or by the Functional Resources and their parameters the procedure shall act on.

4.12.2.2 Concept

The Throw Event procedure is intended for use in Cross Support Transfer Services that involve the modification of operating parameters of a Provider CSSS during the execution of a service package. The procedure may be incorporated into a CSTS to provide a capability to modify:

a) dynamically modifiable configuration parameters of the procedures of a given CSTS instance (refer to 3.13.1.1 b));

b) production parameters associated with the data being transferred by that transfer service (e.g., a service whose primary purpose is to deliver commands to the space element, which uses the Throw Event procedure to modify the link parameters used to deliver those commands); or

c) production parameters of the Provider CSSS independent of any data transfer via that transfer service (e.g., a service whose purpose is to control parameters of production functions that support multiple Cross Support Service instances, and for which no single service instance has the authority to modify those parameters on behalf of the service user).

NOTE – If the directed action cannot be completed by the CSTS provider itself (e.g., it must be performed by the EM of the ESLT), then the service provider must forward the directive to the appropriate EM of the ESLT for completion of the action.

No specific directives or actions that these directives would trigger are defined as part of the Throw Event procedure. Each service that uses this procedure defines the directives and the associated actions. Also, Functional Resources may specify such directives that are intended to modify the Functional Resource configuration.

NOTE – The service itself may defer the definition of the actions to a bilateral agreement between the service providing and service using organizations.

This procedure uses only one operation: the EXECUTE-DIRECTIVE. It allows the service user to transmit a directive to the service provider.

The service user may invoke the EXECUTE-DIRECTIVE operations at any time that the service instance is in the state 'bound'.

4.12.3 BEHAVIOR

4.12.3.1 Activities

NOTE – The service user can request the performance of a predefined action by the EM of an ESLT by invoking the EXECUTE-DIRECTIVE operation.

4.12.3.1.1 The service provider shall acknowledge the receipt of a valid request and deny any invalid request contained in an incoming EXECUTE-DIRECTIVE invocation.

4.12.3.1.2 If the request is valid, the service provider shall subsequently report on the success or failure of the requested action.

4.12.3.1.3 The service using this procedure or the Functional Resources this procedure is acting on shall define the guard conditions required to execute the predefined actions properly and safely.

4.12.3.2 Acknowledging Directives

4.12.3.2.1 If the invocation is a valid directive, the service provider shall:

a) send a positive acknowledgement; and

b) begin performing the requested action (see 4.12.3.3).

4.12.3.2.2 If the invocation is not a valid directive, the service provider shall:

a) send a negative acknowledgement; and

b) not perform the requested action.

4.12.3.3 Performing Directed Actions

4.12.3.3.1 The EM of the ESLT shall attempt to perform the directed action. There are no time constraints imposed by the Throw Event procedure on the completion of the action.

NOTE – Procedures derived from the Throw Event procedure may impose time constraints on the completion of the directed action.

4.12.3.3.2 A single action may involve setting of more than one parameter.

4.12.3.3.2.1 In this case, the individual parameters shall be set in the same sequence as specified in the `directive-qualifier` parameter.

4.12.3.3.2.2 Parameter-specific guard conditions shall be evaluated only after setting of the previous parameter in the sequence has completed.

NOTE – This approach ensures that each guard condition is checked only once preceding configuration changes have taken effect, and therefore the check is performed based on the up-to-date status.

4.12.3.3.3 When the action is successfully completed, the service provider shall send a positive return.

4.12.3.3.4 If the action cannot be successfully completed, the service provider shall send a negative return.

4.12.3.3.5 After having sent the return of the EXECUTE-DIRECTIVE operation, the service provider shall cease performing the operation.

4.12.3.4 Terminating

Upon receipt of a ‘terminate procedure’ event from the Association Control procedure, the procedure shall terminate by:

a) ceasing the performing of any ongoing EXECUTE-DIRECTIVE operations; and

b) releasing the resources.

NOTE – If the execution of directed actions is not yet completed at the time the Throw Event procedure is terminated, completion of these actions will not be reported by this procedure and will have to be determined by other means.

4.12.4 REQUIRED OPERATIONS

Table 4-70: Throw Event Procedure Required Operations

Operations	Source	Extended	Refined	Procedure Blocking/Non-Blocking
EXECUTE-DIRECTIVE	Common	Y	N	Non-Blocking

4.12.4.1 EXECUTE-DIRECTIVE (Acknowledged)

4.12.4.1.1 General

The Throw Event procedure shall extend the EXECUTE-DIRECTIVE operation defined in 3.13 through the addition of one value for the `diagnostic` parameter of the acknowledgement and return.

4.12.4.1.2 Invocation, Acknowledgement, Return, and Parameters

The common parameters of the EXECUTE-DIRECTIVE (acknowledged) operation are defined in 3.13.2.

4.12.4.1.3 `diagnostic` Parameter Extension Value Definition and Syntax

NOTE – Whenever a negative acknowledgement has been sent by the service provider, the action identified by the `directive-identifier` parameter has not been performed. Whenever a negative return has been sent by the service provider, the action identified by the `directive-identifier` parameter has either been performed only partially or not at all. The negative return provides the detailed information regarding the extent to which a requested action has been performed, for example, for which Functional Resource parameters the update has failed.

4.12.4.1.3.1 If an EXECUTE-DIRECTIVE negative acknowledgement is sent, the `diagnostic` parameter shall use one of the `diagnostic` values specified in 3.13.2.3.1.

4.12.4.1.3.2 If an EXECUTE-DIRECTIVE negative return is sent, the `diagnostic` parameter shall use one of the following values:

a) one of the `diagnostic` values specified in 3.13.2.3.3;

b) the value ‘guard condition evaluated to false’—a required condition was found not to be met.

4.12.4.1.3.3 The type `TeExecDirNegReturnDiagnosticExt`, as defined in F3.14, shall specify the syntax of the `diagnostic` parameter of the EXECUTE-DIRECTIVE return, extended as listed in 4.12.4.1.3.2.

4.12.5 CONFIGURATION PARAMETERS

An instance of the Throw Event procedure requires no configuration. Any information related to the capabilities accessible through the Throw Event procedure will be governed by the directives that are supported by the ESLT that hosts the CSTS provider and/or the Service Agreement that establishes the capabilities that are provided to the individual user mission.

4.12.6 PROCEDURE STATE TABLE

Table 4-71: Throw Event Procedure State Table

No.	Incoming Event	State 1 ('inactive')	State 2 ('active')
1	(ExecuteDirectiveInvocation)	IF "valid directive" THEN 'initiate action' (+ExecuteDirectiveAcknowledge) → 2 ELSE (-ExecuteDirectiveAcknowledge) ENDIF	IF "valid directive" THEN 'initiate action' (+ExecuteDirectiveAcknowledge) ELSE (-ExecuteDirectiveAcknowledge) ENDIF
2	'action completed [N]'		(+ExecuteDirectiveReturn) [N] IF 'no other operation invocation is awaiting return' THEN → 1 ENDIF
3	'action not successfully completed [N]'		(-ExecuteDirectiveReturn) [N] IF 'no other operation invocation is awaiting return' THEN → 1 ENDIF
4	'invalid PDU 'xxx''	'procedure to association abort 'xxx''	'procedure to association abort 'xxx'' → 1
5	'terminate procedure'	'terminate itself'	'terminate itself'

Table 4-72: Procedure State Table Incoming Event Description References

Event	Reference
'action completed'	4.12.3.3.3
'action not successfully completed'	4.12.3.3.4
'invalid PDU 'xxx''	3.2.3.6, 4.2.2.4; 'xxx' is one of the `diagnostic` values specified in 4.2.2.5
'terminate procedure'	4.2.3, internal event from the Association Control procedure to all other procedures of the service instance in response to a protocol abort, a PEER-ABORT, or an UNBIND

Table 4-73: Procedure State Table Predicate Definitions

Predicate	Evaluates to TRUE if
"valid directive"	None of the error conditions identified in 4.12.4.1.3.1 is true for the EXECUTE-DIRECTIVE invocation

Table 4-74: Procedure State Table Simple Action References

Name	References
'initiate action'	4.12.3.3.1
'procedure to association abort 'xxx''	4.2.2.3, 4.2.2.5, raise 'procedure to association abort 'xxx'' event with `diagnostic` set to 'xxx' to the Association Control procedure
'terminate itself'	4.12.3.4

ANNEX A

IMPLEMENTATION CONFORMANCE STATEMENT PROFORMA

(NORMATIVE)

A1 INTRODUCTION

A1.1 OVERVIEW

Although it might be possible to implement a software library on the basis of this specification, the prime intent of this document is to provide a framework for the specification of Cross Support Transfer Services. Such a service specification will provide a specification for all elements that are left abstract in this document and can thus be implemented.

As the baseline for the specification of Cross Support Transfer Services, implementation conformance in this Recommended Standard is expressed with regard to the protocol on the interface between the user and the provider of CSTSes. Therefore the ICS is a Protocol ICS (PICS).

As an aide to the creation of the PICS proforma for transfer services developed on the basis of this document, this annex provides the PICS Requirements List (RL) for the elements specified herein. Service specifications are expected to import this RL and to create a service-specific profile on this basis.

The RL support column in this annex is blank. An implementation's completed RL is called the PICS. The PICS states which capabilities and options have been implemented. The following can use the PICS:

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

b) a supplier 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;

c) a user or potential user of the implementation, as a basis for initially checking the possibility of interworking with another implementation (it should be noted that, while interworking can never be guaranteed, failure to interwork can often be predicted from incompatible PICSes);

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

A1.2 ABBREVIATIONS AND CONVENTIONS

The RL consists of information in tabular form. The status of features is indicated using the abbreviations and conventions described below.

Item Column

The item column contains a prefix identifying the element the given table is referring to and sequential numbers for items in the table.

Feature Column

The feature column contains a brief descriptive name for a feature. It implicitly means: 'Is this feature supported by the implementation?'

Status Column

The status column uses the following notations:

a) M mandatory;

b) O optional;

c) O<n> optional, but support of at least one of the group of options labeled by the same numeral <n> is required;

d) C<n> conditional as defined in corresponding expression below the table;

e) X prohibited;

f) N/A not applicable.

Support Column Symbols

The support column is to be used by the implementer to state whether a feature is supported by entering Y, N, or N/A, indicating

a) Y Yes, supported by the implementation;

b) N No, not supported by the implementation;

c) N/A Not applicable.

The support column should also be used, when appropriate, to enter values supported for a given capability.

Allowed Values Column

All PDU parameter types are specified in annex F using ASN.1. The ASN.1 data type specifications constrain among others the permissible value range, and therefore such constraints are not repeated in the Allowed Values column in the tables contained in this

annex. However, if a parameter is constrained for all instances of the given PDU to a subset of the range or set specified for that parameter in annex F, then the subset is identified in the tables that contain the PDU parameters.

Allowed Values Column Symbols

If the specification of allowed values is too large to fit in the Allowed Values cell, the Allowed Values column uses the notation 'AV<n>' as an indication that the allowed values are specified below the table.

Supported Values Column

The Supported Values column is to be used by the implementer to state whether the specified range or set of values for the parameter is supported by entering Y or SV<n>, indicating

a) Y — Yes, the range/set defined in the Recommended Specification is fully supported by the implementation;

b) SV<n> — The range/set defined in the Recommended Standard is not fully supported by the implementation. The supported subset is documented below the table.

A1.3 INSTRUCTIONS FOR COMPLETING THE RL

An implementer shows the extent of compliance to the Recommended Standard by completing the RL; that is, the state of compliance with all mandatory requirements and the options supported are shown. The resulting completed RL is called PICS. The implementer shall complete the RL by entering appropriate responses in the support or values supported column, using the notation described in A1.2. If a conditional requirement is inapplicable, N/A should be used. If a mandatory requirement is not satisfied, exception information must be supplied by entering a reference X*i*, where *i* is a unique identifier to an accompanying rationale for the noncompliance.

A2 PICS PROFORMA FOR XYZ CSTS PROTOCOL (CCSDS 9NM.I-B-K)

A2.1 GENERAL INFORMATION

The PICS for a CSTS implementation shall encompass the filled-in tables A-1 to A-4.

Table A-1: Identification of PICS

Date of Statement (DD/MM/YYYY)	
PICS Serial Number	
System Conformance Statement Cross-Reference	

Table A-2: Identification of Implementation under Test

Implementation name	
Implementation version	
Special configuration	
Other information	

Table A-3: Identification of Supplier

Supplier	
Contact Point for Queries	
Implementation Name(s) and Versions	
Other information necessary for full identification, for example, name(s) and version(s) for machines and/or operating systems, system name(s)	

Table A-4: Identification of Specification

CCSDS 9NM.I-B-K	
Have any exceptions been required? NOTE – A YES answer means that the implementation does not conform to the Recommended Standard. Non-supported mandatory capabilities are to be identified in the PICS, with an explanation of why the implementation is nonconforming.	Yes [ ] No [ ]

A2.2 REQUIREMENTS LIST

This subsection provides the RLs for the elements specified in this Recommended Standard. Depending on which procedures and associated PDUs are actually used in a CSTS specification, the relevant subset of the tables A-5 to A-24 will become part of the service-specific PICS proforma.

Table A-5: Required Procedures

Procedures				
Item	**Description**	**Reference**	**Status**	**Support**
proc-1	Association Control	4.3	M	
proc-2	Unbuffered Data Delivery	4.4	O	
proc-3	Buffered Data Delivery	4.5	O	
proc-4	Data Processing	4.6	O	
proc-5	Buffered Data Processing	4.7	O	
proc-6	Sequence-Controlled Data Processing	4.8	O	
proc-7	Information Query	4.9	O	
proc-8	Cyclic Report	4.10	O	
proc-9	Notification	4.11	O	
proc-10	Throw Event	4.12	O	

Table A-6: Required PDUs

Item	PDU	Ref.	Service Provider System		Service User System	
			Status	Support	Status	Support
pdu-1	BindInvocation	F3.5	M		M	
pdu-2	BindReturn	F3.5	M		M	
pdu-3	PeerAbortInvocation	F3.5	M		M	
pdu-4	UnbindInvocation	F3.5	M		M	
pdu-5	UnbindReturn	F3.5	M		M	
pdu-6	ExecuteDirectiveAcknowledge	F3.4	C1		C1	
pdu-7	ExecuteDirectiveInvocation	F3.4	C1		C1	
pdu-8	ExecuteDirectiveReturn	F3.4	C1		C1	
pdu-9	GetInvocation	F3.4	C2		C2	
pdu-10	GetReturn	F3.4	C2		C2	
pdu-11	NotifyInvocation	F3.4	C3		C3	
pdu-12	ProcessDataInvocation	F3.4	C4		C4	
pdu-13	ProcessDataReturn	F3.4	C5		C5	
pdu-14	StartInvocation	F3.4	C6		C6	
pdu-15	StartReturn	F3.4	C6		C6	
pdu-16	StopInvocation	F3.4	C6		C6	
pdu-17	StopReturn	F3.4	C6		C6	
pdu-18	TransferDataInvocation	F3.4	C7		C7	
pdu-19	ReturnBuffer	F3.7	C8		C8	
pdu-20	ForwardBuffer	F3.9	C9		C9	

C1 IF proc-6 OR proc-10 THEN M ELSE N/A
C2 IF proc-7 THEN M ELSE N/A
C3 IF proc-3 OR proc-4 OR proc-5 OR proc-6 OR proc-9 THEN M ELSE N/A
C4 IF proc-4 OR proc-5 OR proc-6 THEN M ELSE N/A
C5 IF proc-6 THEN M ELSE N/A
C6 IF proc-2 OR proc-3 OR proc-4 OR proc-5 OR proc-6 OR proc-8 OR proc-9 THEN M ELSE N/A
C7 IF proc-2 OR proc-3 OR proc-8 THEN M ELSE N/A
C8 If proc-3 THEN M ELSE N/A
C9 If proc-5 THEN M ELSE N/A

Table A-7: BIND Invocation Parameters

Parameters of the BindInvocation PDU						
					Values	
Item	Parameter	Ref.	Status	Support	Allowed	Supported
bindInv-1	invokerCredentials	F3.3	M			
bindInv-2	invokeId	F3.3	M			
bindInv-3	procedureName	F3.3	M		AV1	
bindInv-4	initiatorIdentifier	F3.5	M			
bindInv-5	responderPortIdentifier	F3.5	M			
bindInv-6	serviceType	F3.5	M			
bindInv-7	versionNumber	F3.5	M			
bindInv-8	serviceInstanceIdentifier	F3.5	M			
bindInv-9	bindInvocationExtension	F3.5	M		AV2	

AV1 For the BIND invocation, the procedureRole element of the parameter bindInv-3 must be set to ‘associationControl’.

AV2 If parameters need to be added to the BIND invocation PDU, the parameter bindInv-9 can be used to do so, but no such extension is defined in this Recommended Standard, and extension of the BIND invocation PDU, although permissible, is discouraged (see 4.3.3.1.13). Except if a procedure derived from the parent Association Control procedure that is using this PDU specifies such extension, the value of this parameter shall be set to ‘notUsed’.

The parameters bindInv-1, bindInv-2, and bindInv-3 are contained in the complex parameter standardInvocationHeader in the `BindInvocation` type shown in F3.5. This parameter is of the type `StandardInvocationHeader` that is specified in F3.3.

Table A-8: BIND Return Parameters

Parameters of the BindReturn PDU						
Item	Parameter	Ref.	Status	Support	Values	
					Allowed	Supported
bindRet-1	performerCredentials	F3.3	M			
bindRet-2	invokeId	F3.3	M			
bindRet-3	result	F3.3	M			
bindRet-4	positive	F3.3	C10		AV3	
bindRet-5	diagnostic	F3.3	C11		AV4	
bindRet-6	negExtension	F3.3	C11		AV5	
bindRet-7	responderIdentifier	F3.5	M			

C10 IF bindRet-3 = 'positive' THEN M ELSE X
C11 IF bindRet-3 = 'negative' THEN M ELSE X

AV3 If parameters need to be added to the positive BIND return PDU, the parameter bindRet-4 can be used to do so, but no such extension is defined in this Recommended Standard, and extension of the BIND return PDU, although permissible, is discouraged (see 4.3.3.1.13). Except if a procedure derived from the parent Association Control procedure and using this PDU specifies such extension, the value of this parameter shall be set to 'notUsed'.

AV4 For the negative BIND return, the parameter bindRet-5 is extended by the type `AssocBindDiagnosticExt`, defined in F3.5. Therefore the parameter bindRet-5 may have (a) any value defined for the `Diagnostic` type in F3.3 except 'diagnosticExtension'; or (b) any value defined by 'diagnosticExtension': 'acBindDiagExt': 'AssocBindDiagnosticExt', defined in F3.5, except 'assocBindDiagnosticExtExtension'. Additional values can be introduced by the further extension 'diagnosticExtension': 'acBindDiagExt': 'AssocBindDiagnosticExt': 'assocBindDiagnosticExtExtension', but such extension is discouraged (see 4.3.3.1.13) and not specified in this Recommended Standard.

AV5 If parameters need to be added to the negative BIND return PDU, the parameter bindRet-6 can be used to do so, but no such extension is defined in this Recommended Standard, and extension of the BIND return PDU, although permissible, is discouraged (see 4.3.3.1.13). Except if a procedure derived from the parent Association Control procedure and using this PDU specifies such extension, the value of this parameter shall be set to 'notUsed'.

All parameters of the BIND return PDU except bindRet-7 are contained the complex parameter of the type `StandardReturnHeader` that is specified in F3.3. Specific extensions are, however, specified in F3.5.

Table A-9: PEER-ABORT Invocation Parameters

Parameters of the PeerAbortInvocation PDU						
Item	Parameter	Ref.	Status	Support	Values	
					Allowed	Supported
peerAbortInv-1	diagnostic	F3.5	M		40 .. 126	

If an implementation uses the PEER-ABORT `diagnostic` value 'other reason' (126), the conditions under which that is done shall be specified (see 3.3.2.7.2).

Table A-10: UNBIND Invocation Parameters

Parameters of the UnbindInvocation PDU						
Item	Parameter	Ref.	Status	Support	Values	
					Allowed	Supported
unbindInv-1	invokerCredentials	F3.3	M			
unbindInv-2	invokeId	F3.3	M			
unbindInv-3	procedureName	F3.3	M		AV6	
unbindInv-4	unbindInvocationExtension	F3.5	M		AV7	

AV6 For the UNBIND invocation, the procedureRole element of the parameter unbindInv-3 must be set to ‘associationControl’.

AV7 If parameters need to be added to the UNBIND invocation PDU, the parameter unbindInv-4 can be used to do so, but no such extension is defined in this Recommended Standard, and extension of the UNBIND invocation PDU, although permissible, is discouraged (see 4.3.3.1.13). Except if a procedure derived from the parent Association Control procedure and using this PDU specifies such extension, the value of this parameter shall be set to 'notUsed'.

The parameters unbindInv-1, unbindInv-2, and unbindInv-3 are contained in the complex parameter standardInvocationHeader in the `UnbindInvocation` type shown in F3.5. This parameter is of the type `StandardInvocationHeader` that is specified in F3.3.

Table A-11: UNBIND Return Parameters

Parameters of the UnbindReturn PDU						
					Values	
Item	Parameter	Ref.	Status	Support	Allowed	Supported
unbindRet-1	performerCredentials	F3.3	M			
unbindRet-2	invokeId	F3.3	M			
unbindRet-3	result	F3.3	M		AV8	

AV8 The value of the parameter unbindRet-3 of the UNBIND return PDU shall always be set to the value 'positive': 'notUsed'; that is, the result is always positive and not extended.

All parameters of the UNBIND return PDU are contained in the complex parameter of the type `StandardReturnHeader` that is specified in F3.3.

Table A-12: EXECUTE-DIRECTIVE Invocation Parameters

Parameters of the ExecuteDirectiveInvocation PDU						
Item	Parameter	Ref.	Status	Support	Values	
					Allowed	Supported
execDirInv-1	invokerCredentials	F3.3	M			
execDirInv-2	invokeId	F3.3	M			
execDirInv-3	procedureName	F3.3	M		AV9	
execDirInv-4	directiveIdentifier	F3.4	M		AV10	
execDirInv-5	localProcDirQualifier	F3.4	C12		AV11	
execDirInv-6	targetprocedureName	F3.4	C13			
execDirInv-7	serviceProcDirQualifierValues	F3.4	C13		AV12	
execDirInv-8	functionalResourceInstanceNumber	F3.4	C14			
execDirInv-9	functionalResourceQualifiers	F3.4	C14		AV13	
execDirInv-10	directiveQualifierExtension	F3.4	C15		AV14	
execDirInv-11	executeDirectiveInvocationExtension	F3.4	M		AV15	

C12 IF execDirInv-4 is set to the Published Identifier of a directive that is registered under the procedure type that shall perform the EXECUTE-DIRECTIVE operation, THEN M ELSE X

C13 IF execDirInv-4 is set to the Published Identifier of a directive that is registered under a procedure type that is associated with the type of service invoking the EXECUTE-DIRECTIVE operation, but different from the procedure type that shall perform the EXECUTE-DIRECTIVE operation, THEN M ELSE X

C14 IF execDirInv-4 is set to the Published Identifier of a directive that is registered under a Functional Resource Type, THEN M ELSE X

C15 IF NOT (C12 OR C13 OR C14), THEN M ELSE X

AV9 The value of the procedureRole element of the parameter execDirInv-3 is constrained to one of the two values ‘prime procedure’ or ‘secondary procedure’.

AV10 The Published Identifier specified in the execDirInv-4 parameter must identify a registered directive.

AV11 The parameter execDirInv-5 will be one of the following: (a) 'directiveQualifier': 'localProcDirQualifier': 'DirectiveQualifierValues': 'sequenceOfParamIdsAndValues', (b) 'directiveQualifier': 'localProcDirQualifier': 'DirectiveQualifierValues': 'parameterlessValues', or (c) 'directiveQualifier': 'localProcDirQualifier': 'DirectiveQualifierValues': 'noQualifierValues'.

AV12 The parameter execDirInv-7 will be one of the following: (a) 'directiveQualifier': 'serviceProcDirQualifier': 'serviceProcDirQualifierValues': 'DirectiveQualifierValues': 'sequenceOfParamIdsAndValues', (b) 'directiveQualifier': 'serviceProcDirQualifier': 'serviceProcDirQualifierValues': 'DirectiveQualifierValues': 'parameterlessValues', or (c) 'directiveQualifier': 'serviceProcDirQualifier': 'serviceProcDirQualifierValues': 'DirectiveQualifierValues': 'noQualifierValues'.

AV13 The parameter execDirInv-9 will be one of the following: (a) 'directiveQualifier': 'functResourceDirQualifier': 'functionalResourceQualifiers': 'DirectiveQualifierValues': 'sequenceOfParamIdsAndValues', (b) 'directiveQualifier': 'functResourceDirQualifier': 'functionalResourceQualifiers': 'DirectiveQualifierValues': 'parameterlessValues', or (c) 'directiveQualifier': 'functResourceDirQualifier': 'functionalResourceQualifiers': 'DirectiveQualifierValues': 'noQualifierValues'.

AV14 The directive qualifier can be defined by the extension 'directiveQualifier': 'directiveQualifierExtension', but no such extension is specified in this Recommended Standard. Except if a procedure using this PDU defines such extension, this parameter must be absent.

AV15 If parameters need to be added to the EXECUTE-DIRECTIVE invocation PDU, the parameter execDirInv-11 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to 'notUsed'.

The parameters execDirInv-1, execDirInv-2, and execDirInv-3 are contained in the complex parameter standardInvocationHeader in the `ExecuteDirectiveInvocation` type shown in F3.4. This parameter is of the type `StandardInvocationHeader` that is specified in F3.3.

Table A-13: EXECUTE-DIRECTIVE Acknowledgement Parameters

Parameters of the ExecuteDirectiveAcknowledge PDU						
Item	Parameter	Ref.	Status	Support	Values	
					Allowed	Supported
execDirAck-1	performerCredentials	F3.3	M			
execDirAck-2	invokeId	F3.3	M			
execDirAck-3	result	F3.3	M			
execDirAck-4	positive	F3.3	C16		AV16	
execDirAck-5	diagnostic	F3.3	C17		AV17	
execDirAck-6	negExtension	F3.3	C17		AV18	

C16 IF execDirAck-3 = ‘positive’, THEN M ELSE X
C17 IF execDirAck-3 = ‘negative’, THEN M ELSE X

AV16 If parameters need to be added to the positive EXECUTE-DIRECTIVE acknowledgement PDU, the parameter execDirAck-4 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to ‘notUsed’.

AV17 For the EXECUTE-DIRECTIVE acknowledgement, the parameter execDirAck-5 is extended by the type `ExecDirNegAckDiagnosticExt` defined in F3.4. Therefore the parameter execDirAck-5 may have (a) any value defined for the `Diagnostic` type in F3.3 except ‘diagnosticExtension’; or (b) any value defined by the extension ‘diagnosticExtension’: ‘execDirAckDiagExt’: ‘ExecDirNegAckDiagnosticExt’ in F3.4 except ‘execDirNegAckDiagnosticExtExtension’. Additional values can be introduced by the further extension ‘diagnosticExtension’: ‘execDirAckDiagExt’: ‘ExecDirNegAckDiagnosticExt’: ‘execDirNegAckDiagnosticExtExtension’, but no such extension is defined in this Recommended Standard.

AV18 If parameters need to be added to the negative EXECUTE-DIRECTIVE acknowledgement PDU, the parameter execDirAck-6 can be used to do so, but no such extension is defined in this Recommended Standard. Except if a procedure using this PDU specifies such extension, the value of this parameter shall be set to ‘notUsed’.

All parameters of the EXECUTE-DIRECTIVE acknowledgement PDU are contained the complex parameter of the type `StandardReturnHeader` that is specified in F3.3. Specific extensions are, however, specified in F3.4.

Table A-14: EXECUTE-DIRECTIVE Return Parameters

Parameters of the ExecuteDirectiveReturn PDU						
Item	Parameter	Ref.	Status	Support	Values	
					Allowed	Supported
execDirRet-1	performerCredentials	F3.3	M			
execDirRet-2	invokeId	F3.3	M			
execDirRet-3	result	F3.3	M			
execDirRet-4	positive	F3.3	C18		AV19	
execDirRet-5	diagnostic	F3.3	C19		AV20	
execDirRet-6	negExtension	F3.3	C19		AV21	

C18 IF execDirRet-3 = ‘positive’ THEN M ELSE X
C19 IF execDirRet-3 = ‘negative’ THEN M ELSE X

AV19 If parameters need to be added to the positive EXECUTE-DIRECTIVE return PDU, the parameter execDirRet-4 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to ‘notUsed’.

AV20 For the negative EXECUTE-DIRECTIVE return PDU, the parameter execDirRet-5 is extended by the type `ExecDirNegReturnDiagnosticExt` defined in F3.4. Therefore the parameter execDirRet-5 may have (a) any standard value defined for the `Diagnostic` type in F3.3 except ‘diagnosticExtension’; or (b) any value defined by the extension ‘diagnosticExtension’: ‘execDirNegReturnDiagnosticExt’: ‘ExecDirNegReturnDiagnosticExt’ defined in F3.4 except ‘execDirNegReturnDiagnosticExtExtension’. Additional values can be introduced by the further extension ‘diagnosticExtension’: ‘execDirNegReturnDiagnosticExt’: ‘ExecDirNegReturnDiagnosticExt’: ‘execDirNegReturnDiagnosticExtExtension’.

If the EXECUTE-DIRECTIVE return PDU is used by the Throw Event procedure, that is, the procedureType element of the parameter execDirInv-3 of the associated EXECUTE-DIRECTIVE invocation has the value ‘throwEvent’, additional values are introduced by the further extension ‘diagnosticExtension’: ‘execDirNegReturnDiagnosticExt’: ‘ExecDirNegReturnDiagnosticExt’: ‘execDirNegReturnDiagnosticExtExtension’: ‘teExecDirDiagExt’: ‘TeExecDirNegReturnDiagnosticExt’, where the type `TeExecDirNegReturnDiagnosticExt` is specified in F3.14. Therefore the

parameter execDirRet-5 may have, in this case, (a) any standard value defined for the `Diagnostic` type in F3.3 except 'diagnosticExtension'; or (b) any value defined by the extension 'diagnosticExtension': 'execDirNegReturnDiagnosticExt': 'ExecDirNegReturnDiagnosticExt' defined in F3.4 except 'execDirNegReturnDiagnosticExtExtension'; or (c) any value defined by the extension 'diagnosticExtension': 'execDirNegReturnDiagnosticExt': 'ExecDirNegReturnDiagnosticExt': 'execDirNegReturnDiagnosticExtExtension': 'teExecDirDiagExt': 'TeExecDirNegReturnDiagnosticExt' defined in F3.14 except 'teExecDirNegReturnDiagnosticExtExtension'. Additional values can be introduced by the further extension 'diagnosticExtension' 'execDirNegReturnDiagnosticExt': 'ExecDirNegReturnDiagnosticExt': 'execDirNegReturnDiagnosticExtExtension': 'teExecDirDiagExt': 'TeExecDirNegReturnDiagnosticExt': 'teExecDirNegReturnDiagnosticExtExtension', but no such extension is defined in this Recommended Standard.

AV21 If parameters need to be added to the negative EXECUTE-DIRECTIVE return PDU, the parameter execDirRet-6 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to 'notUsed'.

All parameters of the EXECUTE-DIRECTIVE return PDU are contained the complex parameter of the type `StandardReturnHeader` that is specified in F3.3. Specific extensions are, however, specified in F3.4 and F3.14.

Table A-15: GET Invocation Parameters

Parameters of the GetInvocation PDU						
					Values	
Item	Parameter	Ref.	Status	Support	Allowed	Supported
getInv-1	invokerCredentials	F3.3	M			
getInv-2	invokeId	F3.3	M			
getInv-3	procedureName	F3.3	M		AV22	
getInv-4	listOfParameters	F3.4	M			
getInv-5	getInvocationExtension	F3.4	M		AV23	

AV22 The value of the procedureRole element of the parameter getInv-3 is constrained to one of the two values 'prime procedure' or 'secondary procedure'.

AV23 If parameters need to be added to the GET invocation PDU, the parameter getInv-5 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to 'notUsed'.

The parameters getInv-1, getInv-2, and getInv-3 are contained in the complex parameter standardInvocationHeader in the `GetInvocation` type shown in F3.4. This parameter is of the type `StandardInvocationHeader` that is specified in F3.3.

Table A-16: GET Return Parameters

Parameters of the GetReturn PDU						
Item	Parameter	Ref.	Status	Support	Values	
					Allowed	Supported
getRet-1	performerCredentials	F3.3	M			
getRet-2	invokeId	F3.3	M			
getRet-3	result	F3.3	M			
getRet-4	positive	F3.3	C20		AV24	
getRet-5	qualifiedParameters	F3.4	C20		AV25	
getRet-6	getPosReturnExtExtension	F3.4	C20		AV26	
getRet-7	diagnostic	F3.3	C21		AV27	
getRet-8	negExtension	F3.3	C21		AV28	

C20 IF getRet-3 = 'positive', THEN M ELSE X
C21 IF getRet-3 = 'negative', THEN M ELSE X

AV24 For the positive GET return, the parameter getRet-4 is set to 'getPosReturnExt': 'GetPosReturnExt' defined in F3.4.

AV25 For the positive GET return, the parameter getRet-5 is specified by 'qualifiedParameters': 'QualifiedParametersSequence'. The type `QualifiedParametersSequence` is defined in F3.4.

AV26 If further parameters need to be added to the positive GET return PDU, the parameter getRet-6 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to 'notUsed'.

AV27 For the negative GET return, the parameter getRet-7 is extended by the type `GetDiagnosticExt` defined in F3.4. Therefore the parameter getRet-7 may have (a) any standard value defined for the `Diagnostic` type in F3.3 except 'diagnosticExtension', or (b) any value defined by the extension 'diagnosticExtension': 'getDiagnosticExt': 'GetDiagnosticExt' defined in F3.4 except 'getDiagnosticExtExtension'. Additional values can be introduced by the further extension 'diagnosticExtension': 'getDiagnosticExt': 'GetDiagnosticExt': 'getDiagnosticExtExtension'.

AV28 If parameters need to be added to the negative GET return PDU, the parameter getRet-8 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to 'notUsed'.

All parameters of the GET return PDU are contained the complex parameter of the type `StandardReturnHeader` that is specified in F3.3. Specific extensions are, however, specified in F3.4.

Table A-17: PROCESS-DATA Invocation Parameters

Parameters of the ProcessDataInvocation PDU						
Item	Parameter	Ref.	Status	Support	Values	
					Allowed	Supported
procDataInv-1	invokerCredentials	F3.3	M			
procDataInv-2	invokeId	F3.3	M			
procDataInv-3	procedureName	F3.3	M		AV29	
procDataInv-4	dataUnitId	F3.4	M			
procDataInv-5	data	F3.4	M			
procDataInv-6	processDataInvocationExtension	F3.4	M		AV30	
procDataInv-7	processCompletionReport	F3.8	C22			
procDataInv-8	dataProcProcDataInvocExtExtension	F3.8	C22		AV31	
procDataInv-9	earliestDataProcessingTime	F3.10	C23			
procDataInv-10	latestDataProcessingTime	F3.10	C23			
procDataInv-11	sequContrDataProcProcDataInvocExtExtension	F3.10	C23		AV32	

C22 IF procDataInv-6 = (‘dpProcDataInvocExt’: ‘DataProcProcDataInvocExt’), THEN M ELSE X

C23 IF procDataInv-8 = (‘scdpProcDataInvocExt’: ‘SequContrDataProcProcDataInvocExt’), THEN M ELSE X

AV29 The value of the procedureRole element of the parameter procDataInv-3 is constrained to one of the two values ‘prime procedure’ or ‘secondary procedure’.

AV30 If the procedureType element of the parameter procDataInv-3 has the value ‘dataProcessing’, ‘bufferedDataProcessing’, or ‘sequenceControlledDataProcessing’, or if the procedure using this PDU is derived from one of these three procedure types, then the parameter procDataInv-6 shall be set to the value ‘dpProcDataInvocExt’: ‘DataProcProcDataInvocExt’. Otherwise, no such extension is specified in this Recommended Standard, and the value of procDataInv-6 shall be set to ‘notUsed’, except if the procedure using this PDU is mentioned above. In case of other procedure types, this parameter shall be set to ‘notUsed’, except if the procedure using this PDU specifies such extension.

AV31 If further parameters need to be added to the PROCESS-DATA invocation PDU, the parameter procDataInv-8 can be used to do so. Such extension is defined for the Sequence-Controlled Data Processing procedure. Except if such extension is defined by the procedure derived from the Data Processing procedure or the Buffered Data Processing procedure and using this PDU, the value of this parameter shall be set to ‘notUsed’. If the PDU is used by the Sequence-Controlled Data Processing procedure, that is, the procedureType element of the parameter procDataInv-3 has the value ‘sequenceControlledDataProcessing’, then the parameter procDataInv-8 shall be set to ‘scdpProcDataInvocExt’: ‘SequContrDataProcProcDataInvocExt’.

AV32 If parameters need to be added to the PROCESS-DATA invocation PDU used by the Sequence-Controlled Data Processing procedure, that is, the procedureType element of the parameter procDataInv-3 has the value ‘sequenceControlledDataProcessing’, the parameter procDataInv-11 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure derived from the parent Sequence-Controlled Data Processing procedure and using this PDU specifies such extension, this parameter shall be set to ‘notUsed’.

The parameters procDataInv-1, procDataInv-2, and procDataInv-3 are contained in the complex parameter standardInvocationHeader in the `ProcessDataInvocation` type shown in F3.4. This parameter is of the type `StandardInvocationHeader` that is specified in F3.3.

Table A-18: PROCESS-DATA Return Parameters

Parameters of the ProcessDataReturn PDU						
					Values	
Item	Parameter	Ref.	Status	Support	Allowed	Supported
procDataRet-1	performerCredentials	F3.3	M			
procDataRet-2	invokeId	F3.3	M			
procDataRet-3	result	F3.3	M			
procDataRet-4	positive	F3.3	C24		AV33	
procDataRet-5	dataUnitIdPosRtn	F3.10	C25			
procDataRet-6	sequContrDataProcProcData PosReturnExtExtension	F3.10	C25		AV34	
procDataRet-7	diagnostic	F3.3	C26		AV35	
procDataRet-8	negExtension	F3.3	C26		AV36	
procDataRet-9	dataUnitIdNegRtn	F3.10	C27			
procDataRet-10	sequContrDataProcProcData NegReturnExtExtension	F3.10	C27		AV37	

C24 IF procDataRet-3 = ‘positive’, THEN M ELSE X

C25 IF procDataRet-3 = ‘positive’ AND procDataRet-4 = ‘scdpProcDataPosReturnExt’: ‘SequContrDataProcProcDataPosReturnExt’, THEN M ELSE X

C26 IF procDataRet-3 = ‘negative’, THEN M ELSE X

C27 IF procDataRet-3 = ‘negative’ AND procDataRet-8 = ‘scdpProcDataNegReturnExt’: ‘SequContrDataProcProcDataNegReturnExt’, THEN M ELSE X

AV33 If parameters need to be added to the positive PROCESS-DATA return PDU, the parameter procDataRet-4 can be used to do so. Such extension is defined for the Sequence-Controlled Data Processing procedure. If this PDU is used by the Sequence-Controlled Data Processing procedure, that is, the procedureType element of parameter procDataInv-3 in the associated invocation has the value ‘sequenceControlledDataProcessing’, then the parameter procDataRet-4 shall be set to ‘scdpProcDataPosReturnExt’: ‘SequContrDataProcProcDataPosReturnExt’. If the procedure using this PDU does not specify such extension, the value of the parameter procDataRet-4 shall be set to ‘notUsed’.

AV34 If further parameters need to be added to the positive PROCESS-DATA return PDU, the parameter procDataRet-6 can be used to do so, but this Recommended Standard does not specify such extension. Except if a procedure derived from the parent Sequence-Controlled Data Processing procedure and using this PDU specifies such extension, the value of this parameter shall be set to 'notUsed'.

AV35 For the PROCESS-DATA return, the parameter procDataRet-7 is extended by the type `SequContrDataProcProcDataDiagnosticExt` defined in F3.10. Therefore the parameter procDataRet-7 may have (a) any standard value defined for the `Diagnostic` type in F3.3 except 'diagnosticExtension'; or (b) any value defined by the extension 'diagnosticExtension': 'scdpProcDataDiagExt': 'SequContrDataProcProcDataDiagnosticExt' defined in F3.10 except 'sequContrDataProcProcDataDiagnosticExtExtension'. Additional values can be introduced by the further extension 'diagnosticExtension': 'scdpProcDataDiagExt': 'SequContrDataProcProcDataDiagnosticExt': 'sequContrDataProcProcDataDiagnosticExtExtension', but no such extension is specified in this Recommended Standard.

AV36 If parameters need to be added to the negative PROCESS-DATA return PDU, the parameter procDataRet-8 can be used to do so. Such extension is defined for the Sequence-Controlled Data Processing procedure. If this PDU is used by the Sequence-Controlled Data Processing procedure, that is, the procedureType element of the parameter procDataInv-3 in the associated invocation has the value 'sequenceControlledDataProcessing', then the parameter procDataRet-8 shall be set to 'scdpProcDataNegReturnExt': 'SequContrDataProcProcDataNegReturnExt'. The type SequContrDataProcProcDataNegReturnExt is defined in F3.10. Except if such extension is defined by a procedure using this PDU, the value of this parameter shall be set to 'notUsed'.

AV37 If further parameters need to be added to the negative PROCESS-DATA return PDU, the parameter procDataRet-10 can be used to do so, but this Recommended Standard does not specify such extension. Except if a procedure derived from the parent Sequence-Controlled Data Processing procedure and using this PDU specifies such extension, the value of this parameter shall be set to 'notUsed'.

This PDU is valid only in case the PROCESS-DATA operation is used by the Sequence-Controlled Data Processing procedure; that is, the procedureType element of the parameter procDataInv-3 of the associated PROCESS-DATA invocation has the value 'sequenceControlledDataProcessing'. The Data Processing procedure and the Buffered Data Processing procedure use the unconfirmed variant of the PROCESS-DATA operation.

All parameters of the PROCESS-DATA return PDU are contained the complex parameter of the type `StandardReturnHeader` that is specified in F3.3. Specific extensions are, however, specified in F3.10.

Table A-19: START Invocation Parameters

Parameters of the StartInvocation PDU						
Item	Parameter	Ref.	Status	Support	Values	
					Allowed	Supported
startInv-1	invokerCredentials	F3.3	M			
startInv-2	invokeId	F3.3	M			
startInv-3	procedureName	F3.3	M		AV38	
startInv-4	startInvocationExtension	F3.4	M		AV39	
startInv-5	startGenerationTime	F3.7	C28			
startInv-6	stopGenerationTime	F3.7	C28			
startInv-7	buffDataDelStartInvocExtExtension	F3.7	C28		AV40	
startInv-8	firstDataUnitId	F3.10	C29			
startInv-9	sequContrDataProcStartInvocExtExtension	F3.10	C29		AV41	
startInv-10	deliveryCycle	F3.12	C30			
startInv-11	listOfParameters	F3.12	C30			
startInv-12	cyclicReportStartInvocExtExtension	F3.12	C30		AV42	
startInv-13	listOfEvents	F3.13	C31			
startInv-14	notificationStartInvocExtExtension	F3.13	C31		AV43	

C28 IF startInv-4 = ‘bddStartInvocExt’: ‘BuffDataDelStartInvocExt’, THEN M ELSE X
C29 IF startInv-4 = ‘scdpStartInvocExt’: ‘SequContrDataProcStartInvocExt’, THEN M ELSE X
C30 IF startInv-4 = ‘crStartInvocExt’: ‘CyclicReportStartInvocExt’, THEN M ELSE X
C31 IF startInv-4 = ‘nStartInvocExt’: ‘NotificationStartInvocExt’ THEN M ELSE X

AV38 The value of the procedureRole element of the parameter startInv-3 is constrained to one of the two values ‘prime procedure’ or ‘secondary procedure’.

AV39 If the procedureType element of the parameter startInv-3 has the value ‘bufferedDataDelivery’, then the parameter startInv-4 shall be set to the value ‘bddStartInvocExt’: ‘BuffDataDelStartInvocExt’.

If the procedureType element of the parameter startInv-3 has the value ‘sequenceControlledDataProcessing’, then the parameter startInv-4 shall be set to the value ‘scdpStartInvocExt’: ‘SequContrDataProcStartInvocExt’.

If the procedureType element of the parameter startInv-3 has the value ‘cyclicReport’, then the parameter startInv-4 shall be set to the value ‘crStartInvocExt’: ‘CyclicReportStartInvocExt’.

If the procedureType element of the parameter startInv-3 has the value ‘notification’, then the parameter startInv-4 shall be set to the value ‘nStartInvocExt’: ‘NotificationStartInvocExt’.

In all other cases in which parameters need to be added to the START invocation PDU, the parameter startInv-4 can be used to do so, but no such extension is defined in this Recommended Standard. Therefore this parameter shall be set to the value ‘notUsed’, except if the procedure using this PDU specifies such extension.

AV40 If parameters need to be added to the START invocation PDU used by a procedure derived from the parent Buffered Data Delivery procedure, the parameter startInv-7 can be used to do so, but no such extension is defined in this Recommended Standard. Except if a procedure derived from the parent Buffered Data Delivery procedure specifies such extension, this parameter shall be set to ‘notUsed’.

AV41 If parameters need to be added to the START invocation PDU used by a procedure derived from the parent Sequence-Controlled Data Processing procedure, the parameter startInv-9 can be used to do so, but no such extension is defined in this Recommended Standard. Except if a procedure derived from the parent Sequence-Controlled Data Processing procedure specifies such extension, this parameter shall be set to ‘notUsed’.

AV42 If parameters need to be added to the START invocation PDU used by a procedure derived from the parent Cyclic Report procedure, the parameter startInv-12 can be used to do so, but no such extension is defined in this Recommended Standard. Except if a procedure derived from the parent Cyclic Report procedure specifies such extension, this parameter shall be set to ‘notUsed’.

AV43 If parameters need to be added to the START invocation PDU used by a procedure derived from the parent Notification procedure, the parameter startInv-14 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure derived from the parent Notification procedure specifies such extension, this parameter shall be set to ‘notUsed’.

The parameters startInv-1, startInv-2, and startInv-3 are contained in the complex parameter standardInvocationHeader in the `StartInvocation` type shown in F3.4. This parameter is of the type `StandardInvocationHeader` that is specified in F3.3.

Table A-20: START Return Parameters

Parameters of the StartReturn PDU						
Item	Parameter	Ref.	Status	Support	Values	
					Allowed	Supported
startRet-1	performerCredentials	F3.3	M			
startRet-2	invokeId	F3.3	M			
startRet-3	result	F3.3	M			
startRet-4	positive	F3.3	C32		AV44	
startRet-5	diagnostic	F3.3	C33		AV45	
startRet-6	negExtension	F3.3	C33		AV46	

C32 IF startRet-3 = 'positive', THEN M ELSE X
C33 IF startRet-3 = 'negative', THEN M ELSE X

AV44 If parameters need to be added to the positive START return PDU, the parameter startRet-4 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to 'notUsed'.

AV45 For the START return PDU, the parameter startRet-5 is extended by the type `StartDiagnosticExt` defined in F3.4. Therefore the parameter startRet-5 may have (a) any standard value defined for the `Diagnostic` type in F3.3 except 'diagnosticExtension'; or (b) any value defined by the extension 'diagnosticExtension': 'startDiagnosticExt': 'StartDiagnosticExt' in F3.4 except 'startDiagnosticExtExtension'.

If the procedureType element of the parameter startInv-3 of the associated START invocation has the value 'bufferedDataDelivery', then parameter startRet-5 is further extended by the type `BuffDataDelStartDiagnosticExt` defined in F3.7. Therefore the parameter startRet-5 may have in this case (a) any standard value defined for the `Diagnostic` type in F3.3 except 'diagnosticExtension'; or (b) any value defined by the extension 'diagnosticExtension': 'startDiagnosticExt': 'StartDiagnosticExt' in F3.4 except 'startDiagnosticExtExtension'; or (c) any value defined by the extension 'diagnosticExtension': 'startDiagnosticExt': 'StartDiagnosticExt': 'startDiagnosticExtExtension': bddStartDiagExt': 'BuffDataDelStartDiagnosticExt' except 'buffDataDelStartDiagnosticExtExtension'.

Additional values can be introduced by the further extension ‘diagnosticExtension’: ‘startDiagnosticExt’: ‘StartDiagnosticExt’: ‘startDiagnosticExtExtension’: bddStartDiagExt’: ‘BuffDataDelStartDiagnosticExt’: ‘buffDataDelStartDiagnosticExtExtension’, but no such extension is specified in this Recommended Standard.

If the procedureType element of the parameter startInv-3 of the associated START invocation has the value ‘cyclicReport’, then parameter startRet-5 is extended by the type `CyclicReportStartDiagnosticExt` defined in F3.12. Therefore the parameter startRet-5 may have in this case (a) any standard value defined for the `Diagnostic` type in F3.3 except ‘diagnosticExtension’; (b) any value defined by the extension ‘diagnosticExtension’: ‘startDiagnosticExt’: ‘StartDiagnosticExt’ in F3.4 except ‘startDiagnosticExtExtension’; or (c) any value defined by the extension ‘diagnosticExtension’: ‘startDiagnosticExt’: ‘StartDiagnosticExt’: ‘startDiagnosticExtExtension’: crStartDiagExt’: ‘CyclicReportStartDiagnosticExt’ defined in F3.12 except ‘cyclicReportStartDiagnosticExtExtension’. Additional values can be introduced by the further extension ‘diagnosticExtension’: ‘startDiagnosticExt’: ‘StartDiagnosticExt’: ‘startDiagnosticExtExtension’: crStartDiagExt’: ‘CyclicReportStartDiagnosticExt’: ‘cyclicReportStartDiagnosticExtExtension’, but no such extension is specified in this Recommended Standard.

If the procedureType element of the parameter startInv-3 of the associated START invocation has the value ‘notification’, then the parameter startRet-5 is extended by the type `NotificationStartDiagnosticExt` defined in F3.13. Therefore the parameter startRet-5 may have in this case (a) any standard value defined for the `Diagnostic` type in F3.3 except ‘diagnosticExtension’; (b) any value defined by the extension ‘diagnosticExtension’: ‘startDiagnosticExt’: ‘StartDiagnosticExt’ in F3.4 except ‘startDiagnosticExtExtension’; or (c) any value defined by the extension ‘diagnosticExtension’: ‘startDiagnosticExt’: ‘StartDiagnosticExt’: ‘startDiagnosticExtExtension’: ‘nStartDiagExt’: ‘NotificationStartDiagnosticExt’ defined in F3.13 except ‘notificationStartDiagnosticExtExtension’. Additional values can be introduced by the further extension ‘diagnosticExtension’: ‘startDiagnosticExt’: ‘StartDiagnosticExt’: ‘startDiagnosticExtExtension’: ‘nStartDiagExt’: ‘NotificationStartDiagnosticExt’: ‘notificationStartDiagnosticExtExtension’, but no such extension is specified in this Recommended Standard.

AV46 If parameters need to be added to the negative START return PDU, the parameter startRet-6 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to ‘notUsed’.

The parameters startRet-1, startRet-2, and startRet-3 of the START return PDU are contained in the complex parameter of the type `StandardReturnHeader` that is specified in F3.3. Specific extensions are, however, specified in F3.4, F3.7, F3.12, and F3.13.

Table A-21: STOP Invocation Parameters

Parameters of the StopInvocation PDU						
Item	Parameter	Ref.	Status	Support	Values	
					Allowed	Supported
stopInv-1	invokerCredentials	F3.3	M			
stopInv-2	invokeId	F3.3	M			
stopInv-3	procedureName	F3.3	M		AV47	
stopInv-4	stopInvocationExtension	F3.4	M		AV48	

AV47 The value of the procedureRole element of the parameter stopInv-3 is constrained to one of the two values ‘prime procedure’ or ‘secondary procedure’.

AV48 If parameters need to be added to the STOP invocation PDU, the parameter stopInv-4 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to ‘notUsed’.

The parameters stopInv-1, stopInv-2, and stopInv-3 are contained in the complex parameter standardInvocationHeader in the `StopInvocation` type shown in F3.4. This parameter is of the type `StandardInvocationHeader` that is specified in F3.3.

Table A-22: STOP Return Parameters

Parameters of the StopReturn PDU						
Item	Parameter	Ref.	Status	Support	Values	
					Allowed	Supported
stopRet-1	performerCredentials	F3.3	M			
stopRet-2	invokeId	F3.3	M			
stopRet-3	result	F3.3	M			
stopRet-4	positive	F3.3	C34		AV49	
stopRet-5	diagnostic	F3.3	C35		AV50	
stopRet-6	negExtension	F3.3	C35		AV51	

C34 IF stopRet-3 = ‘positive’, THEN M ELSE X
C35 IF stopRet-3 = ‘negative’, THEN M ELSE X

AV49 If parameters need to be added to the positive STOP return PDU, the parameter stopRet-4 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to ‘notUsed’.

AV50 The parameter stopRet-5 may have any standard value defined for the `Diagnostic` type in F3.3 except ‘diagnosticExtension’. Additional values can be introduced by the extension ‘diagnosticExtension’, but no such extension is specified in this Recommended Standard.

AV51 If parameters need to be added to the negative STOP return PDU, the parameter stopRet-6 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to ‘notUsed’.

All parameters of the STOP return PDU are contained the complex parameter of the type `StandardReturnHeader` that is specified in F3.3.

Table A-23: NOTIFY Invocation Parameters

Parameters of the NotifyInvocation PDU						
Item	Parameter	Ref.	Status	Support	Values	
					Allowed	Supported
notifyInv-1	invokerCredentials	F3.3	M			
notifyInv-2	invokeId	F3.3	M			
notifyInv-3	procedureName	F3.3	M		AV52	
notifyInv-4	eventTime	F3.4	M			
notifyInv-5	eventName	F3.4	M			
notifyInv-6	eventValue	F3.4	M		AV53	
notifyInv-7	notifyInvocationExtension	F3.4	M		AV54	
notifyInv-8	dataUnitIdLastProcessed	F3.8	C36			
notifyInv-9	dataProcessingStatus	F3.8	C37		AV55	
notifyInv-10	dataProcessingStartTime	F3.8	C37			
notifyInv-11	dataUnitIdLastOk	F3.8	C36			
notifyInv-12	dataProcessingStopTime	F3.8	C38			
notifyInv-13	productionStatus	F3.8	C36		AV56	
notifyInv-14	dataProcNotifyInvocExtExtension	F3.8	C36		AV57	

C36 IF notifyInv-7 = ‘dpNotifyInvocExt’: ‘DataProcNotifyInvocExt’, THEN M ELSE X

C37 IF (notifyInv-7 = ‘dpNotifyInvocExt’: ‘DataProcNotifyInvocExt’) AND (notifyInv-8 is NOT ‘noDataProcessed’), THEN M ELSE X

C38 IF (notifyInv-7 = ‘dpNotifyInvocExt’: ‘DataProcNotifyInvocExt’) AND (notifyInv-11 is NOT ‘noSuccessfulProcessing’), THEN M ELSE X

AV52 The value of the procedureRole element of the notifyInv-3 parameter is constrained to one of the two values ‘prime procedure’ or ‘secondary procedure’.

AV53 The value of the notifyInv-6 parameter can be one of the following: a value that can be expressed using the type `SequenceOfQualifiedValue` defined in F3.3, ‘empty’, or a value that can be defined by means of the extension ‘eventValueExtension’ (see EventValue defined in F3.3), but no such extension is specified in this Recommended Standard. Except if a procedure using this PDU specifies such extension, the value of ‘eventValue’ must not be set to ‘eventValueExtension’.

AV54 If the procedureType element of the parameter notifyInv-3 has the value ‘dataProcessing’, ‘bufferedDataProcessing’, or ‘sequenceControlledDataProcessing’, or a value associated with a procedure derived from the Data Processing procedure, the Buffered Data Processing procedure, or the Sequence-Controlled Data Processing procedure, then the parameter notifyInv-7 shall be set to the value ‘dpNotifyInvocExt’: ‘DataProcNotifyInvocExt’.

In all other cases, if parameters need to be added to the NOTIFY invocation PDU, the parameter notifyInv-7 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, this parameter shall be set to ‘notUsed’.

AV55 If the procedureType element of the parameter notifyInv-3 has the value ‘sequenceControlledFrameDataProcessing’, the parameter notifyInv-9 can take on (a) any value specified for the dataProcessingStatus element of the `DataProcNotifyInvocExt` type defined in F3.8 except ‘dataProcessingStatusExtension’, or (b) any value specified by the `SequContrDataProcStatus` type defined in F3.8 except ‘sequContrDataProcStatusExtension’.

If the procedureType element of the parameter notifyInv-3 has the value ‘bufferedFrameDataProcessing’, the parameter notifyInv-9 can take on any value specified for the dataProcessingStatus element of the `DataProcNotifyInvocExt` type defined in F3.8 except ‘dataProcessingStatusExtension’.

AV56 If the parameter notifyInv-5 has the value ‘productionStatusChange’, the parameter notifyInv-13 shall be set to ‘productionStatusChange’: ‘NULL’; that is, it is absent. For all other values of the parameter notifyInv-5, it shall be set to ‘anyOtherEvent’: ‘ProductionStatus’.

AV57 If parameters need to be added to the NOTIFY invocation PDU used by a procedure derived from the Data Processing procedure, the Buffered Data Processing procedure, or the Sequence-Controlled Data Processing procedure, then the parameter notifyInv-14 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to ‘notUsed’.

The parameters notifyInv-1, notifyInv-2, and notifyInv-3 are contained in the complex parameter standardInvocationHeader in the `NotifyInvocation` type shown in F3.4. This parameter is of the type `StandardInvocationHeader` that is specified in F3.3.

Table A-24: TRANSFER-DATA Invocation Parameters

Parameters of the TransferDataInvocation PDU						
					Values	
Item	Parameter	Ref.	Status	Support	Allowed	Supported
transferDataInv-1	invokerCredentials	F3.3	M			
transferDataInv-2	invokeId	F3.3	M			
transferDataInv-3	procedureName	F3.3	M		AV58	
transferDataInv-4	generationTime	F3.4	M			
transferDataInv-5	sequenceCounter	F3.4	M			
transferDataInv-6	data	F3.4	M		AV59	
transferDataInv-7	qualifiedParameters	F3.12	C39			
transferDataInv-8	cyclicReportTransferDataInvocDataRefExtension	F3.12	C39		AV60	
transferDataInv-9	transferDataInvocationExtension	F3.4	M		AV61	

C39 IF transferDataInv-6 = ‘extendedData’: ‘crTransferDataInvocDataRef’: ‘CyclicReportTransferDataInvocDataRef’, THEN M ELSE X

AV58 The value of the procedureRole element of the parameter transferDataInv-3 is constrained to one of the two values ‘prime procedure’ or ‘secondary procedure’.

AV59 If the procedureType element of the parameter transferDataInv-3 is ‘cyclicReport’, then the parameter transferDataInv-6 shall be set to the value ‘extendedData’: ‘crTransferDataInvocDataRef’: ‘CyclicReportTransferDataInvocDataRef’. The type `CyclicReportTransferDataInvocDataRef` is defined in F3.12.

In all other cases a type different from `OCTET STRING` is needed for the data to be transferred by means of the TRANSFER-DATA invocation PDU, the parameter transferDataInv-6 can be used to specify such a type by means of the extension ‘extendedData’, but except for the Cyclic Report procedure, no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, this parameter shall be set to ‘opaqueString’: ‘OCTET STRING’.

AV60 If parameters need to be added to the TRANSFER-DATA invocation PDU used by a procedure derived from the parent Cyclic Report procedure, then the parameter transferDataInv-8 can be used to do so, but no such extension is defined in this Recommended Standard. Except if a procedure derived from the parent Cyclic

Report procedure and using this PDU specifies such extension, the value of this parameter shall be set to ‘notUsed’.

AV61 If parameters need to be added to the TRANSFER-DATA invocation PDU, the parameter transferDataInv-9 can be used to do so, but no such extension is defined in this Recommended Standard. Except if the procedure using this PDU specifies such extension, the value of this parameter shall be set to ‘notUsed’.

The parameters transferDataInv-1, transferDataInv-2, and transferDataInv-3 are contained in the complex parameter standardInvocationHeader in the `TransferDataInvocation` type shown in F3.4. This parameter is of the type `StandardInvocationHeader` that is specified in F3.3.

ANNEX B

PRODUCTION STATUS AND CONFIGURATION

(NORMATIVE)

B1 PRODUCTION STATUS OVERVIEW

This annex defines the states and transitions of the `production-status` parameter and presents in tabular form the effect of the `production-status` parameter value on the processing of data and operations. The transitions that may occur are illustrated in figure B-1.

This annex also defines how a CSTS determines the configuration of the underlying production and notifies changes in that underlying production configuration.

B2 PRODUCTION STATUS SPECIFICATIONS

B2.1 OVERVIEW

Figure B-1 Illustrates the four values of the production status of a CSTS and the transitions that may occur among them.

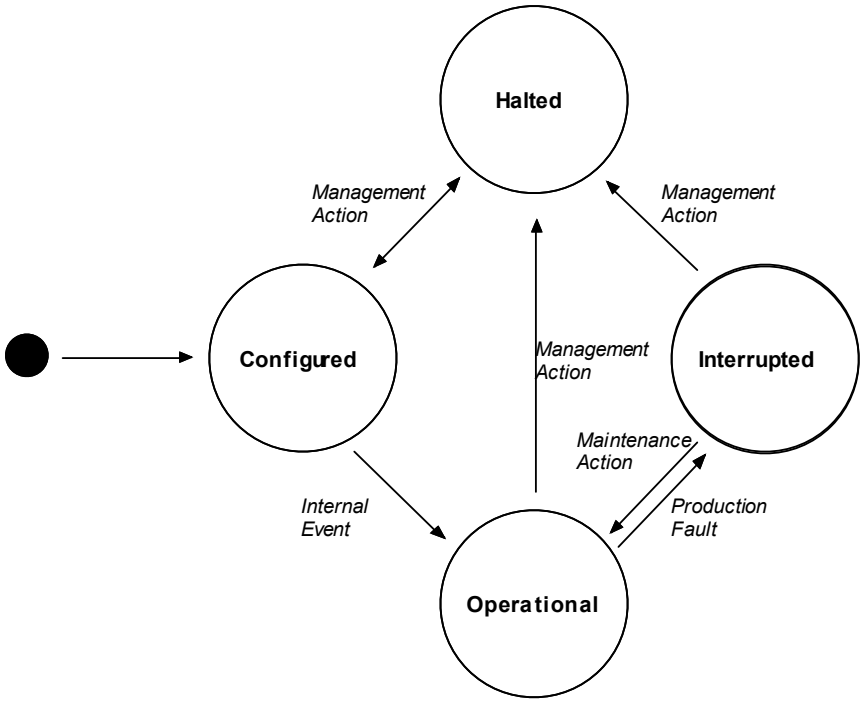

Figure B-1: Production Status Diagram

If absolutely necessary for a service type, refinement of the `production-status` parameter and the introduction of substates is permissible (see 2.2.2.2). In that case the CSTS specification needs to provide the equivalent information of this annex for the service-type-specific `production-status` parameter. Furthermore, for all procedures that in their parent version deal with the standard `production-status` parameter, derived procedures need to be specified capable of handling the service-type-specific non-standard production-status parameter.

The type of the standard `production-status` parameter is

```
ProdStat            ::= ENUMERATED
{
    configured              (0)
,   operational             (1)
,   interrupted             (2)
,   halted                  (3)
}
```

B2.2 REQUIREMENTS

B2.2.1 The `production-status` values shall be set in accordance with the high-level semantic definition listed in table B-1.

B2.2.2 The values used to indicate the `production-status` values shall be those of the type `ProdStat` as specified in the SANA registry located at https://sanaregistry.org/r/functional_resources.

B2.2.3 The Published Identifier for the `production-status` parameter for a CSTS instance is the OID of the `xxxProdstat` parameter of the Functional Resource representing that CSTS instance where 'xxx' is the abbreviated name of the CSTS type.

B2.2.4 A CSTS instance shall change the `production-status` value in accordance with the conditions specified in table B-2. Such change of the `production-status` value shall constitute the procedure state table incoming events 'production status change', and, depending on the actual transition, 'production status change to 'operational'', 'production status change to 'interrupted'', 'production status change to 'halted'', and 'production status change to 'configured'', respectively.

Table B-1: Production Status Semantic

Production Status Value	Semantic Definition
'configured'	The configuration of the service production process has been completed.
'operational'	The service production process is ready to process data.
'interrupted'	The service production process has stopped because of an error condition that may be temporary.
'halted'	The service production process has been stopped by management action.

B2.2.5 The event reporting the change of the production status shall be named using the Functional Resource Instance representing the CSTS instance reporting the 'production status change' event (see 3.11.2.2.3).

NOTES

1 The production status is always a property of the CSTS instance itself and not of the associated production process. The CSTS may specify how the aggregate production status shall be derived, for instance, from the resource status and resource configuration change event reported by the Functional Resources representing the production process.

2 The production status always refers to the current status of the CSTS. Therefore in case a CSTS is used to retrieve data that have been collected using a production process that completed before the CSTS was instantiated, any issues related to the production process that collected the data now being retrieved by the CSTS are not visible from the CSTS production status. However, the CSTS may specify means different from the production status for reporting events that affected the data collection.

Table B-2: Production Status Transitions

Start Status	End Status	Cause of Status Change
'configured'	'operational'	Management action to make the production status 'operational'; this typically includes ensuring the availability of the service provider services.
'operational'	'interrupted'	Occurrence of a production fault detected by the service provider.
'interrupted'	'operational'	Maintenance action typically is required to correct the production fault (e.g., re-configuration to use a backup service production). The production status changes to 'operational' when the service provider detects that the fault is corrected.
[any]	'halted'	Direct management action is required, such as an operator directive causing the service provider to halt production.
'halted'	'configured'	Direct management action is required, such as an operator directive restoring the required configuration and declaring the production status to be 'configured'.

B2.2.6 The transition to the production status 'operational' shall only be notified by a CSTS instance if a different production status value has been notified before and the transition to production status 'operational' has not yet been notified since the most recent successful BIND operation.

B2.2.7 If not stated differently in the specification of the CSTS, the transition to the production status 'interrupted' shall only be notified by a CSTS instance if it is presently affected by the possibly transient production fault.

NOTE – When the production status is 'halted', the BIND operation shall be rejected (see 3.4.2.3.1). The effects of the production status parameter value on the operations other than the BIND are addressed by the procedures using the operation whenever needed.

B3 PRODUCTION CONFIGURATION

B3.1 OVERVIEW

Each CSTS specification defines how the Functional Resource Instances directly associated with the service production of an instance of that CSTS type are determined, and which set of parameters of those Functional Resource Instances form the production configuration. The change of value of any parameter in the production configuration constitutes a 'production configuration change' event for that CSTS instance.

B3.2 REQUIREMENT

The event reporting the change of the production configuration shall be named using the Functional Resource Instance representing the CSTS instance reporting the ‘production configuration change’ event (see 3.11.2.2.3).

ANNEX C

QUALIFIED PARAMETERS

(NORMATIVE)

C1 OVERVIEW

This annex defines the requirements applicable to named parameters used by the GET operation (see 3.12), the Information Query procedure (see 4.9), and the Cyclic Report procedure (see 4.10).

C2 REQUIREMENTS

C2.1 The `qualified-parameters` parameter shall contain for each parameter identified in the `list-of-parameters` the following information:

a) the Parameter Name defined with its Functional Resource Name (see 1.6.1.7.23) or with its procedure name (see the type `FRorProcedureName` specified in F3.3) and its Parameter Identifier (see 1.6.1.7.31);

b) the parameter type (see C2.3);

c) the parameter value(s): a parameter may be expressed as a single value or as multiple values;

d) the parameter qualifier reporting the validity of the parameter value (see C2.4).

C2.2 The Functional Resource Name (see 1.6.1.7.23) shall be defined by its Functional Resource Type (see 1.6.1.7.24) and its Functional Resource Instance Number (see 1.6.1.7.22). The procedure name shall be defined by the procedure type and the procedure role. For secondary procedures, the procedure role shall specify also the procedure instance number.

C2.3 Any parameter shall be of the EMBEDDED PDV type (see F2.2). The syntax and therefore the type of a parameter value shall be identified by the Published Identifier assigned to that parameter.

NOTE – For instance, the `data-transfer-mode` parameter of the Buffered Data Processing procedure has the Published Identifier `pBDPdataTransferMode`, and the syntax is specified by the type `PBDPdataTransferModeType` (see table 4-33 and F3.16).

C2.4 The qualifier of a parameter value shall be one of the following (see `QualifiedValue` in F3.3):

a) ‘valid’—the value is valid;

b) ‘unavailable’—the service provider cannot provide the value;

c) ‘undefined’—in the current service provider context, the value is undefined;

d) ‘error’—the processing of the service provider resulted in an error.

C2.5 If the value qualifier of a given parameter is ‘unavailable’, ‘undefined’, or ‘error’, the value of that parameter shall not be returned.

ANNEX D

OBJECT IDENTIFIERS DEFINITION

(NORMATIVE)

D1 OVERVIEW

This annex defines

a) the OID tree structure that is used (a) by the procedures defined in this Recommended Standard and (b) for registering Functional Resource Types and their associated parameters, events, and directives;

   NOTE – It is not intended to constrain how the data types of, for example, parameters, are implemented or encoded. These definitions are suitable for inclusion in any type of ASN.1 based protocol that implements Cross Support Services.

b) the procedures to be applied for the management of the OIDs defined in this annex and for the creation of new OIDs that need to be defined when new services are specified; and

c) the top-level OIDs themselves.

   NOTE – The complete set of OIDs defined by this Recommended Standard is specified in F3.1 and F3.16. An informative description of the registries relevant to this Recommended Standard and their management is provided in H2. An informative presentation of the overall OID tree structure is provided in annex K.

D2 OBJECT IDENTIFIERS REGISTRATION

D2.1 This document requests the creation of, or uses, registries that will be managed by the Space Assigned Numbers Authority, applying the registration rules outlined in H2. New assignments in these registries will be shown at the SANA registry Web site: http://sanaregistry.org. Therefore, the reader shall look at the SANA Web site for all the assignments contained in these registries.

D2.2 Already registered OIDs shall be affected neither by any extension of registries as requested by this Recommended Standard nor by the definition of new CSTSes.

D2.3 Requests to add assignments to the subtrees of the OID Registry that are owned by the Cross Support Service (CSS) Area beyond those initiated by the CSS Area shall be submitted to SANA and come from a Member Agency, an Observer Agency, a CCSDS

Associate, or an industry partner supported by a Member Agency. The request shall be related to a cross support activity. After evaluation of the request and approval by the CSS Area Director (AD), CSS Deputy Area Directory (DAD), or a person duly authorized by the AD or DAD, a new OID will be allocated and added to the relevant subtree owned by the CSS Area and added in this way to the existing OID Registry.

D3 TREE STRUCTURE

NOTE – The overall tree structure can be found in annex K.

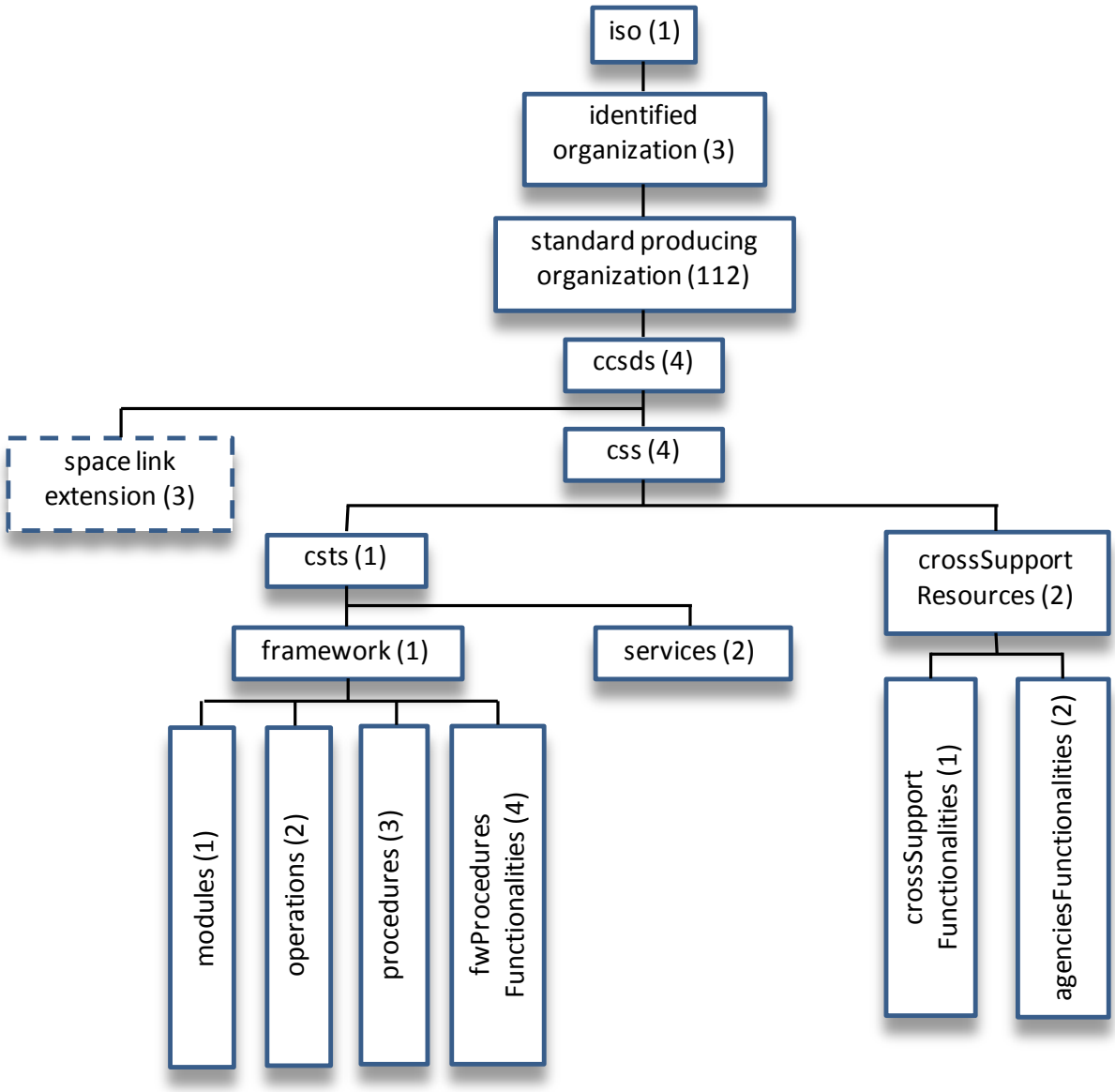

Figure D-1: CSTS and Cross Support Resources Root Object Identifier Tree

D3.1 Under the css node, one OID branch with the top-level node csts shall be allocated for all CSTS information objects.

D3.2 Under the css node, one OID branch with the top-level node crossSupportResources shall be allocated for all Cross Support Resources information objects (see D6).

D3.3 The csts OID branch shall be subdivided into the following subbranches:

a) 'framework', which lists all OIDs that are reserved for the CSTS Specification Framework definition;

b) 'services', which lists all OIDs that are relevant for CSTSes, each new service having its own subbranch.

D3.4 The 'framework' branch (see D4) shall be divided as follows:

a) The 'modules' branch lists the OIDs of the ASN.1 modules defined in this Recommended Standard.

b) The 'operations' branch lists the OIDs applicable to each operation defined in this Recommended Standard and specifies for each operation the OIDs of the extended types defined in this Recommended Standard.

c) The 'procedures' branch lists the OIDs to be used for each procedure. Each CSTS Specification Framework procedure may be further detailed with derived procedures and with parameter extensions of the procedure.

d) The 'fwProceduresFunctionalities' branch lists for each procedure defined in this Recommended Standard in a procedure-specific subbranch the Parameter Identifiers, the Event Identifiers, and the Directive Identifiers. Under each of the procedure-specific subbranches of the 'fwProceduresFunctionalities' branch, there shall be three separate subbranches: one for Parameters, one for Events, and one for Directives, as defined in F3.16.

NOTE – The Parameter Identifiers, Event Identifiers, and Directive Identifiers are used to construct the Parameter, Event, and Directive Names that are transferred via various CSTS operations. Annex E specifies the rules for the construction of names using these identifiers.

D3.5 The 'services' branch (see D5) shall contain the service-type OIDs, and for each of them, it may contain the extended parameters, derived services, newly defined procedures, and service type specific ASN.1 modules.

D4 CSTS SPECIFICATION FRAMEWORK OBJECT IDENTIFIERS REGISTRATION

NOTES

1 This section specifies some rules regarding the construction of the strings that are used to name OIDs. While within this Recommended Standard these rules have been adhered to, this is not meant to impose the same rules on related documents such as CSTS specifications. If deemed more convenient, different rules for the construction of strings naming OIDs may be applied in those documents.

2 To better illustrate how strings used to name OIDs are built, all figures in sections D4 to D6 contain such strings, but these strings are examples only; that is, they do not name actually specified OIDs of service types, procedures, parameters, events, directives, etc. As such, the figures in these sections are informative only.

D4.1 New OIDs in the 'framework' branch shall be allocated only in the context of an update of this Recommended Standard.

D4.2 New OIDs in the 'modules' branch shall be allocated only if new ASN.1 modules are created in this Recommended Standard.

D4.3 New OIDs in the 'operations' branch shall be allocated only in the context of a definition of new operations and additional extensions in the definitions of the operations in this Recommended Standard.

NOTE – Strictly speaking, not the operations as such but the PDUs associated with a given operation are registered in the 'operations' branch. For example, different OIDs are assigned to the START invocation and to the START return PDU, respectively. Likewise, extensions are registered separately for each PDU.

D4.4 New OIDs in the 'procedures' branch (see figure D-2) shall be allocated only in the context of

a) definition of new procedures: in case a new procedure is created in this Recommended Standard, the newly allocated OID shall be complemented with two subbranches:

   1) '(procedure classifier)DerivedProcedures' to register any derivation of a procedure from this newly registered procedure,

   2) '(procedure classifier)ExtProcedureParam' to register operations parameters extensions; and

b) procedure derived from an existing procedure: the created OID shall be complemented with two subbranches:

   1) '(derived procedure classifier)DerivedProcedures' to register the derivation of a procedure from the given parent procedure,

   2) '(derived procedure classifier)ExtProcedureParam' to register the operations parameters extension.

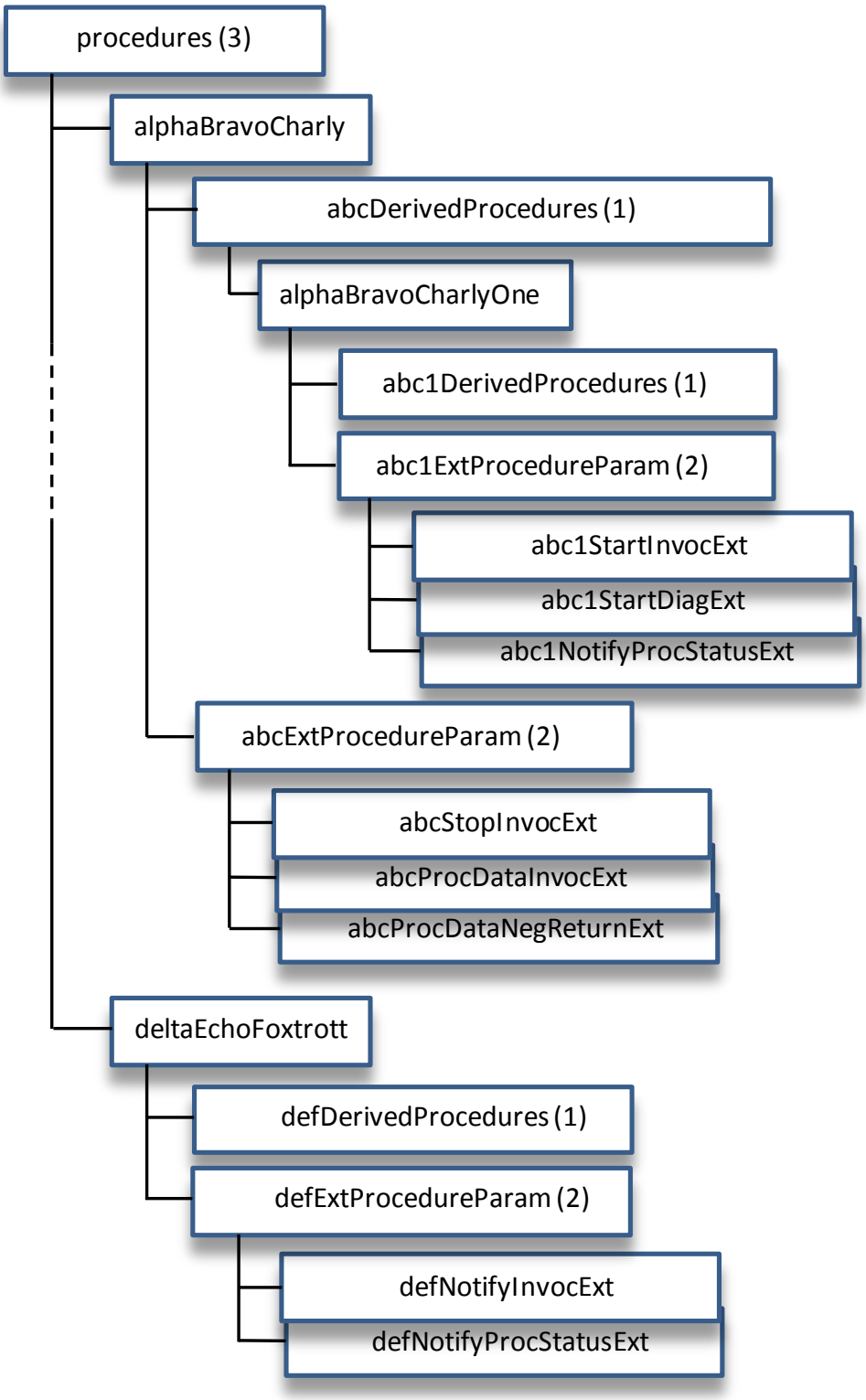

Figure D-2: 'procedures' Subtree

D4.5 New OIDs in the 'proceduresFunctionalities' branch shall be allocated only in the context of the definition of a new procedure in this Recommended Standard. The newly

allocated OID node name shall be of the form ‘proc(Procedure Long Name)’. Each such OID shall be complemented with three subbranches (see figure D-3):

a) ‘p(PROCEDURE SHORT NAME)parametersId’ to register the procedure’s parameters;

b) ‘p(PROCEDURE SHORT NAME)eventsId’ to register the procedure’s events;

c) ‘p(PROCEDURE SHORT NAME)directivesId’ to register the procedure’s directives.

NOTES

1 The <PROCEDURE SHORT NAME> strings used for the procedures contained in this Recommended Standard are specified in F3.16.

2 The <PROCEDURE SHORT NAME> string is the acronym, that is, the initialism of the <procedure long name string> and by definition is therefore ALL CAPS.

3 Figure K-4 illustrates the above specified OID tree structure for all procedures specified within this Recommended Standard.

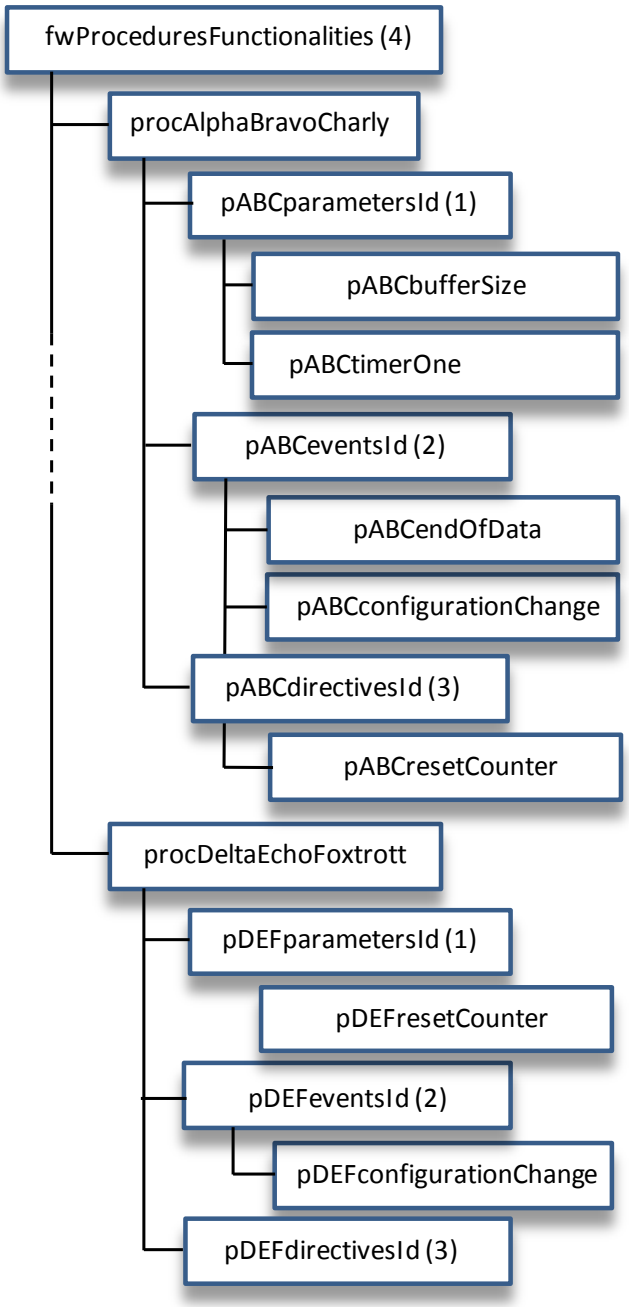

Figure D-3: 'fwProceduresFunctionalities' Subtree

D5 SERVICE OBJECT IDENTIFIERS REGISTRATION

D5.1 Whenever a new non-derived service is specified, a new OID shall be allocated directly under the 'services' node. The allocated OID shall be complemented with five subbranches (see figure D-4):

a) '(service classifier)DerivedServices' to register services derived from the specified service;

b) '(service classifier)ExtendedServiceParameters' to register service type specific parameters;

c) '(service classifier)ServiceProcedures' to register procedures derived for this service;

d) '(service classifier)ServiceModules' to register service type specific ASN.1 modules;

e) '(service classifier)ServiceFrRef' to register the reference to the Functional Resource Type that models the given service type.

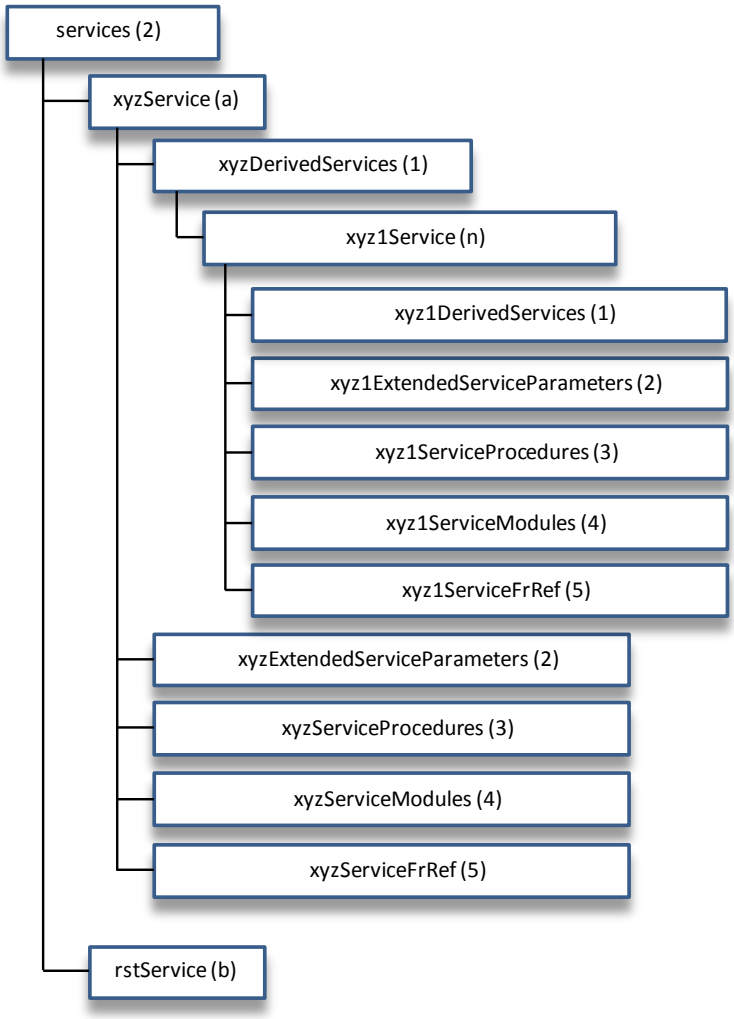

Figure D-4: 'services' Subtree

D5.2 For each procedure defined for a given service, four subbranches shall be created directly under that procedure node as follows (see figure D-5):

a) '(service classifier)SP(service procedure classifier)ExtendedOpsParam' to register extensions of operation parameters used by this procedure;

b) '(service classifier)SP(service procedure classifier)ExtendedProcParam' to register extensions of procedure specific parameters;

c) '(service classifier)SP(service procedure classifier)EventsId' to register procedure type specific events;

d) '(service classifier)SP(service procedure classifier)DirectivesId' to register procedure type specific directives.

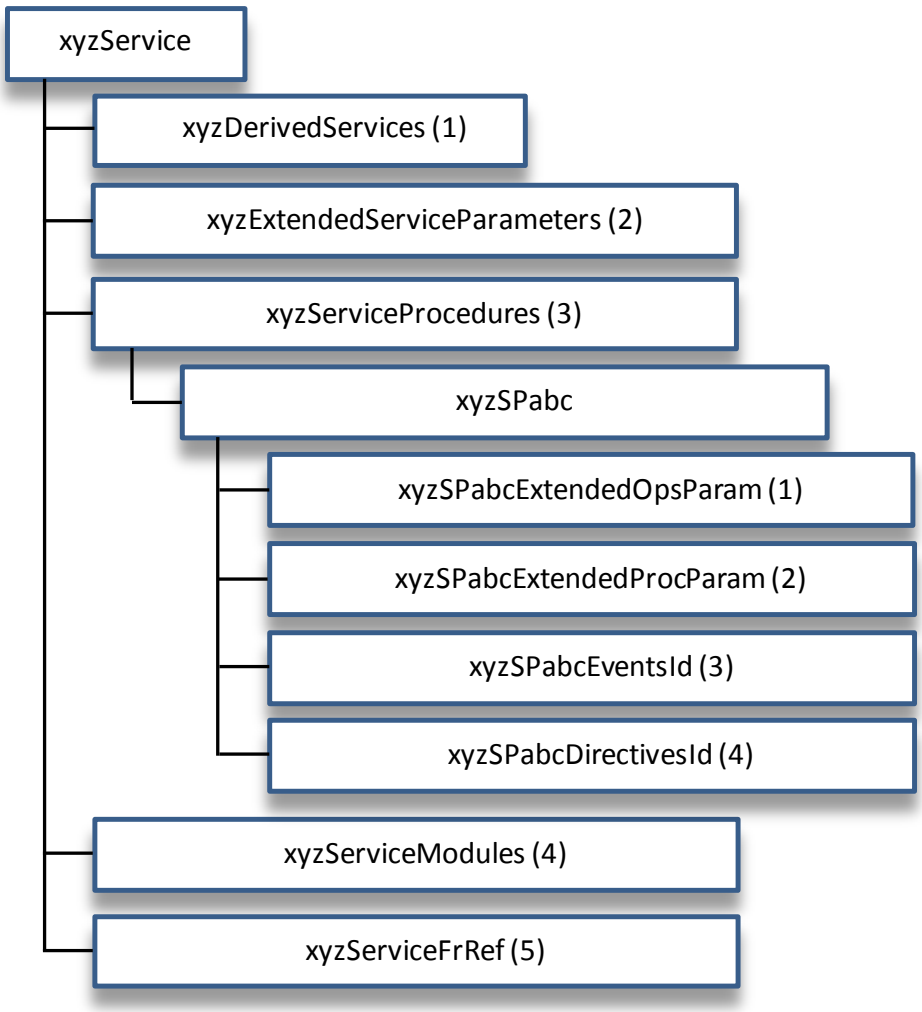

Figure D-5: 'service procedures' Subtree

D5.3 Whenever a new service derived from an existing service is specified, a new OID shall be allocated directly under the service it is derived from (see figure D-4). The allocated OID shall be complemented with five subbranches:

a) '(derived service classifier)DerivedServices' to register services derived from the specified service;

b) ‘(derived service classifier)ExtendedServiceParameters’ to register parameters syntax;

c) ‘(derived service classifier)ServiceProcedures’ to register derived procedures and extended parameters for this service;

d) ‘(derived service classifier)ServiceModules’ to register ASN.1 modules for this service;

e) ‘(derived service classifier)ServiceFrRef’ to register the reference to the Functional Resource Type that models the given service type.

D6 CROSS SUPPORT RESOURCES OBJECT IDENTIFIERS REGISTRATION

D6.1 GENERAL

The Cross Support Resources branch shall be divided into two branches: ‘crossSupportFunctionalities’ and ‘agenciesFunctionalities’.

D6.2 CROSS SUPPORT FUNCTIONALITIES BRANCH

D6.2.1 Published Identifiers in this branch shall be allocated independently of this Recommended Standard.

D6.2.2 Published Identifier registration shall ensure registration of parameters, events, and directives grouped per Functional Resource Type.

D6.2.3 Functional Resource Types, Parameter Identifiers, Event Identifiers, and Directive Identifiers shall be registered as Published Identifiers (see figure D-6).

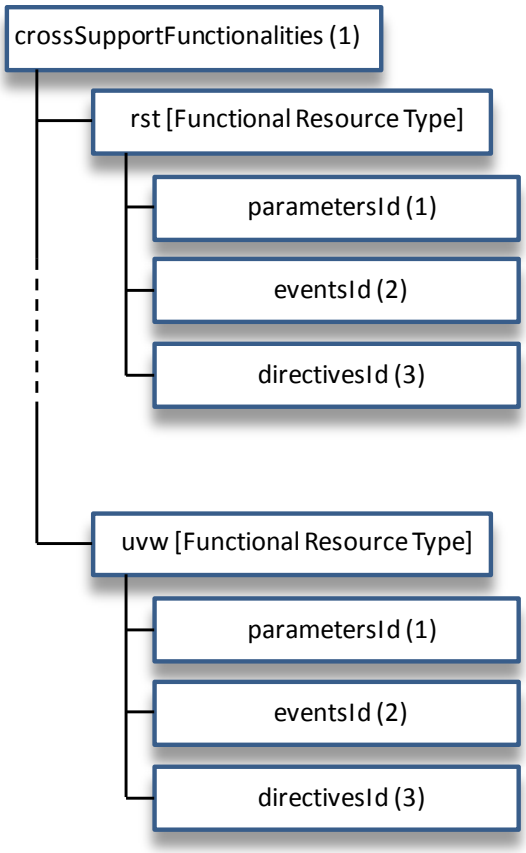

Figure D-6: 'crossSupportFunctionalities' Subtree

D6.2.4 Whenever a new Functional Resource Type is defined,

a) a 'functionalResourceType' Published Identifier shall be allocated for that Functional Resource Type with three subbranches: 'parametersId', 'eventsId', and 'directivesId', as required in D6.2.3;

b) Functional Resource Type specific parameters (in the 'parametersId' branch) shall be allocated reporting the specific parameters that may be defined for that Functional Resource Type;

c) Functional Resource Type specific events (in the 'eventsId' branch) shall be allocated corresponding to the specific events that may be defined for that Functional Resource Type; and

d) Functional Resource Type specific directives (in the 'directivesId' branch) shall be allocated corresponding to the specific directives that may be defined for that Functional Resource Type.

NOTE – The Parameter Identifiers, Event Identifiers, and Directive Identifiers are used to construct the Parameter, Event, and Directive Names that are transferred via various CSTS operations. Annex E specifies the rules for the construction of names using these identifiers.

D6.2.5 Whenever a new CSTS is defined, a 'functionalResourceType' Published Identifier shall be allocated for that CSTS with three branches: 'parametersId', 'eventsId', and 'directivesId', as required in D6.2.3, and populated as specified in D6.2.4.

NOTE – The provision of any CSTS is modeled by means of a service type specific Functional Resource. Therefore the definition of a new CSTS requires also the specification of the related Functional Resource and consequently the allocation of an OID to that Functional Resource Type.

D6.2.6 Whenever a new version of a Functional Resource is required, it shall be assigned a new OID. The old Functional Resource shall remain untouched, but may be deprecated.

D6.2.7 Whenever a new version of a parameter, event, or directive is required, the following steps shall be performed:

a) the existing parameter/event/directive shall be left untouched;

b) a new parameter/event/directive shall be created the OID of which shall be the same as the original parameter except that the appended version number shall be incremented by one;

c) at any point in time, the original parameter may be deprecated.

D6.3 AGENCIES FUNCTIONALITIES

D6.3.1 An Agency may support agency specific Functional Resources that are not covered under the 'crossSupportFunctionalities' branch.

D6.3.2 That Agency may request from SANA a dedicated OID that is allocated to that Agency and is to be registered under the 'agenciesFunctionalities' branch.

D6.3.3 The Agency may then request SANA to register new Functional Resources, parameters, events, and directives, in line with the structure adopted under the 'crossSupportFunctionalities' branch (see figure D-7).

NOTES

1 It is the responsibility of the Agency to allocate and maintain the required Published Identifiers (Functional Resource Types, parameters, events, and directives).

2 It is the responsibility of the Agency not to duplicate already existing 'crossSupportFunctionalities' definitions.

3 The registration rules applying to the 'agenciesFunctionalities' subbranch and to the related registration of Agency-specific Functional Resources are documented in H2.4.

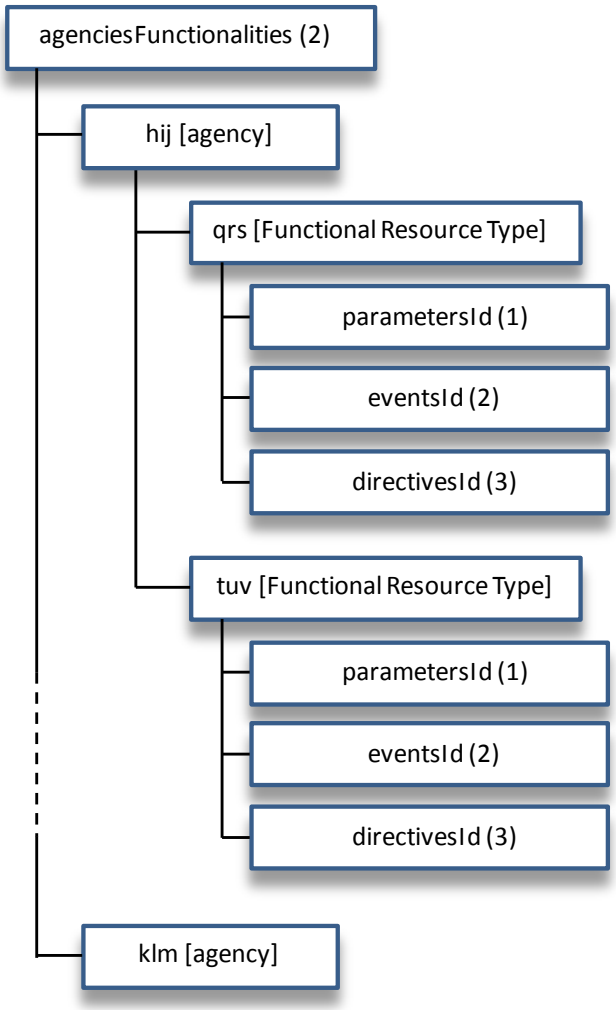

Figure D-7: 'agenciesFunctionalities' Subtree

ANNEX E

COMPOSITION OF PARAMETER, EVENT, AND DIRECTIVE NAMES AND PARAMETER AND EVENT LISTS

(NORMATIVE)

E1 OVERVIEW

Naming Functional Resources and the observable parameters, notifiable events, and directives associated with those Functional Resources is achieved by means of Published Identifiers. Published Identifiers are OIDs that are registered with the Space Assigned Numbers Authority. Use of Published Identifiers allows for the specification of CSTSes that can use parameters or events that have not been identified at the time of the writing of those CSTS specifications.

Those Published Identifiers are used in a variety of ways in the operations and procedures of the CSTS Specification Framework:

a) to form the *Parameter Names* used (1) in the `list-of-parameters` parameter of the GET operation (3.12.2.2.2) and the Cyclic Report procedure START operation (4.10.4.1.2.3), and (2) in the `qualified-parameters` parameter of the GET operation (3.12.2.2.3) and the Cyclic Report procedure TRANSFER-DATA operation (4.10.4.2.2.2);

b) to form the *Event Names* used in the `event-name` parameter of the NOTIFY operation (3.11.2.2.3.1) and the `list-of-events` parameter of the Notification procedure START operation (4.11.4.1.2.2);

c) to form the *Parameter Names* that might be used in the `directive-qualifier` parameter of the EXECUTE-DIRECTIVE operation (3.13.2.2.3);

d) to form the logical records that constitute the named and default parameter lists used in the `list-of-parameters` parameters of the GET operation (3.12.2.2.2) and the Cyclic Report procedure START operation (4.10.4.1.2.3);

e) to form the logical records that constitute the named and default event lists used in the `list-of-events` parameter of the Notification procedure START operation (4.11.4.1.2.2);

f) to form the Functional Resource Names used (1) in the `list-of-parameters` parameters of the GET operation (3.12.2.2.2) and the Cyclic Report procedure START operation (4.10.4.1.2.3), and (2) in the `list-of-events` parameter of the Notification procedure START operation (4.11.4.1.2.2);

g) to form the procedure name used in the `list-of-parameters` parameters of the GET operation (3.12.2.2.2), the Cyclic Report procedure START operation (4.10.4.1.2.3), and the `list-of-events` parameter of the Notification procedure START operation (4.11.4.1.2.2).

This annex defines the composition of Functional Resource Names, Parameter Names, Event Names, Directive Names, Parameter Lists, and Event Lists using Published Identifiers of the appropriate types.

E2 FUNCTIONAL RESOURCE NAME

E2.1 Each Functional Resource Name shall be composed of a Functional Resource Type Published Identifier and a Functional Resource Instance Number.

E2.2 The Functional Resource Type shall be registered on one of the following CCSDS OID subtrees:

a) {ccsds css crossSupportResources crossSupportFunctionalities} for a CCSDS-standard Functional Resource Type; or

b) {ccsds css crossSupportResources agenciesFunctionalities <Agency X>} for an Agency-unique Functional Resource Type.

E3 PROCEDURE NAME

E3.1 Each procedure name shall be composed of a Procedure Type Published Identifier and the procedure role. If the role is 'association control' or 'prime procedure', no instance number is defined, as the procedure is already unambiguously identified. If the role is 'secondary procedure', an instance number is part of the procedure name.

E3.2 The Procedure Type shall be registered on the CCSDS OID subtree {ccsds css csts framework procedures}.

E4 PARAMETER NAME

E4.1 A Parameter Name shall be composed of a Parameter Identifier and either the Functional Resource Name of the Functional Resource Instance that generates the values for that instance of the parameter or the procedure name of the procedure that generates the value for that instance of the procedure configuration parameter.

E4.2 If a Functional Resource Name is used to form the Parameter Name, then the Parameter Identifier shall be registered on one of the following CCSDS OID subtrees:

a) {ccsds css crossSupportResources crossSupportFunctionalities <functional resource type A> parametersId} for a parameter that is specified as part of a CCSDS-standard Functional Resource Type definition;

b) {ccsds css crossSupportResources agenciesFunctionalities <Agency X> <functional resource type A> parametersId} for a parameter that is specified as part of an Agency-unique Functional Resource Type definition or an Agency-specific extension of a CCSDS-standard Functional Resource Type.

NOTE – If an Agency uses the 'agenciesFunctionalities' subtree to register additional Agency-specific parameters for a CCSDS-standard Functional Resource Type, there is not necessarily any relationship between the integer value of the Functional Resource Type node under that Agency's 'agenciesFunctionalities' subtree and the value of the Functional Resource Type node under the 'crossSupportFunctionalities' subtree.

E4.3 If a procedure name is used to form the Parameter Name, then the Parameter Identifier shall be registered on the {ccsds css csts framework fwProceduresFunctionalities proc<Procedure Long Name> p<PROCEDURE SHORT NAME>parametersId} subtree for a configuration parameter that is defined for that framework procedure.

NOTE – The <Procedure Long Name> and <PROCEDURE SHORT NAME> strings used for the procedures contained in this Recommended Standard are formed in accordance with the rules laid down in D4.5 and specified in F3.16.

E5 EVENT NAME

E5.1 An Event Name shall be composed of an Event Identifier and either the Functional Resource Name of the Functional Resource Instance that generates the notifications for that instance of the event or the procedure name of the framework procedure that issues the notification.

E5.2 If a Functional Resource Name is used to form the Event Name, then the Event Identifier shall be registered on one of the following CCSDS OID subtrees:

a) {ccsds css crossSupportResources crossSupportFunctionalities <functional resource type A> eventsId} for an event that is specified as part of a CCSDS-standard Functional Resource Type definition;

b) {ccsds css crossSupportResources agenciesFunctionalities <Agency X> <functional resource type A> eventsId} for an event that is specified as part of an Agency-unique Functional Resource Type definition or an Agency-specific extension of a CCSDS-standard Functional Resource Type.

NOTE – If an Agency uses the 'agenciesFunctionalities' subtree to register additional Agency-specific events for a CCSDS-standard Functional Resource Type, there is not necessarily any relationship between the integer value of the Functional Resource Type node under that Agency's 'agenciesFunctionalities' subtree and the value of the Functional Resource Type node under the 'crossSupportFunctionalities' subtree.

E5.3 If a procedure name is used to form the Event Name, then the Event Identifier shall be registered on the {ccsds css csts framework fwProceduresFunctionalities proc<Procedure Long Name> p<PROCEDURE SHORT NAME>eventsId} subtree for a notification that is defined for a framework procedure.

NOTE – The <Procedure Long Name> and <PROCEDURE SHORT NAME> strings used for the procedures contained in this Recommended Standard are formed in accordance with the rules laid down in D4.5 and specified in F3.16.

E6 DIRECTIVE NAME

E6.1 A Directive Name shall be composed of a Directive Identifier and either the Functional Resource Name of the Functional Resource Instance that receives that instance of the directive or the procedure name of the framework procedure that shall be acted on by that directive.

E6.2 If a Functional Resource is used to form the Directive Name, then the Directive Identifier shall be registered on one of the following CCSDS OID subtrees:

a) {ccsds css crossSupportResources crossSupportFunctionalities <functional resource type A> directivesId} for a directive that is specified as part of a CCSDS-standard Functional Resource Type definition;

b) {ccsds css crossSupportResources agenciesFunctionalities <Agency X> <functional resource type A> directivesId} for a directive that is specified as part of an Agency-unique Functional Resource Type definition or an Agency-specific extension of a CCSDS-standard Functional Resource Type.

NOTE – If an Agency uses the 'agenciesFunctionalities' subtree to register additional Agency-specific directives for a CCSDS-standard Functional Resource Type, there is not necessarily any relationship between the integer value of the Functional Resource Type node under that Agency's 'agenciesFunctionalities' subtree and the value of the Functional Resource Type node under the 'crossSupportFunctionalities' subtree.

E6.3 If a procedure name is used to form the Directive Name, then the Directive Identifier shall be registered on the {ccsds css csts framework fwProceduresFunctionalities proc<Procedure Long Name> p<PROCEDURE SHORT NAME>directivesId} subtree for a directive that is defined for a framework procedure.

NOTE – The <Procedure Long Name> and <PROCEDURE SHORT NAME> strings used for the procedures contained in this Recommended Standard are formed in accordance with the rules laid down in D4.5 and specified in F3.16.

E7 PARAMETER LABEL

A Parameter Label shall be the Published Identifier of that parameter. Given the way that a Published Identifier is constructed, it identifies the parameter and either the associated Functional Resource Type or the associated procedure type.

E8 EVENT LABEL

An Event Label shall be the Published Identifier of that event. Given the way that a Published Identifier is constructed, it identifies the event and either the associated Functional Resource Type or the associated procedure type.

E9 PARAMETER LIST

E9.1 A parameter list shall contain a set of one or more Parameter Labels.

E9.2 Each parameter list record shall represent all instances of the parameter type represented by the Parameter Identifier for all instances of the Functional Resource Type or procedure type that are associated with the service executing the procedure that uses the parameter list.

NOTE – For example, assume that one of the Parameter Labels of a parameter list includes the `return-buffer-size` Parameter Identifier specified for the Buffered Data Delivery procedure. If that parameter list is put into the `list-of-parameters` parameter of the GET invocation or the Cyclic Report procedure START invocation, it signifies that the `return-buffer-size` parameter value is to be sent for every instance of the Buffered Data Delivery procedure that is active and executing within the service instance that executes the Information Query or the Cyclic Report procedure.

E10 EVENT LIST

E10.1 An event list shall logically consist of a set of one or more Event Labels, each of which is an Event Identifier Published Identifier. Because of the way such a Published Identifier is constructed, it identifies the event and the associated Functional Resource Type or procedure type.

E10.2 Each event list record shall represent all instances of the event type represented by the Event Identifier for all instances of the Functional Resource Type or procedure type that are associated with the service instance executing the Notification procedure.

NOTE – For example, assume that one of the Event Labels of an event list includes the `configuration-change` Event Identifier of the Buffered Data Delivery procedure. If that event list is put into the `list-of-parameters` parameter of the Notification procedure START invocation, it signifies that the `pBDPconfigurationChange` notification is to be sent whenever it occurs on any of the instances of the Buffered Data Delivery procedure type that that are active and being executed by the service instance that is executing the Notification procedure.

E11 PARAMETERS, EVENTS, AND DIRECTIVES DEFINITION

E11.1 A parameter, an event, and a directive shall be defined using the following definition:

a) a classifier, that is, a compact string in 'camel case' notation and using standard abbreviations (e.g., 'fwd' for 'forward') indicating the purpose of the parameter, event, or directive;

b) the semantic definition of the parameter in the form of free text;

c) a name in the form of an OID, where the last digit indicates the version of the definition (see Parameter Name definition in 1.6.1.7.33, Event Name definition in 1.6.1.7.18, and Directive Name definition in 1.6.1.7.15);

d) a syntax specification, preferably expressed in ASN.1, and an OID referencing this syntax, in which this OID shall be identical with the OID assigned as per c) above;

e) in the case of a parameter, a flag indicating if this parameter can be configured;

f) in the case of a directive acting on one or more parameters flagged as 'configured', a guard condition; if that guard condition does not evaluate to true, the directive must not be executed on that parameter; that is, the parameter value must not be modified;

g) if applicable and not implied by the data type specification, the range and other constraints (limits) shall be defined;

h) if applicable, the engineering unit(s), preferably SI units, shall be stated;

i) a flag indicating if the definition has been deprecated or not;

NOTE – The deprecation flag indicates that at least one more-recent version of the specification of the given parameter, event, or directive exists. The deprecated version can no longer be expected to be supported by providers of CSTSes. In case one or more parameters, events, or directives of a Functional Resource shall no longer be supported at all regardless of their version, a new Functional Resource Type is to be specified.

j) the authorizing entity (e.g., CSS Area);

k) the creation date.

E11.2 Requests to add assignments to this registry shall be submitted to SANA and come from a member Agency, an observer Agency, a CCSDS Associate, or an industry partner supported by a member Agency. The request shall be related to a cross support activity. After evaluation of the request and approval by the CSS Area chair or deputy or a person duly authorized by her/him, a new OID will be allocated and added to the existing list.

ANNEX F

DATA TYPES DEFINITION

(NORMATIVE)

F1 OVERVIEW

F1.1 This annex defines the data types that are used by the procedures defined in this Recommended Standard. It is intended to provide a clear specification of these data types and to avoid ambiguity. It is not intended to constrain how these data types are implemented or encoded. These definitions are suitable for inclusion in any type of ASN.1-based protocol that implements Cross Support Services.

F1.2 The data type definitions are presented in ASN.1 modules.

F1.3 Conceptually, the CSTS Specification Framework ASN.1 definitions break into three levels (putting aside, for the moment, the top-level data structure that contains a Framework PDU). The lowest level consists of data types; these are used as building blocks. The middle level consists of operation messages (invocations, returns, and acknowledgements); these are generic definitions that can be extended by procedures. The highest level consists of procedure-specific information (e.g., extensions of the generic operation messages).

F1.4 The ASN.1 definitions are broken down into modules. A distinction is made between *general purpose* items (items used by multiple procedures) and *procedure-specific* items (items used by only a single procedure). General purpose building blocks are defined in a single module called 'Cross Support Transfer Service – Common Types' (F3.3). General purpose operation messages are defined in a single module called 'Common Operation PDUs' (F3.4). For each procedure, there is a dedicated module that contains procedure-specific building blocks (if any), procedure-specific operation messages (if any), and procedure-specific extensions to the generic operation messages (if any). Generally, the modules are arranged in bottom-up order:

a) F3.1 – list of OIDs for Framework operations and procedures;

b) F3.2 – building blocks that are used only by the Bind-Invocation message (the Service-Instance-Id);

c) F3.3 – CSTS common types;

d) F3.4 – common operations PDUs;

e) F3.5 – procedure-specific information for the Association Control procedure;

f) F3.6 through F3.14 – procedure-specific information (each subsection dedicated to one procedure);

g) F3.15 – the top-level CSTS Specification Framework PDUs;

h) F3.16 – procedure-specific OIDs associated with parameters, events, and directives.

F1.5 The top-level Framework PDU specifies an operation (e.g., Bind) and a specific message type within that operation (i.e., invocation, return, or acknowledgement). The combination is an operation message (e.g., Bind-Invocation). As mentioned above, if the operation is of a general purpose, its messages will be defined in the 'Common Operation PDUs' subsection (F3.4). Otherwise, the operation's messages will be defined within one of the subsections dedicated to specific procedures; for example, the Bind, Unbind, and PEER-ABORT operations are defined in the subsection dedicated to the Association Control procedure (F3.5).

F2 EXTENSION

F2.1 The extension capability may or may not be used using the `Extended` parameter. If not used, the `Extended` parameter shall carry a 'null' value, also referred to as 'notUsed'.

F2.2 Extension is defined by means of 'EMBEDDED PDV'. The ASN.1 `EMBEDDED PDV` type is a type used to include non-ASN.1 or other data within an ASN.1 encoded message. This type is described using the following ASN.1 SEQUENCE:

```
EMBEDDED PDV ::= [UNIVERSAL 11] SEQUENCE
{   identification              CHOICE
    {   syntaxes                    SEQUENCE
        {   abstract                    OBJECT IDENTIFIER
        ,   transfer                    OBJECT IDENTIFIER
        }
    ,   syntax                      OBJECT IDENTIFIER
    ,   presentation-context-id     INTEGER
    ,   context-negotiation         SEQUENCE
        {   presentation-context-id     INTEGER
        ,   transfer-syntax             OBJECT IDENTIFIER
        }
    ,   transfer-syntax             OBJECT IDENTIFIER
    ,   fixed                       NULL
    }
,   data-value                  OCTET STRING
}
```

F2.3 The extension shall make use of the 'syntax' definition in the 'identification' CHOICE. The 'syntax' is assigned an OID (see F3.1) that is unique for all extensions and clearly indicates that all external syntaxes carried by this definition belong to the Cross Support Services. The effect of this restriction is to make the EMBEDDED PDV type, as it applies to CSTSes, appear as the following type:

```
EMBEDDED PDV ::= [UNIVERSAL 11] SEQUENCE
{   identification              CHOICE
    {   syntax                      OBJECT IDENTIFIER
    }
,   data-value                  OCTET STRING
}
```

F2.4 The syntax used for the extension (see F2.2) shall follow the requirements defined in annex D.

F2.5 The assignment of the OID that identifies the syntax and the specification of the type of the `data-value` shall be placed next to OID assignment of the parameter, event value, or directive qualifier OID assignment in the same ASN.1 module.

F2.6 The name associated with the OID of a procedure parameter, event, or directive shall be formed as specified in D4.5.

F2.7 If a derived procedure has inherited events or directives from the parent procedure, the OIDs identifying these events and event values as well as these directives and directive qualifiers shall be inherited as well.

F2.8 The name associated with the OID of a Functional Resource parameter, event, or directive shall be formed as specified in D6.

NOTE – The names associated with the OIDs associated with Functional Resource parameters, events, or directives are specified in the related SANA registry (see H2.4).

F2.9 The OID of an `event-value` shall be formed by appending a digit to the OID of the associated event; the OID of a directive-qualifier shall be formed by appending a digit to the OID of the associated directive. In both cases, the appended digit shall start from the value '1'. This OID identifies either the syntax of the complete `event-value` or `directive-qualifier`, or of an individual element of the sequence forming the `event-value` or `directive-qualifier` (see `EventValue` in F3.3 and `DirectiveQualifierValues` in F3.4).

NOTE – Complex event values or directive qualifiers can be expressed by defining a complex type, such that the event value is specified by a single type definition. Alternatively, separate types can be specified for the individual elements forming the event value or directive qualifier. In the latter case, the event value or directive qualifier is expressed as a sequence of the individual elements, in which for each type-specification of such an element, an OID needs to be assigned.

F2.10 The name associated with the OID of an `event-value` shall be formed by appending the string 'EvtValue' to the name of the OID of the associated event. If the `event-value` is expressed as a sequence of values with individually specified types, then the last digit of the assigned OID shall be appended to the OID name of each of the elements.

F2.11 The name associated with the OID of a `directive-qualifier` shall be formed by appending the string 'DirQual' to the name of the OID of the associated directive. If the `directive-qualifier` is expressed as a sequence of values with individually specified types, then the last digit of the assigned OID shall be appended to the OID name of each of the elements.

F2.12 The name of a parameter value, an `event-value`, or a `directive-qualifier` shall be formed by appending the string 'Type' to the name of the OID that identifies the parameter, `event-value`, or `directive-qualifier`.

NOTE – As an example for illustration of the above requirements, the 'buffered data delivery configuration change' event as specified in 4.5.4.2.2 is taken here. The OID assigned in F3.16 to this event in F3.16 is

```
pBDDconfigurationChange  OBJECT IDENTIFIER    ::=   {pBDDeventsId 4}
```

The event value specified in 4.5.4.2.2 consists of a sequence of two elements, the `return-buffer-size` and the `delivery-latency-limit`. Due to this specification, two OIDs need to be assigned so that the syntax of each of the elements can be unambiguously identified. Applying the above-specified rules, the OID assignments and type specifications are:

```
pBDDconfigurationChangeEvtValue1   OBJECT IDENTIFIER  ::=
                                                {pBDDreturnBufferSize}
PBDDconfigurationChangeEvtValue1Type   ::=    PBDDreturnBufferSizeType

pBDDconfigurationChangeEvtValue2   OBJECT IDENTIFIER  ::=
                                                {pBDDdeliveryLatencyLimit}
PBDDconfigurationChangeEvtValue2Type   ::=    PBDDdeliveryLatencyLimitType
```

If the event 'buffered data delivery configuration change' were specified to use a single element for forming the `event-value`, then the ASN.1 specification would be:

```
pBDDconfigurationChangeEvtValue    OBJECT IDENTIFIER  ::=
                                                {pBDDconfigurationChange 1}
PBDDconfigurationChangeEvtValueType    ::=    SEQUENCE
{ returnBufferSize       PBDDreturnBufferSizeType
, deliveryLatencyLimit   PBDDdeliveryLatencyLimitType
}
```

F3 DATA TYPE SPECIFICATION

F3.1 LIST OF OBJECT IDENTIFIERS

This module defines the OIDs required for all syntaxes used for the definition of extended types.

```
CCSDS-CSTS-OBJECT-IDENTIFIERS
{   iso(1) identified-organization(3) standards-producing-organization(112)
    ccsds(4) css(4) csts(1) framework(1) modules(1) object-identifiers(1) version(2)
}

DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

EXPORTS    acExtProcedureParam
,          agenciesFunctionalities
,          bddExtProcedureParam
,          bdpExtProcedureParam
,          crExtProcedureParam
,          crossSupportFunctionalities
,          dpExtProcedureParam
,          executeDirectiveAcknowledge
,          executeDirectiveReturn
,          fwProceduresFunctionalities
,          getReturn
,          modules
,          nExtProcedureParam
,          operations
,          procedures
,          scdpExtProcedureParam
,          services
,          startReturn
,          teExtProcedureParam;

css                                OBJECT IDENTIFIER ::= {1 3 112 4 4}

csts                               OBJECT IDENTIFIER ::= {css 1}
crossSupportResources              OBJECT IDENTIFIER ::= {css 2}

framework                          OBJECT IDENTIFIER ::= {csts 1}
services                           OBJECT IDENTIFIER ::= {csts 2}

-- ================================================================
-- FRAMEWORK OBJECT IDENTIFIERS
```

```
fwProceduresFunctionalities         OBJECT IDENTIFIER ::= {framework 4}
modules                             OBJECT IDENTIFIER ::= {framework 1}
operations                          OBJECT IDENTIFIER ::= {framework 2}
procedures                          OBJECT IDENTIFIER ::= {framework 3}
-- ************************************
-- FRAMEWORK OPERATIONS IDENTIFIERS:
bindInvocation                      OBJECT IDENTIFIER ::= {operations 1}
bindReturn                          OBJECT IDENTIFIER ::= {operations 2}
unbindInvocation                    OBJECT IDENTIFIER ::= {operations 3}
unbindReturn                        OBJECT IDENTIFIER ::= {operations 4}
peerAbortInvocation                 OBJECT IDENTIFIER ::= {operations 5}
startInvocation                     OBJECT IDENTIFIER ::= {operations 6}
startReturn                         OBJECT IDENTIFIER ::= {operations 7}
stopInvocation                      OBJECT IDENTIFIER ::= {operations 8}
stopReturn                          OBJECT IDENTIFIER ::= {operations 9}
executeDirectiveInvocation          OBJECT IDENTIFIER ::= {operations 10}
executeDirectiveAcknowledge         OBJECT IDENTIFIER ::= {operations 11}
executeDirectiveReturn              OBJECT IDENTIFIER ::= {operations 12}
getInvocation                       OBJECT IDENTIFIER ::= {operations 13}
getReturn                           OBJECT IDENTIFIER ::= {operations 14}
notifyInvocation                    OBJECT IDENTIFIER ::= {operations 15}
transferDataInvocation              OBJECT IDENTIFIER ::= {operations 16}
processDataInvocation               OBJECT IDENTIFIER ::= {operations 17}
processDataReturn                   OBJECT IDENTIFIER ::= {operations 18}


-- ************************************
-- FRAMEWORK PROCEDURES IDENTIFIERS:
-- Identifiers to be used with the type ProcedureType
-- This branch is used to support all extension definitions required
-- for the operations extended by procedures.
associationControl                  OBJECT IDENTIFIER ::= {procedures 1}
unbufferedDataDelivery              OBJECT IDENTIFIER ::= {procedures 2}
bufferedDataDelivery                OBJECT IDENTIFIER ::= {procedures 3}
dataProcessing                      OBJECT IDENTIFIER ::= {procedures 4}
informationQuery                    OBJECT IDENTIFIER ::= {procedures 5}
notification                        OBJECT IDENTIFIER ::= {procedures 6}
throwEvent                          OBJECT IDENTIFIER ::= {procedures 7}

acDerivedProcedures                 OBJECT IDENTIFIER ::= {associationControl 1}
acExtProcedureParam                 OBJECT IDENTIFIER ::= {associationControl 2}
uddDerivedProcedures                OBJECT IDENTIFIER ::= {unbufferedDataDelivery 1}
uddExtProcedureParam                OBJECT IDENTIFIER ::= {unbufferedDataDelivery 2}
bddDerivedProcedures                OBJECT IDENTIFIER ::= {bufferedDataDelivery 1}
bddExtProcedureParam                OBJECT IDENTIFIER ::= {bufferedDataDelivery 2}
dpDerivedProcedures                 OBJECT IDENTIFIER ::= {dataProcessing 1}
dpExtProcedureParam                 OBJECT IDENTIFIER ::= {dataProcessing 2}
iqDerivedProcedures                 OBJECT IDENTIFIER ::= {informationQuery 1}
iqExtProcedureParam                 OBJECT IDENTIFIER ::= {informationQuery 2}
nDerivedProcedures                  OBJECT IDENTIFIER ::= {notification 1}
nExtProcedureParam                  OBJECT IDENTIFIER ::= {notification 2}
teDerivedProcedures                 OBJECT IDENTIFIER ::= {throwEvent 1}
teExtProcedureParam                 OBJECT IDENTIFIER ::= {throwEvent 2}
bufferedDataProcessing              OBJECT IDENTIFIER ::= {dpDerivedProcedures 1}
cyclicReport                        OBJECT IDENTIFIER ::= {uddDerivedProcedures 1}
sequenceControlledDataProcessing    OBJECT IDENTIFIER ::= {dpDerivedProcedures 2}

crDerivedProcedures                 OBJECT IDENTIFIER ::= {cyclicReport 1}
crExtProcedureParam                 OBJECT IDENTIFIER ::= {cyclicReport 2}
```

```
bdpDerivedProcedures                OBJECT IDENTIFIER ::= {bufferedDataProcessing 1}
bdpExtProcedureParam                OBJECT IDENTIFIER ::= {bufferedDataProcessing 2}

scdpDerivedProcedures               OBJECT IDENTIFIER ::=
                                       {sequenceControlledDataProcessing 1}
scdpExtProcedureParam               OBJECT IDENTIFIER ::=
                                       {sequenceControlledDataProcessing 2}


-- ================================================================
-- CROSS SUPPORT RESOURCES
crossSupportFunctionalities         OBJECT IDENTIFIER ::= {crossSupportResources 1}
agenciesFunctionalities             OBJECT IDENTIFIER ::= {crossSupportResources 2}

END
```

F3.2 SERVICE-INSTANCE-ID

NOTE – This module defines the format of the Service Instance Identifier (see 3.4.2.2.7) used in the BIND operation.

```
CCSDS-CSTS-SERVICE-INSTANCE-ID
{   iso(1) identified-organization(3) standards-producing-organization(112)
    ccsds(4) css(4) csts(1) framework(1) modules(1) service-instance(2)
    version(1)
}

DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

EXPORTS    ServiceInstanceIdentifier
;
IMPORTS    IntUnsigned
,          PublishedIdentifier
    FROM CCSDS-CSTS-COMMON-TYPES
;

ServiceInstanceIdentifier    ::=    SEQUENCE
{   spacecraftId                PublishedIdentifier
,   facilityId                  PublishedIdentifier
,   serviceType                 PublishedIdentifier
,   serviceInstanceNumber       IntUnsigned
}


END
```

F3.3 CROSS SUPPORT TRANSFER SERVICE — COMMON TYPES

```
CCSDS-CSTS-COMMON-TYPES
{   iso(1) identified-organization(3) standards-producing-organization(112)
    ccsds(4) css(4) csts(1) framework(1) modules(1) common-types(3) version(2)
}
```

```
DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

EXPORTS    AbstractChoice
,          AdditionalText
,          AuthorityIdentifier
,          BufferSize
,          ConditionalTime
,          DataTransferMode
,          DataUnitId
,          DeliveryLatencyLimit
,          DeliveryMode
,          Diagnostic
,          Duration
,          Embedded
,          EventValue
,          Extended
,          FRorProcedureName
,          FunctionalResourceInstanceNumber
,          FunctionalResourceName
,          FunctionalResourceType
,          IdentifierString
,          IntPos
,          IntUnsigned
,          InvokeId
,          Label
,          ListOfNamesDiagnosticExt
,          ListOfParametersEvents
,          ListOfParamEventsDiagnostics
,          LogicalPortName
,          Name
,          PortId
,          ProcedureName
,          ProcessingLatencyLimit
,          ProductionStatus
,          PublishedIdentifier
,          QualifiedParameter
,          StandardAcknowledgeHeader
,          StandardInvocationHeader
,          StandardReturnHeader
,          Time
,          TypeAndValue
,          UnknownName
;


-- This type is used by operations allowing the procedures using them to select two
-- possibilities for the definition of the data parameter:
-- 1. opaqueString: direct use, no extension required;
-- 2. extendedData: definition of a complex type using a constructed syntax
AbstractChoice           ::=    CHOICE
{   opaqueString             [0]    OCTET STRING
,   extendedData             [1]    Embedded
}

-- In the event of a negative result, the value of the diagnostic
-- is complemented by the following type (see 3.2.1.7 a)):
AdditionalText           ::=    VisibleString
```

```
-- Appellation is used between the service provider and the service user.
-- This appellation not being formally agreed can only be used for logging
-- or tracing.
Appellation             ::=    VisibleString (SIZE (1 .. 128))

AuthorityIdentifier     ::=    IdentifierString (SIZE (3 .. 16))

BufferSize              ::=    IntPos

ConditionalTime         ::=    CHOICE
{   undefined               [0]    NULL
,   known                   [1]    Time
}

-- If credentials are used, it will be necessary that the internal
-- structure of the octet string is known to both parties. Since the
-- structure will depend on the algorithm used, it is not specified here.
-- However, the peer entities may use ASN.1 encoding to make the internal
-- structure visible.
Credentials             ::=    CHOICE
{   unused                  [0]    NULL
,   used                    [1]    OCTET STRING (SIZE (8 .. 256))
}

DataTransferMode        ::=    ENUMERATED
{   undefined               (0)
,   timely                  (1)
,   complete                (2)
}

DataUnitId              ::=    IntUnsigned

DeliveryLatencyLimit    ::=    IntPos

DeliveryMode            ::=    ENUMERATED
{   undefined               (0)
,   realTime                (1)
,   complete                (2)
}

-- The diagnostics defined here are to be used with all operation returns.
-- Note:
-- By means of the 'diagnosticExtension' CHOICE, additional values of the
-- diagnostic parameter can be introduced if that is necessary for the
-- negative return or acknowledgement of an operation.
Diagnostic          ::=    CHOICE
{   invalidParameterValue [0]    SEQUENCE
    {   text          AdditionalText
    ,   appellation   Appellation   -- of the invalid parameter
    }
,   conflictingValues     [1]    SEQUENCE
    {   text          AdditionalText
    ,   appellations  SEQUENCE OF Appellation
    }
,   otherReason           [2]    AdditionalText
,   unsupportedOption     [3]    AdditionalText
,   diagnosticExtension   [100]  Embedded
}
```

```
-- The Duration maybe expressed in seconds, milliseconds, or microseconds
Duration              ::=    CHOICE
{   seconds              [0]    IntUnsigned
,   milliseconds         [1]    IntUnsigned
,   microseconds         [2]    IntUnsigned
}

Embedded              ::=    EMBEDDED PDV

EventValue            ::=    CHOICE
{   qualifiedValues      [0]    SequenceOfQualifiedValue
,   empty                [1]    NULL
,   eventValueExtension  [100]  Embedded
}

Extended              ::=    CHOICE
{   external             [0]    Embedded
,   notUsed              [1]    NULL
}

FRorProcedureName     ::=    CHOICE
{   functionalResourceName      [0]    FunctionalResourceName
,   procedureName               [1]    ProcedureName
}

FunctionalResourceName          ::=    SEQUENCE
{   functionalResourceType                 FunctionalResourceType
,   functionalResourceInstanceNumber       FunctionalResourceInstanceNumber
}

FunctionalResourceInstanceNumber    ::=    IntPos

FunctionalResourceType   ::=    PublishedIdentifier

IdentifierString      ::=    VisibleString (FROM (ALL EXCEPT " "))

-- 1 to (2^32)-1
IntPos                ::=    INTEGER (1 .. 4294967295)

-- 0 to (2^32)-1
IntUnsigned           ::=    INTEGER (0 .. 4294967295)

InvokeId              ::=    IntUnsigned

-- The Label structure is used to identify:
-- 1. the Label of a parameter
-- 2. the Label of an event
Label                 ::=    PublishedIdentifier

ListOfNamesDiagnosticExt     ::=    CHOICE
{   unknownNames                [0]    SEQUENCE OF UnknownName
,   unknownDefault              [1]    AdditionalText
,   diagnosticExtension         [100]  Embedded
}
```

```
-- The type ListOfParametersEvents is used by the service user to select
-- the parameters by means of the START of the Cyclic Report procedure and
-- by means of the GET operation. While the choices [0] to [5] relate to
-- Cross Support Resources, the choices [6] and [7] relate to the
-- configration parameters registered in the framework resources branch.
ListOfParametersEvents   ::=    CHOICE
{   empty                           [0]    NULL   -- signifying default list
,   listName                        [3]    VisibleString
,   functionalResourceType          [5]    FunctionalResourceType
,   functionalResourceName          [4]    FunctionalResourceName
,   procedureType                   [6]    ProcedureType
,   procedureName                   [7]    ProcedureName
,   paramEventLabels                [2]    SEQUENCE OF Label
,   paramEventNames                 [1]    SEQUENCE OF Name
}

ListOfParamEventsDiagnostics    ::=    CHOICE
{   undefinedDefault                [4]    AdditionalText
,   unknownListName                 [3]    VisibleString
,   unknownFunctionalResourceType   [1]    FunctionalResourceType
,   unknownFunctionalResourceName   [0]    FunctionalResourceName
,   unknownProcedureType            [5]    ProcedureType
,   unknownProcedureName            [6]    ProcedureName
,   unknownParamEventIdentifier     [2]    SEQUENCE OF CHOICE
    {   paramEventLabel                [0]    Label
    ,   paramEventName                 [1]    Name
    }
}

LogicalPortName      ::=    IdentifierString (SIZE (1 .. 128))

MaxFwdBufferSize     ::=    IntPos

-- The Name structure is used to identify:
-- 1. the Name of a parameter
-- 2. the Name of an event
-- 3. the Name of a directive
Name                      ::=    SEQUENCE
{   fRorProcedureName                FRorProcedureName
,   paramOrEventOrDirectiveId    PublishedIdentifier
}

PortId               ::=    LogicalPortName

ProcedureName        ::=    SEQUENCE
{   procedureType                ProcedureType
,   procedureRole                CHOICE
    {   primeProcedure           [0]    NULL
    ,   secondaryProcedure    [1]    IntPos
    ,   associationControl    [2]    NULL
    }
}
-- The ProcedureType is an Object Identifier, the allocation of which is
-- under control of CCSDS. It is declared in the ASN.1 module
-- CCSDS-CSTS-OBJECT-IDENTIFIERS (see F3.1).
ProcedureType        ::=    OBJECT IDENTIFIER

ProcessingLatencyLimit   ::=    IntUnsigned
```

```
ProductionStatus      ::=    ENUMERATED
{   configured     (0)
,   operational    (1)
,   interrupted    (2)
,   halted         (3)
}

-- A PublishedIdentifier is an identifier agreed to between the service
-- provider and the service user. The identifier is registered in SANA
-- (See D2)
PublishedIdentifier      ::=    OBJECT IDENTIFIER

QualifiedValue        ::=    CHOICE
{   valid                [0]    TypeAndValue -- Valid value
,   unavailable          [1]    NULL   -- Unknown or unavailable value
,   undefined            [2]    NULL   -- Undefined in the context
,   error                [3]    NULL   -- Processing resulted in an error
}


-- The definition of the parameters can be found in annex C.
QualifiedParameter    ::=    SEQUENCE
{   parameterName         Name
,   qualifiedValues       SequenceOfQualifiedValue
}


SequenceOfQualifiedValue ::=    SEQUENCE OF QualifiedValue

StandardAcknowledgeHeader    ::=    StandardReturnHeader

StandardInvocationHeader ::=    SEQUENCE
{   invokerCredentials       Credentials
,   invokeId                 InvokeId
,   procedureName            ProcedureName
}

StandardReturnHeader  ::=    SEQUENCE
{   performerCredentials        Credentials
,   invokeId                    InvokeId
,   result                      CHOICE
    {   positive             [0]    Extended   -- To carry the positive results
    ,   negative             [1]    SEQUENCE
        {   diagnostic                  Diagnostic
        ,   negExtension                Extended
                            -- The default value of the negExtension parameter is
                            -- 'notUsed'.
                            -- Unless a PDU that uses the StandardReturnHeader
                            -- explicitly defines an extension type to be used
                            -- as the value of negExtension for that PDU, the
                            -- value shall be 'notUsed'.
        }
    }
}
```

```
Time                    ::=    CHOICE
{   ccsdsFormatMilliseconds      [0]    TimeCCSDSMilli
,   ccsdsFormatPicoseconds       [1]    TimeCCSDSPico
}

TimeCCSDSMilli          ::=    OCTET STRING (SIZE(8))
-- P-field is implicit (not present, defaulted to 41 hex
-- T-field:
-- 2 octets: number of days since 1958/01/01 00:00:00;
-- 4 octets: number of milliseconds of the day;
-- 2 octets: number of microseconds of the millisecond (set to 0 if not used).
-- This definition reflects exactly the format of the CCSDS defined
-- time tag as used in spacelink data units (see reference [5]).


TimeCCSDSPico           ::=    OCTET STRING (SIZE(10))
-- P-field is implicit (not present, defaulted to 42 hex)
-- T-field:
-- 2 octets: number of days since 1958/01/01 00:00:00;
-- 4 octets: number of milliseconds of the day;
-- 4 octets: number of picoseconds of the millisecond (set to 0 if not used).
-- This definition reflects exactly the format of the CCSDS-defined
-- time tag as used in spacelink data units (see reference [5]).


TypeAndValue            ::=    Embedded

UnknownName             ::=    SEQUENCE
{   text                   AdditionalText
,   name                   Name
}

END
```

F3.4 COMMON OPERATIONS PDUS

```
CCSDS-CSTS-COMMON-OPERATIONS-PDUS
{   iso(1) identified-organization(3) standards-producing-organization(112)
    ccsds(4) css(4) csts(1) framework(1) modules(1) common-operations(4) version(2)
}

DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

EXPORTS    ExecuteDirectiveAcknowledge
,          ExecuteDirectiveInvocation
,          ExecuteDirectiveReturn
,          GetInvocation
,          GetReturn
,          NotifyInvocation
,          ProcessDataInvocation
,          ProcessDataReturn
,          StartInvocation
,          StartReturn
,          StopInvocation
,          StopReturn
,          TransferDataInvocation
;

IMPORTS    AbstractChoice
,          AdditionalText
,          DataUnitId
,          EventValue
,          Extended
,          Embedded
,          FunctionalResourceInstanceNumber
,          FunctionalResourceName
,          IntUnsigned
,          ListOfParametersEvents
,          ListOfParamEventsDiagnostics
,          Name
,          ProcedureName
,          PublishedIdentifier
,          QualifiedParameter
,          StandardAcknowledgeHeader
,          StandardInvocationHeader
,          StandardReturnHeader
,          Time
,          TypeAndValue
    FROM CCSDS-CSTS-COMMON-TYPES

           executeDirectiveAcknowledge
,          executeDirectiveReturn
,          getReturn
,          startReturn
    FROM CCSDS-CSTS-OBJECT-IDENTIFIERS
;

-- ============================================================
-- The first part of the module is left empty as there are no PDUs
-- defined in this module.
-- ============================================================
```

```
-- ============================================================
-- The second part of the module defines the common operations
-- the service provider may receive.
-- ============================================================

DirectiveQualifierValues     ::=    CHOICE
{   sequenceOfParamIdsAndValues  [0]    SequenceOfParameterIdsAndValues
,   parameterlessValues          [1]    TypeAndValue
,   noQualifierValues            [2]    NULL
}

ExecuteDirectiveInvocation   ::=    SEQUENCE
{   standardInvocationHeader     StandardInvocationHeader
,   directiveIdentifier          PublishedIdentifier
,   directiveQualifier           CHOICE
    {   localProcDirQualifier        [0]    DirectiveQualifierValues
    ,   serviceProcDirQualifier      [1]    SEQUENCE
        {   targetProcedureName                 ProcedureName
        ,   serviceProcDirQualifierValues       DirectiveQualifierValues
        }
    ,   functResourceDirQualifier[2]    SEQUENCE
        {   functResourceName                   FunctionalResourceName
        ,   functionalResourceQualifiers        DirectiveQualifierValues
        }
    ,   directiveQualifierExtension  [3]    Embedded
    }
,   executeDirectiveInvocationExtension     Extended
}

GetInvocation         ::=    SEQUENCE
{   standardInvocationHeader     StandardInvocationHeader
,   listOfParameters             ListOfParametersEvents   -- See 3.12.2.2.2
,   getInvocationExtension       Extended
}

ProcessDataInvocation     ::=    SEQUENCE
{   standardInvocationHeader        StandardInvocationHeader
,   dataUnitId                      DataUnitId
,   data                            AbstractChoice-- See 3.10.2.2.4.
,   processDataInvocationExtension  Extended
}

SequenceOfParameterIdsAndValues ::=    SEQUENCE OF SEQUENCE
{   parameterIdentifier      PublishedIdentifier
,   parameterValue           TypeAndValue
}

StartInvocation       ::=    SEQUENCE
{   standardInvocationHeader     StandardInvocationHeader
,   startInvocationExtension     Extended
}

StopInvocation        ::=    SEQUENCE
{   standardInvocationHeader     StandardInvocationHeader
,   stopInvocationExtension      Extended
}
```

```
-- ================================================================
-- The third part of the module defines the common operations
-- the service provider may send.
-- ================================================================

ExecuteDirectiveAcknowledge  ::=    StandardAcknowledgeHeader

ExecuteDirectiveReturn    ::=    StandardReturnHeader

GetReturn             ::=    StandardReturnHeader

-- The Published Identifier part of the eventName is defined by Functional Resource type -
- specific Object Identifier (see https://sanaregistry.org/r/functional_resources) or by
-- procedure type specific Published Identifiers(see F3.16).
NotifyInvocation      ::=    SEQUENCE
{   standardInvocationHeader     StandardInvocationHeader
,   eventTime                    Time
,   eventName                    Name
,   eventValue                   EventValue
,   notifyInvocationExtension    Extended
}

ProcessDataReturn     ::=    StandardReturnHeader

StartReturn           ::=    StandardReturnHeader

StopReturn            ::=    StandardReturnHeader

TransferDataInvocation    ::=    SEQUENCE
{   standardInvocationHeader            StandardInvocationHeader
,   generationTime                      Time
,   sequenceCounter                     IntUnsigned
,   data                                AbstractChoice
,   transferDataInvocationExtension     Extended
}

-- ================================================================
-- The fourth part of the module defines the extensions
-- of the common operations the service provider may send.
-- ================================================================

-- *****
-- EXECUTE-DIRECTIVE invocation
-- The EXECUTE-DIRECTIVE invocation is not extended, i.e.,
-- 'ExecuteDirectiveInvocation': 'executeDirectiveInvocationExtension' is
-- set to 'notUsed'.
```

```
-- EXECUTE-DIRECTIVE acknowledgement
-- The EXECUTE-DIRECTIVE positive acknowledgement does not extend
-- ExecuteDirectiveAcknowledge; that is, 'ExecuteDirectiveAcknowledge':
-- 'StandardAcknowledgeHeader': 'StandardReturnHeader': 'result':
-- 'positive' shall be set to 'notUsed'.
-- The EXECUTE-DIRECTIVE negative acknowledgement does not extend
-- ExecuteDirectiveAcknowledge; that is, 'ExecuteDirectiveAcknowledge':
-- 'StandardAcknowledgeHeader': 'StandardReturnHeader': 'result':
-- 'negative': 'negExtension'shall be set to 'notUsed'.
-- The EXECUTE DIRECTIVE negative acknowledgement makes use of: (a) one of
-- the common diagnostics of the StandardReturnHeader type (see diagnostic
-- parameter defined in 3.3.2.7) except 'diagnosticExtension'; or (b) one
-- of the diagnostic values defined by 'ExecuteDirectiveAcknowledge':
-- 'StandardAcknowledgeHeader': 'StandardReturnHeader': 'result':
-- 'negative': 'diagnostic': 'Diagnostic': 'diagnosticExtension':
-- 'execDirAckDiagExt': 'ExecDirNegAckDiagnosticExt' in F3.4 except
-- 'execDirNegAckDiagnosticExtExtension'.
ExecDirNegAckDiagnosticExt   ::=    CHOICE
{   unknownDirective                        [0]    NULL
,   unknownQualifier                        [1]    NULL
,   invalidProcedureName                    [2]    NULL
,   invalidFunctionalResourceName           [3]    NULL
,   invalidFunctionalResourceParameter      [4]    SET OF Name
,   invalidProcedureParameter           [5]    SET OF Name
,   parameterValueOutOfRange                [6]    SET OF Name
,   execDirNegAckDiagnosticExtExtension     [100]  Embedded
}

execDirAckDiagExt     OBJECT IDENTIFIER     ::=    {executeDirectiveAcknowledge 1}

-- EXECUTE-DIRECTIVE return
-- The EXECUTE-DIRECTIVE positive return does not extend
-- ExecuteDirectiveReturn; that is, 'ExecuteDirectiveReturn':
-- 'StandardReturnHeader': 'result': 'positive' shall be set to 'notUsed'.
-- The EXECUTE-DIRECTIVE negative return does not extend
-- ExecuteDirectiveReturn; that is, 'ExecuteDirectiveReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension' shall be
-- set to 'notUsed'.
-- The EXECUTE DIRECTIVE negative return makes use of: (a) one of the
-- common diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7 and F3.3) except
-- 'diagnosticExtension'; or (b) one of the diagnostic values defined by
-- 'ExecuteDirectiveReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic': 'diagnosticExtension':
-- 'execDirNegReturnDiagnosticExt': 'ExecDirNegReturnDiagnosticExt' in F3.4
-- except 'execDirNegReturnDiagnosticExtExtension'.
ExecDirNegReturnDiagnosticExt   ::=    CHOICE
{   actionNotCompleted                      [0]    ActionNotCompletedDiag
,   execDirNegReturnDiagnosticExtExtension     [100]  Embedded
}

execDirNegReturnDiagnosticExt   OBJECT IDENTIFIER     ::=
                                            {executeDirectiveReturn 1}

ActionNotCompletedDiag   ::=    CHOICE
{   parameterNames          [0]    SET OF Name
,   noParameterNames        [1]    NULL
}
```

```
-- *****
-- GET invocation
-- The GET invocation is not extended; that is, 'GetInvocation':
-- 'getInvocationExtension' shall be set to 'notUsed'.

-- GET Return
-- The GET positive return extends the GetReturn by adding the parameters
-- 'qualifiedParameters' and 'getPosReturnExtExtension' defined by
-- 'GetReturn': 'StandardReturnHeader': 'result': 'positive':
-- 'getPosReturnExt': 'GetPosReturnExt'. This extension only defines the
-- 'qualifiedParameters' parameter. 'getPosReturnExtExtension' shall be set
-- to 'notUsed'.
GetPosReturnExt      ::=    SEQUENCE
{   qualifiedParameters          QualifiedParametersSequence
,   getPosReturnExtExtension     Extended
}

getPosReturnExt   OBJECT IDENTIFIER    ::=    {getReturn 1}

-- The GET negative return does not extend GetReturn; that is, 'GetReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension' shall be
-- set to 'notUsed'.
-- The GET negative return makes use of: (a) one of the common diagnostics
-- of 'StandardReturnHeader': 'result': 'negative': 'diagnostic':
-- 'Diagnostic' (see diagnostic parameter defined in 3.3.2.7 and F3.3)
-- except 'diagnosticExtension'; or (b) one of the additional diagnostic
-- values defined by 'GetReturn': 'StandardReturnHeader': 'result':
-- 'negative': 'diagnostic': 'Diagnostic': 'diagnosticExtension':
-- 'getDiagnosticExt': 'GetDiagnosticExt' in F3.4 except
-- 'getDiagnosticExtExtension'.
GetDiagnosticExt     ::=    CHOICE
{   common                       [0]    ListOfParamEventsDiagnostics
,   getDiagnosticExtExtension[100]  Embedded
}

getDiagnosticExt  OBJECT IDENTIFIER    ::=    {getReturn 2}

QualifiedParametersSequence  ::=    SEQUENCE OF QualifiedParameter

-- *****
-- START invocation
-- The START invocation is not extended; that is, 'StartInvocation':
-- 'startInvocationExtension' shall be set to 'notUsed'.
```

```
-- START return
-- The START positive return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'positive' shall be set
-- to 'notUsed'.
-- The START negative return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'negExtension' shall be set to 'notUsed'.
-- The START negative return makes use of: (a) one of the common
-- diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7 and F3.3) except
-- 'diagnosticExtension'; or (b) one of the additional values specified by
-- 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic': 'diagnosticExtension': 'startDiagnosticExt':
-- 'StartDiagnosticExt' in F3.4 except 'startDiagnosticExtExtension'.
StartDiagnosticExt    ::=    CHOICE
{   unableToComply              [0]    AdditionalText
,   outOfService                [1]    AdditionalText
,   startDiagnosticExtExtension  [100]  Embedded
}

startDiagnosticExt    OBJECT IDENTIFIER    ::=    {startReturn 1}

-- *****
-- STOP invocation
-- The STOP invocation is not extended; that is, 'StopInvocation':
-- 'stopInvocationExtension' shall be set to 'notUsed'.

-- STOP return
-- The STOP positive return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'positive' shall be set to 'notUsed'.
-- The STOP negative return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension' shall be
-- set to 'notUsed'.
-- The STOP negative return makes use of one of the common diagnostics
-- of 'StandardReturnHeader': 'result': 'negative': 'diagnostic':
-- 'Diagnostic' (see 3.3.2.7 and F3.3) except 'diagnosticExtension'.

END
```

F3.5 PROCEDURE — ASSOCIATION CONTROL PDUS

```
CCSDS-CSTS-ASSOCIATION-CONTROL-TYPES
{   iso(1) identified-organization(3) standards-producing-organization(112)
    ccsds(4) css(4) csts(1) framework(1) modules(1)
    associationControlPdus(5) version(1)
}

DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

EXPORTS    AssociationPdu
,          BindInvocation
,          BindReturn
,          PeerAbortInvocation
,          UnbindInvocation
,          UnbindReturn
;

IMPORTS    AdditionalText
,          AuthorityIdentifier
,          Embedded
,          Extended
,          IntPos
,          PortId
,          PublishedIdentifier
,          StandardInvocationHeader
,          StandardReturnHeader
    FROM CCSDS-CSTS-COMMON-TYPES

           ServiceInstanceIdentifier
    FROM CCSDS-CSTS-SERVICE-INSTANCE-ID

           acExtProcedureParam
    FROM CCSDS-CSTS-OBJECT-IDENTIFIERS

           CstsFrameworkPdu
    FROM CCSDS-CSTS-PDUS
;

-- ================================================================
-- The first part of the module definition defines the PDU containing
-- the operations used by the Association Control procedure.
-- ================================================================
AssociationPdu       ::=    CstsFrameworkPdu (WITH COMPONENTS
    {   bindInvocation
    ,   bindReturn
    ,   unbindInvocation
    ,   unbindReturn
    ,   peerAbortInvocation
    }
)

-- ================================================================
-- The second part of the module defines the operations
-- of the Association Control procedure.
-- ================================================================
```

```
BindInvocation        ::=    SEQUENCE
{   standardInvocationHeader     StandardInvocationHeader
,   initiatorIdentifier          AuthorityIdentifier
,   responderPortIdentifier      PortId
,   serviceType                  ServiceType
,   versionNumber                VersionNumber
,   serviceInstanceIdentifier    ServiceInstanceIdentifier
,   bindInvocationExtension      Extended
}

BindReturn            ::=    SEQUENCE
{   standardReturnHeader         StandardReturnHeader
,   responderIdentifier          AuthorityIdentifier
}

PeerAbortInvocation   ::=    SEQUENCE
{   diagnostic                   PeerAbortDiagnostic
}

UnbindInvocation      ::=    SEQUENCE
{   standardInvocationHeader     StandardInvocationHeader
,   unbindInvocationExtension    Extended
}

UnbindReturn          ::=    StandardReturnHeader

-- ================================================================
-- The third part contains the types used by the operations
-- defined in the second part.
-- ================================================================
```

```
-- Peer Abort diagnostic values definition:
-- 0-39: SLE (usage: 0-8 and 127 for all SLE services)
-- 128-199: ISP
-- 200-250: Application
-- PEER-ABORT diagnostic definition:
-- The diagnostic of the PEER-ABORT only allows to carry one
-- octet of information (see ISP1, reference [2]).
-- The following ASN.1 definition is a dummy definition:
PeerAbortDiagnostic   ::=    OCTET STRING (SIZE(1))
-- The standard Association Control procedure as defined in this
-- Recommended Standard reserves the PEER-ABORT diagnostic values from 40
-- to 69 and 126. The values of the PEER-ABORT diagnostic defined (as
-- integers) by the Association Control procedure (see 3.6.2.2) are:
--     accessDenied                    (40)
--     unexpectedResponderId           (41)
--     operationalRequirement          (42)
--     protocolError                   (43)
--     communicationsFailure           (44)
--     encodingError                   (45)
--     responseTimeout                 (46)
--     endOfServiceProvisionPeriod     (47)
--     unsolicitedInvokeId             (48)
--     duplicateInvokeId               (49)
--     invalidProcedureName            (50)
--     unrecognizedType                (51)
--  otherReason                        (126)
-- As per 4.3.3.1.11.4, a procedure may trigger an abort of the association
-- and passing the desired diagnostic value. Values in the range from 70 to
-- 125 are reserved for this purpose.

-- The Buffered Data Processing procedure may use the following diagnostic:
--     forwardBufferTooLarge               (70)

-- Future versions of this Recommended Standard may specify additional
-- procedures and additional procedure-specific diagnostic values in the
-- range 71 to 125.

-- Service-type specific procedures (derived or service-original) may use
-- diagnostic values in the range reserved for applications 200 to 250.
-- Since the service type is known when a PEER-ABORT is received,
-- different service types can choose the diagnostic values independently
-- of each other.

-- The Service Type is an OID, the allocation of which is
-- under control of CCSDS. See the example in annex K.

ServiceType           ::=    PublishedIdentifier

VersionNumber         ::=    IntPos

-- ================================================================
-- The fourth part of the module definition contains the Extended
-- types used by the operations defined in the second part.
-- ================================================================

-- *****
-- BIND invocation
-- The BIND invocation is not extended; that is, 'BindInvocation':
-- 'bindInvocationExtension' shall be set to 'notUsed'.
```

```
-- BIND return
-- The BIND positive return does not extend BindReturn; that is, 'BindReturn':
-- 'StandardReturnHeader': 'result': 'positive' shall be set to 'notUsed'.
-- The BIND negative return does not extend BindReturn; that is, 'BindReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension' shall be
-- set to 'notUsed'.
-- The BIND negative return makes use of: (a) one of the common
-- diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7 and F3.3) except
-- 'diagnosticExtension'; or (b) one of the additional diagnostics
-- specified by 'BindReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic': 'diagnosticExtension': 'acBindDiagExt':
-- 'AssocBindDiagnosticExt' in F3.5 except
-- 'assocBindDiagnosticExtExtension'.
AssocBindDiagnosticExt   ::=    CHOICE
{   accessDenied                    [1]    AdditionalText
,   serviceTypeNotSupported         [2]    AdditionalText
,   versionNotSupported             [3]    AdditionalText
,   noSuchServiceInstance           [4]    AdditionalText
,   alreadyBound                    [5]    AdditionalText
,   siNotAccessibleToThisInitiator  [6]    AdditionalText
,   inconsistentServiceType         [7]    AdditionalText
,   outOfService                    [8]    AdditionalText
,   assocBindDiagnosticExtExtension [100]  Embedded
}

acBindDiagExt  OBJECT IDENTIFIER    ::=    {acExtProcedureParam 1}

-- *****
-- UNBIND invocation
-- The UNBIND invocation is not extended; that is, 'UnbindInvocation':
-- 'unbindInvocationExtension' shall be set to 'notUsed'.

-- UNBIND return
-- The UNBIND positive return does not extend UnbindReturn; that is,
-- 'UnbindReturn': 'StandardReturnHeader': 'result': 'positive' shall be
-- set to 'notUsed'.
-- The UNBIND negative return is not used; that is, 'UnbindReturn':
-- 'StandardReturnHeader': 'result' must not be set to 'negative'.

END
```

F3.6 PROCEDURE — UNBUFFERED DATA DELIVERY PDUS

```
CCSDS-CSTS-UNBUFFERED-DATA-DELIVERY-PDUS
{   iso(1) identified-organization(3) standards-producing-organization(112)
    ccsds(4) css(4) csts(1) framework(1) modules(1) unbuffDataDeliveryPdus(6)
    version(1)
}

DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

-- Main PDU exported to allow possible extension by derived procedures
EXPORTS    UnbufferedDataDeliveryPdu
;

IMPORTS    CstsFrameworkPdu
    FROM CCSDS-CSTS-PDUS
;

-- ================================================================
-- The first part of the module definition defines the PDU containing
-- the operations used by the Unbuffered Data Delivery procedure.
-- ================================================================
UnbufferedDataDeliveryPdu    ::=    CstsFrameworkPdu (WITH COMPONENTS
    {   startInvocation
    ,   startReturn
    ,   stopInvocation
    ,   stopReturn
    ,   transferDataInvocation
    }
)

-- ================================================================
-- The second part of the module defines the operations
-- of the Unbuffered Data Delivery procedure.
-- ================================================================
-- All operations are defined in the module
-- CCSDS-CSTS-COMMON-OPERATIONS-PDUS (see F3.4)

-- ================================================================
-- The third part contains the types used by the operations
-- defined in the second part.
-- ================================================================
-- This procedure does not have specific definitions.

-- ================================================================
-- The fourth part of the module definition contains the Extended
-- types used by the operations defined in the second part.
-- ================================================================

-- *****
-- START invocation
-- The START invocation is not extended; that is, 'StartInvocation':
-- 'startInvocationExtension' shall be set to 'notUsed'.
```

```
-- START return
-- The START positive return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'positive' shall be set
-- to 'notUsed'.
-- The START negative return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'negExtension' shall be set to 'notUsed'.
-- The START negative return makes use of: (a) one of the common
-- diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7 and F3.3) except
-- 'diagnosticExtension'; or (b) one of the additional values specified by
-- 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic': 'diagnosticExtension': 'startDiagnosticExt':
-- 'StartDiagnosticExt' in F3.6 except 'startDiagnosticExtExtension'.

-- *****
-- STOP Invocation
-- The STOP invocation is not extended; that is, 'StopInvocation':
-- 'stopInvocationExtension' shall be set to 'notUsed'.

-- STOP return
-- The STOP positive return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'positive' shall be set to 'notUsed'.
-- The STOP negative return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension' shall be
-- set to 'notUsed'.
-- The STOP negative return makes use of one of the common diagnostics
-- of 'StandardReturnHeader': 'result': 'negative': 'diagnostic':
-- 'Diagnostic' (see 3.3.2.7 and F3.3) except 'diagnosticExtension'.

-- TRANSFER-DATA Invocation
-- The TRANSFER-DATA invocation is not extended; that is,
-- 'TransferDataInvocation': 'transferDataInvocationExtension' shall be set
-- to 'notUsed'.

END
```

F3.7 PROCEDURE — BUFFERED DATA DELIVERY PDUS

```
CCSDS-CSTS-BUFFERED-DATA-DELIVERY-PDUS
{   iso(1) identified-organization(3) standards-producing-organization(112)
    ccsds(4) css(4) csts(1) framework(1) modules(1) buffDataDeliveryPdus(7)
    version(1)
}

DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

-- Main PDU exported to allow possible extension by derived procedures
EXPORTS    BufferedDataDeliveryPdu
,          ReturnBuffer
;

IMPORTS    AdditionalText
,          ConditionalTime
,          Embedded
,          Extended
    FROM CCSDS-CSTS-COMMON-TYPES

           bddExtProcedureParam
    FROM CCSDS-CSTS-OBJECT-IDENTIFIERS

           NotifyInvocation
,          TransferDataInvocation
    FROM CCSDS-CSTS-COMMON-OPERATIONS-PDUS

           CstsFrameworkPdu
    FROM CCSDS-CSTS-PDUS
;

-- ================================================================
-- The first part of the module definition defines the PDU containing
-- the operations used by the Buffered Data Delivery procedure.
-- ================================================================
BufferedDataDeliveryPdu      ::=    CstsFrameworkPdu (WITH COMPONENTS
    {   startInvocation
    ,   startReturn
    ,   stopInvocation
    ,   stopReturn
    ,   returnBuffer
    }
)

-- ================================================================
-- The second part of the module defines the operations
-- of the Buffered Data Delivery procedure.
-- ================================================================
-- All operations are defined in the module
-- CCSDS-CSTS-COMMON-OPERATIONS-PDUS (see F3.4)

-- The ReturnBuffer is not an operation but is the concatenation of
-- one or more TRANSFER-DATA and/or NOTIFY invocations.
ReturnBuffer          ::=    SEQUENCE OF TransferDataOrNotification
```

```
-- ================================================================
-- The third part contains the types used by the operations
-- defined in the second part.
-- ================================================================

TransferDataOrNotification   ::=    CHOICE
{   transferDataInvocation              [0]    TransferDataInvocation
,   notifyInvocation                    [1]    NotifyInvocation
}

-- ================================================================
-- The fourth part of the module definition contains the Extended
-- types used by the operations defined in the second part.
-- ================================================================

-- *****
-- START invocation
-- The START invocation is extended with the additional parameters
-- 'startGenerationTime' and 'stopGenerationTime'. This extension is
-- defined by 'StartInvocation': 'startInvocationExtension':
-- 'bddStartInvocExt': 'BuffDataDelStartInvocExt'. No further parameters
-- are added; that is, 'StartInvocation': 'startInvocationExtension':
-- 'bddStartInvocExt': 'BuffDataDelStartInvocExt':
-- 'buffDataDelStartInvocExtExtension' shall be set to 'notUsed'.
BuffDataDelStartInvocExt     ::=    SEQUENCE
{   startGenerationTime                 ConditionalTime
,   stopGenerationTime                  ConditionalTime
,   buffDataDelStartInvocExtExtension   Extended
}

bddStartInvocExt  OBJECT IDENTIFIER     ::=    {bddExtProcedureParam 1}

-- START return
-- The START positive return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'positive' shall be set
-- to 'notUsed'.
-- The START negative return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'negExtension' shall be set to 'notUsed'.
-- The START negative return makes use of: (a) one of the common
-- diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7 and F3.3) except
-- 'diagnosticExtension'; (b) one of the additional diagnostics defined
-- by 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic': 'diagnosticExtension': 'startDiagnosticExt':
-- 'StartDiagnosticExt' in F3.4 except 'startDiagnosticExtExtension'; or
-- (c) one of the additional values defined by 'StartReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'diagnostic':
-- 'Diagnostic': 'diagnosticExtension': 'startDiagnosticExt':
-- 'StartDiagnosticExt': 'startDiagnosticExtExtension': 'bddStartDiagExt':
-- 'BuffDataDelStartDiagnosticExt' in F3.7 except
-- 'buffDataDelStartDiagnosticExtExtension'.
BuffDataDelStartDiagnosticExt    ::=    CHOICE
{   missingTimeValue                            [1]    AdditionalText
,   invalidStartGenerationTime                  [2]    AdditionalText
,   invalidStopGenerationTime                   [3]    AdditionalText
,   inconsistentTime                            [4]    AdditionalText
,   buffDataDelStartDiagnosticExtExtension      [100]  Embedded
}
```

```
bddStartDiagExt   OBJECT IDENTIFIER    ::=    {bddExtProcedureParam 2}

-- *****
-- STOP Invocation
-- The STOP invocation is not extended; that is, 'StopInvocation':
-- 'stopInvocationExtension' shall be set to 'notUsed'.

-- STOP return
-- The STOP positive return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'positive' shall be set to 'notUsed'.
-- The STOP negative return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension' shall be
-- set to 'notUsed'.
-- The STOP negative return makes use of one of the common diagnostics
-- of 'StandardReturnHeader': 'result': 'negative': 'diagnostic':
-- 'Diagnostic' (see 3.3.2.7 and F3.3) except 'diagnosticExtension'.

-- NOTIFY:
-- The NotifyInvocation type is defined in F3.4.
-- NOTIFY invocation extension:
-- This procedure defines the additional eventName values 'data discarded due to
-- excessive backlog', 'bdd recording buffer overflow', 'end of data' and
-- 'buffered data delivery configuration change'(see
-- 4.5.4.2.2.1). The associated Published Identifiers are
-- pBDDdataDiscardedExcessBacklog, pBDDrecordingBufferOverflow,
-- pBDDendOfData, and pBDDconfigurationChange, as defined in F3.16).
-- No other extension is specified; that is, 'notifyInvocationExtension' shall
-- be set to 'notUsed'.

-- TRANSFER-DATA Invocation
-- The TRANSFER-DATA invocation is not extended; that is,
-- 'TransferDataInvocation': 'transferDataInvocationExtension' shall be set
-- to 'notUsed'.

END
```

F3.8 PROCEDURE — DATA PROCESSING PDUS

```
CCSDS-CSTS-DATA-PROCESSING-PDUS
{   iso(1) identified-organization(3) standards-producing-organization(112) ccsds(4)
    css(4) csts(1) framework(1) modules(1) dataProcessingPdus(8) version(1)
}

DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

-- Main PDU exported to allow possible extension by derived procedures
EXPORTS    DataProcessingPdu
;

IMPORTS    DataUnitId
,          Embedded
,          Extended
,          ProductionStatus
,          Time
    FROM CCSDS-CSTS-COMMON-TYPES

           dpExtProcedureParam
    FROM CCSDS-CSTS-OBJECT-IDENTIFIERS

           CstsFrameworkPdu
    FROM CCSDS-CSTS-PDUS
;

-- ================================================================
-- The first part of the module definition defines the PDU containing
-- the operations used by the Data Processing procedure.
-- ================================================================
DataProcessingPdu     ::=    CstsFrameworkPdu (WITH COMPONENTS
    {   startInvocation
    ,   startReturn
    ,   stopInvocation
    ,   stopReturn
    ,   processDataInvocation
    ,   notifyInvocation
    }
)

-- ================================================================
-- The second part of the module defines the operations
-- of the Data Processing procedure.
-- ================================================================
-- All operations are defined in the module
-- CCSDS-CSTS-COMMON-OPERATIONS-PDUS (see F3.4)

-- ================================================================
-- The third part contains the types used by the operations
-- defined in the second part.
-- ================================================================
-- This procedure does not have specific definition.
```

```
-- ================================================================
-- The fourth part of the module definition contains the Extended
-- types used by the operations defined in the second part.
-- ================================================================

-- *****
-- START Invocation
-- The START invocation is not extended; that is, 'StartInvocation':
-- 'startInvocationExtension' shall be set to 'notUsed'.

-- START return
-- The START positive return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'positive' shall be set
-- to 'notUsed'.
-- The START negative return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'negExtension' shall be set to 'notUsed'.
-- The START negative return makes use of: (a) one of the common
-- diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7 and F3.3) except
-- 'diagnosticExtension'; or (b) one of the additional values specified by
-- 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic': 'diagnosticExtension': 'startDiagnosticExt':
-- 'StartDiagnosticExt' in F3.4 except 'startDiagnosticExtExtension'.

-- *****
-- STOP Invocation
-- The STOP invocation is not extended; that is, 'StopInvocation':
-- 'stopInvocationExtension' shall be set to 'notUsed'.

-- STOP return
-- The STOP positive return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'positive' shall be set to 'notUsed'.
-- The STOP negative return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension' shall be
-- set to 'notUsed'.
-- The STOP negative return makes use of one of the common diagnostics
-- of 'StandardReturnHeader': 'result': 'negative': 'diagnostic':
-- 'Diagnostic' (see 3.3.2.7 and F3.3) except 'diagnosticExtension'.

-- *****
-- PROCESS-DATA Invocation
-- The PROCESS-DATA invocation is extended with the additional parameter
-- 'processCompletionReport'. This extension is defined by
-- 'ProcessDataInvocation': 'processDataInvocationExtension':
-- 'dpProcDataInvocExt': 'DataProcProcDataInvocExt':
-- 'processCompletionReport'. No further parameters are added; that is,
-- 'ProcessDataInvocation': 'processDataInvocationExtension':
-- 'dpProcDataInvocExt': 'DataProcProcDataInvocExt':
-- 'dataProcProcDataInvocExtExtension' shall be set to 'notUsed'.
DataProcProcDataInvocExt ::=    SEQUENCE
{   processCompletionReport         CHOICE
    {   doNotProduceReport          [0]    NULL
    ,   produceReport                   [1]    NULL
    }
,   dataProcProcDataInvocExtExtension   Extended
}

dpProcDataInvocExt    OBJECT IDENTIFIER    ::=    {dpExtProcedureParam 1}
```

```
-- *****
-- NOTIFY:
-- The NotifyInvocation type is defined in F3.4.
-- NOTIFY invocation extension:
-- This procedure defines the additional eventName values 'data processing
-- completed' and 'data processing configuration change'(see 4.6.4.2.3).
-- The associated Published Identifiers are pDPdataProcessingCompleted and
-- pDPconfigurationChange, as defined in F3.16).
-- The NOTIFY invocation is extended with the additional parameters
-- 'dataUnitIdLastProcessed', 'dataUnitIdLastOk', and 'productionStatus'.
-- This extension is defined by 'NotifyInvocation':
-- 'notifyInvocationExtension': 'dpNotifyInvocExt':
-- 'DataProcNotifyInvocExt'. No further parameters are added; that is,
-- 'NotifyInvocation': 'notifyInvocationExtension': 'dpNotifyInvocExt':
-- 'DataProcNotifyInvocExt': 'dataProcNotifyInvocExtExtension' shall be set
-- to 'notUsed'. No additional values of the 'dataProcessingSstatus'
-- parameter are defined; that is, 'NotifyInvocation':
-- 'notifyInvocationExtension': 'dpNotifyInvocExt':
-- 'DataProcNotifyInvocExt': 'dataUnitIdLastProcessed':
-- 'dataUnitLastProcessed': 'dataProcessingStatus' must not be set to
-- 'dataProcessingStatusExtension'.
DataProcNotifyInvocExt   ::=    SEQUENCE
{   dataUnitIdLastProcessed         CHOICE
    {   noDataProcessed                 [0]    NULL
    ,   dataUnitLastProcessed           [1]    SEQUENCE
        {   lastProcessedDataUnitId         DataUnitId
        ,   dataProcessingStatus            CHOICE
            {   successfullyProcessed           [0]    DataProcessingStartTime
            ,   processingInterrupted           [1]    DataProcessingStartTime
            ,   processingStarted               [2]    DataProcessingStartTime
            ,   dataProcessingStatusExtension   [100]  Embedded
            }
        }
    }
,   dataUnitIdLastOk                CHOICE
    {   noSuccessfulProcessing          [0]    NULL
    ,   dataUnitLastOk                  [1]    SEQUENCE
        {   lastOkdataUnitId                DataUnitId
        ,   dataProcessingStopTime          Time
        }
    }
,   productionStatus                CHOICE
    {   productionStatusChange          [0]    NULL
    ,   anyOtherEvent                   [1]    ProductionStatus
    }
,   dataProcNotifyInvocExtExtension     Extended
}

dpNotifyInvocExt  OBJECT IDENTIFIER    ::=    {dpExtProcedureParam 2}

-- The following type is not an extension but a type used by
-- DataProcNotifyInvocExt
DataProcessingStartTime  ::=    Time

END
```

F3.9 PROCEDURE — BUFFERED DATA PROCESSING PDUS

```
CCSDS-CSTS-BUFFERED-DATA-PROCESSING-PDUS
{   iso(1) identified-organization(3) standards-producing-organization(112) ccsds(4)
    css(4) csts(1) framework(1) modules(1) bufferedDataProcessingPdus(9) version(2)
}

DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

-- Main PDU exported to allow possible extension by derived procedures
EXPORTS    BufferedDataProcessingPdu
,          ForwardBuffer
;

IMPORTS    DataProcessingPdu
    FROM CCSDS-CSTS-DATA-PROCESSING-PDUS

           ProcessDataInvocation
    FROM CCSDS-CSTS-COMMON-OPERATIONS-PDUS;

-- ================================================================
-- The first part of the module definition defines the PDU containing
-- the operations used by the Buffered Data Processing procedure.
-- ================================================================
-- The Buffered Data Processing procedure is derived from the
-- Data Processing procedure. Its PDU is cast as the type of the PDU
-- defined in the Data Processing procedure: DataProcessingPdu type defined
-- in F3.8.
BufferedDataProcessingPdu    ::=    DataProcessingPdu

-- The Buffered Data Processing Procedure uses an additional diagnostic
-- passed to the Association Contol Procedure in case a too large Transfer
-- Buffer PDU is receceived. The diagnostic value 70 shall signify
-- 'forward buffer too large'.
-- ================================================================
-- The second part of the module defines the operations
-- of the Data Processing procedure.
-- ================================================================
-- All operations are defined in the module
-- CCSDS-CSTS-COMMON-OPERATIONS-PDUS (see F3.4)

-- ================================================================
-- The third part contains the types used by the operations
-- defined in the second part.
-- ================================================================

ForwardBuffer        ::=    SEQUENCE OF ProcessDataInvocation

-- ================================================================
-- The fourth part of the module definition contains the Extended
-- types used by the operations defined in the second part.
-- ================================================================

-- *****
-- START invocation
-- The START invocation is not extended; that is, 'StartInvocation':
-- 'startInvocationExtension' shall be set to 'notUsed'.
```

```
-- START return
-- The START positive return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'positive' shall be set
-- to 'notUsed'.
-- The START negative return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'negExtension' shall be set to 'notUsed'.
-- The START negative return makes use of: (a) one of the common
-- diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7 and F3.3) except
-- 'diagnosticExtension'; or (b) one of the additional values specified by
-- 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic': 'diagnosticExtension': 'startDiagnosticExt':
-- 'StartDiagnosticExt' in F3.4 except 'startDiagnosticExtExtension'.

-- *****
-- STOP invocation
-- The STOP invocation is not extended; that is, 'StopInvocation':
-- 'stopInvocationExtension' shall be set to 'notUsed'.

-- STOP return
-- The STOP positive return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'positive' shall be set to 'notUsed'.
-- The STOP negative return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension' shall be
-- set to 'notUsed'.
-- The STOP negative return makes use of one of the common diagnostics
-- of 'StandardReturnHeader': 'result': 'negative': 'diagnostic':
-- 'Diagnostic' (see 3.3.2.7 and F3.3) except 'diagnosticExtension'.

-- *****
-- PROCESS-DATA Invocation
-- The PROCESS-DATA invocation is extended with the additional parameter
-- 'processCompletionReport'. This extension is defined by
-- 'ProcessDataInvocation': 'processDataInvocationExtension':
-- 'dpProcDataInvocExt': 'DataProcProcDataInvocExt':
-- 'processCompletionReport' inherited from the parent Data Processing
-- procedure. No further extension is defined; that is,
-- 'ProcessDataInvocation': 'processDataInvocationExtension':
-- 'dpProcDataInvocExt': 'DataProcProcDataInvocExt':
-- 'dataProcProcDataInvocExtExtension' shall be set to 'notUsed'.

-- *****
-- NOTIFY Invocation
-- The extension of the NOTIFY invocation is inherited from the Data
-- Processing procedure. The extension is defined in F3.8.

END
```

F3.10 PROCEDURE — SEQUENCE-CONTROLLED DATA PROCESSING PDUS

```
CCSDS-CSTS-SEQUENCE-CONTROLLED-DATA-PROCESSING-PDUS
{   iso(1) identified-organization(3) standards-producing-organization(112) ccsds(4)
    css(4) csts(1) framework(1) modules(1) sequenceControlledDataProcessingPdus(10)
    version(1)
}

DEFINITIONS
IMPLICIT TAGS
::=     BEGIN

-- Main PDU exported to allow possible extension by derived procedures
EXPORTS     SequContrDataProcessingPdu
;

IMPORTS     ConditionalTime
,           DataUnitId
,           Embedded
,           Extended
    FROM CCSDS-CSTS-COMMON-TYPES

            scdpExtProcedureParam
    FROM CCSDS-CSTS-OBJECT-IDENTIFIERS

            CstsFrameworkPdu
    FROM CCSDS-CSTS-PDUS
;

-- ================================================================
-- The first part of the module definition defines the PDU containing
-- the operations used by the Sequence-Controlled Data Processing
-- procedure.
-- ================================================================
SequContrDataProcessingPdu   ::=    CstsFrameworkPdu (WITH COMPONENTS
    {   startInvocation
    ,   startReturn
    ,   stopInvocation
    ,   stopReturn
    ,   processDataInvocation
    ,   processDataReturn
    ,   notifyInvocation
    ,   executeDirectiveInvocation
    ,   executeDirectiveAcknowledge
    ,   executeDirectiveReturn
    }
)

-- ================================================================
-- The second part of the module defines the operations
-- of the Data Processing procedure.
-- ================================================================
-- All operations are defined in the module
-- CCSDS-CSTS-COMMON-OPERATIONS-PDUS (see F3.4).
```

```
-- ================================================================
-- The third part contains the types used by the operations
-- defined in the second part.
-- ================================================================
-- This procedure does not have specific definitions.

-- ================================================================
-- The fourth part of the module definition contains the Extended
-- types used by the operations defined in the second part.
-- ================================================================

-- *****
-- START Invocation
-- The START invocation is extended with the additional parameter
-- 'firstDataUnitId'. This extension is defined by 'StartInvocation':
-- 'startInvocationExtension': 'scdpStartInvocExt':
-- 'SequContrDataProcStartInvocExt'. No further parameters are added; that is,
-- 'StartInvocation': 'startInvocationExtension': 'scdpStartInvocExt':
-- 'SequContrDataProcStartInvocExt':
-- 'sequContrDataProcStartInvocExtExtension' shall be set to 'notUsed'.
SequContrDataProcStartInvocExt   ::=    SEQUENCE
{   firstDataUnitId                             DataUnitId
,   sequContrDataProcStartInvocExtExtension     Extended
}

scdpStartInvocExt     OBJECT IDENTIFIER     ::=    {scdpExtProcedureParam 1}

-- START return
-- The START positive return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'positive' shall be set
-- to 'notUsed'.
-- The START negative return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'negExtension' shall be set to 'notUsed'.
-- The START negative return makes use of: (a) one of the common
-- diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7 and F3.3) except
-- 'diagnosticExtension'; or (b) one of the additional values specified by
-- 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic': 'diagnosticExtension': 'startDiagnosticExt':
-- 'StartDiagnosticExt' in F3.4 except 'startDiagnosticExtExtension'.

-- *****
-- STOP Invocation
-- The STOP invocation is not extended; that is, 'StopInvocation':
-- 'stopInvocationExtension' shall be set to 'notUsed'.

-- STOP return
-- The STOP positive return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'positive' shall be set to 'notUsed'.
-- The STOP negative return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension' shall be
-- set to 'notUsed'.
-- The STOP negative return makes use of one of the common diagnostics
-- of 'StandardReturnHeader': 'result': 'negative': 'diagnostic':
-- 'Diagnostic' (see 3.3.2.7 and F3.3) except 'diagnosticExtension'.
```

```
-- *****
-- PROCESS-DATA Invocation
-- The PROCESS-DATA invocation is extended with the additional parameter
-- 'processCompletionReport' inherited from the parent Data Processing
-- procedure. This extension is defined in F3.8. The PROCESS-DATA
-- invocation is further extended with the parameters
-- 'earliestDataProcessingTime' and 'latestDataProcessingTime' defined by
-- 'ProcessDataInvocation': 'processDataInvocationExtension':
-- 'dpProcDataInvocExt': 'DataProcProcDataInvocExt':
-- 'dataProcProcDataInvocExtExtension': 'scdpProcDataInvocExt':
-- 'SequContrDataProcProcDataInvocExt'. No further parameters are added to
-- the PROCESS-DATA invocation; that is, 'ProcessDataInvocation':
-- 'processDataInvocationExtension': 'dpProcDataInvocExt':
-- 'DataProcProcDataInvocExt': 'dataProcProcDataInvocExtExtension':
-- 'scdpProcDataInvocExt': 'SequContrDataProcProcDataInvocExt':
-- 'sequContrDataProcDataInvocExtExtension' shall be set to 'notUsed'.
SequContrDataProcProcDataInvocExt   ::=    SEQUENCE
{   earliestDataProcessingTime                ConditionalTime
,   latestDataProcessingTime                  ConditionalTime
,   sequContrDataProcDataInvocExtExtension    Extended
}

scdpProcDataInvocExt     OBJECT IDENTIFIER    ::=    {scdpExtProcedureParam 2}

-- *****
-- PROCESS-DATA positive return
-- The PROCESS-DATA positive return extends ProcessDataReturn defined below
-- by adding the parameter 'dataUnitId'. This extension is
-- defined by 'ProcessDataReturn': 'StandardReturnHeader': 'result':
-- 'positive': 'scdpProcDataPosReturnExt':
-- 'SequContrDataProcProcDataPosReturnExt'. No further parameters are added
-- to the PDU; that is, 'ProcessDataReturn': 'StandardReturnHeader': 'result':
-- 'positive': 'scdpProcDataPosReturnExt':
-- 'SequContrDataProcProcDataPosReturnExt':
-- 'sequContrDataProcProcDataPosReturnExtExtension' shall be set to
-- 'notUsed'.

SequContrDataProcProcDataPosReturnExt   ::=    SEQUENCE
{   dataUnitIdPosRtn                                    DataUnitId
,   sequContrDataProcProcDataPosReturnExtExtension    Extended
}

scdpProcDataPosReturnExt OBJECT IDENTIFIER    ::=    {scdpExtProcedureParam 3}

-- The PROCESS-DATA negative return extends ProcessDataReturn by adding the
-- parameter 'dataUnitId'. This extension is defined by
-- 'ProcessDataReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'negExtension': 'scdpProcDataNegReturnExt':
-- 'SequContrDataProcProcDataNegReturnExt'. No further parameters are added
-- to the ProcessDataReturn; that is, 'ProcessDataReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension':
-- 'scdpProcDataNegReturnExt': 'SequContrDataProcProcDataNegReturnExt':
-- 'sequContrDataProcProcDataNegReturnExtExtension'shall be set to
-- 'notUsed'.
SequContrDataProcProcDataNegReturnExt   ::=    SEQUENCE
{   dataUnitIdNegRtn                                    DataUnitId
,   sequContrDataProcProcDataNegReturnExtExtension    Extended
}
```

```
scdpProcDataNegReturnExt OBJECT IDENTIFIER    ::=    {scdpExtProcedureParam 5}

-- The PROCESS-DATA negative return makes use of: (a) one of the common
-- diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7 and F3.3) except
-- 'diagnosticExtension'; or (b) one of the additional diagnostics defined
-- by 'ProcessDataReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic': 'diagnosticExtension':
-- 'scdpProcDataDiagExt': 'SequContrDataProcProcDataDiagnosticExt' in F3.10
-- except 'sequContrDataProcProcDataDiagnosticExtExtension'.
SequContrDataProcProcDataDiagnosticExt    ::=    CHOICE
{   unableToProcess                                 [0]      NULL
,   serviceInstanceLocked                           [1]      NULL
,   outOfSequence                                   [2]      NULL
,   inconsistentTimeRange                           [3]      NULL
,   invalidTime                                     [4]      NULL
,   lateData                                        [5]      NULL
,   dataError                                       [6]      NULL
,   unableToStore                                   [7]      NULL
,   sequContrDataProcProcDataDiagnosticExtExtension [100]    Embedded
}

scdpProcDataDiagExt   OBJECT IDENTIFIER    ::=    {scdpExtProcedureParam 4}

-- *****
-- NOTIFY:
-- The NotifyInvocation type is defined in F3.4.
-- NOTIFY invocation extension:
-- This procedure defines the additional eventName values 'expired' and
-- 'locked' (see 4.8.4.3.4). The associated Published Identifiers are
-- pSCDPexpired and pSCDPlocked as defined in F3.16.

-- Further NOTIFY invocation extensions are inherited from the parent Data
-- Processing procedure. These extensions are defined in F3.8. Additional
-- values of the processing-status parameter are introduced by means of the
-- extension 'NotifyInvocation': 'notifyInvocationExtension':
-- 'dpNotifyInvocExt': 'DataProcNotifyInvocExt': 'dataUnitIdLastProcessed':
-- 'dataUnitLastProcessed': 'dataProcessingStatus':
-- 'dataProcessingStatusExtension': 'scdpNotifyProcStatusExt':
-- 'SequContrDataProcStatus'. No further processing-status values are
-- added; that is, 'NotifyInvocation': 'notifyInvocationExtension':
-- 'dpNotifyInvocExt': 'DataProcNotifyInvocExt': 'dataUnitIdLastProcessed':
-- 'dataUnitLastProcessed': 'dataProcessingStatus':
-- 'dataProcessingStatusExtension': 'scdpNotifyProcStatusExt':
-- 'SequContrDataProcStatus' must not be set to
-- 'sequContrDataProcStatusExtension'.
SequContrDataProcStatus  ::=    CHOICE
{   expired                             [0]    NULL
,   processingNotStarted                [1]    NULL
,   sequContrDataProcStatusExtension    [100]  Embedded
}

scdpNotifyProcStatusExt  OBJECT IDENTIFIER    ::=    {scdpExtProcedureParam 6}
```

```
-- *****
-- EXECUTE-DIRECTIVE invocation
-- In the EXECUTE-DIRECTIVE invocation the parameter directive-identifier
-- shall be set to 'reset'; that is, 'ExecuteDirectiveInvocation':
-- 'directiveIdentifier' shall be set to the Published Identifier
-- {pSCDPdirectivesId 1}.
-- The directive-qualifier parameter is defined by
-- 'ExecuteDirectiveInvocation': 'directiveQualifier':
-- 'localProcDirQualifier': 'DirectiveQualifierValues':
-- 'parameterlessValues':
-- 'TypeAndValue': 'Embedded': 'EMBEDDED PDV'. OID and type of this parameter are
-- pSCDPdataUnitId and PSCDPdataUnitIdType, respectively (see F3.16).
-- The 'directiveQualifier' parameter is not extended and must therefore not be set to
-- 'directiveQualifierExtension'.

-- The EXECUTE-DIRECTIVE invocation is not extended; that is,
-- 'ExecuteDirectiveInvocation': 'execute DirectiveInvocationExtension'
-- shall be set to 'notUsed'.

-- EXECUTE-DIRECTIVE acknowledgement
-- The EXECUTE-DIRECTIVE positive acknowledgement is not extended; that is,
-- 'ExecuteDirectiveAcknowledge': 'StandardAcknowledgeHeader':
-- 'StandardReturnHeader': 'result': 'positive' shall be set to 'notUsed'.
-- The EXECUTE-DIRECTIVE negative acknowledgement is not extended; that is,
-- 'ExecuteDirectiveAcknowledge': 'StandardAcknowledgeHeader':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension' shall be
-- set to 'notUsed'.
-- The EXECUTE-DIRECTIVE negative acknowledgement makes use of: (a) one of
-- the common diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7); or (b) one of the additional
-- diagnostics defined by 'ExecuteDirectiveAcknowledge':
-- 'StandardAcknowledgeHeader': 'StandardReturnHeader': 'result':
-- 'negative': 'diagnostic': 'Diagnostic': 'diagnosticExtension':
-- 'execDirAckDiagExt': 'ExecDirNegAckDiagnosticExt'. No further
-- diagnostics are specified; that is, 'ExecuteDirectiveAcknowledge':
-- 'StandardAcknowledgeHeader': 'StandardReturnHeader': 'result':
-- 'negative': 'diagnostic': 'Diagnostic': 'diagnosticExtension':
-- 'execDirAckDiagExt': 'ExecDirNegAckDiagnosticExt' must not be set to
-- 'execDirNegAckDiagnosticExtExtension'.

-- EXECUTE-DIRECTIVE return
-- The EXECUTE-DIRECTIVE positive return is not extended; that is,
-- 'ExecuteDirectiveReturn': 'StandardReturnHeader': 'result': 'positive'
-- shall be set to 'notUsed'.
-- The EXECUTE-DIRECTIVE negative return is not extended; that is,
-- 'ExecuteDirectiveReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'negExtension' shall be set to 'notUsed'.
-- The EXECUTE-DIRECTIVE negative return makes use of: (a) one of the
-- common diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7 and F3.3) except
-- 'diagnosticExtension'; or (b) one of the additional diagnostics defined
-- by 'ExecuteDirectiveReturn': 'StandardReturnHeader': 'result':
-- 'negative': 'diagnostic': 'Diagnostic': 'diagnosticExtension':
-- 'execDirNegReturnDiagnosticExt': 'ExecDirNegReturnDiagnosticExt'in F3.4
-- except 'execDirNegReturnDiagnosticExtExtension'.

END
```

F3.11 PROCEDURE — INFORMATION QUERY PDUS

```
CCSDS-CSTS-INFORMATION-QUERY-PDUS
{   iso(1) identified-organization(3) standards-producing-organization(112)
    ccsds(4) css(4) csts(1) framework(1) modules(1) informationQueryPdus(11)
    version(1)
}

DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

-- Main PDU exported to allow possible extension by derived procedures
EXPORTS    InformationQueryPdu
;

IMPORTS    CstsFrameworkPdu
    FROM CCSDS-CSTS-PDUS
;

-- ================================================================
-- The first part of the module definition defines the PDU containing
-- the operations used by the Information Query procedure.
-- ================================================================
InformationQueryPdu   ::=    CstsFrameworkPdu (WITH COMPONENTS
    {   getInvocation
    ,   getReturn
    }
)

-- ================================================================
-- The second part of the module defines the operations
-- of the Information Query procedure.
-- ================================================================
-- All operations are defined in the module
-- CCSDS-CSTS-COMMON-OPERATIONS-PDUS (see F3.4)

-- ================================================================
-- The third part contains the types used by the operations
-- defined in the second part.
-- ================================================================
-- This procedure does not have specific definitions.

-- ================================================================
-- The fourth part of the module definition contains the Extended
-- types used by the operations defined in the second part.
-- ================================================================
-- This procedure does not have specific extensions.

END
```

F3.12 PROCEDURE — CYCLIC REPORT PDUS

```
CCSDS-CSTS-CYCLIC-REPORT-PDUS
{   iso(1) identified-organization(3) standards-producing-organization(112)
    ccsds(4) css(4) csts(1) framework(1) modules(1) cyclicReportPdus(12) version(2)
}

DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

-- Main PDU exported to allow possible extension by derived procedures
EXPORTS    CyclicReportPdu
;

IMPORTS    AdditionalText
,          Embedded
,          Extended
,          IntPos
,          ListOfParametersEvents
,          ListOfParamEventsDiagnostics
,          QualifiedParameter
    FROM CCSDS-CSTS-COMMON-TYPES

           UnbufferedDataDeliveryPdu
    FROM CCSDS-CSTS-UNBUFFERED-DATA-DELIVERY-PDUS

           crExtProcedureParam
    FROM CCSDS-CSTS-OBJECT-IDENTIFIERS
;

-- ================================================================
-- The first part of the module definition defines the PDU containing
-- the operations used by the Cyclic Report procedure.
-- ================================================================
-- The Cyclic Report procedure is derived from the Unbuffered Data
-- Delivery procedure. Its PDU is cast as the type of the PDU defined in
-- the Unbuffered Data Delivery procedure: UnbufferedDataDeliveryPdu type
-- defined in F3.6.

CyclicReportPdu       ::=    UnbufferedDataDeliveryPdu

-- ================================================================
-- The second part of the module defines the operations
-- of the Cyclic Report procedure.
-- ================================================================
-- All operations are defined in the module
-- CCSDS-CSTS-COMMON-OPERATIONS-PDUS (see F3.4)

-- ================================================================
-- The third part contains the types used by the operations
-- defined in the second part.
-- ================================================================
-- This procedure does not have specific definitions.

-- ================================================================
-- The fourth part of the module definition contains the Extended
-- types used by the operations defined in the second part.
-- ================================================================
```

```
-- *****
-- START Invocation
-- The START invocation is extended with the additional parameters
-- 'deliveryCycle' and 'listOfParameters'. This extension is defined by
-- 'StartInvocation': 'startInvocationExtension': 'crStartInvocExt':
-- 'CyclicReportStartInvocExt'. No further parameters are added to
-- StartInvocation; that is, 'StartInvocation': 'startInvocationExtension':
-- 'crStartInvocExt': 'CyclicReportStartInvocExt':
-- 'cyclicReportStartInvocExtExtension' shall be set to 'notUsed'.
CyclicReportStartInvocExt    ::=    SEQUENCE
{   deliveryCycle                       IntPos
,   listOfParameters                    ListOfParametersEvents
,   cyclicReportStartInvocExtExtension  Extended
}

crStartInvocExt   OBJECT IDENTIFIER    ::=    {crExtProcedureParam 1}

-- START return
-- The START positive return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'positive' shall be set
-- to 'notUsed'.
-- The START negative return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'negExtension' shall be set to 'notUsed'.
-- The START negative return makes use of: (a) one of the common
-- diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7 and F3.3) except
-- 'diagnosticExtension'; (b) one of the additional
-- diagnostics defined by 'StartReturn': 'StandardReturnHeader': 'result':
-- 'negative': 'diagnostic': 'Diagnostic': 'diagnosticExtension':
-- 'startDiagnosticExt': 'StartDiagnosticExt' in F3.4 except
-- 'startDiagnosticExtExtension'; or (c) one of the additional
-- diagnostics defined by 'StartReturn': 'StandardReturnHeader': 'result':
-- 'negative': 'diagnostic': 'Diagnostic': 'diagnosticExtension':
-- 'startDiagnosticExt': 'StartDiagnosticExt':
-- 'startDiagnosticExtExtension': 'crStartDiagExt':
-- 'CyclicReportStartDiagnosticExt' in F3.12 except
-- 'cyclicReportStartDiagnosticExtExtension'.
CyclicReportStartDiagnosticExt  ::=    CHOICE
{   common                                      [0]    ListOfParamEventsDiagnostics
,   outOfRange                                  [1]    AdditionalText
,   cyclicReportStartDiagnosticExtExtension     [100]  Embedded
}

crStartDiagExt    OBJECT IDENTIFIER    ::=    {crExtProcedureParam 2}

-- *****
-- STOP Invocation
-- The STOP invocation is not extended; that is, 'StopInvocation':
-- 'stopInvocationExtension' shall be set to 'notUsed'.
```

```
-- STOP return
-- The STOP positive return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'positive' shall be set to 'notUsed'.
-- The STOP negative return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension' shall be
-- set to 'notUsed'.
-- The STOP negative return makes use of one of the common diagnostics
-- of 'StandardReturnHeader': 'result': 'negative': 'diagnostic':
-- 'Diagnostic' (see 3.3.2.7 and F3.3) except 'diagnosticExtension'.

-- *****
-- TRANSFER-DATA invocation
-- The TRANSFER-DATA invocation is not extended; that is,
-- 'TransferDataInvocation': 'transferDataInvocationExtension' shall be set
-- to 'notUsed'.
-- The TRANSFER-DATA invocation parameter data is refined to carry
-- the list-of-parameters-values information and is therefore defined as
-- follows: 'TransferDataInvocation': 'data': 'AbstractChoice':
-- 'extendedData': 'crTransferDataInvocDataRef':
-- 'CyclicReportTransferDataInvocDataRef'. 'TransferDataInvocation':
-- 'data': 'AbstractChoice': 'extendedData': 'crTransferDataInvocDataRef':
-- 'CyclicReportTransferDataInvocDataRef':
-- 'cyclicReportTransferDataInvocDataRefExtension' shall be set to
-- 'notUsed'.
CyclicReportTransferDataInvocDataRef    ::=    SEQUENCE
{   qualifiedParameters                              SEQUENCE OF QualifiedParameter
,   cyclicReportTransferDataInvocDataRefExtension    Extended
}

crTransferDataInvocDataRef   OBJECT IDENTIFIER    ::=    {crExtProcedureParam 3}

END
```

F3.13 PROCEDURE — NOTIFICATION PDUS

```
CCSDS-CSTS-NOTIFICATION-PDUS
{   iso(1) identified-organization(3) standards-producing-organization(112)
    ccsds(4) css(4) csts(1) framework(1) modules(1) notificationPdus(13)
    version(1)
}

DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

-- Main PDU exported to allow possible extension by derived procedures
EXPORTS    NotificationPdu
;

IMPORTS    Embedded
,          Extended
,          ListOfParametersEvents
,          ListOfParamEventsDiagnostics
    FROM CCSDS-CSTS-COMMON-TYPES

           nExtProcedureParam
    FROM CCSDS-CSTS-OBJECT-IDENTIFIERS
```

```
            CstsFrameworkPdu
    FROM CCSDS-CSTS-PDUS
;

-- ================================================================
-- The first part of the module definition defines the PDU containing
-- the operations used by the Notification procedure.
-- ================================================================
NotificationPdu       ::=    CstsFrameworkPdu (WITH COMPONENTS
    {   startInvocation
    ,   startReturn
    ,   stopInvocation
    ,   stopReturn
    ,   notifyInvocation
    }
)

-- ================================================================
-- The second part of the module defines the operations
-- of the Notification procedure.
-- ================================================================
-- All operations are defined in the module
-- CCSDS-CSTS-COMMON-OPERATIONS-PDUS (see F3.4)

-- ================================================================
-- The third part contains the types used by the operations
-- defined in the second part.
-- ================================================================
-- This procedure does not have specific definitions.

-- ================================================================
-- The fourth part of the module definition contains the Extended
-- types used by the operations defined in the second part.
-- ================================================================

-- *****
-- START Invocation
-- The START invocation is extended with the additional parameter
-- 'listOfEvents' specifying the events that shall be notified. This
-- extension is defined by 'StartInvocation': 'startInvocationExtension':
-- 'nStartInvocExt': 'NotificationStartInvocExt'. No further parameters are
-- added to StartInvocation; that is, 'StartInvocation':
-- 'startInvocationExtension': 'nStartInvocExt':
-- 'NotificationStartInvocExt': 'notificationStartInvocExtExtension' shall
-- be set to 'notUsed'.
NotificationStartInvocExt    ::=    SEQUENCE
{   listOfEvents                        ListOfParametersEvents
,   notificationStartInvocExtExtension  Extended
}

nStartInvocExt    OBJECT IDENTIFIER    ::=    {nExtProcedureParam 1}
```

```
-- START return
-- The START positive return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'positive' shall be set
-- to 'notUsed'.
-- The START negative return does not extend StartReturn; that is,
-- 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'negExtension' shall be set to 'notUsed'.
-- The START negative return makes use of: (a) one of the common
-- diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7 and F3.3) except
-- 'diagnosticExtension'; (b) one of the additional diagnostics defined
-- by 'StartReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic': 'diagnosticExtension': 'startDiagnosticExt':
-- 'StartDiagnosticExt' in F3.4 except 'startDiagnosticExtExtension'; or
-- (c) one of the additional diagnostics defined by 'StartReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'diagnostic':
-- 'Diagnostic': 'diagnosticExtension': 'startDiagnosticExt':
-- 'StartDiagnosticExt': 'startDiagnosticExtExtension': 'nStartDiagExt':
-- 'NotificationStartDiagnosticExt' in F3.13 except
-- 'notificationStartDiagnosticExtExtension'.
NotificationStartDiagnosticExt  ::=    CHOICE
{   common                                  [0]    ListOfParamEventsDiagnostics
,   notificationStartDiagnosticExtExtension    [100]  Embedded
}

nStartDiagExt     OBJECT IDENTIFIER     ::=    {nExtProcedureParam 2}

-- *****
-- STOP Invocation
-- The STOP invocation is not extended; that is, 'StopInvocation':
-- 'stopInvocationExtension' shall be set to 'notUsed'.

-- STOP return
-- The STOP positive return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'positive' shall be set to 'notUsed'.
-- The STOP negative return does not extend StopReturn; that is, 'StopReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension' shall be
-- set to 'notUsed'.
-- The STOP negative return makes use of one of the common diagnostics
-- of 'StandardReturnHeader': 'result': 'negative': 'diagnostic':
-- 'Diagnostic' (see 3.3.2.7 and F3.3) except 'diagnosticExtension'.

END
```

F3.14 PROCEDURE — THROW EVENTS PDUS

```
CCSDS-CSTS-THROW-EVENT-PDUS
{   iso(1) identified-organization(3) standards-producing-organization(112) ccsds(4)
    css(4) csts(1) framework(1) modules(1) throwEventPdus(14) version(1)
}

DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

-- Main PDU exported to allow possible extension by derived procedures
EXPORTS    ThrowEventPdu
;

IMPORTS    Embedded
    FROM CCSDS-CSTS-COMMON-TYPES

           teExtProcedureParam
    FROM CCSDS-CSTS-OBJECT-IDENTIFIERS

           CstsFrameworkPdu
    FROM CCSDS-CSTS-PDUS
;

-- ================================================================
-- The first part of the module definition defines the PDU containing
-- the operations used by the Throw Event procedure.
-- ================================================================
ThrowEventPdu         ::=    CstsFrameworkPdu (WITH COMPONENTS
    {   executeDirectiveInvocation
    ,   executeDirectiveAcknowledge
    ,   executeDirectiveReturn
    }
)

-- ================================================================
-- The second part of the module defines the operations
-- of the Throw Event procedure.
-- ================================================================
-- All operations are defined in the module
-- CCSDS-CSTS-COMMON-OPERATIONS-PDUS (see F3.4)

-- ================================================================
-- The third part contains the types used by the operations
-- defined in the second part.
-- ================================================================
-- This procedure does not have specific definition.

-- ================================================================
-- The fourth part of the module definition contains the Extended
-- types used by the operations defined in the second part.
-- ================================================================

-- *****
-- EXECUTE-DIRECTIVE invocation
-- The EXECUTE-DIRECTIVE invocation is not extended; that is,
-- 'ExecuteDirectiveInvocation': 'executeDirectiveInvocationExtension'
-- shall be set to 'notUsed'.
```

```
-- EXECUTE-DIRECTIVE acknowledgement
-- The EXECUTE-DIRECTIVE positive acknowledgement is not extended; that is,
-- 'ExecuteDirectiveAcknowledge': 'StandardAcknowledgeHeader':
-- 'StandardReturnHeader': 'result': 'positive' shall be set to 'notUsed'.
-- The EXECUTE-DIRECTIVE negative acknowledgement is not extended; that is,
-- 'ExecuteDirectiveAcknowledge': 'StandardAcknowledgeHeader':
-- 'StandardReturnHeader': 'result': 'negative': 'negExtension' shall be
-- set to 'notUsed'.
-- The EXECUTE-DIRECTIVE negative acknowledgement makes use of: (a) one of
-- the common diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7); or (b) one of the additional
-- diagnostics defined by 'ExecuteDirectiveAcknowledge':
-- 'StandardReturnHeader': 'result': 'negative': 'diagnostic':
-- 'Diagnostic': 'diagnosticExtension': 'execDirAckAckDiagExt':
-- 'ExecDirNegAckDiagnosticExt'. No further diagnostics are specified; that is,
'ExecuteDirectiveAcknowledge': 'StandardReturnHeader': 'result':
-- 'negative': 'diagnostic': 'Diagnostic': 'diagnosticExtension':
-- 'execDirAckAckDiagExt': 'ExecDirNegAckDiagnosticExt' must not be set to
-- 'execDirNegAckDiagnosticExtExtension'.

-- EXECUTE-DIRECTIVE return
-- The EXECUTE-DIRECTIVE positive return is not extended; that is,
-- 'ExecuteDirectiveReturn': 'StandardReturnHeader': 'result': 'positive'
-- shall be set to 'notUsed'.
-- The EXECUTE-DIRECTIVE negative return is not extended; that is,
-- 'ExecuteDirectiveReturn': 'StandardReturnHeader': 'result': 'negative':
-- 'negExtension' shall be set to 'notUsed'.
-- The EXECUTE-DIRECTIVE negative return makes use of: (a) one of the
-- common diagnostics of 'StandardReturnHeader': 'result': 'negative':
-- 'diagnostic': 'Diagnostic' (see 3.3.2.7 and F3.3) except
-- 'diagnosticExtension'; (b) one of the additional diagnostics defined
-- by 'ExecuteDirectiveReturn': 'StandardReturnHeader': 'result':
-- 'negative': 'diagnostic': 'Diagnostic': 'diagnosticExtension':
-- 'execDirNegReturnDiagnosticExt': 'ExecDirNegReturnDiagnosticExt' in F3.4
-- except 'execDirNegReturnDiagnosticExtExtension'; or (c) one of the
-- additional diagnostics defined by 'ExecuteDirectiveReturn':
-- 'StandardReturnHeader': 'result': 'negative': 'diagnostic':
-- 'Diagnostic': 'diagnosticExtension': 'execDirNegReturnDiagnosticExt':
-- 'ExecDirNegReturnDiagnosticExt':
-- 'execDirNegReturnDiagnosticExtExtension': 'teExecDirDiagExt':
-- 'TeExecDirNegReturnDiagnosticExt' in F3.14 except
-- 'teExecDirNegReturnDiagnosticExtExtension'.
TeExecDirNegReturnDiagnosticExt ::=    CHOICE
{   guardConditionEvaluatedToFalse             [0]    NULL
,   teExecDirNegReturnDiagnosticExtExtension   [100]  Embedded
}

teExecDirDiagExt  OBJECT IDENTIFIER    ::=    {teExtProcedureParam 1}

END
```

F3.15 CSTS FRAMEWORK PROTOCOL DATA UNITS

```
CCSDS-CSTS-PDUS
{   iso(1) identified-organization(3) standards-producing-organization(112)
    ccsds(4) css(4) csts(1) framework(1) modules(1) cstsFrameworkPdus(15)
    version(1)
}

DEFINITIONS
IMPLICIT TAGS
::=    BEGIN

EXPORTS    CstsFrameworkPdu
;

IMPORTS    BindInvocation
,          BindReturn
,          PeerAbortInvocation
,          UnbindInvocation
,          UnbindReturn
    FROM CCSDS-CSTS-ASSOCIATION-CONTROL-TYPES

           ExecuteDirectiveAcknowledge
,          ExecuteDirectiveInvocation
,          ExecuteDirectiveReturn
,          GetInvocation
,          GetReturn
,          NotifyInvocation
,          ProcessDataInvocation
,          ProcessDataReturn
,          StartInvocation
,          StartReturn
,          StopInvocation
,          StopReturn
,          TransferDataInvocation
    FROM CCSDS-CSTS-COMMON-OPERATIONS-PDUS

           ReturnBuffer
    FROM CCSDS-CSTS-BUFFERED-DATA-DELIVERY-PDUS

           ForwardBuffer
    FROM CCSDS-CSTS-BUFFERED-DATA-PROCESSING-PDUS
;
```

```
-- ============================================================
-- The Pdu type lists all possible PDUs that can be exchanged between
-- the user and the provider.
-- ============================================================
CstsFrameworkPdu      ::=    CHOICE
{   bindInvocation               [0]    BindInvocation
,   bindReturn                   [1]    BindReturn
,   unbindInvocation             [2]    UnbindInvocation
,   unbindReturn                 [3]    UnbindReturn
,   peerAbortInvocation          [4]    PeerAbortInvocation
,   startInvocation              [10]   StartInvocation
,   startReturn                  [11]   StartReturn
,   stopInvocation               [20]   StopInvocation
,   stopReturn                   [21]   StopReturn
,   executeDirectiveInvocation   [30]   ExecuteDirectiveInvocation
,   executeDirectiveAcknowledge  [31]   ExecuteDirectiveAcknowledge
,   executeDirectiveReturn       [32]   ExecuteDirectiveReturn
,   getInvocation                [40]   GetInvocation
,   getReturn                    [41]   GetReturn
,   notifyInvocation             [50]   NotifyInvocation
,   processDataInvocation        [60]   ProcessDataInvocation
,   processDataReturn            [61]   ProcessDataReturn
,   forwardBuffer                [62]   ForwardBuffer
,   transferDataInvocation       [70]   TransferDataInvocation
,   returnBuffer                 [71]   ReturnBuffer
}

-- ============================================================
-- The CstsFrameworkTopPdu is another name for CstsFrameworkPdu.
-- This type is added to cover the case of a compiler that does not
-- recognize CstsFrameworkPdu as being a top level PDU.
-- ============================================================
CstsFrameworkTopPdu   ::=    CstsFrameworkPdu

END
```

F3.16 CSTS SPECIFICATION FRAMEWORK PROCEDURE PARAMETERS, EVENTS, AND DIRECTIVES

```
CCSDS-CSTS-FW-PROCEDURE-PARAMETERS-EVENTS-DIRECTIVES
{   iso(1) identified-organization(3) standards-producing-organization(112)
    ccsds(4) css(4) csts(1) framework(1) modules(1)
    procedureParamEventDirective(16) version(2)
}

DEFINITIONS
IMPLICIT TAGS
::=     BEGIN

EXPORTS     procAssociationControl
,           pACparametersId
,           pACeventsId
,           pACdirectivesId
,           pACinitiatorId
,           PACinitiatorIdType
,           pACresponderId
,           PACresponderIdType
,           pACresponderPortId
,           PACresponderPortIdType
,           pACserviceInstanceId
,           PACserviceInstanceIdType
,           procUnbuffDataDelivery
,           pUBDDparametersId
,           pUBDDeventsId
,           pUDDdirectivesId
,           procBuffDataDelivery
,           pBDDparametersId
,           pBDDeventsId
,           pBDDdirectivesId
,           pBDDreturnBufferSize
,           PBDDreturnBufferSizeType
,           pBDDdeliveryMode
,           PBDDdeliveryModeType
,           pBDDdeliveryLatencyLimit
,           PBDDdeliveryLatencyLimitType
,           pBDDconfigurationChange
,           pBDDconfigurationChangeEvtValue1
,           PBDDconfigurationChangeEvtValue1Type
,           pBDDconfigurationChangeEvtValue2
,           PBDDconfigurationChangeEvtValue2Type
,           procDataProcessing
,           pDPparametersId
,           pDPeventsId
,           pDPdirectivesId
,           pDPinputQueueSize
,           PDPinputQueueSizeType
,           pDPdataProcessingCompleted
,           pDPconfigurationChange
,           pDPconfigurationChangeEvtValue
,           PDPconfigurationChangeEvtValueType
,           procBufferDataProcessing
,           pBDPparametersId
,           pBDPeventsId
,           pBDPdirectivesId
,           pBDPdataTransferMode
```

```
,          PBDPdataTransferModeType
,          pBDPmaxForwardBufferSize
,          PBDPmaxForwardBufferSizeType
,          pBDPprocessingLatencyLimit
,          PBDPprocessingLatencyLimitType
,          pBDPdataProcessingCompleted
,          pBDPconfigurationChange
,          pBDPconfigurationChangeEvtValue1
,          PBDPconfigurationChangeEvtValue1Type
,          pBDPconfigurationChangeEvtValue2
,          PBDPconfigurationChangeEvtValue2Type
,          pBDPconfigurationChangeEvtValue3
,          PBDPconfigurationChangeEvtValue3Type
,          procSeqControlledDataProcess
,          pSCDPparametersId
,          pSCDPeventsId
,          pSCDPdirectivesId
,          pSCDPdataUnitId
,          PSCDPdataUnitId
,          pSCDPdataProcessingCompleted
,          pSCDPexpired
,          pSCDPlocked
,          pSCDPconfigurationChange
,          pSCDPconfigurationChangeEvtValue
,          PSCDPconfigurationChangeEvtValueType
,          pSCDPresetDirective
,          pSCDPresetDirectiveDirQual
,          PSCDPresetDirectiveDirQualType
,          procInformationQuery
,          pIQparametersId
,          pIQeventsId
,          pIQdirectivesId
,          pIQnamedLabelLists
,          PIQnamedLabelListsType
,          pIQnamedLabelListsTypeExt
,          procCyclicReport
,          pCRparametersId
,          pCReventsId
,          pCRdirectivesId
```

```
,          pCRnamedLabelLists
,          pCRminimumAllowedDeliveryCycle
,          PCRnamedLabelListsType
,          PCRminimumAllowedDeliveryCycleType
,          pCRnamedLabelListsTypeExt
,          procNotification
,          pNparametersId
,          pNeventsId
,          pNdirectivesId
,          pNnamedLabelLists
,          PNnamedLabelListsType
,          pNnamedLabelListsTypeExt
,          procThrowEvent
,          pTEparametersId
,          pTEeventsId
,          pTEdirectivesId
;

IMPORTS    fwProceduresFunctionalities
    FROM CCSDS-CSTS-OBJECT-IDENTIFIERS

           ServiceInstanceIdentifier
    FROM CCSDS-CSTS-SERVICE-INSTANCE-ID

           AuthorityIdentifier
,          BufferSize
,          DataTransferMode
,          DataUnitId
,          DeliveryLatencyLimit
,          DeliveryMode
,          IdentifierString
,          IntPos
,          IntUnsigned
,          Label
,          ProcessingLatencyLimit
,          ProductionStatus
    FROM CCSDS-CSTS-COMMON-TYPES
;

-- ============================================================
-- FRAMEWORK CROSS SUPPORT IDENTIFIERS


-- ASSOCIATION CONTROL
procAssociationControl   OBJECT IDENTIFIER ::=
                                           {fwProceduresFunctionalities 1}
pACparametersId          OBJECT IDENTIFIER    ::=    {procAssociationControl 1}
pACeventsId              OBJECT IDENTIFIER    ::=    {procAssociationControl 2}
pACdirectivesId          OBJECT IDENTIFIER    ::=    {procAssociationControl 3}
pACinitiatorId           OBJECT IDENTIFIER    ::=    {pACparametersId 1}
PACinitiatorIdType                            ::=    AuthorityIdentifier
pACresponderId           OBJECT IDENTIFIER    ::=    {pACparametersId 2}
PACresponderIdType                            ::=    AuthorityIdentifier
pACresponderPortId       OBJECT IDENTIFIER ::=    {pACparametersId 3}
PACresponderPortIdType                           ::=    IdentifierString (SIZE (3 .. 16))
pACserviceInstanceId     OBJECT IDENTIFIER    ::=    {pACparametersId 4
PACserviceInstanceIdType                         ::=    ServiceInstanceIdentifier
```

```
-- UNBUFFERED DATA DELIVERY
procUnbuffDataDelivery   OBJECT IDENTIFIER    ::=
                                              {fwProceduresFunctionalities 2}
pUBDDparametersId        OBJECT IDENTIFIER    ::=    {procUnbuffDataDelivery 1}
pUBDDeventsId            OBJECT IDENTIFIER    ::=    {procUnbuffDataDelivery 2}
pUDDdirectivesId         OBJECT IDENTIFIER    ::=    {procUnbuffDataDelivery 3}


-- BUFFERED DATA DELIVERY
procBuffDataDelivery     OBJECT IDENTIFIER    ::=
                                              {fwProceduresFunctionalities 3}
pBDDparametersId         OBJECT IDENTIFIER    ::=    {procBuffDataDelivery 1}
pBDDeventsId             OBJECT IDENTIFIER    ::=    {procBuffDataDelivery 2}
pBDDdirectivesId         OBJECT IDENTIFIER    ::=    {procBuffDataDelivery 3}
pBDDreturnBufferSize     OBJECT IDENTIFIER    ::=    {pBDDparametersId 1}
PBDDreturnBufferSizeType                      ::=    BufferSize
pBDDdeliveryMode         OBJECT IDENTIFIER    ::=    {pBDDparametersId 2}
PBDDdeliveryModeType                          ::=    DeliveryMode
pBDDdeliveryLatencyLimit OBJECT IDENTIFIER    ::=    {pBDDparametersId 3}
PBDDdeliveryLatencyLimitType                  ::=    DeliveryLatencyLimit
pBDDconfigurationChange  OBJECT IDENTIFIER    ::=    {pBDDeventsId 1}
pBDDconfigurationChangeEvtValue1   OBJECT IDENTIFIER  ::=      {pBDDreturnBufferSize}
PBDDconfigurationChangeEvtValue1Type          ::=    PBDDreturnBufferSizeType
pBDDconfigurationChangeEvtValue2   OBJECT IDENTIFIER  ::=      {pBDDdeliveryLatencyLimit}
PBDDconfigurationChangeEvtValue2Type          ::=    PBDDdeliveryLatencyLimitType


-- DATA PROCESSING
procDataProcessing    OBJECT IDENTIFIER       ::=    {fwProceduresFunctionalities 4}
pDPparametersId       OBJECT IDENTIFIER       ::=    {procDataProcessing 1}
pDPeventsId           OBJECT IDENTIFIER       ::=    {procDataProcessing 2}
pDPdirectivesId       OBJECT IDENTIFIER       ::=    {procDataProcessing 3}
pDPinputQueueSize     OBJECT IDENTIFIER       ::=    {pDPparametersId 1}
PDPinputQueueSizeType                         ::=    BufferSize
pDPdataProcessingCompleted   OBJECT IDENTIFIER    ::=    {pDPeventsId 1}
pDPconfigurationChange   OBJECT IDENTIFIER    ::=    {pDPeventsId 2}
pDPconfigurationChangeEvtValue  OBJECT IDENTIFIER ::=    {pDPinputQueueSize}
PDPconfigurationChangeEvtValueType            ::=    PDPinputQueueSizeType

-- BUFFERED DATA PROCESSING
procBufferDataProcessing OBJECT IDENTIFIER    ::=    {fwProceduresFunctionalities 5}
pBDPparametersId         OBJECT IDENTIFIER    ::=    {procBufferDataProcessing 1}
pBDPeventsId             OBJECT IDENTIFIER    ::=    {procBufferDataProcessing 2}
pBDPdirectivesId         OBJECT IDENTIFIER    ::=    {procBufferDataProcessing 3}
pBDPdataTransferMode     OBJECT IDENTIFIER    ::=    {pBDPparametersId 1}
PBDPdataTransferModeType                      ::=    DataTransferMode
pBDPmaxForwardBufferSize OBJECT IDENTIFIER    ::=    {pBDPparametersId 2}
PBDPmaxForwardBufferSizeType                  ::=    BufferSize
pBDPprocessingLatencyLimit   OBJECT IDENTIFIER    ::=    {pBDPparametersId 3}
PBDPprocessingLatencyLimitType                ::=    ProcessingLatencyLimit
pBDPdataProcessingCompleted  OBJECT IDENTIFIER    ::=    {pDPdataProcessingCompleted}
pBDPconfigurationChange  OBJECT IDENTIFIER    ::=    {pDPconfigurationChange}
pBDPconfigurationChangeEvtValue1    OBJECT IDENTIFIER ::= {pBDPmaxForwardBufferSize}
PBDPconfigurationChangeEvtValue1Type          ::=    PBDPmaxForwardBufferSizeType
pBDPconfigurationChangeEvtValue2    OBJECT IDENTIFIER ::= {pDPinputQueueSize}
PBDPconfigurationChangeEvtValue2Type              ::=    PDPinputQueueSizeType
pBDPconfigurationChangeEvtValue3    OBJECT IDENTIFIER ::= {pBDPprocessingLatencyLimit}
PBDPconfigurationChangeEvtValue3Type          ::=    PBDPprocessingLatencyLimitType
```

```
-- SEQUENCE-CONTROLLED DATA PROCESSING
procSeqControlledDataProcess     OBJECT IDENTIFIER    ::=
                                          {fwProceduresFunctionalities 6}
pSCDPparametersId                OBJECT IDENTIFIER    ::=
                                          {procSeqControlledDataProcess 1}
pSCDPeventsId                    OBJECT IDENTIFIER    ::=
                                          {procSeqControlledDataProcess 2}
pSCDPdirectivesId                OBJECT IDENTIFIER     ::=
                                          {procSeqControlledDataProcess 3}
pSCDPdataUnitId                  OBJECT IDENTIFIER    ::=      {pSCDPparametersId 1}
PSCDPdataUnitId                           ::=    DataUnitId
pSCDPdataProcessingCompleted     OBJECT IDENTIFIER    ::=
                                          {pDPdataProcessingCompleted}
pSCDPexpired                     OBJECT IDENTIFIER    ::=      {pSCDPeventsId 1}
pSCDPlocked                      OBJECT IDENTIFIER    ::=      {pSCDPeventsId 2}
pSCDPconfigurationChange         OBJECT IDENTIFIER    ::=      {pDPconfigurationChange}
pSCDPconfigurationChangeEvtValue    OBJECT IDENTIFIER ::=      {pDPinputQueueSize}
PSCDPconfigurationChangeEvtValueType                  ::=      PDPinputQueueSizeType
pSCDPresetDirective              OBJECT IDENTIFIER    ::=      {pSCDPdirectivesId 1}
pSCDPresetDirectiveDirQual       OBJECT IDENTIFIER    ::=      {pSCDPresetDirective 1}
PSCDPresetDirectiveDirQualType                        ::=      PSCDPdataUnitId


-- INFORMATION QUERY
procInformationQuery             OBJECT IDENTIFIER    ::=
                                          {fwProceduresFunctionalities 7}
pIQparametersId                  OBJECT IDENTIFIER    ::=      {procInformationQuery 1}
pIQeventsId                      OBJECT IDENTIFIER    ::=      {procInformationQuery 2}
pIQdirectivesId                  OBJECT IDENTIFIER    ::=      {procInformationQuery 3}
pIQnamedLabelLists           OBJECT IDENTIFIER    ::=    {pIQparametersId 1}
-- Note: In case the service user selects a procedure type in the GET
-- invocation, the service provider will return the same list of
-- parameters for each active instance of the selected procedure type.
PIQnamedLabelListsType    ::=    LabelListSet
pIQnamedLabelListsTypeExt        OBJECT IDENTIFIER    ::=      {pIQnamedLabelLists 1}


-- CYCLIC REPORT
procCyclicReport                 OBJECT IDENTIFIER    ::=
                                          {fwProceduresFunctionalities 8}
pCRparametersId                  OBJECT IDENTIFIER    ::=      {procCyclicReport 1}
pCReventsId                      OBJECT IDENTIFIER    ::=      {procCyclicReport 2}
pCRdirectivesId                  OBJECT IDENTIFIER    ::=      {procCyclicReport 3}
pCRnamedLabelLists               OBJECT IDENTIFIER    ::=      {pCRparametersId 1}
pCRminimumAllowedDeliveryCycle   OBJECT IDENTIFIER    ::=      {pCRparametersId 2}
PCRnamedLabelListsType    ::=    LabelListSet
PCRminimumAllowedDeliveryCycleType                    ::=      IntPos
pCRnamedLabelListsTypeExt        OBJECT IDENTIFIER    ::=      {pCRnamedLabelLists 1}


-- NOTIFICATION
procNotification                 OBJECT IDENTIFIER    ::=
                                          {fwProceduresFunctionalities 9}
pNparametersId                   OBJECT IDENTIFIER    ::=      {procNotification 1}
pNeventsId                       OBJECT IDENTIFIER    ::=      {procNotification 2}
pNdirectivesId                   OBJECT IDENTIFIER    ::=      {procNotification 3}
pNnamedLabelLists                OBJECT IDENTIFIER    ::=      {pNparametersId 1}
PNnamedLabelListsType     ::=    LabelListSet
pNnamedLabelListsTypeExt         OBJECT IDENTIFIER    ::=      {pNnamedLabelLists 1}
```

```
-- THROW EVENT
procThrowEvent                  OBJECT IDENTIFIER    ::=
                                                {fwProceduresFunctionalities 10}
pTEparametersId                 OBJECT IDENTIFIER    ::=       {procThrowEvent 1}
pTEeventsId                     OBJECT IDENTIFIER    ::=       {procThrowEvent 2}
pTEdirectivesId                 OBJECT IDENTIFIER    ::=       {procThrowEvent 3}


-- Additional types for the Framework Functional Resource
LabelList             ::=    SEQUENCE
{   name                  VisibleString
,   defaultList           BOOLEAN
,   labels                SEQUENCE OF Label
}

LabelListSet          ::=    SET OF LabelList
-- Only one list in the set can be the default list.

END
```

ANNEX G

SERVICE STATE TABLES

(NORMATIVE)

G1 OVERVIEW

The state machine of a CSTS is determined by the state type (stateless or stateful) of the procedure that the service uses for its prime procedure instance.

G2 STATE MACHINE FOR CSTSES WITH A STATELESS PRIME PROCEDURE INSTANCE

NOTE – This subsection specifies the state machine for a CSTS that has a stateless prime procedure.

G2.1 An instance of the CSTS shall have one of two states: 'unbound' (state 1) or 'bound' (state 2).

G2.2 When the state of a CSTS instance's Association Control procedure is 'unbound', the state of the CSTS shall be 'unbound' (state 1).

G2.3 When the state of a CSTS instance's Association Control procedure is 'bound', the state of the CSTS shall be 'bound' (state 2).

NOTE – The states of the Association Control procedure and the events that cause transitions between them are defined in 4.3.6.

G2.4 Each CSTS with stateless prime procedure shall conform to the state table for a CSTS instance defined in table G-1, supported by information in tables G-2 through G-4.

G2.5 The state table for a CSTS with a stateless prime procedure instance may be extended to support extended behavior that is associated with that CSTS.

Table G-1: State Table for CSTSes with a Stateless Prime Procedure Instance

No.	Incoming Event	State 1 ('unbound')	State 2 ('bound)
1	(BindInvocation)	IF "positive result" THEN (+BindReturn) → 2 ELSE (-BindReturn) ENDIF	(PeerAbortInvocation 'protocol error') {clean up }→ 1
2	'abort-causing event xxx'	[ignore]	(PeerAbortInvocation 'xxx') {clean up} → 1
3	(UnbindInvocation)	[ignore]	IF "positive result" THEN (+unbindReturn) {clean up} → 1 ELSE (-unbindReturn) ENDIF
4	(PeerAbortInvocation)	[ignore]	{clean up} → 1
5	'not authenticated PDU'	[ignore]	[ignore]
6	'invalid PDU 'xxx''	(PeerAbortInvocation 'xxx') → 1	(PeerAbortInvocation 'xxx') → 1
7	'protocol abort'	[ignore]	{clean up} → 1

Table G-2: State Table for CSTSes with a Stateless Prime Procedure Instance: Event Description References

Event	Reference
'abort-causing event xxx'	An event has occurred that is designated as resulting in an abort of the association (3.6). The values for 'xxx' are identical to the values of the `diagnostic` parameter of the PEER-ABORT operation defined in 3.6.2.2.
'not authenticated PDU'	3.2.4.5.1
'invalid PDU 'xxx''	3.2.3.6, 4.2.2.4. 'xxx' is one of the `diagnostic` values specified in 4.2.2.5
'protocol abort'	4.3.3.1.11.5

Table G-3: State Table for CSTSes with a Stateless Prime Procedure Instance: Predicate Descriptions

Predicate	Evaluates to TRUE if
"positive result"	No reason for sending a negative return has been detected.

Table G-4: State Table for CSTSes with a Stateless Prime Procedure Instance: Compound Action Definitions

Name	Actions Performed
{clean up}	stop all response timers reset parameter values to those specified in the service package

G3 STATE MACHINE FOR CSTSES WITH A STATEFUL PRIME PROCEDURE INSTANCE

NOTE – This subsection specifies the state machine for a CSTS that has a stateful prime procedure instance.

G3.1 An instance of a CSTS shall have one of two states: 'unbound' (state 1) or 'bound' (state 2).

G3.2 When the state of a CSTS instance's Association Control procedure is 'unbound', the state of the CSTS shall be 'unbound' (1).

NOTE – In state 1, all resources required to enable the provision of the service have been allocated, and all objects required to provide the service have been instantiated. However, no association yet exists between the service user and the service provider (i.e., the transfer service provider port is not bound).

G3.3 When the state of a CSTS instance's Association Control procedure is 'bound', the state of the CSTS shall be 'bound' (2).

NOTE – The states of the Association Control procedure and the events that cause transitions between them are defined in 4.3.6. In state 2, an association has been established between the service user and the service provider, and they may interact by means of the operations defined by the procedures of which the service is composed.

G3.4 An instance of a CSTS in the 'bound' state shall have one of two substates: 'bound.ready' (substate 2.1) or 'bound.active' (substate 2.2).

G3.5 The CSTS instance shall be in the 'bound.ready' (2.1) substate when the CSTS instance is in the 'bound' state and its prime procedure instance is in the 'inactive' state (see 4.2.4).

G3.6 The CSTS instance shall be in the 'bound.active' (2.2) substate when the CSTS instance is in the 'bound' state and its prime procedure instance is in the 'active' state.

NOTE – Secondary procedures required by the service may act independently of the substates 2.1 and 2.2 of the state 'bound'.

G3.7 The service instance shall remain in the 'bound.active' substate until the prime procedure instance transitions back to the state 'inactive' as specified 4.2.4.

G3.8 Each CSTS that has a stateful prime procedure instance shall conform to the state table for a CSTS defined in table G-5, supported by information in tables G-6 through G-8.

G3.9 The state table for a CSTS with a stateful prime procedure instance may be extended to support extended behavior that is associated with that CSTS.

Table G-5: State Table for CSTSes with a Stateful Prime Procedure Instance

No.	Incoming Event	State 1 ('unbound')	State 2.1 ('bound.ready')	State 2.2 ('bound.active')
1	(BindInvocation)	IF "positive result" THEN (+BindReturn) → 2.1 ELSE (-BindReturn) ENDIF	(PeerAbortInvocation 'protocol error') {clean up} → 1	(PeerAbortInvocation 'protocol error') {clean up} → 1
2	'transition to active'	[ignore]	→ 2.2	Not applicable
3	'transition to inactive'	[ignore]	(PeerAbortInvocation 'protocol error') → 1	→ 2.1
4	'abort-causing event xxx'	[ignore]	(PeerAbortInvocation 'xxx') → 1	(PeerAbortInvocation 'xxx') → 1
5	(UnbindInvocation)	[ignore]	IF "positive result" THEN (+unbindReturn) {clean up} → 1 ELSE (-unbindReturn) ENDIF	(PeerAbortInvocation 'protocol error') → 1
6	(PeerAbortInvocation)	[ignore]	{clean up} → 1	{clean up} → 1
7	'protocol abort'	[ignore]	{clean up} → 1	{clean up} → 1
8	'not authenticated PDU'	[ignore]	[ignore]	[ignore]
9	'invalid PDU 'xxx''	[ignore]	(PeerAbortInvocation 'xxx') → 1	(PeerAbortInvocation 'xxx') → 1

Table G-6: State Table for CSTSes with a Stateful Prime Procedure Instance: Event Description References

Event	Reference
'abort-causing event xxx'	An event has occurred that is designated as resulting in an abort of the association (3.6). The values for 'xxx' are identical to the values of the `diagnostic` parameter of the PEER-ABORT operation defined in 3.6.2.2.
'transition to active'	Internal event from the prime procedure signaling that it has transitioned to the 'active' state (see 4.2.4).
'transition to inactive'	Internal event from the prime procedure signaling that it has transitioned to the 'inactive' state (see 4.2.4).
'protocol abort'	4.3.3.1.11.5
'not authenticated PDU'	3.2.4.5.1
'invalid PDU 'xxx''	3.2.3.6, 4.2.2.4. 'xxx' is one of the `diagnostic` values specified in 4.2.2.5.

Table G-7: State Table for CSTSes with a Stateful Prime Procedure Instance: Predicate Descriptions

Predicate	Evaluates to TRUE if
"positive result"	No reason for sending a negative return has been detected.

Table G-8: State Table for CSTSes with a Stateful Prime Procedure Instance: Compound Action Definitions

Name	Actions Performed
{clean up}	stop all response timers reset parameter values to those specified in the service package

ANNEX H

SECURITY, SANA, AND PATENT CONSIDERATIONS

(INFORMATIVE)

H1 SECURITY ASPECTS OF CROSS SUPPORT TRANSFER SERVICES

H1.1 INTRODUCTION

This subsection describes security aspects that are common to CSTSes based on the CSTS Specification Framework. Service specifications based on this Recommended Standard are expected to reference the description provided herein and add a discussion of service-specific security aspects as applicable.

H1.2 SECURITY BACKGROUND AND OVERVIEW

The CSTS Specification Framework explicitly provides authentication and access control for CSTSes. Additional security capabilities, if required, are levied on the underlying communications services that support the CSTS. CSTSes are defined as layered application services operating over underlying communications services that must meet certain requirements, but which are otherwise unspecified. Selection of the underlying communications services over which real CSTS implementations connect is based on the requirements of the communicating parties and/or the availability of CCSDS-standard communications technology profiles and proxy specifications. Different underlying communications technology profiles are intended to address not only different performance requirements but also different security requirements. Missions and service providers are expected to select from these technology profiles to acquire the performance and security capabilities appropriate to the mission. Specification of the various underlying communications technologies, and in particular their associated security provisions, are outside the scope of this Recommended Standard.

NOTE – A CCSDS-standard communications technology profile for CSTS is specified by reference [2].

H1.3 STATEMENTS OF SECURITY CONCERNS

H1.3.1 General

This subsection identifies the support of the CSTS Specification Framework for capabilities that respond to security concerns in the areas of data privacy, data integrity, authentication, access control, availability of resources, and auditing.

H1.3.2 Data Privacy (Also Known as Confidentiality)

The CSTS Specification Framework does not define explicit data privacy requirements or capabilities to ensure data privacy. Data privacy is expected to be ensured outside of the Cross Support Transfer Service layer in the underlying communications service.

H1.3.3 Data Integrity

The CSTS Specification Framework defines and enforces a strict sequence of operations that constrain the ability of a third party to inject operation invocations or responses into the transfer service association between a service user and service provider. This constrains the ability of a third party to seize control of an active Cross Support Transfer Service instance without detection.

NOTE – The CSTS Specification Framework requires that the underlying communications service transfer data in sequence, completely and with integrity, without duplication, with flow control that notifies the application layer in the event of congestion, and with notification to the application layer in the event that communication between the service user and the service provider is disrupted (see 1.3.1). The ISP1 protocol (reference [2]) that is assumed to be used does provide these capabilities.

H1.3.4 Authentication

This CSTS Specification Framework defines authentication requirements (see 3.2.4) and defines `initiator-identifier`, `responder-identifier`, `invoker-credentials`, and `performer-credentials` parameters of the service operation invocations and responses that are used to perform CSTS authentication. The CSTS authentication capability can be selectively set to authenticate at one of three levels: authenticate every CSTS PDU, authenticate only the BIND operation invocation and return, or perform no authentication. Depending upon the inherent authentication available from the underlying communication network, the security environment in which the service user and service provider are operating, and the security requirements of the spaceflight mission, the CSTS authentication level can be adapted by choosing the operation invocation and responses that are to be authenticated. Furthermore, the mechanism used for generating and checking the credentials, and thus the level of protection against masquerading (simple or strong authentication) can be selected in accordance with the results of a threat analysis. One possible procedure by which operation invocations and responses are authenticated is described in the CSS Green Book (reference [I2]). Although the procedure as presented in reference [I2] refers to SLE transfer services, it can equally be applied to CSTSes.

H1.3.5 Access Control

This CSTS Specification Framework defines access control requirements (see 4.3.3.1.12), and defines `initiator-identifier` and `responder-identifier` parameters of operation invocations and responses that are used to perform CSTS access control. The procedure by which access to a CSTS is controlled is described in the CSS Green Book (reference [I2]). Although the procedure as presented in reference [I2] refers to SLE transfer services, it can equally be applied to CSTSes.

H1.3.6 Availability of Resources

CSTSes are provided via communication networks that have some limit with respect to the resources available to support those services. If these resources can be diverted from their support of the CSTS (in what is commonly known as 'denial of service') then the performance of the CSTS may be curtailed or inhibited. This CSTS Specification Framework does not define explicit capabilities to prevent denial of service. Resource availability is expected to be ensured by appropriate capabilities in the underlying communications service. The specific capabilities will be dependent upon the technologies used in the underlying communications service and the security environment in which the transfer service user and service provider operate.

H1.3.7 Auditing

This CSTS Specification Framework does not define explicit security auditing requirements or capabilities. Security auditing is expected to be negotiated and implemented bilaterally between the spaceflight mission and the service provider.

H1.4 POTENTIAL THREATS AND ATTACK SCENARIOS

H1.4.1 Breach of Privacy

CSTSes, in general, depend on unspecified mechanisms operating in the underlying communications service or on privacy-ensuring capabilities in the service-specific application processes that interoperate through the CSTS Specification Framework procedures, to ensure data privacy (confidentiality). If no such mechanisms are actually implemented, or the mechanisms selected are inadequate or inappropriate to the network environment in which the mission is operating, an attacker could read the data contained in the CSTS PDUs as they traverse the WAN between service user and service provider.

H1.4.2 PDU Interception and Replay

The CSTS Specification Framework constrains the ability of a third party to seize control of an active CSTS instance, but it does not specify mechanisms that would prevent an attacker from intercepting the PDUs and replacing the contents of the parameter carried by the PDUs. The

prevention of such a replacement attack depends on unspecified mechanisms in the underlying communications service, in unspecified (with respect to the CSTS Specification Framework) mechanisms in the service-specific application processes that interoperate through the CSTS Specification Framework procedures (e.g., specifying an encryption function as part of the standard service), or some combination of the two. If no such mechanisms are actually implemented, or the mechanisms selected are inadequate or inappropriate to the network environment in which the mission is operating, an attacker could substitute data transferred between the service user and the service provider without detection.

H1.4.3 Unauthenticated Access

If the CSTS authentication capability is not used and if authentication is not ensured by the underlying communications service, attackers could somehow obtain valid `initiator-identifier` values and use them to initiate CSTS instances by which they could gain access to the data transferred via these services or inject data for further processing by the service provider.

H1.4.4 Denial of Service Attacks

A CSTS depends on unspecified mechanisms operating in the underlying communications service to ensure that the supporting network has sufficient resources to provide sufficient support to legitimate service users. If no such mechanisms are actually implemented, or the mechanisms selected are inadequate or inappropriate to the network environment in which the mission is operating, an attacker could prevent legitimate service users from using the CSTS.

H1.4.5 Failure to Detect Breach Attempts

If the service provider of a CSTS provides no security auditing capabilities, or if a service user chooses not to employ auditing capabilities that do exist, then attackers may delay or escape detection while stealing or altering data exchanged via the service.

H1.5 CONSEQUENCES OF NOT APPLYING SECURITY

The consequences of not applying security to CSTS are possible degradation and loss of ability to use the service, or the substitution of altered data that are exchanged between the service user and service provider of that service. In particular, the alteration of telecommand and telemetry data may seriously affect spacecraft safety.

H2 SANA CONSIDERATION

H2.1 GENERAL

This Recommended Standard requests SANA to extend the already existing registry for OIDs and to create a new registry to capture the specification of Functional Resources, as described below. New assignments in these registries, in conformance with the modifications identified, will be shown at the global SANA registry Web site: https://sanaregistry.org. Therefore the reader shall look at the SANA Web site for all the assignments contained in these registries.

Already registered registry entries shall not be affected by this Recommended Standard.

This Recommended Standard makes extensive use of OIDs. All OIDs specified by this Recommended Standard are part of OID subtrees, the management of which has been delegated to the CSS Area. The structure of those subtrees is specified in annex D and extends the already existing OID registry that is administered by SANA on behalf of CCSDS. The SANA registry containing the overall CCSDS OID tree may be found at https://sanaregistry.org/oid/tree/. The CSTS OID subtree is registered at https://sanaregistry.org/r/oid.

Annex K provides a graphical representation of the OID tree structure and the OID values relevant in the context of this Recommended Standard. The normative specification of the OID tree structure is provided in annex D, while the normative assignment of OID values can be found in annex F.

H2.2 FRAMEWORK OBJECT IDENTIFIERS

Within the OID subtree, the management of which has been delegated to the CSS Area, a subtree shall be created that accommodates all OIDs associated with the present version of this Recommended Standard. The root node of this subtree shall be:

```
{iso(1) identified-organization(3) standards-producing-
organization(112) ccsds(4) css(4) csts(1) framework(1)}
```

The recommendations in this document request SANA to update the CCSDS OID registry by adding the OIDs and labels contained in the above-identified subtree. The CSS Area provides the new OIDs and their labels to SANA in an Extensible Markup Language (XML) formatted file.

The assignment policy for this ‘framework’ subtree is that

a) new OIDs shall be constructed as specified in D4; and

b) new OIDs shall be allocated in one or more of the subbranches of this subtree only in the context of the definition of new operations and additional extensions of the definitions of the operations, in case of new or additional derived procedures and/or new ASN.1 modules specified in this Recommended Standard, that is, only when a new issue of this Recommended Standard is approved.

NOTE – The engineering review of the proposed registry changes is implied by the CCSDS publication and approval process for Recommended Standards.

H2.3 SERVICE SPECIFIC OIDS

Within the OID subtree, the management of which has been delegated to the CSS Area, a subtree shall be created that accommodates all OIDs of CSTS types and, as applicable, service-type-specific derived services, parameters, procedures, and ASN.1 modules, and the reference to the Functional Resource Type modeling the given service type. The root node of this subtree shall be:

```
{iso(1) identified-organization(3) standards-producing-
organization(112) ccsds(4) css(4) csts(1) services(2)}
```

The recommendations in those Recommended Standards that specify CSTSes based on this document request SANA to update the CCSDS OID registry by adding the OIDs and labels contained in the CSTS type specific part of the above identified subtree. The CSS Area provides the new OIDs and their labels to SANA in an XML-formatted file.

The assignment policy for this 'services' subtree is that

a) new Object Identifiers shall be constructed as specified in D5; and

b) new Object Identifiers shall be allocated within the subtree part allocated to the given CSTS type, only in the context of the publication of the Recommended Standard specifying that CSTS type.

NOTE – The engineering review of the proposed registry changes is implied by the CCSDS publication and approval process for Recommended Standards.

H2.4 FUNCTIONAL RESOURCE REGISTRY

Some of the operations specified in this Recommended Standard interact with Functional Resources, for instance, to read or set parameters, or to be informed of the occurrence of certain events. The necessary identification and definition of these Functional Resource Types and the associated parameters, events, and directives are provided in a dedicated registry.

The recommendations in this document have created the local SANA registry named 'Functional Resources' located at https://sanaregistry.org/r/functional_resources.

Informative material regarding the detailed content of this registry is provided in annex L of this Recommended Standard. The normative specification of how the records of this registry shall be built is provided in D6.

Within the OID subtree, the management of which has been delegated to the CSS Area, a subtree has been created that accommodates all the OIDs used to identify each record of the 'Functional Resources' registry. The root node of this subtree is:

```
{iso(1) identified-organization(3) standards-producing-
organization(112) ccsds(4) css(4) crossSupportResources(2)}
```

There are two subnodes under the 'crossSupportResources' node: 'crossSupportFunctionalities' and 'agenciesFunctionalities', used to register CCSDS-standard Functional Resource Types and Agency-specific Functional Resource Types, respectively. Under each Functional Resource Type OID, the parameters, events, and directives are registered, each under a dedicated subnode.

The Functional Resource registry is owned by the CSS Area, which therefore is also the Review Authority of this registry. Any proposed change under the 'crossSupportFunctionalities' node of this registry, be it the addition of a Functional Resource type or the modification of a Functional Resource's parameters, events, or directives, shall originate from the CSS Area and require engineering review by that Area or an Expert Group, if such group is appointed by the CCSDS Management Council (CMC). The way that records of this part of the registry and in particular the associated OIDs shall be built is specified in D6.2.

Service providers, typically some Agency, may implement 'private' capabilities that are outside the functionality covered by the CCSDS specified Functional Resources but still of potential interest to cross support. Such a service provider may, via a designated control authority, propose the registration of Functional Resources associated with such 'private' capabilities. If approved following the engineering review by the Review Authority, that is, the CSS Area, SANA will be requested to update the registry part under the 'agenciesFunctionalities' node accordingly. The way that records of this part of the registry and in particular the associated OIDs shall be built is specified in D6.3.

H2.5 REGISTRIES OF ELEMENTS OF SERVICE INSTANCE IDENTIFIERS

A specific CSTS instance that a user wants to bind to is identified by means of the `service-instance-identifier` parameter of the BIND invocation. As specified in 3.4.2.2.7, the parameter consists of the following elements:

a) an identifier of the spacecraft being supported by the given CSTS instance;

b) an identifier of the facility where the CSTS provider is located;

c) an identifier of the CSTS type;

d) the service instance number.

The elements a), b), and c) are Object Identifiers.

Spacecraft are listed in the enterprise Spacecraft Registry located at https://sanaregistry.org/r/spacecraft. This registry, among others, assigns an OID to each spacecraft where all these OIDs are elements of the subtree, the root node of which is:

```
{iso(1) identified-organization(3) standards-producing-
organization(112) ccsds(4) spacecraft(7)}
```

Facilities are listed in the enterprise Service Site and Aperture Registry located at https://sanaregistry.org/r/service_sites_apertures. This registry, among others, assigns an OID to each service site. The service site may be further broken down as applicable per aperture, per forward link and return link of that aperture, and per frequency band supported on the forward link and return link, respectively. Depending on the entity the CSTS provider is associated with, the OID of the CSTS provider location is formed. The root node in any case is:

```
{iso(1) identified-organization(3) standards-producing-
organization(112) ccsds(4) service-site-and-aperture(6)}
```

Agencies or other service providers implementing CSTSes will also decide where these providers will be located in terms of facilities operated by this service provider and if such a provider is common with respect to the given site or only available in conjunction with a specific aperture or frequency band of that aperture. The Registration Authority of this registry is SANA. Requests for the addition of new entries to this registry must come from the official representative, that is, an Agency Representative (AR) with the service site Point of Contact (PoC) role, of a space agency or other organization that is a member of the CCSDS and desires to have a site registered as supporting the given CSTS type. The SANA Steering Group (SSG) serves as the Review Authority of this registry.

The subtree containing the CSTS type OIDs is addressed in H2.3. Regarding the link between service sites discussed above and the CSTS types available at those sites, it is recommended that service providing organizations offering CSTSes should add the OIDs of those CSTSes to the list of services available at that site in addition to the Initial Service Types listed in reference [6].

H3 PATENT CONSIDERATIONS

No patent rights are known to adhere to any of the specifications of this Recommended Standard.

ANNEX I

INFORMATIVE REFERENCES

(INFORMATIVE)

[I1] *Organization and Processes for the Consultative Committee for Space Data Systems*. Issue 4. CCSDS Record (Yellow Book), CCSDS A02.1-Y-4. Washington, D.C.: CCSDS, April 2014.

[I2] *Cross Support Concept—Part 1: Space Link Extension Services*. Issue 3. Report Concerning Space Data System Standards (Green Book), CCSDS 910.3-G-3. Washington, D.C.: CCSDS, March 2006.

[I3] *Guidelines for the Specification of Cross Support Transfer Services*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 921.2-M-1. Washington, D.C.: CCSDS, March 2019.

[I4] *Extensible Space Communication Cross Support—Service Management—Concept*. Issue 1. Report Concerning Space Data System Standards (Green Book), CCSDS 902.0-G-1. Washington, D.C.: CCSDS, September 2014.

[I5] *Cross Support Transfer Service Concept*. Issue 1. Report Concerning Space Data System Standards (Green Book), CCSDS 920.0-G-1. Washington, D.C.: CCSDS, forthcoming.

[I6] *Cross Support Service Management—Service Management Utilization Request Formats*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 902.9-B-1. Washington, D.C.: CCSDS, forthcoming.

[I7] *Cross Support Service Management—Simple Schedule Format Specification*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 902.1-B-1. Washington, D.C.: CCSDS, May 2018.

[I8] *Space Communications Cross Support—Architecture Description Document*. Report Concerning Space Data System Standards (Green Book), CCSDS 901.0-G-1. Washington, D.C.: CCSDS, November 2013.

[I9] *Space Communications Cross Support—Architecture Requirements Document*. Recommended Practice for Space Data Systems (Magenta Book), CCSDS 901.1-M-1. Washington, D.C.: CCSDS, May 2015.

[I10] *Cross Support Service Management—Service Package Data Formats*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 902.4-B-1. Washington, D.C.: CCSDS, forthcoming.

[I11] *Cross Support Transfer Services—Monitored Data Service*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 922.1-B-1. Washington, D.C.: CCSDS, April 2017.

[I12] *Cross Support Transfer Services—Forward Frame Service.* Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 922.3-B-1. Washington, D.C.: CCSDS, forthcoming.

ANNEX J

ABBREVIATIONS

(INFORMATIVE)

This annex lists the acronyms used in this Recommended Standard.

AD	Area Director
AR	Agency Representative
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
B	blocking operation
BER	Basic Encoding Rules (ASN.1)
C	conditional
CCSDS	Consultative Committee for Space Data Systems
CMC	CCSDS Management Council
CSS	CCSDS Cross Support Services Area
CSSS	Cross Support Service System
CSTS	Cross Support Transfer Services
DAD	Deputy Area Director
EM	Element Management (of an ESLT)
ESLT	Earth Space Link Terminal
ICS	implementation conformance statement
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
ISP	Internet Protocol for Transfer Services
M	mandatory
N/A	not applicable
NB	non-blocking operation

OID	Object Identifier
OSI	Open Systems Interconnection
PDU	protocol data unit
PICS	protocol implementation conformance statement
PM	Provision Management (of a Provider CSSS)
PoC	point of contact
RAF	Return All Frames (SLE)
RCF	Return Channel Frame (SLE)
RL	requirements list
ROCF	Return Operational Control Field (SLE)
SANA	Space Assigned Numbers Authority
SF	stateful
SI	International System of Units
SL	stateless
SLE	Space Link Extension
TCP	Transmission Control Protocol
UM	Utilization Management (of an Earth User CSSS)
UTC	Universal Coordinated Time
WG	working group
XML	Extensible Markup Language

ANNEX K

OBJECT IDENTIFIERS

(INFORMATIVE)

Object Identifiers are used for a unique and unambiguous identification of information exchanged by the services derived from the CSTS Specifications Framework. The OIDs defined for the use by Cross Support Services, as registered with SANA, are shown in figure K-1.

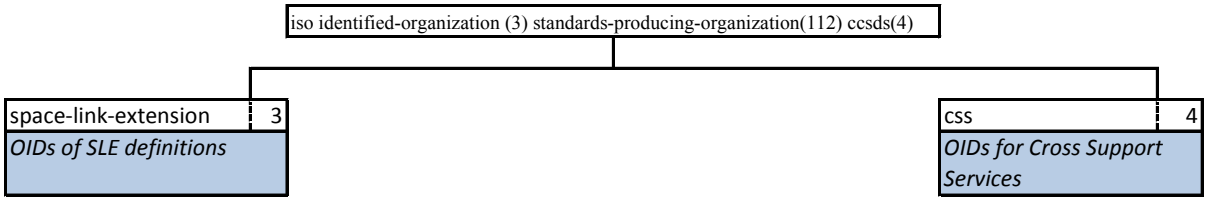

Figure K-1: Cross Support Services Part of the CCSDS Object Identifiers Tree

The OIDs defined for the Cross Support Services are in turn broken down into two groups (see figure K-2):

a) the OIDs defined for the use by CSTSes;

b) the OIDs defined for the use of Cross Support Resources.

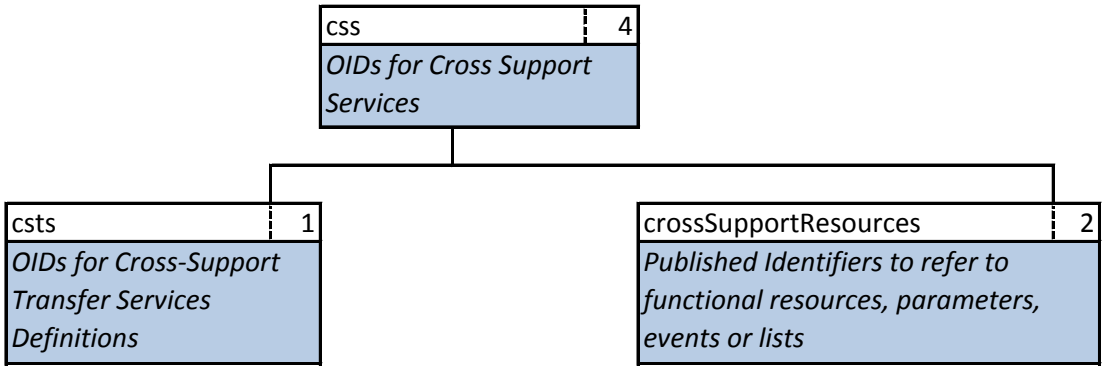

Figure K-2: CSS Object Identifiers Tree

The CSTS OIDs are in turn subdivided into two subgroups, those required by this Recommended Standard, that is, the CSTS Specification Framework, and those required by new services (see figure K-3).

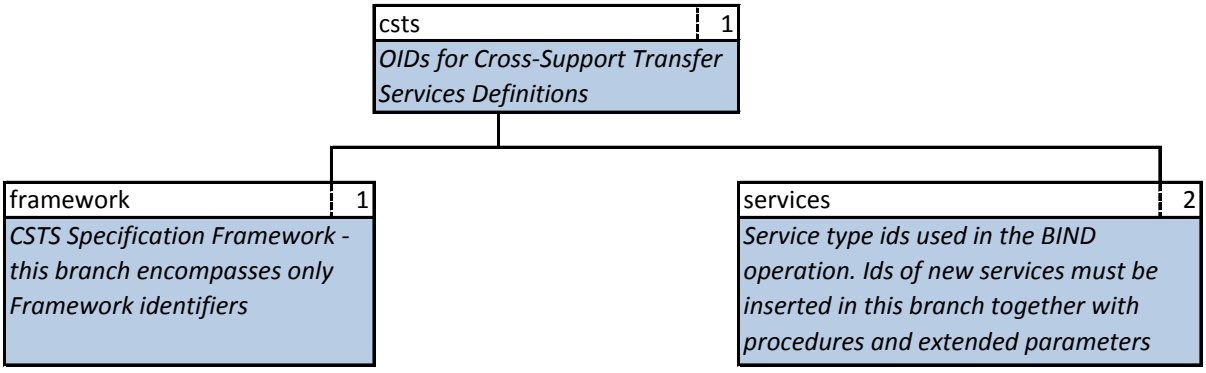

Figure K-3: CSTS Object Identifiers Tree

The OIDs defined for this Recommended Standard cover

a) the modules: each ASN.1 module has its own identifier including a version;

b) the identifiers of the operations specified in this Recommended Standard;

c) the identifiers of all procedures and associated extensions in terms of additional parameters;

d) procedure identifiers, as used in the operations' standard operation header, and the identifiers of Framework Functional Resources of the Framework.

The Framework OID branches are shown in figure K-4.

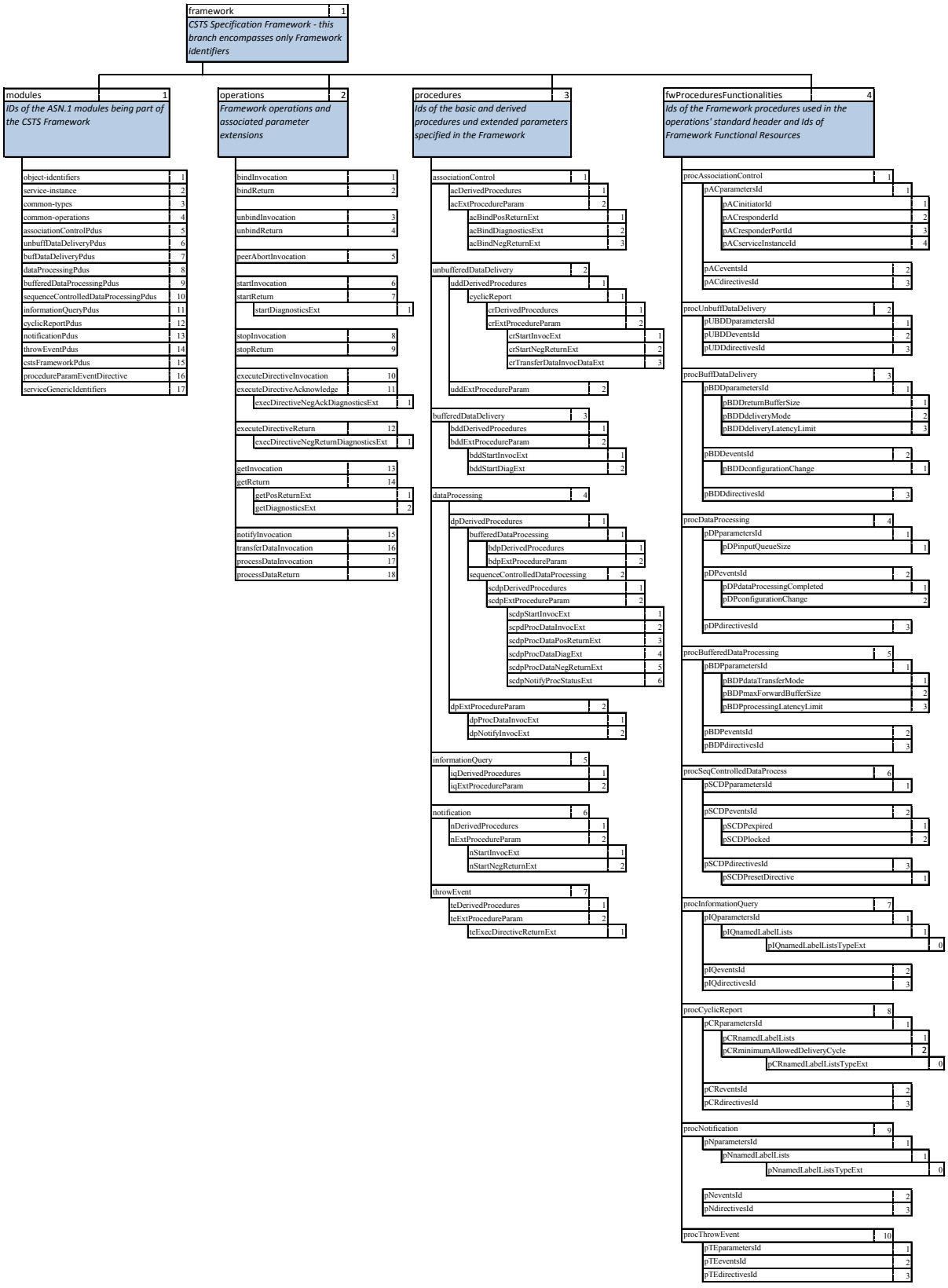

Figure K-4: CSTS Specification Framework Object Identifiers Tree

This Recommended Standard defines the root OIDs of new CSTSes (figure K-5). It is the responsibility of the service-defining organization to specify individual OIDs required by the new service, including the OIDs of the derived procedures and of the syntax used by the service (e.g., ASCII, XML, binary, …), as well as the OID of the reference towards the Functional Resource type modeling the given service type.

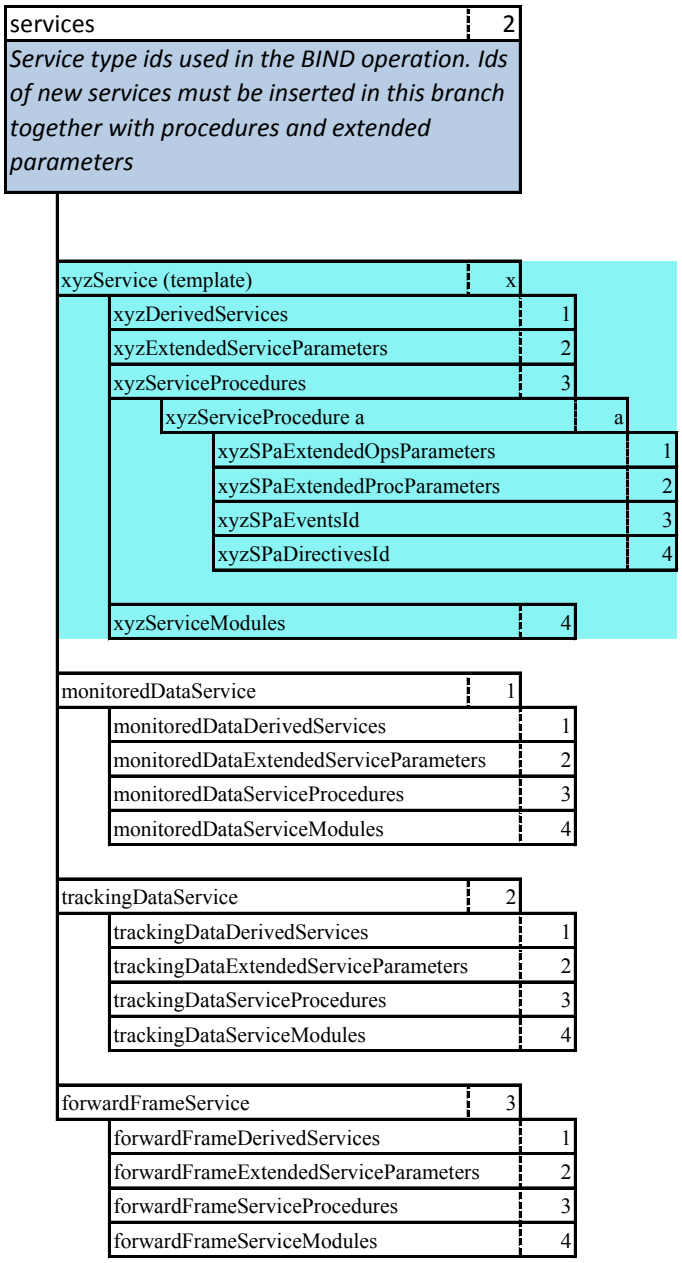

Figure K-5: CSTS Services Object Identifiers Tree

As an example, the ASN.1 module, as it would have to be created for the xyzService illustrated in figure K-5, is shown below. It should be noted that this module does not reflect any real OID assignment; however, it is consistent with figure K-5. The formal requirements for allocating the OIDs can be found in annex D.

The OIDs defined for individual services shall be used in the BIND operation for identifying the service type (see `ServiceType` definition in F3.5).

```
CCSDS-XYZ-SERVICE-OBJECT-IDENTIFIERS
{   iso(1) identified-organization(3) standards-producing-organization(112)
    ccsds(4) css(4) csts(1) services(2) xyzService(200)
    xyzServiceModules(4) object-identifiers(1)
}

DEFINITIONS
IMPLICIT TAGS
::= BEGIN

IMPORTS    services
    FROM CCSDS-CSTS-OBJECT-IDENTIFIERS

           PublishedIdentifier
    FROM CCSDS-CSTS-COMMON-TYPES
;

-- Object Identifiers definition for the service types (see annex D)

-- ================================================================
-- XYZ SERVICE OBJECT IDENTIFIERS

xyzService                      OBJECT IDENTIFIER ::= {services 200}
xyzDerivedServices           OBJECT IDENTIFIER ::= {xyzService 1}
xyzExtendedServiceParameters    OBJECT IDENTIFIER ::= {xyzService 2}
xyzServiceProcedures            OBJECT IDENTIFIER ::= {xyzService 3}
xyzServiceModules            OBJECT IDENTIFIER ::= {xyzService 4}
xyzServiceFrRef                 OBJECT IDENTIFIER ::= {xyzService 5}

XyzServiceFrRef                                   ::= PublishedIdentifier

xyzExtSvcStartDataId            OBJECT IDENTIFIER ::= {xyzExtendedServiceParameters 1}
xyzExtSvcTransferDataId         OBJECT IDENTIFIER ::= {xyzExtendedServiceParameters 2}

badd                            OBJECT IDENTIFIER ::= {xyzServiceProcedures 1}
baddExtendedOpsParameters    OBJECT IDENTIFIER ::= {badd 1}
baddExtendedProcParameters      OBJECT IDENTIFIER ::= {badd 2}
baddEEventsId                   OBJECT IDENTIFIER ::= {badd 3}
baddDirectivesId                OBJECT IDENTIFIER ::= {badd 4}

END
```

The OIDs defined for Cross Support Resources cover the cross-support functionalities defined by CCSDS and the specific functionalities defined by the Agencies making use of Cross Support Services (see figure K-6).

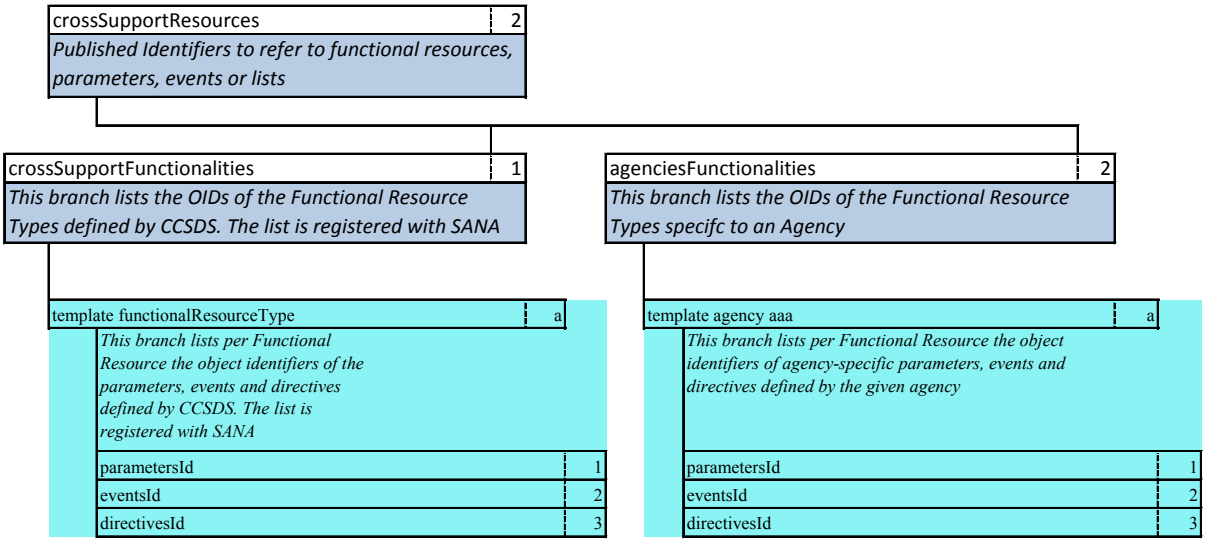

Figure K-6: CSTS Published Identifiers—Object Identifiers Tree

ANNEX L

PUBLISHED IDENTIFIERS FOR FUNCTIONAL RESOURCES REGISTERED UNDER THE CROSSSUPPORTFUNCTIONALITIES NODE

(INFORMATIVE)

An important aspect of cross support between agencies is that both parties have the same understanding of the applied configuration as well as of the status of service production and service provisioning. The details of service production in general depend on the specificities of the equipment deployed, and the make of equipment varies with the service provider. A common understanding of configuration and status therefore must not be equipment specific. Rather, a certain level of abstraction needs to be applied while maintaining an adequate level of detail to overcome equipment-specific representation of the required information. To that end, CCSDS has developed a data dictionary specifying parameters, events, and directives needed to configure and monitor Cross Support Service production and provisioning.

In addition to identifying suitable parameters as such, these need to be grouped such that one can model the existence of multiple instances of the same functionality. For example, within the scope of a service package, two physical return links may be processed, one in X-band and the other in Ka-band, each band handled by a different receiver. Consequently, one will need two instances of the parameters associated with a receiver. The granularity of the grouping of parameters must be chosen such that parameters associated with a functionality of which several instances may be required are grouped accordingly. Such a functionality is referred to as a Functional Resource.

Fundamental to the concept of Functional Resources is that each one represents a cohesive, atomic set of space communication functionality with which can be associated single instances of configuration parameters, monitored parameters, directives, and event notifications.

Functional Resources are not the physical resources (e.g., transmitters and receivers) that real systems comprise. Rather, they represent the functions or capabilities that are provided by those physical resources. A Functional Resource may be realized by several physical entities that work cooperatively to perform that function. Alternatively, for some types of Functional Resources, a single physical resource may be designed such that it instantiates several Functional Resources.

The concept of Functional Resources has been adopted as a core concept of this Recommended Standard, with standard Parameter Names being defined as having a Functional Resource identifier component. This Recommended Standard also defines a registration subtree for Functional Resource Type OIDs under the CCSDS registration tree. Besides monitored parameters, the Functional Resource registration tree is used to register OIDs for *notifiable events* and *directives* associated with each Functional Resource Type.

This annex is intended only to illustrate the registration tree by means of examples. The normative tree is registered with SANA, and the examples provided in this annex may deviate from the SANA registry, which will evolve over time, for example, because of the introduction of novel capabilities.

The root element of the registration tree is the crossSupportFunctionalResources node (cf. figure D-1). The branch under that node has a leaf for each of the Functional Resource Types. For each Functional Resource Type, the associated parameters, notifiable events, and directives are specified. A subset of the Functional Resources is shown in figure L-1.

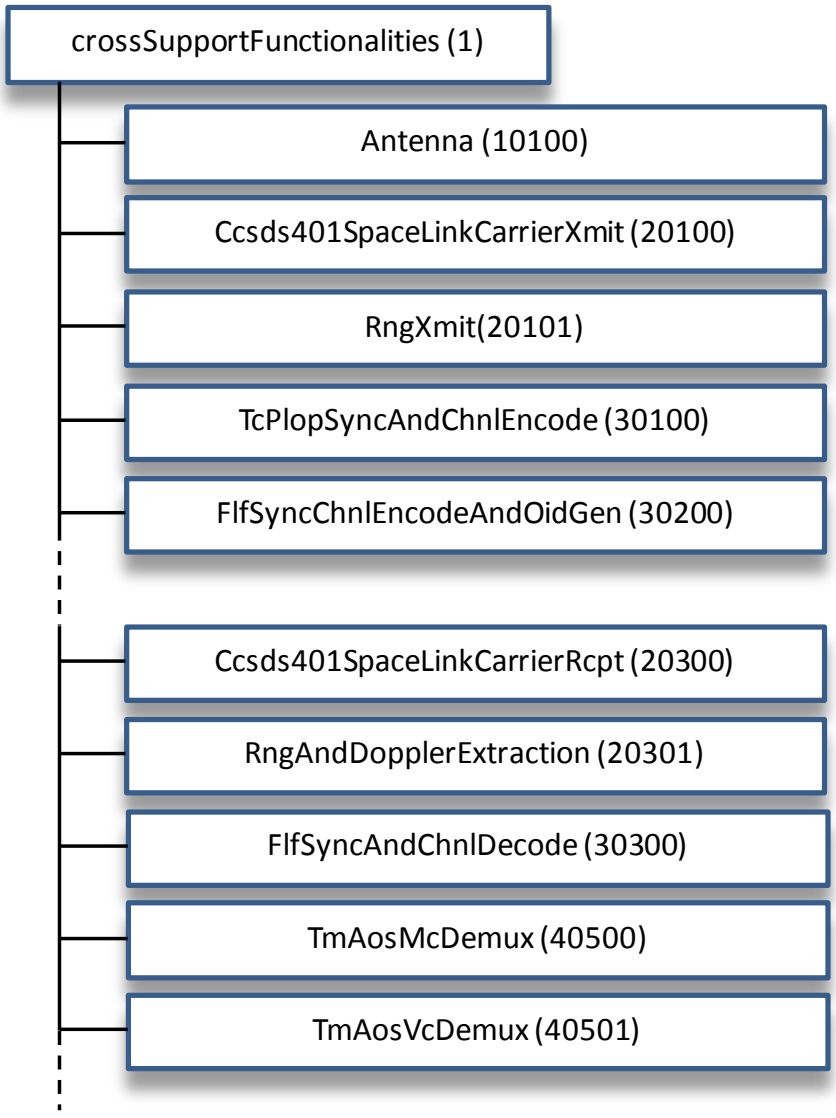

Figure L-1: Example Cross Support Functional Resources

Since the number of parameters associated with a subcarrier on a return space link is moderate, this subset of parameters of the rtn401SpaceLinkCarrierReception is taken here as an example. For this Functional Resource, neither notifiable events nor directives are discussed here. The parameters are shown in figure L-2.

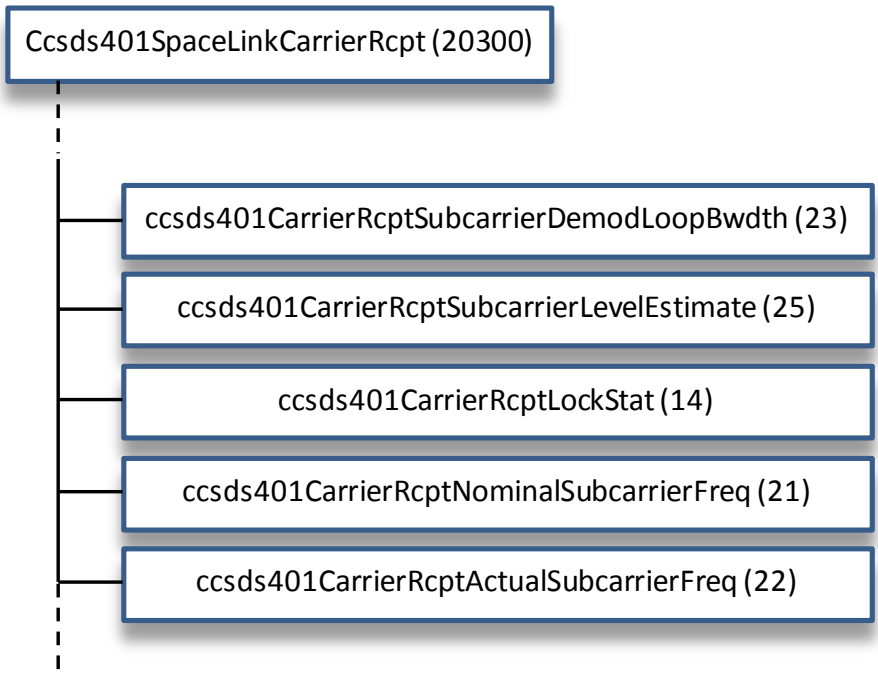

Figure L-2: Subcarrier Related Parameters of the Rtn401SpaceLinkCarrierRecpt Functional Resource

The structure shown so far illustrates how the OIDs are constructed, and with that information, individual Functional Resources and individual parameters can be accessed. For the information exchange between the parties participating in cross support, more information on the parameters is required. As specified in E11.1, for each parameter, event, and directive, the following items are to be specified:

a) a classifier;

b) a semantic definition;

c) a name in the form of an OID;

NOTE – If a suitable parameter already exists on a subbranch different from the one where the parent Functional Resource is registered, the OID specified on the other subbranch will be used to name such parameter.

d) a syntax specification;

NOTE – The syntax specification shown in tables L-1 and L-2 show how the parameter will be presented in the return of the GET operation querying that parameter. The Functional Resource specification will only provide the type specification, for example, `INTEGER (-20000 .. 0)`.

e) the value range;

f) the engineering unit(s);

g) a flag indicating if the definition has been deprecated;

NOTE – The deprecation flag indicates that at least one more recent version of the specification of the given parameter, event, or directive exists. The deprecated version can no longer be expected to be supported by providers of CSTSes.

h) a flag indicating if the given parameter can configured, that is, if the Functional Resource has a directive that permits the setting of the value of this parameter;

NOTE – This flag does not imply that in a specific cross-support scenario the given parameter can actually be set or modified by a CSTS user. The service provider may not have implemented such control capability or may decide not to grant such control in the context of a specific cross-support scenario. The parameters that can be configured by a CSTS user should be identified in a service agreement governing the cross-support arrangement.

i) the creation date.

Conceptually, all parameters specified for a given Functional Resource Type can be monitored. However, this does not imply that, in a specific cross-support scenario, the given parameter can actually be monitored by a CSTS user. The service provider may not generate that parameter or may decide not to disclose it in the context of a specific cross-support scenario. The parameters that are accessible by a CSTS user should be identified in a service agreement governing the cross-support arrangement.

Tables L-1 and L-2 illustrate this using two parameters of the Rtn401SpaceLinkCarrierRecpt Functional Resource Type. The semantic definition in table L-1 shows that the parameter is expressed in dB. This is not an engineering unit per se, but a way to scale a dimensionless ratio. To ensure that the applied scaling is not overlooked, dB is shown as the applicable engineering unit.

Table L-1: Specification of the Subcarrier Level Estimate Parameter

Classifier	ccsds401CarrierRcptSubcarrierLevelEstimate
Authorizing Entity	CSS Area
Creation Date	2020-03-30
Deprecation Flag	Not deprecated
Configured Flag	No
Object Identifier	1 3 112 4 4.2 1 20300 1 25 1
Semantic Definition	This parameter reports the subcarrier to carrier power ratio expressed in 1/100 dBc. If the applicable modulation scheme does not use a subcarrier, this parameter shall be flagged as 'undefined'.
Syntax Specification (see CCSDS 921.1)	‘QualifiedValue’: ‘valid’: “TypeAndValue’: ‘Embedded’: ’EMBEDDED PDV’. The PDV is expressed as the ASN.1 type Ccsds401CarrierRcptSubcarrierLevelEstimate ::= INTEGER (-20000 .. 0)
Value Range	Implied by syntax specification
Engineering Unit(s)	1/100 dBc

Table L-2: Specification of the Subcarrier Lock Status Parameter

Classifier	ccsds401CarrierRcptLockStat
Authorizing Entity	CSS Area
Creation Date	2020-03-30
Deprecation Flag	Not deprecated
Configured Flag	No
Object Identifier	1 3 112 4 4 2 1 20300 1 14 1
Semantic Definition	This parameter reports the lock status for the carrier, and, if applicable, for the subcarrier, and for the symbol stream.
Syntax Specification (see CCSDS 921.1)	'QualifiedValue': 'valid': 'TypeAndValue': 'Embedded': 'EMBEDDED PDV'. The PDV carries the ASN.1 type Ccsds401CarrierRcptLockStat ::= LockStat LockStat ::= SEQUENCE { carrierLock ENUMERATED { notLocked (0) , locked (1) } , subcarrierLock ENUMERATED { notLocked (0) , locked (1) , notApplicable (2) } , symbolStreamLock ENUMERATED { notLocked (0) , locked (1) } }
Value Range	Implied by syntax specification
Engineering Unit(s)	N/A

All the above shown information items can be found for all parameters in the associated registry maintained by SANA.

ANNEX M

ASN.1 CONSTRUCTION OF QUALIFIED PARAMETERS

(INFORMATIVE)

M1 INTRODUCTION

M1.1 OVERVIEW

Parameters of Cross Support Transfer Services can be represented in two forms: as *CSTS procedure parameters* or as *Functional Resource parameters*.

M1.2 CSTS PROCEDURE PARAMETER FORM

The *CSTS procedure parameter* form is used to represent the parameters of the procedures of the CSTS in the operations that are executed on the interface between the provider of an instance of that CSTS and the peer user entity of that service instance. CSTS procedure parameters are identified to the granularity of the individual procedure type (e.g., Data Processing procedure) and procedure instance (prime, secondary instance, or association control). Every implementation of a CSTS must support access (e.g., through inclusion of a Cyclic Report and/or Information Query procedure) to the CSTS procedure parameters that are defined for that CSTS.

NOTE – A CSTS may have optional procedures that may or may not be present in a given implementation. For such a CSTS, an implementation is only required to include the CSTS procedure parameters that belong to the procedures that are implemented.

The CSTS procedure form of a parameter can be thought of as an internal representation of the parameter, in that it is a form used within the association between the service provider and service user.

CSTS procedure parameters use the Object Identifiers (OIDs) assigned to the procedure parameter identifiers and associated data types that are defined in the specification of the given CSTS. These procedure parameters may be inherited from the parent procedures specified in this Specification Framework (e.g., the Data Processing procedure configuration parameters specified in table 4-26), or they may be defined for the derived or service-specific procedures of the CSTS.

M1.3 FUNCTIONAL RESOURCE PARAMETER FORM

The Functional Resource parameter form is used to represent parameters of the Functional Resource that represents the CSTS instance as a whole. The Functional Resource form is used to represent the parameters of a Functional Resource as accessed through means that are *external to* that Functional Resource. In particular, the Functional Resource form of parameters is used to access the *service management parameter*s of those Functional Resources, and is the form of representation used by the Monitored Data CSTS (reference [I11]) and the planned Service Control CSTS.

Unlike the CSTS procedure parameter form, which identifies procedure details which are specific to CSTSes, the Functional Resource form presents a "black box" view of the parameters because this form is used to represent parameters of all Functional Resource types (of which the CSTSes are a subset). The Functional Resource form identifies the Functional Resource type and Functional Resource Instance Number (FRIN) of the Functional Resource instance to which the parameter belongs.

The Functional Resource form of a parameter can be thought of as the external representation of the parameter, in that it is the form used to represent the parameter to entities outside of the Functional Resource itself.

Functional Resource parameters use the OIDs assigned to the parameter identifiers and associated data types that are specified in the definition of the given Functional Resource in the SANA registry named 'Functional Resources' located at https://sanaregistry.org/r/functional_resources.

M2 QUALIFIED PARAMETERS IN THE CSTS PROCEDURE PARAMETER AND FUNCTIONAL RESOURCE PARAMETER FORMS

M2.1 OVERVIEW

As specified in annex E, a `qualified-parameter` parameter contains for each reported parameter the following information:

a) the Parameter Name, which is defined as the combination of its Parameter Identifier and either its Functional Resource Name or its procedure name;

b) the parameter type;

c) the parameter value(s); and

d) the parameter qualifier.

The Parameter Name element of the qualified parameter contains the Functional Resource Name if the parameter is being reported in the context of the Functional Resource that represents the CSTS or the procedure name if the parameter is a procedure parameter that is being reported to the peer user entity of the CSTS provider instance.

The following subsections provide examples of the ASN.1 structure of qualified parameters, first for the CSTS procedure parameter form, followed by the Functional Resource parameter form. The examples use the same underlying parameter, represented in the first case as qualified parameter using the CSTS procedure parameter form, and in the second case using the functional resource form.

The CSTS procedure parameter and Functional Resource form examples use the `initiator-identifier` parameter of the Association Control (AC) procedure of the Forward Frame CSTS. This is a configuration parameter that is inherited from the Framework AC procedure (see table 4-2). The Forward Frame CSTS configures the AC procedure parameter `initiator-identifier` through the service management parameter `ffInitiatorId`. Both examples illustrate the reporting of 'valid' values of the parameter being reported.

M2.2 CSTS PROCEDURE PARAMETER FORM OF THE QUALIFIED PARAMETER FOR THE INITIATOR-IDENTIFIER PARAMETER

Beginning with the ASN.1 types from F3.3 that are used to define the qualified parameter:

```
QualifiedParameter       ::=    SEQUENCE
{   parameterName     Name
,   qualifiedValues   SequenceOfQualifiedValue
}

Name                     ::=    SEQUENCE
{   fRorProcedureName              FRorProcedureName
,   paramOrEventOrDirectiveId      PublishedIdentifier
}

FRorProcedureName     ::=    CHOICE
{   functionalResourceName       [0]    FunctionalResourceName
,   procedureName                [1]    ProcedureName
}

ProcedureName         ::=    SEQUENCE
{   procedureType                ProcedureType
,   procedureRole                CHOICE
    {   primeProcedure           [0]    NULL
    ,   secondaryProcedure       [1]    IntPos
    ,   associationControl       [2]    NULL
    }
}

-- The ProcedureType is an Object Identifier, the allocation of which is
-- under control of CCSDS. It is declared in the ASN.1 module
-- CCSDS-CSTS-OBJECT-IDENTIFIERS (see F3.1).
ProcedureType         ::=    OBJECT IDENTIFIER

PublishedIdentifier   ::=    OBJECT IDENTIFIER

SequenceOfQualifiedValue ::=    SEQUENCE OF QualifiedValue
```

```
QualifiedValue        ::=    CHOICE
{   valid                [0]    TypeAndValue -- Valid value
,   unavailable          [1]    NULL   -- Unknown or unavailable value
,   undefined            [2]    NULL   -- Undefined in the context
,   error                [3]    NULL   -- Processing resulted in an error
}

TypeAndValue          ::=    Embedded

Embedded              ::=    EMBEDDED PDV

IntPos                ::=    INTEGER (1 .. 4294967295)
```

As described in F2.3, only the 'syntax' choice of the EMBEDDED PDV type is applicable to CSTSes, so the effective form of that data type is:

```
EMBEDDED PDV ::= [UNIVERSAL 11] SEQUENCE
{   identification              CHOICE
    {   syntaxes                        SEQUENCE
        {   abstract                        OBJECT IDENTIFIER
        ,   transfer                        OBJECT IDENTIFIER
        }
    ,   syntax                          OBJECT IDENTIFIER
    ,   presentation-context-id         INTEGER
    ,   context-negotiation             SEQUENCE
        {   presentation-context-id         INTEGER
        ,   transfer-syntax                 OBJECT IDENTIFIER
        }
    ,   transfer-syntax                 OBJECT IDENTIFIER
    ,   fixed                           NULL
    }
,   data-value                  OCTET STRING
}
```

The procedure that contains the parameter is the FF-CSTS AC procedure, which is directly adopted from the Framework. From F3.1, we get the OID for the ProcedureType for the AC procedure:

```
associationControl                OBJECT IDENTIFIER ::= {procedures 1}
```

Expanding the OID for the `associationControl` procedure type results in

```
{css (1 3 112 4 4}   csts(1)   framework(1)   procedures(3)    1}
= {1.3.112.4.4.1.1.3.1}
```

Because the FF-CSTS directly adopts the Framework AC procedure, the FF-CSTS AC procedure uses the `initiator-identifier` parameter from the Framework AC procedure. Because this is the CSTS procedure parameter form of the qualified parameter, the OID for the AC procedure parameter `pACinitiatorId` (table 4-2 and F3.16) is used for both the `parameterName` and `EMBEDDED PDV` syntax OIDs (`pACinitiatorId` is

the name of the procedure parameter as well as the OID associated with the data type `PACinitiatorId`). From F3.16:

```
pACinitiatorId          OBJECT IDENTIFIER    ::=    {pACparametersId 1}
PACinitiatorIdType                          ::=    AuthorityIdentifier
```

Expanding the OID for `pACinitiatorId` results in

```
{css(1 3 112 4 4)  csts(1) framework(1)   fwProceduresFunctionalities(4)
procAssociationControl(1)  pACparametersId(1)  1} = {1.3.112.4.4.1.1.4.1.1.1}
```

For the AC procedure, the procedure role is 'associationControl'.

Expanding the `parameterName` element of the `QualifiedParameter` type and eliminating (by strikethrough) the choices that are not applicable to the procedure parameter form:

```
parameterName     SEQUENCE
{   fRorProcedureName                CHOICE
    {   functionalResourceName       [0]    FunctionalResourceName
    ,   procedureName                [1]    SEQUENCE
        {   procedureType               OBJECT IDENTIFIER =  {1.3.112.4.4.1.1.3.1}
               -- OID for Association Control procedure
        ,   procedureRole               CHOICE
            {   primeProcedure          [0]    NULL
            ,   secondaryProcedure      [1]    IntPos
            ,   associationControl      [2]    NULL
            }
        }
    }
,   paramOrEventOrDirectiveId       OBJECT IDENTIFIER = {1.3.112.4.4.1.1.4.1.1.1}
-- OID for pACinitiatorId
}
```

For this example, the value of the parameter `initiator-identifier` is 'XenoFfUser', encoded as ASN.1 type VisibleString where the constraint of the alphabet does not permit spaces to be part of the string. The related ASN.1 Type specifications are from F3.3

```
AuthorityIdentifier      ::=    IdentifierString (SIZE (3 .. 16))

IdentifierString         ::=    VisibleString (FROM (ALL EXCEPT " "))
```

Expanding the `qualifiedValues` element of the `QualifiedParameter` type and excluding (by strikethrough) the choices that are not relevant to the 'valid' case :

```
qualifiedValues   SEQUENCE OF
    CHOICE – The parameter value is valid
    {   valid                 [0]    [UNIVERSAL 11] SEQUENCE
        {   identification              CHOICE
            {   syntaxes                        SEQUENCE
            {   abstract                            OBJECT IDENTIFIER
            ,   transfer                            OBJECT IDENTIFIER
            }
        ,   syntax                          OBJECT IDENTIFIER = {1.3.112.4.4.1.1.4.1.1.1
                                            -- OID for pACinitiatorId

        ,   presentation-context-id         INTEGER
        ,   context-negotiation             SEQUENCE
            {   presentation-context-id         INTEGER
            ,   transfer-syntax                 OBJECT IDENTIFIER
            }
        ,   transfer-syntax                 OBJECT IDENTIFIER
        ,   fixed                           NULL
            }
        ,   data-value                  OCTET STRING  = ('1A0A58656E6F466655736572'H)
                                            -- The octet string in this case (BER
                                            -- encoding) is
                                            -- 1A 0A 58 65 6E 6F 46 66 55 73 65 72 hex
                                            -- where 1A specifies the data type
                                            -- (VisibleString), 0A is the short
                                            -- form length of the value (10 octets) and
                                            -- 58 65 6E 6F 46 66 55 73 65 72
                                            -- is the ASCII representation of the value
                                            -- (the string 'XenoFfUser') of the
                                            -- PACinitiatorId parameter.
        }
    ,   unavailable             [1]     NULL    -- Unknown or unavailable value
    ,   undefined               [2]     NULL    -- Undefined in the context
    ,   error                   [3]     NULL    -- Processing resulted in an error
    }
```

Combining the `parameterName` and `qualifiedValues` elements into the `QualifiedParameter` type, and again eliminating the non-applicable choices:

```
QualifiedParameter       ::= SEQUENCE
{   parameterName     SEQUENCE
    {   fRorProcedureName               CHOICE
        {   functionalResourceName      [0]    FunctionalResourceName
        ,   procedureName               [1]    SEQUENCE
            {   procedureType               OBJECT IDENTIFIER =  {1.3.112.4.4.1.1.3.1}
                                            -- OID for Association Control procedure
            ,   procedureRole               CHOICE
                {   primeProcedure      [0]    NULL
                ,   secondaryProcedure  [1]    IntPos
                ,   associationControl  [2]    NULL
                }
            }
        }
    }
        ,   paramOrEventOrDirectiveId       OBJECT IDENTIFIER = {1.3.112.4.4.1.1.4.1.1.1} -
            - OID for pACinitiatorId
}
,   qualifiedValues   SEQUENCE OF
        CHOICE - The parameter value is valid
        {   valid                [0]    [UNIVERSAL 11] SEQUENCE
            {   identification              CHOICE
                {   syntaxes                    SEQUENCE
                {   abstract                        OBJECT IDENTIFIER
                ,   transfer                        OBJECT IDENTIFIER
                }
                ,   syntax                      OBJECT IDENTIFIER =
                          {1.3.112.4.4.1.1.4.1.1.1} -- OID for pACinitiatorId
                ,   presentation-context-id     INTEGER
                ,   context negotiation         SEQUENCE
                {   presentation-context-id         INTEGER
                ,   transfer-syntax                 OBJECT IDENTIFIER
                }
                ,   transfer-syntax             OBJECT IDENTIFIER
                ,   fixed                       NULL

                }
            ,   data-value                  OCTET STRING = ('1A0A58656E6F466655736572'H)
            }
        ,   unavailable          [1]    NULL    -- Unknown or unavailable value
        ,   undefined            [2]    NULL    -- Undefined in the context
        ,   error                [3]    NULL    -- Processing resulted in an error
        }
}
```

M2.3 FUNCTIONAL RESOURCE PARAMETER FORM OF THE QUALIFIED PARAMETER FOR THE INITIATOR-IDENTIFIER PARAMETER

The Functional Resource form of the qualified parameter uses the same ASN.1 data types and OIDs as the CSTS procedure parameter form, except that instead of the `ProcedureName` and `ProcedureType` data types the `FunctionalResourceName`, `FunctionalResourceInstanceNumber`, and `FunctionalResourceType` data types are used. From F3.3:

```
FunctionalResourceName          ::=    SEQUENCE
{   functionalResourceType                FunctionalResourceType
,   functionalResourceInstanceNumber      FunctionalResourceInstanceNumber
}

FunctionalResourceInstanceNumber    ::=    IntPos

FunctionalResourceType    ::=    OBJECT IDENTIFIER
```

For the Forward Frame CSTS Provider Functional Resource, the data type of the `ffInitiatorId` parameter is `FfInitiatorId`, defined as

```
FfInitiatorId   ::=   AuthorityIdentifier
```

where

```
AuthorityIdentifier ::=  VisibleString (FROM (ALL EXCEPT " ")) (SIZE (3 .. 16))
```

For the Functional Resource form of the parameter name, the value of the `paramOrEventOrDirectiveId` element is the OID assigned to the `ffInitiatorId` parameter of the Forward Frame CSTS Provider Functional Resource. For the purposes of this example, the following OID assignments are assumed for the Functional Resource Type of the Forward Frame CSTS Provider Functional Resource (classifier `ffCstsProvider`), the `ffInitiatorId` parameter, and the `ffInitiatorId` data type:

a) the OID for `ffCstsProvider` is {1.3.112.4.4.2.1.80300},

b) the OID assigned to `ffInitiatorId` is {1.3.112.4.4.2.1.80300.1.4.1}, and

c) for Functional Resource parameters, the data type of the parameter has its own OID. The `FfInitiatorId` data type has the "typeOID" (syntax OID) {1.3.112.4.4.2.1.80300.1.4.1.1}.

For this example, the FRIN is arbitrarily assumed to be 3.

Expanding the parameterName element of the QualifiedParameter type and excluding (by strikethrough) the choices that are not relevant to the Functional Resource form:

```
parameterName     SEQUENCE
{   fRorProcedureName  CHOICE -- only the functionalResourceName choice
                                  -- applies
    {   functionalResourceName   [0] SEQUENCE
        {   functionalResourceType   PublishedIdentifier = {1.3.112.4.4.2.1.80300}
                                        -- OID for FR Type FwdFrameCstsProvider
        ,   functionalResourceInstanceNumber     IntPos = 3
        }
    ,   procedureName      [1]    ProcedureName
    }
,   paramOrEventOrDirectiveIdOBJECT IDENTIFIER = {1.3.112.4.4.2.1.80300.1.4.1}
                                     -- OID for ffInitiatorId parameter
}
```

NOTE – The Functional Resource form of the qualified value uses a separate ‘typeOID’ value for the syntax subelement of the qualifiedValues element. This is different from the CSTS procedure form of the qualified parameter, which uses the same procedure parameter OID for both the paramOrEventOrDirectiveId subelement of the parameterName element and the syntax subelement of the qualifiedValue element.

Expanding the `qualifiedValues` element of the `QualifiedParameter` type and excluding (by strikethrough) the choices that are not relevant to the 'valid' case:

```
qualifiedValues   SEQUENCE OF
   CHOICE - The parameter valuie is valid
   {  valid                [0]    [UNIVERSAL 11] SEQUENCE
      {  identification              CHOICE
         {  ~~syntaxes~~                        ~~SEQUENCE~~
         ~~{  abstract                          OBJECT IDENTIFIER~~
         ~~,  transfer                          OBJECT IDENTIFIER~~
         ~~}~~
         ~~,~~ syntax                           OBJECT IDENTIFIER =
                  {1.3.112.4.4.2.1.80300.1.4.1.1} -- OID for FfInitiatorId data type
         ~~,  presentation-context-id       INTEGER~~
         ~~,  context-negotiation           SEQUENCE~~
         ~~{  presentation-context-id       INTEGER~~
         ~~,  transfer-syntax               OBJECT IDENTIFIER~~
         ~~}~~
         ~~,  transfer-syntax               OBJECT IDENTIFIER~~
         ~~,  fixed                         NULL~~
         }
      ,  data-value                  OCTET STRING = ('1A0A58656E6F466655736572'H)
                                        -- The octet string in this case (BER
                                        -- encoding) is
                                        -- 1A 0A 58 65 6E 6F 46 66 55 73 65 72 hex
                                        -- where 1A specifies the data type
                                        -- (VisibleString), 0A is the short
                                        -- form length of the value (10 octets) and
                                        -- 58 65 6E 6F 46 66 55 73 65 72
                                        -- is the ASCII representation of value (the
                                        -- string 'XenoFfUser') of the
                                        -- FfInitiatorId parameter.
      }

   ~~,  unavailable          [1]    NULL    -- Unknown or unavailable value~~
   ~~,  undefined            [2]    NULL    -- Undefined in the context~~
   ~~,  error                [3]    NULL    -- Processing resulted in an error~~
   }
```

Combining the parameterName and qualifiedValues elements into the QualifiedParameter type:

```
QualifiedParameter       ::= SEQUENCE
{   parameterName     SEQUENCE
    {   fRorProcedureName  CHOICE -- only the functionalResourceName choice
                                   -- applies
        {   functionalResourceName    [0] SEQUENCE
            {   functionalResourceType   PublishedIdentifier = {1.3.112.4.4.2.1.80300}
                                            -- OID for FR Type FwdFrameCstsProvider
            ,   functionalResourceInstanceNumber     IntPos = 3
            }
        ,   procedureName      [1]     ProcedureName
        }
    ,   paramOrEventOrDirectiveIdOBJECT IDENTIFIER = {1.3.112.4.4.2.1.80300.1.4.1}
                                        -- OID for ffInitiatorId parameter
    }
,   qualifiedValues   SEQUENCE OF
        CHOICE - The parameter value is valid
        {   valid                [0]    [UNIVERSAL 11] SEQUENCE
            {   identification               CHOICE
                {   syntaxes                        SEQUENCE
                {   abstract                        OBJECT IDENTIFIER
                ,   transfer                        OBJECT IDENTIFIER
                }
            ,   syntax                          OBJECT IDENTIFIER =
                    (1.3.112.4.4.2.1.80200.1.4.1.1) -- OID for FfInitiatorId data type
            ,   presentation-context-id         INTEGER
            ,   context-negotiation             SEQUENCE
                {   presentation-context-id      INTEGER
                ,   transfer-syntax              OBJECT IDENTIFIER
                }
            ,   transfer-syntax                 OBJECT IDENTIFIER
            ,   fixed                           NULL
                }
            ,   data-value                   OCTET STRING = ('1A0A58656E6F466655736572'H)
                                             -- ('1A0A58656E6F466655736572'H) ('020105'H)
                                             -- is the BER encoding of
                                             -- VisibleString (value of 'XenoFfUser'),
                                             -- which is the base type of the
                                             -- FfInitiatorId data type.
            }
        ,   unavailable          [1]     NULL    -- Unknown or unavailable value
        ,   undefined            [2]     NULL    -- Undefined in the context
        ,   error                [3]     NULL    -- Processing resulted in an error
        }
}
```