The CORDET Framework
PUS Extension

P&P Software GmbH
High Tech Center 1
8274 Tägerwilen (CH)

Web site: www.pnp-software.com
E-mail: pnp-software@pnp-software.com

Written By:	Alessandro Pasetti
Checked By:	Marcel Opprecht
Document Ref.:	PP-DF-COR-0003
Issue:	0.2 (In Progress)
Created On:	30/01/2019, at: 12:55

Contents

List of Figures

List of Tables

1 Change History

This section lists the changes made in the current and previous revisions. Changes are classified according to their type. The change type is identified in the second column in the table according to the following convention:

- "**E**": Editorial or stylistic change
- "**L**": Clarification of existing text
- "**D**": A requirement has in whole or in part been deleted
- "**C**": A requirement has been modified
- "**N**": A new requirement has been introduced
- "**T**": A TBD or TBC has been resolved

Text which is new or has been modified in the current revision is in red font. If a figure has been modified, then its caption is in red font. Section header numbers do not change from one revision to the next (but new sections may, of course, be introduced). However, figure and table numbers may change and these changes are not tracked. Changes in the appendices are not tracked as they are consequences of changes in the main body of the document.

Table 1.1: Detailed List of Changes in Issue 0.1

Sec	Type	Description of Change
n.a.	L	This is the first release of this document

2 Applicable and Reference Documents

The documents in table 2.1 form an integral part of the present document. The documents in table 2.2 are referenced in the present document and are for information only.

Table 2.1: Applicable Documents

ID	Title, Reference Number, Revision Number
[CR-SP]	The CORDET Framework - Specification, PP-DF-COR-00002, Revision 1.6, P&P Software GmbH, Switzerland, 2012, Available from: www.pnp-software.com/cordetfw
[FW-SP]	The Framework Profile, PP-DF-COR-00001, Revision 1.3, P&P Software GmbH, Switzerland, 2012, Available from: www.pnp-software.com/fwprofile
[PS-SP]	Ground Systems and Operations – Telemetry and Telecommand Packet Utilization Standard, ECSS-E-70-41C, April 2016, European Cooperation for Space Standardization (ECSS)
[PX-SP]	The present document

Table 2.2: Reference Documents

ID	Title, Reference Number, Revision Number
[PS-WEB]	The CORDET Framework Project Web Site, www.pnp-software.com/cordetfw

3 Introduction

The CORDET Framework is a software framework for service-oriented embedded applications. It is specified in [CR-SP] as a set of components to manage the services which an application provides to other applications and uses from other applications. A C-language implementation of this specification is available from [CR-WEB].

The CORDET Framework only covers the management of generic services but does not specify any concrete services. The service concept of the CORDET Framework is the same as the service concept of the *Packet Utilization Standard* (PUS). The PUS is an application-level interface standard for space-based distributed systems. It is defined in [PS-SP].

The PUS pre-defines a number of services. This document extends the CORDET Framework to support a subset of those services. The document specifies the components which implements them. The components are specified by providing their behavioural model. The behavioural models are defined using the FW Profile. The FW Profile is a UML profile for reusable software components. It is defined in [FW-SP].

The set of components specified in this document are called the *PUS Extension of the CORDET Framework*. When there is no danger of ambiguity, the shorter names "framework extension" or "PUS extension" are also used as synonyms of PUS Extension of the CORDET Framework.

In terms of the classical software lifecycle, the specification presented in this document is at the level of software requirements in the sense that it defines a complete and unambiguous logical model of the components implementing the PUS extension of the CORDET Framework.

This document assumes the reader to be familiar with the specification of the CORDET Framework in [CR-SP].

3.1 Scope of CORDET Framework

A *CORDET service* is a set of logically and functionally related capabilities that an application offers to other applications. The CORDET Service concept sees an application as a *provider of services* to other applications and as a *user of services* from other applications (see figure 3.1).

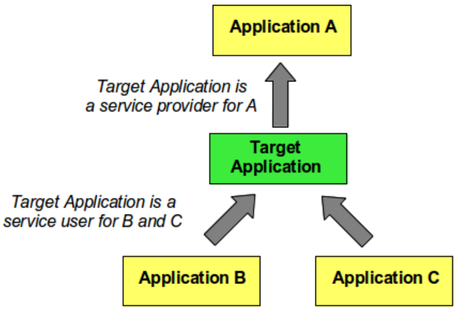


Fig. 3.1: Applications as Providers and Users of Services

The user of a service controls the service by sending *commands* to the service provider. A command is a data exchange between a service user and a service provider to control the execution of a particular activity within the service provider.

The provider of a service sends *reports* to the user of the service. A report is a data exchange between a service provider (the report initiator) and a service user to provide information relating to the execution of a service activity.

Thus, a service consists of a set of commands which the user of the service sends to the provider of the service and of a set of reports which the service provider sends back to its user. A command defines actions to be executed by the service provider. A report encapsulates information about the internal state of the service provider.

Against this background, the CORDET Framework of [CR-SP] fulfils two objectives:

- It provides a formal definition of the abstract command concept and of the abstract report concept by building behavioural models of commands and reports which:
  - capture the aspects of the behaviour of commands and reports which are common to all commands and reports, and
  - identify the adaptation points where service- and implementation-specific behaviour can be added.
- It specifies the component (the *CORDET Components*) which implement the abstract command and report concept.

The CORDET Components cover, on the service user side, the sending of commands and the reception and distribution of reports and, on the service provider side, the processing of incoming commands and the generation of reports but do not cover the implementation of any concrete services.

3.2 Scope of PUS Extension of CORDET Framework

Developers of a CORDET application are expected to deploy the CORDET components and complement them with application-specific components which implement the services of interest to them. The PUS extension of the CORDET Framework facilitates the task of application developers by offering them a set of pre-defined components which implement a set of *Standard Services*. A standard service in this context is a service which implements commonly used functions within a certain domain.

The standard services of the PUS Extension are taken from the Packet Utilization Standard (PUS) of [PS-SP]. The target domain of the PUS Extension is therefore that of space-borne service-provider applications but it is worth stressing that the set of services selected from the PUS are those which are least dependent on the space context and it is therefore expected that the services implemented by the PUS Extension may be of interest to other application domains.

The standard services are defined by defining their commands and reports and the commands and reports are defined as specializations of the abstract command and report concepts of the CORDET Framework. Thus, a standard service is defined by “closing” the adaptation points identified in the abstract command and report concepts.

The CORDET Framework is ultimately intended to foster reuse (at both specification and implementation level) in the field of service-oriented embedded applications. The reuse model it promotes is illustrated in figure 3.2. At the top layer, there is the abstract definition of commands and reports of the CORDET Framework of [CR-SP]. This definition is entirely generic and applicable to all services in all applications. At the intermediate level, standard services are defined which capture concrete behaviour which is common to a large number of applications. The present document specifies one such set of standard services. Finally, at the bottom level, end-applications define their own services which are entirely specific to their needs. The application-level services may be either taken over from the standard services or they may be created as instantiations of the generic service concept (if they are entirely application-specific).

Note that the PUS Extension of the CORDET Framework specifies several services. These services are specified to be independent of each other so that the user may choose only a subset of these services. Similarly, each service is specified in terms of the commands and reports which implement it. Dependencies among the commands and reports of a service are minimized so that users may be free to import into their application just a subset of the commands and reports of a given service.

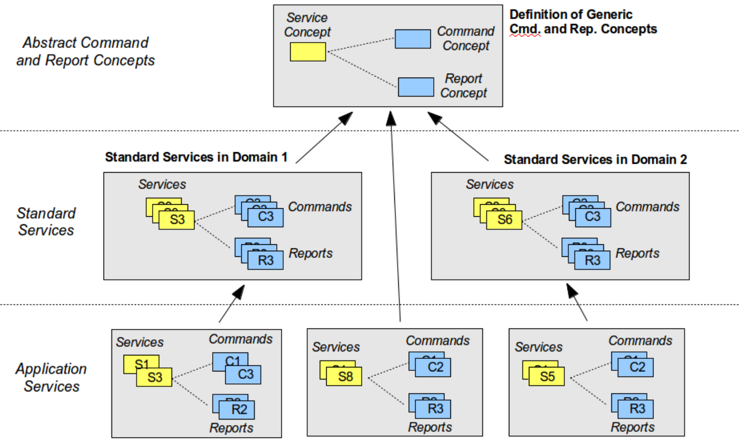


Fig. 3.2: Hierarchical Definition of Services

3.2.1 Overview of Supported Services

Table 3.1 lists the services supported by the PUS Extension of the CORDET Framework. The first column gives the service type identifier. The last column points to the section in this document where the support for the service is specified.

Table 3.1: Services Supported by PUS Extension

N	Service	Section
1	Request Verification Service	8
3	Housekeeping Service	9
5	Event Reporting Service	10
12	On-Board Monitoring Service	12
13	Large Packet Transfer Service	13
17	Test Service	14
19	Event Action Service	15

3.3 Specification Format

This document specifies the PUS Extension of the CORDET Framework. The framework is specified by defining its requirements. The requirements of the framework are of four types:

- *Standard Requirements* which define a desired feature of the framework extension. They are analogous in scope and format to the user requirements of a conventional (non-framework) application.
- *Adaptation Requirement* which define the points where a component offered by the framework extension can be extended by the application developers. In some cases, the definition of an adaptation point is accompanied by the definition of the default options offered by the framework extension for that adaptation point.
- *Use Constraint Requirements* which define the constraints on how the components offered by the framework extension may be used by application developers.
- *Property Requirements* which define behavioural properties which are guaranteed to hold on all applications which: (a) are instantiated from the CORDET Framework and its extension by closing their adaptation points, and (b) comply with the framework's use constraints.

To each framework requirement an *identifier* is attached. The requirement identifier takes the following form: x-y/t where 'x' is an acronym identifying the function to which the requirement applies; 'y' is a unique identifier within that function; and 't' identifies the requirement type. The type is designated by one single letter as follows: 'S' for the Standard Requirements, 'A' for the Adaptation Requirements, 'C' for the Use Constraint Requirements and 'P' for the Property Requirements.

The specification of the framework extension includes a *behavioural model* of the framework which describes its behaviour and identifies the adaptation points where application developers can extend this behaviour to match their requirements.

The behavioural model of the framework extension is defined using the FW Profile of [FW-PS]. It therefore consists of a set of *state machines* (represented as state charts) and *procedures* (represented as activity diagrams). Familiarity with the FW Profile is essential for a full understanding of the framework requirements.

Wherever possible, the framework extension requirements simply make the state machines and procedures applicable. In other words, the state charts representing state machines and the activity diagrams representing procedures are treated as normative and no attempt is made to translate them into a comprehensive set of equivalent requirements.

In accordance with the FW Profile, the activity diagrams and state diagrams identify the framework adaptation points using the $\ll AP \gg$ stereotype (but note that not all adaptation points are identified explicitly in activity or state diagrams). For convenience, all adaptation points with their default options are listed in dedicated tables. In most cases, the adaptation requirements simply make the items in such tables applicable. By default, the implementation mechanism for the adaptation points is left open and is not covered by this specification.

Some of the components specified by the framework extension are defined as extensions of CORDET components. In such cases, the extended component is derived from the base component by either *overriding* or *closing* some of its adaptation points. A derived component overrides an adaptation point of its base component when it changes the default behaviour associated to that adaptation point (but applications can still change that behaviour). A derived component closes an adaptation point of its base component when it defines in a final way the behaviour associated to that adaptation point (i.e. applications can no longer change that behaviour).

3.4 Compliance to PUS Requirements

The PUS Extension of the CORDET Framework implements a subset of the standard PUS services of [PS-SP]. In order to provide visibility over the level of compliance to the PUS requirements of [PS-SP], appendix D presents a statement of compliance to these requirements. This demonstrates that, for the selected services, the PUS Extension is compliant to the PUS requirements. Some points related to the compliance to the PUS deserve a special discussion which is presented below.

There are some terminological differences between PUS and CORDET. For clarity, table 3.2 lists PUS-specific terms and gives the corresponding term or concept in the CORDET world.

3.4.1 Multi-Instruction Requests

In the PUS, a request (command) contains one or more instructions. In the CORDET Framework, the concept of Instruction does not exist: instructions are implicitly embedded within commands. Instructions therefore only arise in the definition of the individual commands. With reference to clause 5.3.3.2, two points need to be noted. Firstly, for multi-instruction commands, the PUS Extension does not impose an upper boundary on the number of instructions in a command. Such an upper boundary, if needed, must be enforced by the user (e.g. in the SRDB). Secondly, for commands pre-defined by the PUS Extension, if a command can hold more than one instruction, then all these instructions are of the same type (i.e. a situation where the same command instance may hold instructions of different types is not allowed).

3.4.2 Acknowledgement Flags

In the PUS, each request (command) carries four flags which determine whether successful acceptance, start, progress and completion of that request should be reported to the request originator. The CORDET Framework defines four flags with the same semantics. It is important to stress that, in accordance with clause 5.4.11.2.2, the acknowledge flags only concern the reporting of verifications performed at the level of the request. The PUS is silent about the conditions under which the outcome of instruction-level verifications should be reported. The PUS Extension by-passes this issue by mapping instruction-level checks to request-level checks (see next point).

3.4.3 Verification of Multi-Instruction Requests

The request execution model of the PUS foresees the generation of verification reports both in response to request-level execution checks and in response to instruction-level execution checks[1]. The request-level verification is covered by the CORDET Framework: the Start Action, Progress Action and Termination Action of an InCommand have an outcome which determine whether the command is successfully started, executed or terminated. The

[1] See, for instance, clauses 5.3.5.2.3a and b which specify that start of execution must be verified both for a request as a whole and for the instructions it contains

CORDET Framework ensures that a verification report is generated in response to each execution outcome. The PUS Extension of the CORDET Framework adds instruction-level verification reports as follows:

- For requests which only contain one single instruction, the instruction-level verification check is subsumed in the request-level check.
- For requests which contain multiple instructions which are verified together (i.e. a request passes a verification stage only if all instructions pass the same verification stage), the instruction-level verification check is subsumed in the request-level check.
- For requests which contain multiple instructions which are verified individually, then the instructions are treated as accepted and successfully started if the request was accepted and successfully started and each instruction is executed in a dedicated progress step. The latter implies that success and failure of execution of an instruction are reported through progress-level execution reports. If one or more instructions have failed their execution, then the request is considered to have failed its execution.

As an example of the last bullet, consider the (3,5) command to enable a set of housekeeping reports. This command carries the SIDs of the reports to be enabled. Each SID defines one 'instruction'. Each SID is processed in a dedicated progress step. If a SID is illegal, then a (1,6) report is generated. If a SID is legal, then it is enabled and, if acknowledgement of successful progress-of-execution is enabled, a (1,5) report is generated. After all SIDs have been processed, the request is declared successful if all SIDs were legal and were enabled (in which case, if acknowledgement of successful completion-of-execution is enabled, a (1,7) report is generated); if, instead, one or more SIDs were illegal, then the request is declared to have failed its execution and a (1,8) is generated for it. Note that, with this approach, it is possible to have a situation where a (1,8) report is generated for a request but some instructions in that request have been successfully executed.

3.4.4 Reporting Failed Progress of Execution

Clause 5.4.9a gives a choice between reporting failed progress of execution through Failed-Progress-Of-Execution notification reports or through Completion-Of-Execution notification reports. In general, both options are compatible with the CORDET Framework: in the former case the notification report is generated by the Report Progress Failed Operation of the framework (adaptation point ICM-14); in the latter case, the notification is generated by the Report Termination Failed Operation of the framework (adaptation point ICM-16). By default, the PUS Extension chooses the former option but application developers can override this choice if they wish.

3.4.5 Disabling Failure Verification Reports

The PUS is not always clear about the conditions for the generation of service 1 reports in response to the commands from its pre-defined services. The approach taken by the PUS Extension is to generate a wide range of verification reports. At instantiation time, applications can restrict this range by selectively disabling verification reports through the enable mechanism of the OutRegistry component of the CORDET Framework. It is recalled that this mechanism allows the OutRegistry to be configured to disable out-going reports by 'kind' where the kind of a report is defined by the triplet: [type, sub-type, discriminant]. In the case of service 1 failure reports, the discriminant is the failure code.

3.4.6 Time-Tagging of Reports

Clause 5.4.2.1 of the PUS leaves applications the option to generate the time-tag of a report either before or after the time the report collects its data. In the CORDET Framework, the time-stamp of a report represents the time when an application makes a request to issue that report (this is after the report data have been collected).

Table 3.2: Terminological Mapping PUS-CORDET

PUS Term	Corresponding CORDET Term
Application Process	In the PUS, an application process is an entity which hosts one or more sub-services. In the CORDET Framework, the equivalent concept is that of group (each command or report in a CORDET application must belong to a group). See also section 4.2.
Instruction	In the PUS, a request (command) contains one or more instructions. Instructions do not exist in the CORDET Framework. They are implicit to commands. In the PUS Extension, instructions therefore arise when individual commands are defined. If a command has multiple instructions, then each instruction is executed in a dedicated progress step of that command.
Message	In the PUS, a message is either a report or a request and its type is defined by the pair [service type, service sub-type]. The CORDET Framework directly supports the concepts of service types and sub-types and adds to them the concept of discriminant (see section 4.1).
Notification	In the PUS, a report contains one or more notifications. The notifications in one report must be of the same type. Notifications do not exist in the CORDET Framework. They are implicit to reports. In the PUS Extension, notifications therefore arise when individual reports are defined.
Parameter	In a generic sense, PUS parameters are mapped to command and report parameters. In the specific context of service 3, parameters are mapped to data items.
Progress Step	In the PUS, the Progress Step is an enumerated type. In the CORDET Framework, a command is executed in a sequence of *execution steps*. A set of logically related execution steps is a *progress step*. It is up to each command to define what constitutes a progress step.
Request	The PUS Request is the same as the CORDET Command
Subservice	A PUS Subservice is a group of related capabilities which are defined within a service. The concept of Subservice does not exist in the CORDET Framework. In its PUS Extension it arises as part of the definition of the commands and reports which implement a service.
Transaction	In the PUS, a transaction is an exchange between a service provider and a service user which consists of one of the following: (a) a request followed by the report triggered by the request; (b) a data report autonomously generated by the service provider; or (c) an event report autonomously generated by a service provider. The CORDET Framework only defines individual commands and reports. The PUS Extension implicitly defines transactions when it specifies links between a command and the reports it triggers or when it specifies the conditions under which data or event reports are generated.

4 Report and Command Attributes

The CORDET Framework defines a number of attributes for commands and reports. Table 4.1 shows how they are mapped to the command and report attributes defined by the PUS.

The PUS Extension of the CORDET Framework extends the range of command and report attributes to include all command and report attributes defined by the PUS: the components which implement PUS commands and reports provide operations to access all the attributes defined at PUS level.

Within the framework, commands and reports are handled as instances of components of type InReport (for incoming reports), InCommand (for incoming command), or OutComponent (for out-going commands and reports). Commands and reports arrive at and leave the framework through the OutStream and InStream components, which constitute the external interfaces of the framework. At these interfaces, commands and reports are encapsulated in packets (sequences of bytes which carry all the data in the report or command). In the framework extension, these packets comply to the command and report layout defined by the PUS and the PUS Extension provides operation to encode and decode the packets, i.e. to set and read the values of any PUS-defined parameter in a packet.

Table 4.1: Mapping of CORDET Attributes to PUS Attributes

Attribute	Mapping to PUS Attribute
`Src`	Commands: source field of data field header; Reports: PID
`Dest`	Commands: PID; Reports: Destination Identifier (process user identifier of application process addressed by the report)
`SeqCnt`	Sequence Count field in packet header
`CmdRepType`	Packet Type bit in packet header
`Length`	Related to Packet Length Field (which is the length of the packet data field minus 1)
`ProgressStep`	The progress step of an incoming command
`TimeStamp`	Time field in data field header of telemetry packets; not present in telecommand packets
`Discriminant`	Service-specific mapping to parameter which determines command or report layout, see section 4.1
`ServType`	Service Type field in data field header
`ServSubType`	Message Sub-Type field in data field header
`Group`	Related to CAT part of the APID, see section 4.2
`CmdRepId`	Not present
`AcceptAck`	Bit 3 of acknowledge field in data field header
`StartAck`	Bit 2 of acknowledge field in data field header
`ProgressAck`	Bit 1 of acknowledge field in data field header
`TermAck`	Bit 0 of acknowledge field in data field header
`ParStart`	The parameter area starts where the Application Data starts, namely at byte 11 of a command packet and at byte 17 of a report packet
`ParLength`	The parameter length is the total packet length (in bytes) minus 10 for command packets and the total packet length (in bytes) minus 16 for report packets

4.1 Mapping of Discriminant Attribute

The CORDET discriminant is an optional attribute of a command or report. It is defined when the layout or the behaviour of a command or report are not exclusively determined by the command or report type and sub-type. In such cases, the discriminant becomes the determinant of the command or report layout and behaviour. The PUS does not have the concept of discriminant but some of its services use a particular field for the same purpose. For instance, the Event Identifier (EID) of service 5 reports determines the layout of a service 5 report and hence serves the same purpose as the CORDET discriminant. Similarly, some commands or reports carry variable-length blocks of data; in such cases, the parameter which defines the length of the data block acts as a discriminant. Bearing in mind these considerations, the PUS Extension maps the CORDET discriminant to the following PUS parameters:

- The Failure Identifier (FID) for service 1 failure reports
- The Structure Identifier (SID) for (3,25) reports
- The Event Identifier (EID) for reports (5,1) to (5,4)

4.2 Mapping of Group Attribute

The CORDET Framework does not have the concept of APID but it uses the concept of group to represent it. More precisely, the CORDET Framework assigns sequence counters to commands and reports and assigns commands and reports going through an InStream or OutStream to 'groups'. The CORDET sequence counters are initialized to 1 and are incremented by 1 within each group (i.e. for each group in an OutStream, a counter is maintained which is incremented by 1 whenever a command or report belonging to that group is issued by the OutStream; and for each group in an InStream, a counter is maintained which is incremented by 1 whenever a command or report belonging to that group is received by the InStream).

The CORDET Framework requires that, for each destination for out-going commands or reports, an OutStream be defined and that, for each source of incoming commands or reports, an InStream be defined.

Bearing in mind the above, compliance with the PUS rules for the management of the APIDs and sequence counters requires that the following rules be adopted for the assignment of the groups:

- If an application sends commands or reports to the same destination with different APIDs, then for each such APID, a group must be defined
- If an application receives commands or reports from the same source with different APIDs, then for each such APID, a group must be defined

4.3 Requirements

The table in this section lists the requirements for the command and report attributes.

Table 4.2: Requirements for Command and Report Attributes

Req. ID	Requirement Text
P-CRA-1/S	*Components encapsulating a command or a report shall implement all attributes defined for them by the PUS*
P-CRA-2/S	*Components encapsulating a command or a report shall provide operations to access in read and write mode all their PUS-defined attributes*
P-CRA-3/S	*The PUS Extension of the CORDET Framework shall provide operations to encode and decode any PUS-defined attribute in a packet carrying a command or report*
P-CRA-4/C	*If an application sends commands or reports to the same destination with different APIDs, then for each such APID, a CORDET Group shall be defined*
P-CRA-5/C	*If an application receives commands or reports from the same source with different APIDs, then for each such APID, a CORDET group shall be defined*

5 The Data Pool Component

The Data Pool Component is a pre-defined component offered by the PUS Extension of the CORDET Framework. It is used by all services supported by the framework extension and it is therefore defined independently of these services.

5.1 Data Pool Concepts

The Data Pool Component provides read-write access to a set of *Data Items*. A Data Item is characterized by the following attributes:

- *Default Value*: the value of the data item when the data pool is reset
- *Current Value*: the value of the data item at a particular point in time
- *Identifier*: a positive integer which uniquely identifies the Data Item within the Data Pool
- *Type*: an enumerated value which determines the range of possible values of the Data Item and its representation in the Data Pool

With reference to the last bullet, it is noted that the set of supported types is defined at implementation level. The data items can be of two kinds:

- *Parameters*: data items whose value is under the control of an entity external to the host application
- *Variables*: data items whose value is autonomously updated by the host application as part of its normal operation

In practice, the data pool is the means through which a component can access data belonging to other components. Note that this specification is silent about the physical location of the data items in the data pool, which can be either the components which own the data item (in which case the data pool only offers a link to the data items), or the data pool itself, or a mixed solution where some data items reside in the data pool and others in peripheral components.

This specification is similarly silent about the internal structure of data items and, in particular, it neither restricts them to be of primitive type nor does it mandate an array-like structure for them. Any such restrictions or options must be introduced at implementation level.

5.2 Data Pool Behaviour

The Data Pool Component - like all other CORDET Components - is an extension of the Base Component of section 3.2 of [CR-SP]. It does not add any behaviour to the Base Component but it specializes some of its adaptation points as described below.

The Initialization Procedure[2] of the Data Pool Component creates the data structures needed by the component. At one extreme, if an implementation chooses to locate all data items inside the Data Pool Component, then its Initialization Procedure is responsible for creating the data structures which host the data items. At the other extreme, in an im-

[2] It is recalled that the Base Component defines three procedures: the *Initialization Procedure*, the *Configuration Procedure*, and the *Execution Procedure*. These two procedures are inherited by all components derived from the Base Component. They are therefore also inherited by the Data Pool Component.

plementation where data items remain located in their originating components and where the data pool only acts as a kind of data switch-board, the Initialization Procedure does nothing and always returns "initialization successful".

When the Data Pool is reset, the current values of its data items are initialized with their default values. The Configuration Procedure is therefore responsible for initializing the data item values with their default values.

This specification does not say where the default values of the data items are stored in relation to their current values. At implementation level, two basic options are possible:

1. The default values are stored alongside the current values (i.e. in RAM)
2. The default values are stored in some other memory area (e.g. in an EEPROM or in a remote location)

In the first case, the initialization of the data items simply involves a copy across two locations in RAM. in the second case, the initialization may be a potentially lengthy process involving the retrieval of the data item values from an external memory bank or from a remote location. The Data Pool Component covers both options and its Configuration Procedure is therefore defined as follows:

- The Configuration Action starts the process whereby the default values of the data items are acquired and copied to their current values
- The Configuration Check returns "success" if the initialization of the data item values can be done in zero logical execution time[3] or else when the initialization has completed

In the case where the initialization of the data item values is not an operation with zero logical execution time, then the Data Pool Component must be sent at least two `Reset` commands before it can enter the CONFIGURED state: the first `Reset` command starts the acquisition of the default values of the data items and the second `Reset` command verifies that the acquisition has terminated. Obviously, there is nothing to stop an application from using a "polling" approach and sending a sequence of `Reset` commands until the Data Pool Component has entered its CONFIGURED state. Note that, in line with requirements AST-5 and AST-7 in [CR-SP], it is the responsibility of the application to send as many `Reset` commands as needed to the Data Pool Component during the application start-up and application reset process.

The data items in the data pool should be kept up-to-date. Two options are possible in this respect: (a) the data items are refreshed by the components which own them or (b) the data items are periodically refreshed by the data pool itself. In case (a), the data pool is entirely passive. In case (b), it must implement the refresh function. A mixed solution where some data items are refreshed by the data pool component while others are refreshed by external components is also possible. Since refreshing should only be done when the data pool is in state CONFIGURED, it is natural to allocate the refresh function to the Execution Procedure of the Data Pool Component.

The framework uses option (a) for all data items under its control with the exception of the debug variables of service 3 (see section 9.2). Users are free to choose between the two

[3] The concept of *logical execution time* is introduced in [FW-SP] as part of the FW Profile Definition. The logical execution time of a behaviour is the execution time of that behaviour on a processor with infinite speed and in the absence of pre-emption by higher-priority activities or blocking by lower-priority activities. Essentially, a behaviour has zero logical execution time if it includes neither "wait" operations nor synchronization operations with external devices or threads.

options for their data items. If they choose option (b), they must extend the Execution Procedure of the Data Pool Component accordingly.

Finally, the Data Pool Component offers an `update` operation to support service 3 and it offers operations to give read-write access to the current values of the data items. The mode of access to these values (through functions which returns pointers to the data items or through functions which return their values) is not specified and is left to the implementation to decide. Also, no limitation is specified on which components can access the data items in the data pool: any component can access any data item in read-write mode. Such limitations, if needed, may be added at implementation level.

5.3 Service Observability Concept

The data pool plays a key role in service 3 (see section 9) but it is also used by other services as the repository through which service observables are accessed. Each service defines a number of *service observables*. These are data items which the service is responsible for keeping up-to-date and which reflect its current state. The service observables are assigned to the data pool which means that they can be accessed using service 3. Note that some of this service status information may also be accessible using service-specific reports (i.e. there may be a degree of redundancy in the observability of the service).

5.4 Service Parameterization Concept

Each service defines a number of *service parameters*. These are data items which control the behaviour of the service and whose value is set either by the user of the application hosting the service (e.g. the ground) or by other services in the application. Service parameters are assigned to the data pool which means that they can be accessed using service 3. Note that some of the service parameters may also be controlled using service-specific commands (i.e. there may be a degree of redundancy in the commandability of a service).

5.5 Adaptation Points

The table in this section lists the adaptation points for the Data Pool Component.

Table 5.1: Adaptation Points for Data Pool Component

AP ID	Adaptation Point	Close-Out Value
P-DP-1	Initialization Check in Initialization Procedure of Data Pool Component (Overrides BAS-1)	Return 'success' if there are adequate resources for creating the data structures for the data items
P-DP-2	Initialization Action in Initialization Procedure of Data Pool Component (Overrides BAS-2)	Create the data structures required for the data items and return 'success' if creation was successful
P-DP-3	Configuration Check in Reset Procedure of Data Pool Component (Overrides BAS-3)	Return 'success' if current values of data items can be initialized with their default values in zero logical execution time or else return 'success' if initialization of current value of data items has completed
P-DP-4	Configuration Action in Reset Procedure of Data Pool Component (Overrides BAS-4)	Start initialization of current values of data items with their default values and return 'success' if the initialization has completed

AP ID	Adaptation Point	Close-Out Value
P-DP-5	Shutdown Action of Data Pool Component (Overrides BAS-5)	Same value as in Base Component
P-DP-6	Execution Procedure of Data Pool Component (Overrides BAS-6)	Refresh values of debug variables in data pool
P-DP-7	Definition of Data Items in the Data Pool Component (New AP)	No default defined at framework level
P-DP-8	Operation to access the Current Value of a Data Item (New AP)	No default defined at framework level
P-DP-9	Operation to update the Current Value of a Data Item (New AP)	No default defined at framework level

5.6 Requirements

The table in this section lists the requirements for the Data Pool Component.

Table 5.2: Requirements for Data Pool Component

Req. ID	Requirement Text
P-DP-1/S	*The PUS Extension of the CORDET Framework shall provide a Data Pool component as an extension of the Base Component*
P-DP-2/A	*The Data Pool Component shall support the adaptation points specified in table 5.1*
P-DP-3/S	*When it is configured, the Data Pool Component shall provide operations to let other components access the current value of its data items*
P-DP-4/S	*When it is configured, the Data Pool Component shall provide operations to let other components update the current value of its data items*
P-DP-5/S	*Deleted*
P-DP-5/C	*An application shall instantiate the Data Pool Component only once*
P-DP-6/C	*An application shall extend the Data Pool Execution Procedure to refresh all data items in the data pool which are not refreshed by other means*

6 Report and Command Factories

Command and report components must be instantiated dynamically as the need arises to generate or process them. For this purpose, the CORDET Framework defines the OutFactory and InFactory components to encapsulate the instantiation process of, respectively, OutComponents and InCommands/InReports. Both kinds of components provide two operations: `Make` to create an instance of a command or report of a given kind (as given by the triplet [type, sub-type, discriminant]) and `Release` to release command or report instance.

The CORDET Framework specifies the interface of the factory components but does not actually provide them because it does not provide any concrete command or report components. The framework extension provides concrete commands and reports and is therefore required to also provide implementations of the two factory components.

The process through which the command and report components are created by the factories is not specified. In particular, the allocation policy for the memory for the instantiated components is left open for the implementation to decide.

6.1 Observables

The table in this section lists the variables which are maintained and made accessible through the data pool by the two factory components.

Table 6.1: Observables for Verification Service

Name	Description
`nOfAllocatedInCmd`	Number of currently allocated InCommands (i.e. successfully created by the InFactory and not yet released)
`nOfAllocatedInRep`	Number of currently allocated InReports (i.e. successfully created by the InFactory and not yet released)
`nOfAllocatedOutCmp`	Number of currently allocated OutComponents (i.e. successfully created by the OutFactory and not yet released)
`nOfFailedInCmd`	Number of InCommands whose creation by the InFactory failed
`nOfFailedInRep`	Number of InReports whose creation by the InFactory failed
`nOfFailedOutCmp`	Number of OutComponents whose creation by the OutFactory failed
`nOfTotAllocatedInCmd`	Number of InCommands successfully created by the InFactory since application start
`nOfTotAllocatedInRep`	Number of InReports successfully created by the InFactory since application start
`nOfTotAllocatedOutCmp`	Number of OutComponents successfully created by the InFac- tory since application start

6.2 Adaptation Points

The `Make` and `Release` operations for the two factory components are adaptation points because the command and report instantiation policies are not defined at framework extension level. These two adaptation points are, however, already defined at CORDET Framework level (see adaptation points FAC-1 and FAC-2 in [CR-SP]) and do not therefore need to be defined again here.

Similarly, the factory components are defined in [CR-SP] as extension of the Base Component and they therefore inherit all the adaptation points of the Base Components but no further specialization of these adaptation points in done in the PUS Extension of the CORDET Framework.

6.3 Requirements

The table in this section lists the requirements for the factory components.

Table 6.2: Requirements for Factory Components

Req. ID	**Requirement Text**
P-FAC-1/S	*The PUS Extension of the CORDET Framework shall provide an InFactory component capable of creating an instance of any of the command or report types defined by the framework*
P-FAC-2/S	*The PUS Extension of the CORDET Framework shall provide an OutFactory component capable of creating an instance of any of the command or report types defined by the framework*
P-FAC-3/S	*The two factory components shall maintain and make accessible through the data pool the observables listed in table* ??

7 Definition of PUS Services

The PUS Extension of the CORDET Framework supports a subset of the PUS services and, for these services, it specifies the components which implement their reports and commands. Since the framework extension covers the provision of PUS services, it is only concerned with incoming commands and out-going reports.

In the CORDET Framework, incoming commands are encapsulated by InCommand components and out-going components are encapsulated by OutComponent components. The InCommand and OutComponent components define abstract commands and reports. These two components implement the invariant behaviour which is common to, respectively, all incoming commands and all out-going reports and they offer adaptation points where the behaviour which is specific to each concrete command or report must be inserted. A concrete command or a concrete report is specified by closing the adaptation points of, respectively, the InCommand component or of the OutComponent component.

This concept is illustrated in figure 7.1 for the case of incoming commands. The component at the top is the InCommand component which is used as a base from which the components implementing concrete commands are derived. The component at the top is provided by the CORDET Framework. The components at the bottom are provided by its PUS Extension.

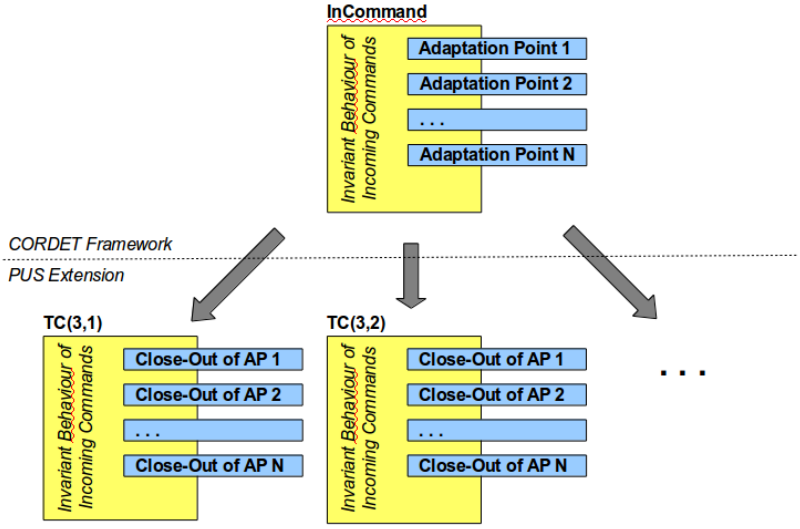


Fig. 7.1: Extension of InCommand Component

The components of the CORDET Framework are defined as models which comply with the FW Profile of [FW-SP]. By way of example, figure 7.2 shows the model of the InCommand (the figure is taken from [CR-SP]). This consists of a state machine where some guards and actions are marked as "Adaptation Points". Concrete commands are defined by attaching a concrete behaviour to these actions and guards.

Thus, for each supported PUS command, the framework extension defines an extension of the InCommand component which closes all the InCommand adaptation points. Similarly, for each supported PUS report, it defines an extension of the OutComponent component which closes all the OutComponent adaptation points.

Table 7.5 and 7.6 list the PUS services supoorted (either in full or in part) and the command and report components provided by the PUS Extension of the CORDET Framework to implement them. The first column in table 7.6 gives the [type,sub-type] pair which identifies the command or report; the second column gives the name of the CORDET component which implements the command or report; the last column gives its PUS names as it is given in section 8 of [PS-SP].

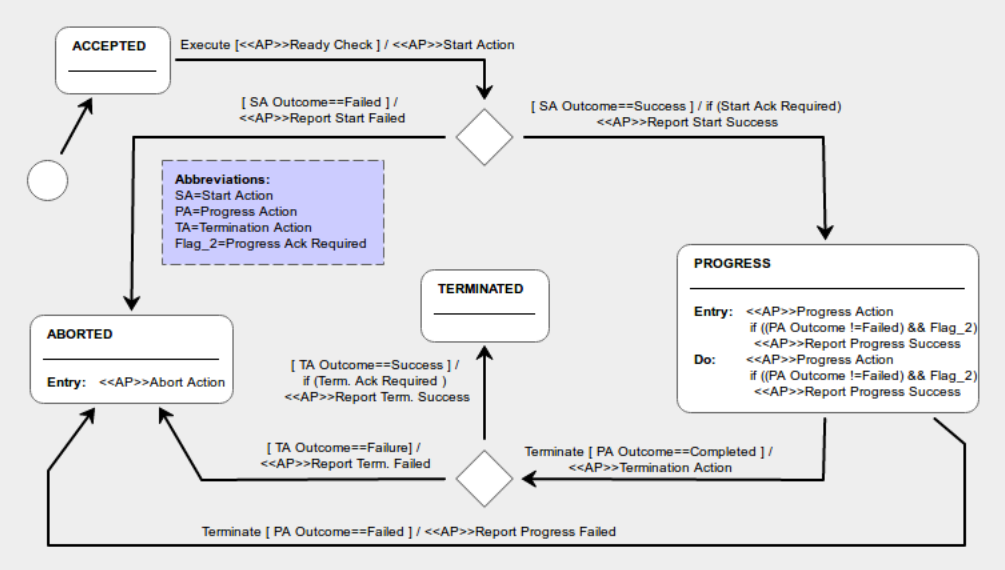

ACCEPTED
Execute [<<AP>>Ready Check] / <<AP>>Start Action
[SA Outcome==Failed] /
<<AP>>Report Start Failed
[SA Outcome==Success] / if (Start Ack Required)
<<AP>>Report Start Success
Abbreviations:
SA=Start Action
PA=Progress Action
TA=Termination Action
Flag_2=Progress Ack Required
PROGRESS
Entry: <<AP>>Progress Action
if ((PA Outcome !=Failed) && Flag_2)
<<AP>>Report Progress Success
Do: <<AP>>Progress Action
if ((PA Outcome !=Failed) && Flag_2)
<<AP>>Report Progress Success
TERMINATED
ABORTED
Entry: <<AP>>Abort Action
[TA Outcome==Success] /
if (Term. Ack Required)
<<AP>>Report Term. Success
[TA Outcome==Failure] /
<<AP>>Report Term. Failed
Terminate [PA Outcome==Completed] /
<<AP>>Termination Action
Terminate [PA Outcome==Failed] / <<AP>>Report Progress Failed

Fig. 7.2: Model of InCommand Component

7.1 Report and Command Adaptation Points

Tables 7.1 and 7.2 list the adaptation points of, respectively, the OutComponent component and the InCommand component. These adaptation points are defined by the CORDET Framework in [CR-SP]. The tables show how they are closed for the concrete commands and reports supported by the PUS Extension of the framework. In some cases, the adaptation point is closed in the same way for all framework reports/commands. In other cases, the close-out is report- or command-specific and is then described in the later sections of this document which define the individual PUS services. Thus, for instance, the close-out of the report-specific adaptation points for the service 1 reports can be found in section 8.3.

The following considerations apply to the data in table 7.1 concerning OutComponents:

- The OutComponent components are created by the OutFactory and it can be assumed that they are created such that they can be successfully initialized and configured. Their initialization and configuration procedures (adaptation points OCM-1 to 4) can therefore be just dummies that do not perform any action. The same applies to the shutdown procedure (adaptation point OCM-5).
- The adaptation point OCM-6 related to the execution procedure is already closed at CORDET Framework level because OutComponents have no execution procedure.
- The adaptation points OCM-7 and 8 related to the setting of the report type and sub-type are closed in accordance with the discussion in section 4 by setting the CORDET types and sub-types equal to the PUS type and sub-type.
- The adaptation points OCM-9 and OCM-10 to 12 related to the setting of the report

discriminant, destination and parameters are closed for each individual report type in the following sections of this document.

- The adaptation point OCM-10 related to the acknowledge level is only relevant to out-going commands and is therefore not applicable to the PUS reports defined in this document (which is only concerned with incoming commands).
- The adaptation points OCM-13 to 16 related to the report checks and actions are closed for each specific report type in the following sections of this document.
- The adaptation point OCM-17 related to the serialize operation is closed to create a packet layout which complies with the layout defined by the PUS in [PS-SP].
- The adaptation point OCM-18 covers the response to a report having an invalid destination. By design, this situation should never arise and the adaptation point is closed with the generation of an error report.

The following considerations apply to the data in table 7.2 concerning InCommands:

- The InCommand components are created by the InFactory but are then initialized and configured by the InLoader. Their initialization and configuration procedures (adaptation points ICM-1 to 4) are therefore implementation-specific. The same applies to the shutdown procedure (adaptation point ICM-5).
- The adaptation point ICM-3 implements the acceptance check for the command. This verifies the correctness of the command length and CRC.
- The adaptation point related to the execution procedure (ICM-6) is already closed at CORDET Framework level because InCommand do not have any execution procedure.
- The adaptation points ICM-7 to 11 related to the command checks and actions are closed for each specific command type in the following sections of this document.
- The adaptation points ICM-12 to 17 related to the generation of success and failure reports for the command are closed as part of the service 1 definition in section 8.3.
- The adaptation points ICM-18 and 19 related to the setting of the command type and sub-type are closed in accordance with the discussion in section 4 by setting the CORDET types and sub-types equal to the PUS type and sub-type.
- The adaptation points ICM-20 and 21 related to the command discriminant and parameters are closed for each individual command type in the following sections of this document.

Table 7.1: Adaptation Points for PUS Reports

AP ID	Adaptation Point	Close-Out Value
P-OCM-1	Initialization Check in Initialization Procedure of OutComponent (Closes OCM-1)	Always returns 'check successful'
P-OCM-2	Initialization Action in Initialization Procedure of OutComponent (Closes OCM-2)	Do nothing and return 'action successful'
P-OCM-3	Configuration Check in Reset Procedure of OutComponent (Closes OCM-3)	Always returns 'check successful'
P-OCM-4	Configuration Action in Reset Procedure of OutComponent (Closes OCM-4)	Do nothing and return 'action successful'

AP ID	Adaptation Point	Close-Out Value
P-OCM-5	Shutdown Action in Base Component of OutComponent (Closes OCM-5)	Do nothing
P-OCM-6	Execution Procedure of OutComponent (Closes OCM-6)	Do nothing
P-OCM-7	Service Type Attribute of OutComponent (Closes OCM-7)	Set equal to PUS service type
P-OCM-8	Command/Report Sub-Type Attribute of OutComponent (Closes OCM-8)	Set equal to PUS service sub-type
P-OCM-9	Destination Attribute of OutComponent (Closes OCM-9)	See definition of individual reports
P-OCM-10	Acknowledge Level Attribute of OutComponent (Closes OCM-10)	Not relevant to out-going report
P-OCM-11	Discriminant Attribute of OutComponent (Closes OCM-11)	See definition of individual reports
P-OCM-12	Parameter Attribute of OutComponent (Closes OCM-12)	See definition of individual reports
P-OCM-13	Enable Check Operation of OutComponent (Closes OCM-13)	See definition of individual reports
P-OCM-14	Ready Check Operation of OutComponent (Closes OCM-14)	See definition of individual reports
P-OCM-15	Repeat Check Operation of OutComponent (Closes OCM-15)	See definition of individual reports
P-OCM-16	Update Action of OutComponent (Closes OCM-16)	See definition of individual reports
P-OCM-17	Serialize Operation of OutComponent (Closes OCM-17)	Build a packet with the layout specified by the PUS
P-OCM-18	Operation to Report Invalid Destination of an OutComponent (Closes OCM-18)	Generate SNDPCKT_INV_DEST Error Report

Table 7.2: Adaptation Points for PUS Commands

AP ID	Adaptation Point	Close-Out Value
P-ICM-1	Initialization Check in Initialization Procedure of InCommand (Closes ICM-1)	Returns ’check successful’ if information for initializing InCommand using data in incoming packet is valid
P-ICM-2	Initialization Action in Initialization Procedure of InCommand (Closes ICM-2)	Use information in incoming packet to initialize InCommand and return “action successful’
P-ICM-3	Configuration Check in Reset Procedure of InCommand (Closes ICM-3)	Returns ’check successful’ if packet length and checksum are correct
P-ICM-4	Configuration Action in Reset Procedure of InCommand (Closes ICM-4)	Use information in incoming packet to configure InCommand and return “action successful’
P-ICM-5	Shutdown Action of InCommand (Closes ICM-5)	Release all resources allocated to the InCommand
P-ICM-6	Execution Procedure of InCommand (Closes ICM-6)	Do nothing
P-ICM-7	Ready Check of InCommand (Closes ICM-7)	See definition of individual commands
P-ICM-8	Start Action of InCommand (Closes ICM-8)	See definition of individual commands
P-ICM-9	Progress Action of InCommand (Closes ICM-9)	See definition of individual commands
P-ICM-10	Termination Action of InCommand (Closes ICM-10)	See definition of individual commands
P-ICM-11	Abort Action of InCommand (Closes ICM-11)	See definition of individual commands
P-ICM-12	Operation to Report Start Failed for InCommand (Closes ICM-12)	See definition of service 1
P-ICM-13	Operation to Report Start Successful for InCommand (Closes ICM-13)	See definition of service 1
P-ICM-14	Operation to Report Progress Failed for InCommand (Closes ICM-14)	See definition of service 1
P-ICM-15	Operation to Report Progress Successful for InCommand (Closes ICM-15)	See definition of service 1
P-ICM-16	Operation to Report Termination Failed for InCommand (Closes ICM-16)	See definition of service 1
P-ICM-17	Operation to Report Report Termination Successful for InCommand (Closes ICM-17)	See definition of service 1

AP ID	Adaptation Point	Close-Out Value
P-ICM-18	Service Type Attribute of InCommand (Closes ICM-18)	Set equal to PUS service type
P-ICM-19	Command Sub-Type Attribute of InCommand (Closes ICM-19)	Set equal to PUS service sub-type
P-ICM-20	Discriminant Attribute of InCommand (Closes ICM-20)	See definition of individual commands
P-ICM-21	Parameter Attributes of InCommand (Closes ICM-21)	See definition of individual commands

7.2 Dependencies Between Services

A service S1 depends on another service S2 if the decision by an application to deploy service S1 requires the same application to also deploy service S2. The services defined in this document minimize this kind of dependencies. Table 7.3 lists the service dependencies. These are limited to:

- Services 12 and 13 generate event reports and therefore need service 5

Note that, although dependencies between services are minimized, all services depend on the data pool because the data pool holds the variable and parameters which are used by the services. Hence, all applications instantiated from the CORDET Extension of the PUS Framework must deploy the data pool component of section 5.

Table 7.3: Service Dependencies

N	Service Name	Dependencies
1	Request Verification Service	None
3	Housekeeping Service	None
5	Event Reporting Service	None
12	On-Board Monitoring Service	Requires Service 5
13	Large Packet Transfer Service	Requires Service 5
17	Test Service	None
19	Event Action Service	TBD

7.3 Requirements

The requirements in table 7.4 make the adaptation points defined in the previous two sections applicable to all command and report components provided by the framework extension.

Table 7.4: Requirements for Framework Extension Commands and Reports

Req. ID	**Requirement Text**
P-PCR-1/A	*The InCommand components provided by the PUS Extension of the CORDET Framework shall close the InCommand adaptation points as stated in table 7.1*
P-PCR-2/A	*The OutComponent components provided by the PUS Extension of the CORDET Framework shall close the OutComponent adaptation points as stated in table 7.2*
P-PCR-3/C	*Applications shall comply with the service dependencies listed in table 7.3*

Table 7.5: List of Supported Services

Type	Acron.	Name
1	Ver	Request Verification Service
3	Hk	Housekeeping Service
5	Evt	Event Reporting Service
11	Scd	Time Based Scheduling Service
12	Mon	On Board Monitoring Service
13	Lpt	Large Packet Transfer Service
17	Tst	Test Service
255	Dum	Dummy Service

Table 7.6: List of Supported Commands/Reports

Type	CORDET Name	PUS Name
TM(1,1)	SuccAccRep	Successful Acceptance Verification Report
TM(1,2)	FailedAccRep	Failed Acceptance Verification Report
TM(1,3)	SuccStartRep	Successful Start of Execution Verification Report
TM(1,4)	FailedStartRep	Failed Start of Execution Verification Report
TM(1,5)	SuccPrgrRep	Successful Progress of Execution Verification Report
TM(1,6)	FailedPrgrRep	Failed Progress of Execution Verification Report
TM(1,7)	SuccTermRep	Successful Completion of Execution Verification Report
TM(1,8)	FailedTermRep	Failed Completion of Execution Verification Report
TM(1,10)	FailedRoutingRep	Failed Routing Verification Report
TC(3,1)	CreHkCmd	Create a Housekeeping Parameter Report Structure
TC(3,2)	CreDiagCmd	Create a Diagnostic Parameter Report Structure
TC(3,3)	DelHkCmd	Delete a Housekeeping Parameter Report Structure
TC(3,4)	DelDiagCmd	Delete a Diagnostic Parameter Report Structure
TC(3,5)	EnbHkCmd	Enable Periodic Generation of a Housekeeping Parameter Report Structure
TC(3,6)	DisHkCmd	Disable Periodic Generation of a Housekeeping Parameter Report Structure
TC(3,7)	EnbDiagCmd	Enable Periodic Generation of a Diagnostic Parameter Report Structure
TC(3,8)	DisDiagCmd	Disable Periodic Generation of a Diagnostic Parameter Report Structure
TC(3,9)	RepStructHkCmd	Report Housekeeping Parameter Report Structure
TM(3,10)	RepStructHkRep	Housekeeping Parameter Report Structure Report
TC(3,11)	RepStructDiagCmd	Report Diagnostic Parameter Report Structure
TM(3,12)	RepStructDiagRep	Diagnostic Parameter Report Structure Report
TM(3,25)	Rep	Housekeeping Parameter Report
TM(3,26)	DiagRep	Diagnostic Parameter Report
TC(3,27)	OneShotHkCmd	Generate One-Shot Report for Housekeeping Parameters
TC(3,28)	OneShotDiagCmd	Generate One-Shot Report for Diagnostic Parameters

Type	CORDET Name	PUS Name
TM(5,1)	Rep1	Informative Event Report (Level 1)
TM(5,2)	Rep2	Low Severity Event Report (Level 2)
TM(5,3)	Rep3	Medium Severity Event Report (Level 3)
TM(5,4)	Rep4	High Severity Event Report (Level 4)
TC(5,5)	EnbCmd	Enable Generation of Event Identifiers
TC(5,6)	DisCmd	Disable Generation of Event Identifiers
TC(5,7)	RepDisCmd	Report the List of Disabled Event Identifiers
TM(5,8)	DisRep	Disabled Event Identifier Report
TC(11,1)	EnbTbsCmd	Enable Time-Based Schedule Execution Function
TC(11,2)	DisTbsCmd	Disable Time-Based Schedule Execution Function
TC(11,3)	ResTbsCmd	Reset Time-Based Schedule
TC(11,4)	InsTbaCmd	Insert Activities into Time-Based Schedule
TC(11,5)	DelTbaCmd	Delete Activities from Time-Based Schedule
TC(11,20)	EnbSubSchedCmd	Enable Time-Based Sub-Schedules
TC(11,21)	DisSubSchedCmd	Disable Time-Based Sub-Schedules
TC(11,22)	CreGrpCmd	Create Time-Based Scheduling Groups
TC(11,23)	DelGrpCmd	Delete Time-Based Scheduling Groups
TC(11,24)	EnbGrpCmd	Enable Time-Based Scheduling Groups
TC(11,25)	DisGrpCmd	Disable Time-Based Scheduling Groups
TC(11,26)	RepGrpCmd	Report Status of Time-Based Scheduling Groups
TM(11,27)	GrpRep	Time-Based Scheduling Group Status Report
TC(12,1)	EnbParMonDefCmd	Enable Parameter Monitoring Definitions
TC(12,2)	DisParMonDefCmd	Disable Parameter Monitoring Definitions
TC(12,3)	ChgTransDelCmd	Change Maximum Transition Reporting Delay
TC(12,4)	DelAllParMonCmd	Delete All Parameter Monitoring Definitions
TC(12,5)	AddParMonDefCmd	Add Parameter Monitoring Definitions
TC(12,6)	DelParMonDefCmd	Delete Parameter Monitoring Definitions
TC(12,7)	ModParMonDefCmd	Modify Parameter Monitoring Definitions
TC(12,8)	RepParMonDefCmd	Report Parameter Monitoring Definitions
TM(12,9)	RepParMonDefRep	Parameter Monitoring Definition Report
TC(12,10)	RepOutOfLimitsCmd	Report Out Of Limit Monitors
TM(12,11)	RepOutOfLimitsRep	Out Of Limit Monitors Report
TM(12,12)	CheckTransRep	Check Transition Report
TC(12,13)	RepParMonStatCmd	Report Status of Parameter Monitors
TM(12,14)	RepParMonStatRep	Parameter Monitor Status Report
TC(12,15)	EnbParMonFuncCmd	Enable Parameter Monitoring Function
TC(12,16)	DisParMonFuncCmd	Disable Parameter Monitoring Function
TC(12,17)	EnbFuncMonCmd	Enable Functional Monitoring Function
TC(12,18)	DisFuncMonCmd	Disable Functional Monitoring Function

Type	CORDET Name	PUS Name
TC(12,19)	EnbFuncMonDefCmd	Enable Functional Monitoring Definitions
TC(12,20)	DisFuncMonDefCmd	Disable Functional Monitoring Definitions
TC(12,21)	ProtFuncMonDefCmd	Protect Functional Monitoring Definitions
TC(12,22)	UnprotFuncMonDefCmd	Unprotect Functional Monitoring Definitions
TC(12,23)	AddFuncMonDefCmd	Add Functional Monitoring Definitions
TC(12,24)	DelFuncMonDefCmd	Delete Functional Monitoring Definitions
TC(12,25)	RepFuncMonDefCmd	Report Functional Monitoring Definitions
TM(12,26)	RepFuncMonDefRep	Report Functional Monitoring Definitions
TC(12,27)	RepFuncMonStatCmd	Report Status of Functional Monitors
TM(12,28)	RepFuncMonStatRep	Status of Functional Monitors Report
TM(13,1)	DownFirstRep	First Downlink Part Report
TM(13,2)	DownInterRep	Intermediate Downlink Report
TM(13,3)	DownLastRep	Last Downlink Part Report
TC(13,9)	UpFirstCmd	First Uplink Part
TC(13,10)	UpInterCmd	Intermediate Uplink Part
TC(13,11)	UpLastCmd	Last Uplink Part
TM(13,16)	UpAbortRep	Large Packet Uplink Abortion Report
TC(13,129)	StartDownCmd	Trigger Large Packet Down-Transfer
TC(13,130)	AbortDownCmd	Abort Large Packet Down-Transfer
TC(17,1)	AreYouAliveCmd	Perform Are-You-Alive Connection Test
TM(17,2)	AreYouAliveRep	Are-You-Alive Connection Report
TC(17,3)	ConnectCmd	Perform On-Board Connection Test
TM(17,4)	ConnectRep	On-Board Connection Test Report
TC(255,1)	Sample1	Sample 1 Command

8 Request Verification Service

The service type of the Request Verification Service is 1. The PUS Extension of the CORDET Framework supports this service in full.

The Request Verification Service is implemented by nine reports which are issued in response to notifications generated by a service provider application. The notifications cover different stages of the processing of an incoming command. More precisely:

- The report (1,10) is triggered in response to notifications of a routing failure for an incoming command (Routing and Reporting Sub-Service)
- The report (1,1) and (1,2) are triggered in response to notifications of the failure or success of the acceptance of an incoming command (Acceptance and Reporting Sub-Service)
- The reports (1,3) to (1,8) are triggered in response to notifications of the failure or success of execution of an incoming command (Execution and Reporting Sub-Service)

The notifications listed above are generated by the CORDET Framework infrastructure. The operations which generate them are defined as adaptation points. The PUS Extension closes these adaptation points to generate the service 1 reports.

An example may help clarify the mechanism through which the service 1 reports are generated. The InCommand state machine of the CORDET Framework defines the generic behaviour of incoming commands. Among other things, this state machine stipulates that, when the execution of an incoming command has been successfully completed, the Report-Termination-Successful Operation is called to notify other parts of the application that the command has successfully terminated. At the level of the CORDET Framework, this operation is defined as an adaptation point (because, at this level, it is not possible to define how and to whom the notification of successful completion should be distributed). At the level of the PUS Extension this adaptation point is closed by having the Report-Termination-Successful Operation generate a service 1 report of type (1,7).

The notifications generated by the CORDET Framework are generated in response to checks performed on incoming commands. However, the PUS stipulates that execution notifications may also be generated in response to checks performed on individual instructions embedded within a command. These notifications cannot be generated by the CORDET Framework which only handles abstract commands. These execution notifications are therefore generated by individual commands as part of their processing of their own instructions. An example may again help clarify this logic. The PUS command of type (3,5) carries several instructions each of which enables one housekeeping report. The processing of these instructions is done by the actions associated to the command itself and the generation of the instruction-level notifications is therefore done by these actions. Note that, as discussed in section 3.4, for instructions, only execution failures are reported.

By way of summary, table 8.1 lists the sources of all notifications which may trigger service 1 reports. For notifications which are issued by the CORDET Framework infrastructure, the rightmost column in the table identifies the corresponding adaptation point.

Table 8.1: Sources of Routing, Acceptance and Execution Notifications

Notification	**Source**	**AP**
Routing Failure Notification	This notification is issued by the *Report Packet Destination Invalid Operation* which is called by the InLoader Execution Procedure when an application has received a command or report with a destination which is neither the application itself nor some other known application.	ILD-12
Acceptance Failure Notification	This notification is issued by the *Report Acceptance Failure Operation* which is called by the InLoader Load Command/Report Procedure when: (a) an incoming command has failed its acceptance check, or (b) the InCommand component could not be created either to a lack or resources or to the triplet [type,sub-type,discriminant] being illegal, or (c) the InCommand component could not be loaded into its InManager due to the InManager Pending Command/Report List (PCRL) being full.	ILD-14
Acceptance Success Notification	This notification is issued by the *Report Acceptance Success Operation* which is called by the InLoader Load Command/Report Procedure when an incoming command has passed its Acceptance Check and that command has requested acknowledgement of successful acceptance.	ILD-15
Execution Start Success Notification	This notification is issued by the *Report Start Successful for InCommand Operation* which is called by the InCommand State Machine when the Start Action of an incoming command has a 'success' outcome and that command has requested acknowledgement of successful start of execution.	ICM-13
Execution Start Failure Notification	This notification is issued by the *Report Start Failed for InCommand Operation* which is called by the InCommand State Machine when the Start Action of an incoming command has a 'failure' outcome. The same operation may also be called by the implementation of the Start Action of a command to report the failure of an instruction within the command.	ICM-12
Execution Progress Success Notification	This notification is issued by the *Report Progress Successful for InCommand Operation* which is called by the InCommand State Machine when the Progress Action of an incoming command has a 'success' outcome and that command has requested acknowledgement of successful progress of execution.	ICM-15
Execution Progress Failure Notification	This notification is issued by the *Report Progress Failed for InCommand Operation* which is called by the InCommand State Machine when the Progress Action of an incoming command has a 'failure' outcome. The same operation may also be called by the implementation of the Progress Action of a command to report the failure of the execution step of an instruction within the command.	ICM-14

Notification	Source	AP
Execution Termination Success Notification	This notification is issued by the *Report Termination Successful for InCommand Operation* which is called by the InCommand State Machine when the Termination Action of an incoming command has a 'success' outcome and that command has requested acknowledgement of successful termination of execution.	ICM-17
Execution Termination Failure Notification	This notification is issued by the *Report Termination Failed for InCommand Operation* which is called by the InCommand State Machine when the Termination Action of an incoming command has a 'failure' outcome.	ICM-16

The framework extension closes the adaptation points in table 8.1 with behaviour which generates the service 1 verification reports. The first row in the table corresponds to a situation where a packet cannot be re-routed, which if the packet contains a command, is the situation where the PUS prescribes that a (1,10) report should be generated. The other rows correspond to situations where an incoming command has either failed or passed one of its processing checks and they are therefore closed with the generation of the service 1 reports (1,1) to (1,8).

The close-out behaviour for the adaptation points is defined in table 8.12. It consists of running a procedure which creates the service 1 report, configures it, and then loads it into the OutLoader. The report is created by calling the `Make` operation of the OutFactory. This may fail if the OutFactory has run out of resources for new reports. In that case, error report OUTFACTORY_FAIL is generated. Procedures which report failures also update the relevant observables (see section 8.2).

The failure code of failure reports in service 1 is treated as a discriminant. This allows applications to selectively disable certain failure reports by using the enable mechanism of the OutRegistry component of the CORDET Framework. It is recalled that this mechanism allows the OutRegistry to be configured to disable out-going reports by 'kind' where the kind of a report is defined by the triplet: [type, sub-type, discriminant].

8.1 Service 1 Report and Command Definition

There are no commands in service 1. The service is only implemented by reports. In the CORDET Framework an out-going report is encapsulated in an OutComponent component. The framework extension offers, for each service 1 report, an component to encapsulate it. These components are implemented as extension of the OutComponent component. They are therefore defined by the way they close the adaptation points of the OutComponent. The tables in this section list the OutComponent adaptation points and show how they are closed for the service 1 components.

The PUS defines the content of the service 1 reports in section 8.1 of AD-3. The 'success' reports carry the packet identifier of the command being verified. The 'failure' reports carry, in addition to the packet identifier, a failure code and an undefined set of failure-related data. The framework extension restricts this flexibility by stipulating that the failure-related data consist of:

- For all failure reports: the triplet [type,sub-type,discriminant] for the command being verified

- For all failure reports but (1,10) reports: the *Verification Failure Data* as a single data item which contains command-specific information about the failure
- For (1,10) reports only: the destination of the command which failed its routing check
- For (1,5) reports only: the identifier of the step which failed its progress check

The Verification Failure Code and the Verification Failure Data are stored in data pool items `verFailCode` and `verFailData`. The purpose of `verFailData` is to provide additional information about the nature of the failure being reported by the failure report. This data item has a fixed size but its syntactical type is command-specific. Its value is set by the entity which performs the verification check. If no failure data are defined for a given verification check, then the value of `verFailData` is "don’t care".

To illustrate, consider the case of a command (3,5) which enables a housekeeping report. This command carries the Structure Identifier (SID) of the report to be enabled. The Start Action of this command checks the legality of the SID (see section 9). If the SID is found to be illegal, the command is rejected with a (1,4) report and the illegal SID value is used as Verification Failure Data. The Start Action of the (1,4) command loads the illegal SID into data pool item `verFailData` and the Command Verification Failure Procedure which creates the (1,4) report takes the Verification Failure Data from `verFailData`.

The Verification Failure Codes which are supported by the PUS Extension are listed in appendix C. These failure codes cover the failure conditions for the commands defined by the PUS Extension. For each failure code, the associated verification failure data is also defined. Applications should extend the table in appendix C with the failure codes for their own commands.

The tables in this section formally specify the service 1 components by specifying how the actions, checks and attributes of a generic out-going report are specialized for service 1 (see section 7). The following remarks apply:

- Service 1 reports retrieve their enable status from the OutRegistry.
- Service 1 reports are generated as soon as the condition which triggered them occur and hence their ready check always returns ’ready’
- Service 1 reports are ’one-off’ reports and hence their repeat check always returns ’no repeat’

With reference to the first bullet, it is recalled that the OutRegistry component of the CORDET Framework stores the enable status of out-going reports as a function of the report’s type, sub-type and discriminant. By default, all out-going reports are enabled. Users who wish to disable a specific service 1 sub-type can do so by setting its status to ’disabled’ in the OutRegistry. This is a run-time operation which would typically be done as part of an application’s initialization.

Table 8.2: Specification of VerSuccAccRep Component

Name	VerSuccAccRep(1,1)
Description	Report generated to mark the successful acceptance of an incoming command
Parameters	Packet identifier and packet sequence control of telecommand being acknowledged
Discriminant	None

Name	VerSuccAccRep(1,1)
Destination	The destination of service 1 reports is set equal to the source of the command being verified
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 8.3: Specification of VerFailedAccRep Component

Name	VerFailedAccRep(1,2)
Description	Report generated to mark the acceptance failure of an incoming command
Parameters	Packet version number followed by information on the command being acknowledged: packet identifier, packet sequence counter, type, sub-type and discriminant, failure code and one single item of failure data (specific to each failure code).
Discriminant	Failure Identification Code
Destination	The destination of service 1 reports is set equal to the source of the command being verified
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 8.4: Specification of VerSuccStartRep Component

Name	VerSuccStartRep(1,3)
Description	Report generated to mark the successful start of execution of an incoming command
Parameters	Packet identifier and packet sequence control of telecommand being acknowledged
Discriminant	None
Destination	The destination of service 1 reports is set equal to the source of the command being verified
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 8.5: Specification of VerFailedStartRep Component

Name	VerFailedStartRep(1,4)
Description	Report generated to mark the start of execution failure of an incoming command
Parameters	Packet version number followed by information on the command being acknowledged: packet identifier, packet sequence counter, type, sub-type and discriminant, failure code and one single item of failure data (specific to each failure code).
Discriminant	Failure Identification Code
Destination	The destination of service 1 reports is set equal to the source of the command being verified
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 8.6: Specification of VerSuccPrgrRep Component

Name	VerSuccPrgrRep(1,5)
Description	Report generated to mark the successful completion of an execution step of an incoming command
Parameters	Packet identifier and packet sequence control of telecommand being acknowledged
Discriminant	None
Destination	The destination of service 1 reports is set equal to the source of the command being verified
Enable Check	#TM(1,5)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 8.7: Specification of VerFailedPrgrRep Component

Name	VerFailedPrgrRep(1,6)
Description	Report generated to mark the failure of an execution step of an incoming command
Parameters	Packet version number followed by information on the command being acknowledged: packet identifier, packet sequence counter, type, sub-type and discriminant, failure code and one single item of failure data (specific to each failure code); identifier of progress step which failed
Discriminant	Failure Identification Code
Destination	The destination of service 1 reports is set equal to the source of the command being verified
Enable Check	#TM(1,6)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 8.8: Specification of VerSuccTermRep Component

Name	VerSuccTermRep(1,7)
Description	Report generated to mark the successful completion of execution of an incoming command
Parameters	Packet identifier and packet sequence control of telecommand being acknowledged
Discriminant	None
Destination	The destination of service 1 reports is set equal to the source of the command being verified
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 8.9: Specification of VerFailedTermRep Component

Name	VerFailedTermRep(1,8)
Description	Report generated to mark the failure to complete execution of an incoming command
Parameters	Packet version number followed by information on the command being acknowledged: packet identifier, packet sequence counter, type, sub-type and discriminant, failure code and one single item of failure data (specific to each failure code).
Discriminant	Failure Identification Code
Destination	The destination of service 1 reports is set equal to the source of the command being verified
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 8.10: Specification of VerFailedRoutingRep Component

Name	VerFailedRoutingRep(1,10)
Description	Report generated to mark the failure to route an incoming command to its final destination
Parameters	Packet version number followed by information on the command whose routing failed: packet identifier, packet sequence counter, type, sub-type and discriminant, and invalid destination
Discriminant	None
Destination	The destination of service 1 reports is set equal to the source of the command being verified
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

8.2 Service 1 Observables

Service 1 maintains and makes available in the data pool various information related to the generation of the failure reports. No information related to the generation of the success reports is maintained because these reports are optional and the conditions under which they are generated depend on the setting of the verification acknowledge flags which are under external control (they are set by the user of a service). Table 8.11 lists the data pool data items which are maintained by service 1.

Table 8.11: Observables for Verification Service

Name	Description
`failCodeAccFailed`	Failure code of last command which failed its Acceptance
`failCodePrgrFailed`	Failure code of last command which failed its Progress Check
`failCodeStartFailed`	Failure code of last command which failed its Start Check
`failCodeTermFailed`	Failure code of last command which failed its Termination
`invDestRerouting`	Destination of last command for which re-routing failed
`nOfAccFailed`	Number of commands which have failed their acceptance check
`nOfPrgrFailed`	Number of commands which have failed their Progress Check
`nOfReroutingFailed`	Number of commands for which re-routing failed
`nOfStartFailed`	Number of commands which have failed their Start Check since
`nOfTermFailed`	Number of commands which have failed their Termination
`pcktIdAccFailed`	Packet identifier of last command which failed its Acceptance
`pcktIdPrgrFailed`	Packet identifier of last command which failed its Progress
`pcktIdReroutingFailed`	Packet identifier of last command for which re-routing failed

Name	Description
`pcktIdStartFailed`	Packet identifier of last command which failed its Start Check
`pcktIdTermFailed`	Packet identifier of last command which failed its Termination
`stepPrgrFailed`	Step identifier of last command which failed its Progress Check
`verFailCode`	Verification Failure Code
`verFailData`	Verification Failure Data (data item of fixed size but variable meaning)

8.3 Service 1 Adaptation Points

Table 8.12 lists the CORDET Framework adaptation points which are closed or overridden by the request verification service.

Table 8.12: Adaptation Points for Service 1 (Request Verification)

AP ID	Adaptation Point	Close-Out Value
P-S1-1	Operation to Report Packet Destination Invalid by InLoader (Closes ILD-12)	Run the Packet Re-Routing Failure Procedure of figure 8.1
P-S1-2	Operation to Report Acceptance Failure by InLoader (Closes ILD-14)	Run the Packet Acceptance Failure Procedure of figure 8.2
P-S1-3	Operation to Report Acceptance Success by InLoader (Closes ILD-13)	Run the Command Verification Success Procedure of figure 8.3
P-S1-4	Operation to Report Start Failed for InCommand (Closes ICM-12)	Run the Command Verification Failure Procedure of figure 8.4
P-S1-5	Operation to Report Start Successful for InCommand (Closes ICM-13)	Run the Command Verification Success Procedure of figure 8.5
P-S1-6	Operation to Report Progress Failed for InCommand (Closes ICM-14)	Run the Command Progress Failure Procedure 8.6
P-S1-7	Operation to Report Progress Successful for InCommand (Closes ICM-15)	Run the Command Progress Success Procedure 8.5
P-S1-8	Operation to Report Termination Failed for InCommand (Closes ICM-16)	Run the Command Verification Failure Procedure of figure 8.3
P-S1-9	Operation to Report Report Termination Successful for InCommand (Closes ICM-17)	Run the Command Verification Success Procedure of figure 8.5

8.4 Service 1 Requirements

The table in this section lists requirements for the request verification service.

Table 8.13: Requirements for Service 1 (Request Verification)

Req. ID	**Requirement Text**
P-S1-1/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, a ReqVerRep component to encapsulate a service 1 report*
P-S1-2/A	*The ReqVerRep component shall close the OutComponent adaptation points as indicated in table* ??
P-S1-3/A	*The service 1 implementation of the PUS Extension of the CORDET Framework shall close or override the InLoader and InCommand adaptation points listed in table 8.12*
P-S1-4/S	*The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 8.11*
P-S1-5/S	*The PUS Extension of the CORDET Framework shall support the service 1 failure codes listed in table C.1*
P-S1-6/C	*If an application performs a verification check for a command and the check fails, it shall update the Verification Failure Data in the data pool with either zero or with a command-specific failure data item*
P-S1-7/C	*Applications shall be responsible for configuring the OutRegistry component to selectively disable failure verification reports which they do not need*

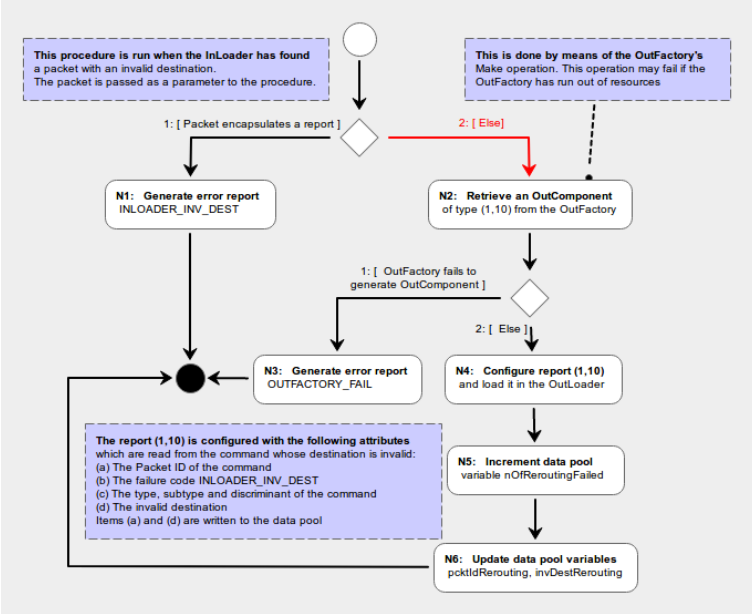

This procedure is run when the InLoader has found
a packet with an invalid destination.
The packet is passed as a parameter to the procedure.
This is done by means of the OutFactory's
Make operation. This operation may fail if the
OutFactory has run out of resources
1: [Packet encapsulates a report]
2: [Else]
N1: Generate error report
INLOADER_INV_DEST
N2: Retrieve an OutComponent
of type (1,10) from the OutFactory
1: [OutFactory fails to
generate OutComponent]
2: [Else]
N3: Generate error report
OUTFACTORY_FAIL
N4: Configure report (1,10)
and load it in the OutLoader
The report (1,10) is configured with the following attributes
which are read from the command whose destination is invalid:
(a) The Packet ID of the command
(b) The failure code INLOADER_INV_DEST
(c) The type, subtype and discriminant of the command
(d) The invalid destination
Items (a) and (d) are written to the data pool
N5: Increment data pool
variable nOfReroutingFailed
N6: Update data pool variables
pcktIdRerouting, invDestRerouting

Fig. 8.1: Packet Rerouting Failure Procedure

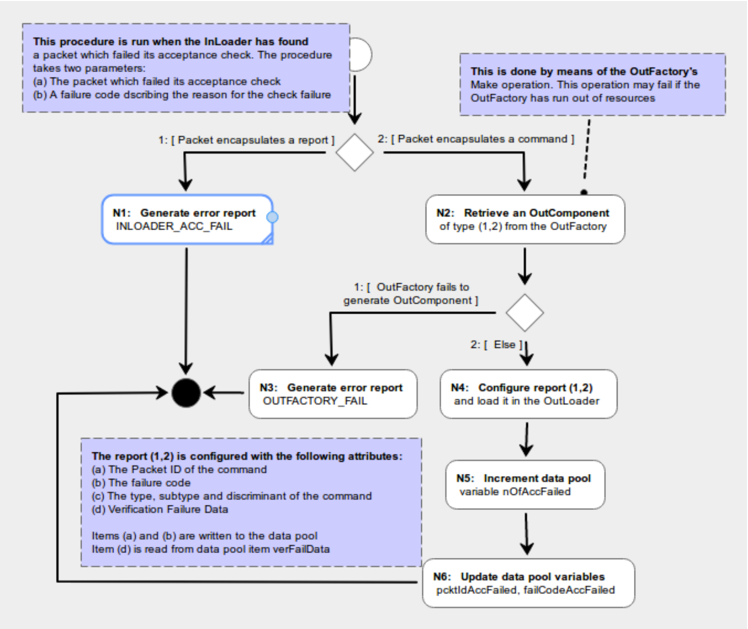

This procedure is run when the InLoader has found
a packet which failed its acceptance check. The procedure
takes two parameters:
(a) The packet which failed its acceptance check
(b) A failure code dscribing the reason for the check failure
This is done by means of the OutFactory's
Make operation. This operation may fail if the
OutFactory has run out of resources
1: [Packet encapsulates a report]
2: [Packet encapsulates a command]
N1: Generate error report
INLOADER_ACC_FAIL
N2: Retrieve an OutComponent
of type (1,2) from the OutFactory
1: [OutFactory fails to
generate OutComponent]
2: [Else]
N3: Generate error report
OUTFACTORY_FAIL
N4: Configure report (1,2)
and load it in the OutLoader
The report (1,2) is configured with the following attributes:
(a) The Packet ID of the command
(b) The failure code
(c) The type, subtype and discriminant of the command
(d) Verification Failure Data
Items (a) and (b) are written to the data pool
Item (d) is read from data pool item verFailData
N5: Increment data pool
variable nOfAccFailed
N6: Update data pool variables
pcktIdAccFailed, failCodeAccFailed

Fig. 8.2: Packet Acceptance Failure Procedure

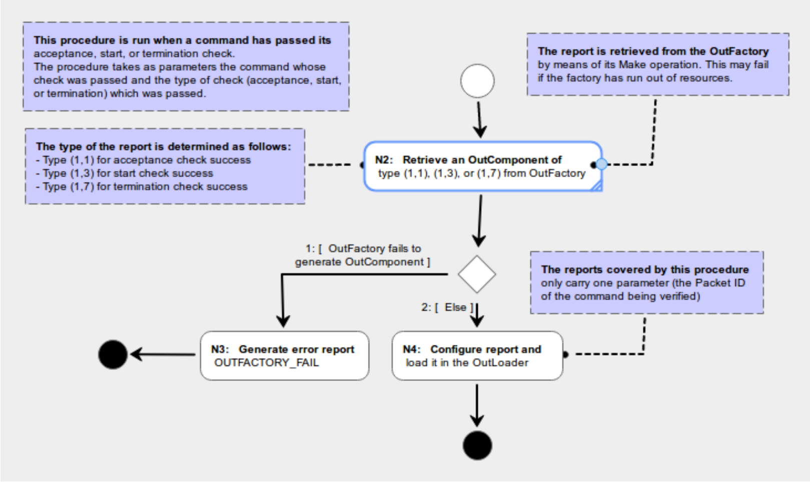

This procedure is run when a command has passed its acceptance, start, or termination check. The procedure takes as parameters the command whose check was passed and the type of check (acceptance, start, or termination) which was passed.
The report is retrieved from the OutFactory by means of its Make operation. This may fail if the factory has run out of resources.
The type of the report is determined as follows:
- Type (1,1) for acceptance check success
- Type (1,3) for start check success
- Type (1,7) for termination check success
N2: Retrieve an OutComponent of type (1,1), (1,3), or (1,7) from OutFactory
1: [OutFactory fails to generate OutComponent]
2: [Else]
The reports covered by this procedure only carry one parameter (the Packet ID of the command being verified)
N3: Generate error report OUTFACTORY_FAIL
N4: Configure report and load it in the OutLoader

Fig. 8.3: Command Verification Success Procedure

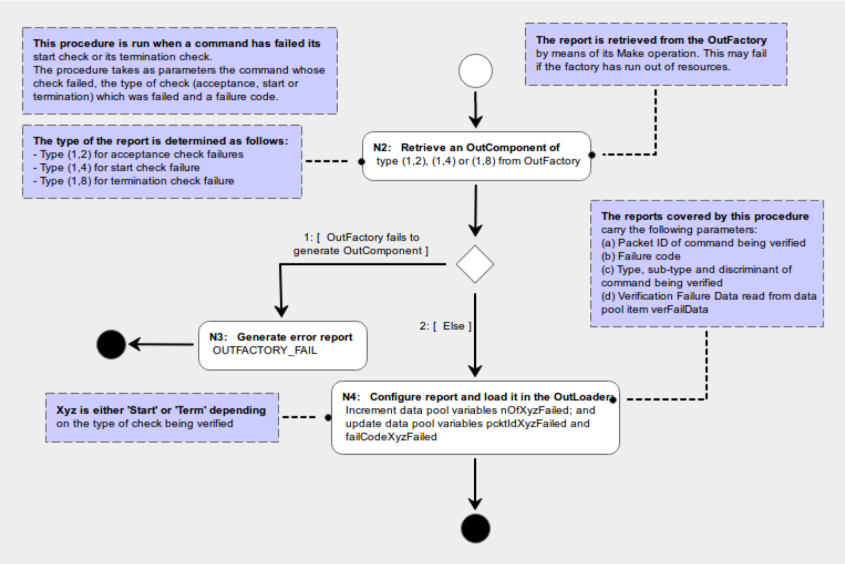

This procedure is run when a command has failed its start check or its termination check. The procedure takes as parameters the command whose check failed, the type of check (acceptance, start or termination) which was failed and a failure code.
The report is retrieved from the OutFactory by means of its Make operation. This may fail if the factory has run out of resources.
The type of the report is determined as follows:
- Type (1,2) for acceptance check failures
- Type (1,4) for start check failure
- Type (1,8) for termination check failure
N2: Retrieve an OutComponent of type (1,2), (1,4) or (1,8) from OutFactory
The reports covered by this procedure carry the following parameters:
(a) Packet ID of command being verified
(b) Failure code
(c) Type, sub-type and discriminant of command being verified
(d) Verification Failure Data read from data pool item verFailData
1: [OutFactory fails to generate OutComponent]
2: [Else]
N3: Generate error report OUTFACTORY_FAIL
Xyz is either 'Start' or 'Term' depending on the type of check being verified
N4: Configure report and load it in the OutLoader
Increment data pool variables nOfXyzFailed; and update data pool variables pcktIdXyzFailed and failCodeXyzFailed

Fig. 8.4: Command Verification Failure Procedure

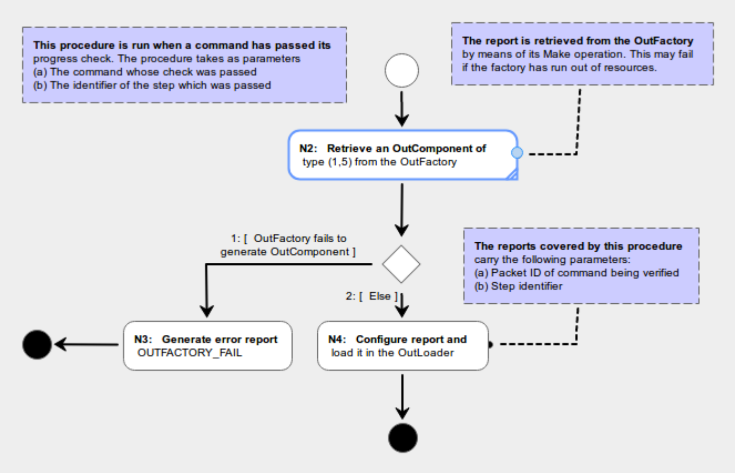

This procedure is run when a command has passed its progress check. The procedure takes as parameters
(a) The command whose check was passed
(b) The identifier of the step which was passed
The report is retrieved from the OutFactory by means of its Make operation. This may fail if the factory has run out of resources.
N2: Retrieve an OutComponent of type (1,5) from the OutFactory
1: [OutFactory fails to generate OutComponent]
2: [Else]
The reports covered by this procedure carry the following parameters:
(a) Packet ID of command being verified
(b) Step identifier
N3: Generate error report OUTFACTORY_FAIL
N4: Configure report and load it in the OutLoader

Fig. 8.5: Command Progress Success Procedure

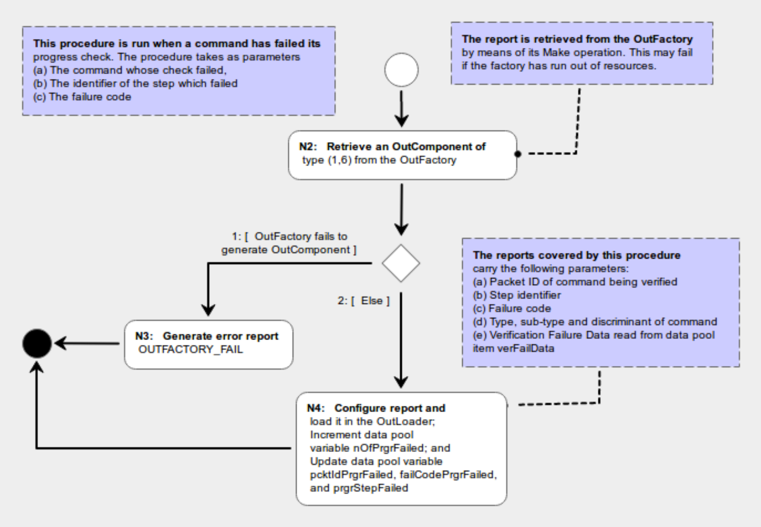

This procedure is run when a command has failed its progress check. The procedure takes as parameters
(a) The command whose check failed,
(b) The identifier of the step which failed
(c) The failure code
The report is retrieved from the OutFactory by means of its Make operation. This may fail if the factory has run out of resources.
N2: Retrieve an OutComponent of type (1,6) from the OutFactory
1: [OutFactory fails to generate OutComponent]
2: [Else]
The reports covered by this procedure carry the following parameters:
(a) Packet ID of command being verified
(b) Step identifier
(c) Failure code
(d) Type, sub-type and discriminant of command
(e) Verification Failure Data read from data pool item verFailData
N3: Generate error report OUTFACTORY_FAIL
N4: Configure report and load it in the OutLoader; Increment data pool variable nOfPrgrFailed; and Update data pool variable pcktIdPrgrFailed, failCodePrgrFailed, and prgrStepFailed

Fig. 8.6: Command Progress Failure Procedure

9 Housekeeping Service

The service type of the Housekeeping Service is 3. The PUS Extension of the CORDET Framework supports this service only in part.

The housekeeping service provides the capability to create, delete and control housekeeping and diagnostic reports. The service 3 commands and reports in the PUS are duplicated being defined once for housekeeping reports and once for diagnostic reports. The PUS framework supports both sets of commands and reports but does not otherwise make any distinction between housekeeping and diagnostic reports. It is essentially up to the user to decide which service 3 reports should be treated as 'housekeeping reports' and which ones should instead be treated as 'diagnostic reports'.

A housekeeping/diagnostic report carries the values of a set of data pool items[4]. Any data pool item may be included in a housekeeping/diagnostic report.

At any given time, an application generates several kinds of housekeeping/diagnostic reports which differ for the set of data items they hold and for the frequency with which they are generated. The housekeeping/diagnostic reports use the discriminant attribute to manage this variability. Thus, two different kinds of housekeeping/diagnostic reports are distinguished by different values of discriminant attribute. In keeping with the PUS convention, the discriminant attribute of a housekeeping/diagnostic report is called *Structure Identifier* or SID. The SID must be a positive integer in the range: 1..HK_MAX_SID.

Since no distinction is made between housekeeping and diagnostic reports, the SID must be unique within the set of all housekeeping/diagnostic reports (i.e. it is not possible for a housekeeping report and a diagnostic report to have the same SID).

Housekeeping/diagnostic reports may be generated periodically or in "one-shot" mode. For periodic reports, the *Collection Period* is the period with which the report is generated. The Collection Period is expressed as an integer multiple of a minimum period HK_COLLECT_PER which is an application constant. A value of zero for the Collection Period indicates that the report must be generated in "one-shot" mode.

A data item in a housekeeping/diagnostic report is either *simply commutated* or *super-commutated*. The value of a simply-commutated data item appears only once in the housekeeping/diagnostic report and it represents the value of the data item at the time the report is generated.

The value of a super-commutated data item instead appears multiple times within a housekeeping/diagnostic report. Super-commutated data items in a report are divided into *groups*. To each group, a *sample repetition number* N is associated: a report carries N values of the data items in the super-commutated group. These N values have been generated by sampling the data items at N distinct points in time within the collection period. The PUS stipulates that the N collection points must be equally spaced within the collection interval but this constraint is not enforced by the framework (but may be enforced at application-level).

The PUS also stipulates that, within a housekeeping/diagnostic report definition, each data item appears only once, either as a simply commutated parameter or as a super-commutated parameter. This restriction is not enforced by the framework.

[4] The PUS uses the term 'parameter' to designate the data pool items whose values are carried by the housekeeping and diagnostic reports.

9.1 Report Definition List (RDL)

The Reporting Definition List or RDL is a data structure which holds the current configuration of the housekeeping/diagnostic reports. The content of the RDL is updated by the service 3 commands and, on request, it may be reported by service 3 reports.

The RDL holds HK_N_REP_DEF *Report Definitions*. The value of HK_N_REP_DEF is an application constant. It represents the maximum number of housekeeping/diagnostic reports which may be defined at a given time.

Each Report Definition defines one housekeeping/diagnostic report in terms of the fields listed in table 9.1. Rows 6 to 9 determine the content of the report. The data items in a housekeeping/diagnostic report are arranged as a sequence of data item values according to the layout specified in clause 6.3.3.3 of [PS-SP]. The total number of reported data items is: (nSimple+nRep[1]+ .. +nRep[nGroup]), of which the first nSimple are simply-commutated whereas the others are split into nGroup groups of super-commutated data items. For each data item in the i-th group, rep[i] values are reported which have been collected at rep[i] times within the collection interval. The total number of data item values in a report therefore is: (nSimple+nRep[1]*rep[1]+ .. +nRep[nGroup]*rep[nGroup])

The parameters HK_MAX_* are application constants. Applications which do not need super-commutated data can set HK_MAX_N_GR to zero.

The sampling buffer mentioned in the last row in table 9.1 is discussed in the next section.

Several service 3 commands operate on a set of RDL entries (e.g. command (3,3) requests that a list of SIDs be deleted from the RDL). In such cases, it is useful to "mark" an entry in the RDL. For this purpose, flag isMarked has been introduced in table 9.1.

Table 9.1: Fields in Report Definition Data Structure

Field Name	Description	Constraint
sid	Structure identifier (SID)	Integer in range: 1..HK_MAX_SID
period	Collection period in units of HK_COLLECT_PER	Positive integer (periodic reports) or zero (one-shot reports)
cycleCnt	Cycle counter (see definition of service 3 reports and commands)	Integer in the range: 0..(period-1)
isEnabled	True if the report is enabled	None
dest	The identifier of the application to which the report is sent	None
nSimple	Number of simply-commutated data items in the report	Integer in range: 1..HK_MAX_N_SIMPLE
lstSampleRep	List of super commutated sample repetition numbers (rep[1] .. rep[nGroup])	The number of groups is in the range: 0..HK_MAX_N_GR and each repetition number is in the range: 1..HK_MAX_REP
lstNSampRep	List of numbers (nRep[1] .. nRep[nGroup]) of data items in each super-commutated group	Each nRep[i] is in range: 1..HK_MAX_N_REP
lstId	List of identifiers of data items in the report	Not more than HK_MAX_N_ITEMS data items and each identifier is in range: 1..HK_MAX_ID

Field Name	Description	Constraint
`sampleBufId`	The identifier of the sampling buffer holding the super-commutated data item values	An integer in the range: 1..HK_N_SAMP_BUF
`isMarked`	Marker flag	Boolean flag

9.2 Management of Super-Commutated Data Items

The housekeeping service is responsible for collecting the values of the data items in housekeeping/diagnostic packets. For simply-commutated data items, the values are collected directly from the data pool. For super-commutated data items, the values are collected from a *Sampling Buffer*. Each sampling buffer holds the values of the super-commutated data items for a given housekeeping/diagnostic report.

The super-commutated data items in a report are arranged in `nGroup` groups. The i-th group covers `nRep[i]` items which are sampled `nRep[i]` times within a collection period. Hence, in each collection period, the i-th group contributes: `nRep[i]*rep[i]` data item values. The sampling buffer for a given housekeeping/diagnostic report must be large enough to hold the data item values collected in one collection period for all super-commutated groups in that report.

The number of sampling buffers is HK_N_SAMP_BUF. The value of HK_N_SAMP_BUF is an application constant. It represents the maximum number of housekeeping/diagnostic reports with super-commutated data items which may be defined at a given time. This may be smaller than the maximum number HK_N_REP_DEF of housekeeping/diagnostic reports. Thus, for instance, an application might stipulate that there may be up to 10 housekeeping/diagnostic reports but only two of these may contain super-commutated data items. This application would set HK_N_REP_DEF to 10 and HK_N_SAMP_BUF to 2.

The association between housekeeping/diagnostic report and its sampling buffer is done dynamically: if a report has super-commutated data items, the last field in its report definition contains a pointer to its sampling buffer (see table 9.1).

The periodic collection of the values of the simply-commutated data items is done by the components hkRep which encapsulate a housekeeping/diagnostic report (see section 9.4). These components are executed once per collection interval. They therefore cannot collect the values of the super-commutated data items which are sampled several times per collection period. Responsibility for the collection of the values of the super-commutated data items rests with the application instantiated from the framework.

The framework offers the following functions to manipulate a sampling buffer:

- *Sampling Buffer Configuration Function* to configure a sampling buffer as a function of the number of groups, the number of data items in each group and the repetition number for each group.
- Sampling Buffer Setter Function to load the i-th value of the j-th data item in the k-th group in the sampling buffer.
- Sampling Buffer Getter Function to retrieve the i-th value of the j-th data item in the k-th group in the sampling buffer.

The Configuration Function is used when a housekeeping/diagnostic report which contains super-commutated data items is created (either at application initialization time for a pre-defined report or in response to a (3,1)/(3,2) command for a dynamically defined report). The Setter Function is used by the application to load the super-commutated values in the sampling buffer. The Getter Function is used in the Update Action of the (3,25) and (3,26) reports to update the content of a housekeeping/diagnostic report.

9.3 Debug Variables

Service 3 offers visibility over the internal state of the IFSW by allowing periodic or sporadic access to the data items in the data pool. The data items in the data pool are defined at design time and should cover all application functions. For situations where additional visibility is required (e.g. in case of debugging during AIT activities), the framework the concept of *debug variables* is introduced. A debug variable is a variable of 4 bytes of length whose address in RAM is a data pool parameter. More precisely, a total of N_DEBUG_VAR debug variables are defined which are encapsulated in data pool variables `debugVar_x` where x ranges from 1 to HK_N_DEBUG_VAR. Additionally, data pool parameters `debugVarAddr_x` are defined to hold the address of `debugVar_x`. The Execution Procedure of the data pool (see section 5.2) loads the values of the memory locations pointed at by the elements of `debugVarAddr` into the elements of `debugVar`.

In order to illustrate the use of the debug variables, consider a situation where the user wishes to have read access to two memory locations holding two integers:

1. The user loads the addresses of the desired locations into the first two elements of `debugVarAddr`
2. The user uses command (3,1) or (3,2) to define a new housekeeping report packet holding `debugVar_1` and `debugVar_2`
3. The users uses command (3,6) or (3,7) to enable the newly defined housekeeping packet and receives the values of `debugVar_1` and `debugVar_2` .

9.4 Service 3 Report and Command Definition

The tables in this section formally specify the service 3 commands and reports by specifying how the actions, checks and attributes of generic out-going commands and reports are specialized for service 3 (see section 7). The following remarks apply:

- In the PUS, service 3 commands and reports appear twice: once for housekeeping reports and once for diagnostic reports. The PUS Extension of the CORDET Framework does not distinguish between housekeeping and diagnostic reports/commands and therefore each CORDET report/command component implements two PUS reports/commands.
- Several commands in this service (e.g. the commands to delete a housekeeping/diagnostic report definition) carry multiple instructions which are executed independently of each other. In keeping with the general strategy outlined in section 3.4, their start action evaluates the instructions one by one and, in case of invalidity, it generates a (1,4) report for each individual instruction.
- The (3,9) and (3,27) commands carry a sequence of SIDs. The command's Start Action removes invalid SIDs. The valid SIDs are then processed by the command's Progress Action. Each SID is processed in a progress step. In keeping with the strategy of section 3.4, only step failures are reported through service 1 reports. The command

is deemed to have completed successfully if at least one SID has been successfully processed.

- For the housekeeping/diagnostic reports (3,25) and (3,26), two components are provided of which one is used when the reports are generated on a periodic basis and the other is used when the reports are generated in óne-shot' mode in response to a (3,27) or (3,28) command.

Table 9.2: Specification of HkCreHkCmd Component

Name	HkCreHkCmd(3,1)
Description	Create a housekeeping report structure
Parameters	SID, collection interval and identifiers of parameters of the report to be created
Discriminant	The structure identifier (SID) of the packet to be created
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of HkCreate Command of figure 9.1
Progress Action	Add the definition of the new report to the RDL, set its enabled status to 'disabled', and set the action outcome to 'completed'
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 9.3: Specification of HkCreDiagCmd Component

Name	HkCreDiagCmd(3,2)
Description	Create a diagnostic report structure
Parameters	SID, collection interval and identifiers of parameters of the diagnostic report to be created
Discriminant	The structure identifier (SID) of the packet to be created
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of HkCreate Command of figure 9.1
Progress Action	Add the definition of the new report to the RDL, set its enabled status to 'disabled', and set the action outcome to 'completed'
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 9.4: Specification of HkDelHkCmd Component

Name	HkDelHkCmd(3,3)
Description	Delete one or more housekeeping report definitions
Parameters	List of SIDs of reports whose definition is to be deleted
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure of 9.2 to identify the valid SIDs in the command argument
Progress Action	Delete the entries in the RDL corresponding to the SIDs which have been identified as valid by the Start Action and then set the action outcome to 'completed'
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 9.5: Specification of HkDelDiagCmd Component

Name	HkDelDiagCmd(3,4)
Description	Delete one or more diagnostic report definitions
Parameters	List of SIDs of reports whose definition is to be deleted
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure of 9.2 to identify the valid SIDs in the command argument
Progress Action	Delete the entries in the RDL corresponding to the SIDs which have been identified as valid by the Start Action and then set the action outcome to 'completed'
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 9.6: Specification of HkEnbHkCmd Component

Name	HkEnbHkCmd(3,5)
Description	Enable the periodic generation of one or more housekeeping report structures
Parameters	List of SIDs to be enabled
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of Multi-SID Command of figure 9.3
Progress Action	For the entries in the RDL corresponding to the SIDs which have been identified as valid by the Start Action: set enabled flag to true and set the cycle counter to 0. Set the action outcome to 'completed'
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 9.7: Specification of HkDisHkCmd Component

Name	HkDisHkCmd(3,6)
Description	Disable the periodic generation of one or more housekeeping report structures
Parameters	List of SIDs to be disabled
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of Multi-SID Command of figure 9.3
Progress Action	Set to false the enable flag of the entries in the RDL corresponding to the SIDs which have been identified as valid by the Start Action and then set the action outcome to 'completed'
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 9.8: Specification of HkEnbDiagCmd Component

Name	HkEnbDiagCmd(3,7)
Description	Enable the periodic generation of one or more diagnostic report structures
Parameters	List of SIDs to be enabled
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of Multi-SID Command of figure 9.3
Progress Action	For the entries in the RDL corresponding to the SIDs which have been identified as valid by the Start Action: set enabled flag to true and set the cycle counter to 0. Set the action outcome to 'completed'
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 9.9: Specification of HkDisDiagCmd Component

Name	HkDisDiagCmd(3,8)
Description	Disable the periodic generation of one or more diagnostic report structures
Parameters	List of SIDs to be disabled
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of Multi-SID Command of figure 9.3
Progress Action	#TM(3,60
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 9.10: Specification of HkRepStructHkCmd Component

Name	HkRepStructHkCmd(3,9)
Description	This command carries a list of SIDs. For each SID, it triggers the generation of a (3,10) report with the definition of the housekeeping report structure for that SID.
Parameters	List of SIDs whose structure is to be reported
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of Multi-SID Command of figure 9.3
Progress Action	Run the procedure Progress Action of Report Service 3 Structure of figure 9.4
Termination Action	Set action outcome to 'success' if all valid SIDs in the command were successfully processed by the progress action; set it to 'failure' otherwise
Abort Action	Default implementation (set action outcome to 'success')

Table 9.11: Specification of HkRepStructHkRep Component

Name	HkRepStructHkRep(3,10)
Description	Report carrying the definition of a housekeeping report structure generated in response to a (3,9) command.
Parameters	SID of the housekeeping report, flag indicating whether periodic generation of the report is enabled, number of simply commutated parameters in the report and their identifiers, number of super-commutated groups and, for each group, number of parameters in the group and their identifiers
Discriminant	Structure Identifier
Destination	The destination is set equal to the source of the (3,9) command which triggers the report.
Enable Check	The enable status is read from the isEnabled field of the Report Definition corresponding to the report's SID
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Load the SID definition from the RDL

Table 9.12: Specification of HkRepStructDiagCmd Component

Name	HkRepStructDiagCmd(3,11)
Description	This command carries a list of SIDs. For each SID, it triggers the generation of a (3,12) report with the definition of the diagnostic report structure for that SID.
Parameters	List of SIDs whose structure is to be reported
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of Multi-SID Command of figure 9.3
Progress Action	Run the procedure Progress Action of Report Service 3 Structure of figure 9.4
Termination Action	Set action outcome to 'success' if all valid SIDs in the command were successfully processed by the progress action; set it to 'failure' otherwise
Abort Action	Default implementation (set action outcome to 'success')

Table 9.13: Specification of HkRepStructDiagRep Component

Name	HkRepStructDiagRep(3,12)
Description	Report carrying the definition of a diagnostic report structure generated in response to a (3,11) command.
Parameters	SID of the diagnostic report, flag indicating whether periodic generation of the report is enabled, number of simply commutated parameters in the report and their identifiers, number of super-commutated groups and, for each group, number of parameters in the group and their identifiers
Discriminant	Structure Identifier
Destination	The destination is set equal to the source of the (3,11) command which triggers the report.
Enable Check	The enable status is read from the isEnabled field of the Report Definition corresponding to the report's SID
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Load the SID definition from the RDL

Table 9.14: Specification of HkRep Component

Name	HkRep(3,25)
Description	Periodic housekeeping report
Parameters	The values of the data items associated to the report's SID in the RDL
Discriminant	Structure Identifier
Destination	For pre-defined housekeeping reports, the initial destination is pre-defined. For all housekeeping reports, the destination is the source of the last (3,5) or (3,7) report enable command.
Enable Check	The enable status is read from the isEnabled field of the Report Definition corresponding to the report's SID
Ready Check	Run the procedure of figure 9.5
Repeat Check	Default implementation (report is not repeated)
Update Action	Load the value of the simply-commutated data items from the data pool and that of the super-commutated data items from the Sampling Buffer associated to the report's SID according to the Report Definition

Table 9.15: Specification of HkDiagRep Component

Name	HkDiagRep(3,26)
Description	Periodic Diagnostic Report (3,26)
Parameters	The values of the data items associated to the report's SID in the RDL
Discriminant	Structure Identifier
Destination	For pre-defined diagnostic reports, the initial destination is pre-defined. For all diagnostic reports, the destination is the source of the last (3,5) or (3,7) report enable command.
Enable Check	The enable status is read from the isEnabled field of the Report Definition corresponding to the report's SID
Ready Check	Run the procedure of figure 9.5
Repeat Check	Default implementation (report is not repeated)
Update Action	Load the value of the simply-commutated data items from the data pool and that of the super-commutated data items from the Sampling Buffer associated to the report's SID according to the Report Definition

Table 9.16: Specification of HkOneShotHkCmd Component

Name	HkOneShotHkCmd(3,27)
Description	Command (3,27) to generate a one-shot housekeeping report
Parameters	The list of SIDs for which the one-shot report is to be generated
Discriminant	SID to be generated in one-shot mode
Ready Check	Return 'command is ready'
Start Action	Run the procedure Start Action of Multi-SID Command of figure 9.3
Progress Action	Run the procedure Progress Action of Generate One-Shot Housekeeping Report of figure 9.6
Termination Action	Set action outcome to 'success' if all valid SIDs in the command were successfully processed by the progress action; set it to 'failure' otherwise
Abort Action	Do nothing

9.5 Service 3 Constants

The service 3 constants are listed in table 9.17.

Table 9.17: Constants for Housekeeping Service

Name	Description
`HK_N_REP_DEF`	Number of Report Definitions in the Report Definition List (maximum number of housekeeping/diagnostic reports)
`HK_MAX_SID`	Maximum value of a service 3 Structure Identifier (SID)
`HK_MAX_N_ITEMS`	Maximum number of data items in a housekeeping/diagnostic report
`HK_COLLECT_PER`	Minimum collection period for service 3 reports
`HK_MAX_N_SIMPLE`	Maximum number of simply-commutated parameters in a housekeeping or diagnostic report
`HK_MAX_N_GR`	Maximum number of super-commutated groups in a housekeeping/diagnostic report
`HK_MAX_REP`	Maximum value of the repetition number of a super-commutated group in a housekeeping/diagnostic report
`HK_MAX_N_REP`	Maximum number of data items in a super-commutated groups in a housekeeping/diagnostic report
`HK_MAX_ID`	Maximum value of a data pool item identifier
`HK_N_SAMP_BUF`	Number of service 3 Sampling Buffers
`HK_N_DEBUG_VAR`	Number of debug variables

9.6 Service 3 Observables and Parameters

The service 3 internal state is defined by the content of the Report Definition List (RDL). Most of its content is visible through reports (3,10) and (3,11). The observables defined by the framework only cover the non-visible part of the RDL state.

Similarly, the service 3 configuration is defined by the content of the Report Definition

List (RDL). This configuration is mostly controlled through commands (3,1)/(3,2) and (3,5)/(3,7) and is partially observable through reports (3,10) and (3,11). The service 3 configuration parameters which are either not controllable through service 3 commands and/or not observable through service 3 reports are defined as data pool parameters.

The observables and the parameters for service 3 are listed in table 9.18.

Table 9.18: Observables and Parameters for Housekeeping Service

Name	Kind	Description	Multiplicity
`debugVarAddr`	par	Address of Debug Variables	HK_N_DEBUG_VAR
`dest`	par	Destination of report definitions in the RDL	HK_N_REP_DEF
`isEnabled`	par	Enable status of report definitions in the RDL	HK_N_REP_DEF
`period`	par	Periods of report definitions in the RDL	HK_N_REP_DEF
`sid`	par	SIDs of report definitions in the RDL	HK_N_REP_DEF
`cycleCnt`	var	Cycle Counter for Reports in RDL	HK_N_REP_DEF
`debugVar`	var	Value of Debug Variables	HK_N_DEBUG_VAR
`sampleBufId`	var	Identifiers of Sampling Buffers	HK_N_REP_DEF

9.7 Service 3 Requirements

The table in this section lists requirements for the test service.

Table 9.19: Requirements for Service 3 (Housekeeping Service)

Req. ID	**Requirement Text**
P-S3-1/S	*The PUS Extension of the CORDET Framework shall implement a Report Definition List (RDL) consisting of HK_N_REP_DEF Report Definitions with the fields defined in table 9.1*
P-S3-2/S	*The PUS Extension of the CORDET Framework shall implement HK_N_SAMPLE_BUF Sampling Buffers capable of holding the values of the super-commutated data items for a given housekeeping/diagnostic report*
P-S3-3/S	*The PUS Extension of the CORDET Framework shall provide a Sampling Buffer Configuration Function to configure a sampling buffer for a given report*
P-S3-4/S	*The PUS Extension of the CORDET Framework shall provide a Sampling Buffer Setter Function to load a data item value in a sampling buffer*
P-S3-5/S	*The PUS Extension of the CORDET Framework shall provide a Sampling Buffer Getter Function to retrieve a data item value from a sampling buffer*
P-S3-6/C	*Application shall be responsible for loading a sampling buffer with the values of super-commutated data items*
P-S3-7/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, a hkCreateCmd component to encapsulate a (3,1) or (3,2) command*
P-S3-8/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, a hkDeleteCmd component to encapsulate a (3,3) or (3,4) command*

Req. ID	**Requirement Text**
P-S3-9/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, a hkEnableCmd component to encapsulate a (3,5) or (3,7) command*
P-S3-10/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, a hkDisableCmd component to encapsulate a (3,6) or (3,8) command*
P-S3-11/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, a hkRepStructCmd component to encapsulate a (3,9) or (3,11) command*
P-S3-12/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, a hkRepStructRep component to encapsulate a (3,10) or (3,12) report*
P-S3-13/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, a hkRep component to encapsulate a periodic (3,25) or (3,26) report*
P-S3-14/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, a hkRepOneShot component to encapsulate a one-shot (3,25) or (3,26) report*
P-S3-15/S	*The hkCreateCmd component shall close the InCommand adaptation points as indicated in table* ??
P-S3-16/S	*The hkDeleteCmd component shall close the InCommand adaptation points as indicated in table* ??
P-S3-17/S	*The hkEnableCmd component shall close the InCommand adaptation points as indicated in table* ??
P-S3-18/S	*The hkDisableCmd component shall close the InCommand adaptation points as indicated in table* ??
P-S3-19/S	*The hkRepStructCmd component shall close the InCommand adaptation points as indicated in table* ??
P-S3-20/S	*The hkRepStructRep component shall close the OutComponent adaptation points as indicated in table* ??
P-S3-21/S	*The hkRep component shall close the OutComponent adaptation points as indicated in table* ??
P-S3-22/S	*The hkRepOneShot component shall close the OutComponent adaptation points as indicated in table* ??
P-S3-23/S	*The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 9.18*
P-S3-24/S	*The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the parameters listed in table* ??

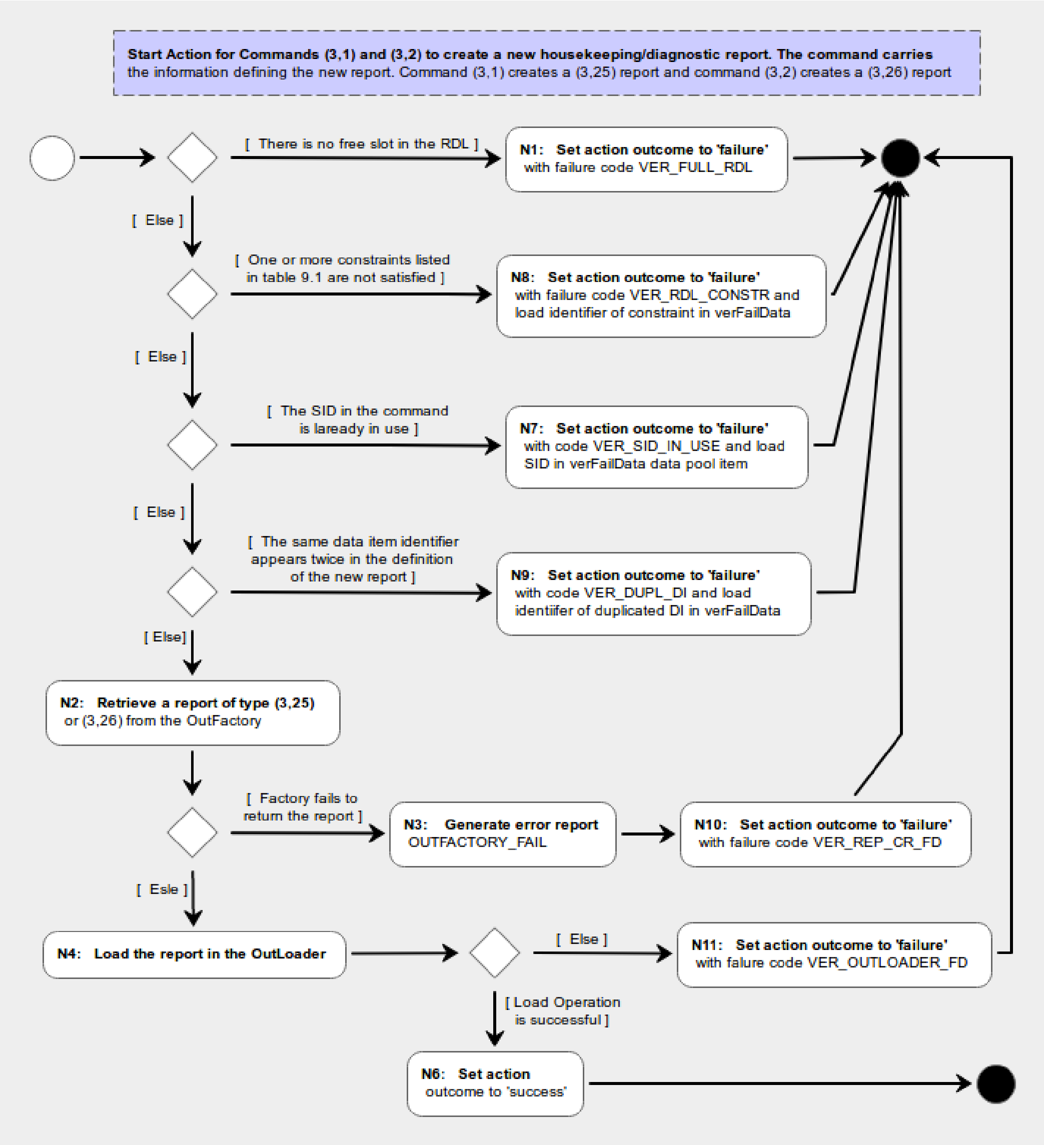

Start Action for Commands (3,1) and (3,2) to create a new housekeeping/diagnostic report. The command carries the information defining the new report. Command (3,1) creates a (3,25) report and command (3,2) creates a (3,26) report
[There is no free slot in the RDL]
N1: Set action outcome to 'failure' with failure code VER_FULL_RDL
[Else]
[One or more constraints listed in table 9.1 are not satisfied]
N8: Set action outcome to 'failure' with failure code VER_RDL_CONSTR and load identifier of constraint in verFailData
[Else]
[The SID in the command is laready in use]
N7: Set action outcome to 'failure' with code VER_SID_IN_USE and load SID in verFailData data pool item
[Else]
[The same data item identifier appears twice in the definition of the new report]
N9: Set action outcome to 'failure' with code VER_DUPL_DI and load identiifer of duplicated DI in verFailData
[Else]
N2: Retrieve a report of type (3,25) or (3,26) from the OutFactory
[Factory fails to retum the report]
N3: Generate error report OUTFACTORY_FAIL
N10: Set action outcome to 'failure' with failure code VER_REP_CR_FD
[Esle]
N4: Load the report in the OutLoader
[Else]
N11: Set action outcome to 'failure' with failure code VER_OUTLOADER_FD
[Load Operation is successful]
N6: Set action outcome to 'success'

Fig. 9.1: Start Action of Command to Create a Service 3 Packet

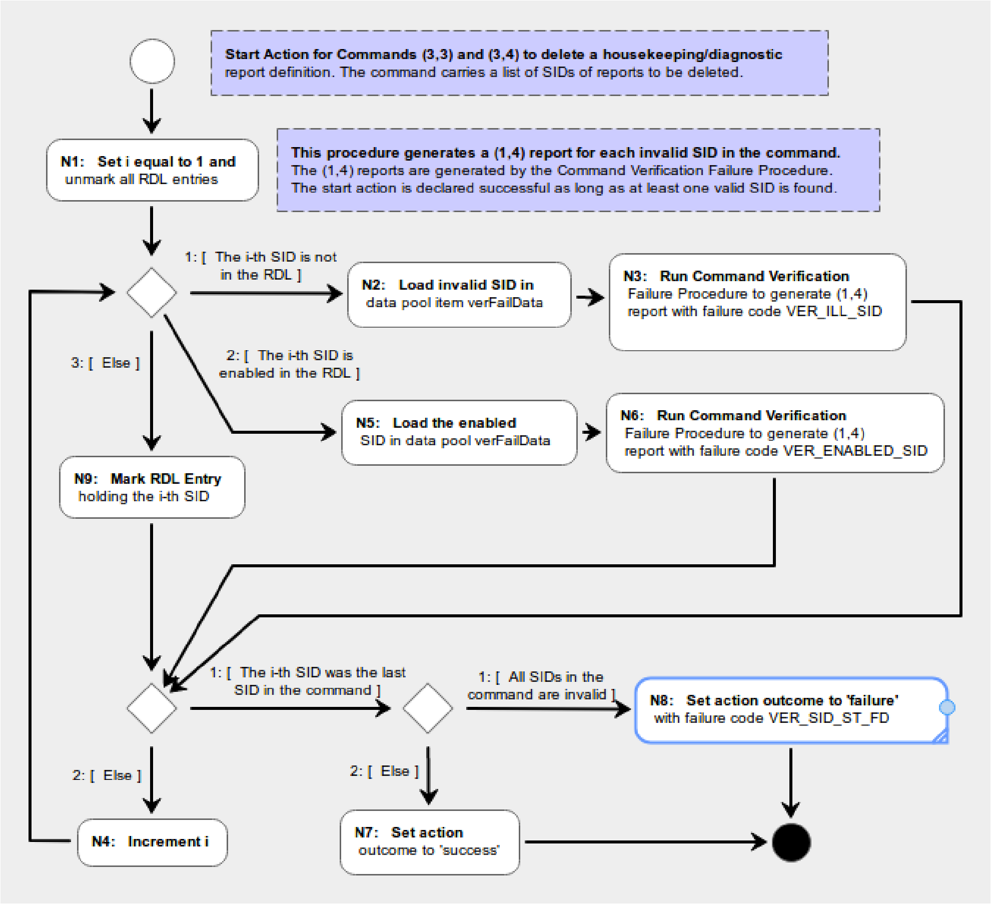
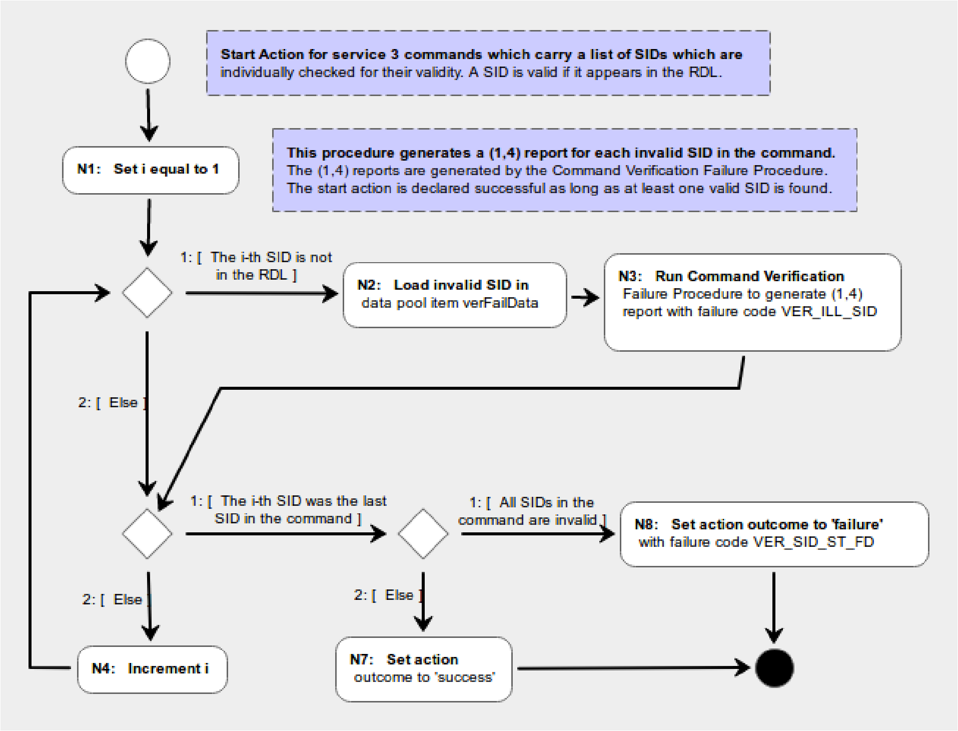

Start Action for Commands (3,3) and (3,4) to delete a housekeeping/diagnostic report definition. The command carries a list of SIDs of reports to be deleted.
This procedure generates a (1,4) report for each invalid SID in the command. The (1,4) reports are generated by the Command Verification Failure Procedure. The start action is declared successful as long as at least one valid SID is found.
N1: Set i equal to 1 and unmark all RDL entries
1: [The i-th SID is not in the RDL]
N2: Load invalid SID in data pool item verFailData
N3: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_SID
3: [Else]
2: [The i-th SID is enabled in the RDL]
N5: Load the enabled SID in data pool verFailData
N6: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ENABLED_SID
N9: Mark RDL Entry holding the i-th SID
1: [The i-th SID was the last SID in the command]
1: [All SIDs in the command are invalid]
N8: Set action outcome to 'failure' with failure code VER_SID_ST_FD
2: [Else]
2: [Else]
N4: Increment i
N7: Set action outcome to 'success'

Fig. 9.2: Start Action of Command to Delete a Service 3 Packet



Start Action for service 3 commands which carry a list of SIDs which are individually checked for their validity. A SID is valid if it appears in the RDL.

This procedure generates a (1,4) report for each invalid SID in the command. The (1,4) reports are generated by the Command Verification Failure Procedure. The start action is declared successful as long as at least one valid SID is found.

N1: Set i equal to 1
1: [The i-th SID is not in the RDL]
N2: Load invalid SID in data pool item verFailData
N3: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_SID
2: [Else]
1: [The i-th SID was the last SID in the command]
1: [All SIDs in the command are invalid]
N8: Set action outcome to 'failure' with failure code VER_SID_ST_FD
2: [Else]
2: [Else]
N4: Increment i
N7: Set action outcome to 'success'

Fig. 9.3: Start Action of Multi-SID Commands

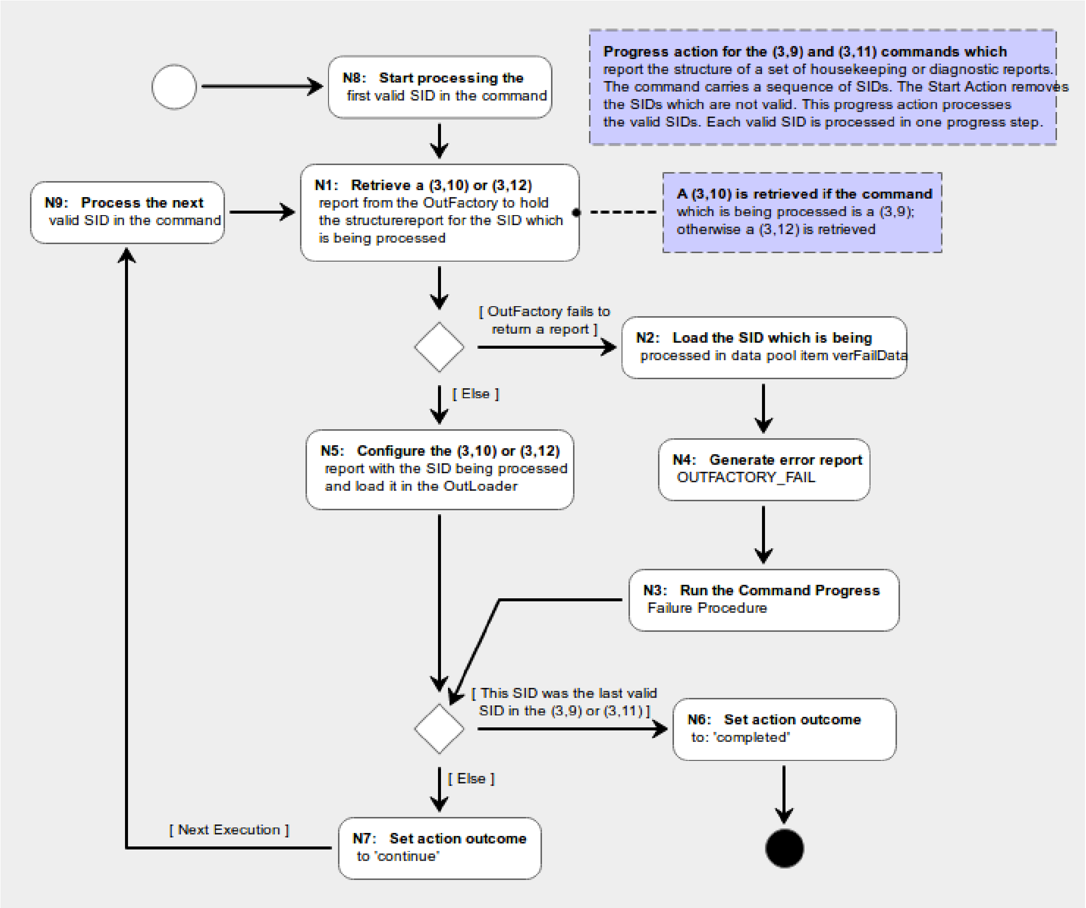

Progress action for the (3,9) and (3,11) commands which report the structure of a set of housekeeping or diagnostic reports. The command carries a sequence of SIDs. The Start Action removes the SIDs which are not valid. This progress action processes the valid SIDs. Each valid SID is processed in one progress step.
N8: Start processing the first valid SID in the command
N1: Retrieve a (3,10) or (3,12) report from the OutFactory to hold the structurereport for the SID which is being processed
A (3,10) is retrieved if the command which is being processed is a (3,9); otherwise a (3,12) is retrieved
N9: Process the next valid SID in the command
[OutFactory fails to return a report]
N2: Load the SID which is being processed in data pool item verFailData
[Else]
N5: Configure the (3,10) or (3,12) report with the SID being processed and load it in the OutLoader
N4: Generate error report OUTFACTORY_FAIL
N3: Run the Command Progress Failure Procedure
[This SID was the last valid SID in the (3,9) or (3,11)]
N6: Set action outcome to: 'completed'
[Else]
[Next Execution]
N7: Set action outcome to 'continue'

Fig. 9.4: Progress Action of Command to Report the Structure of a Service 3 Packet

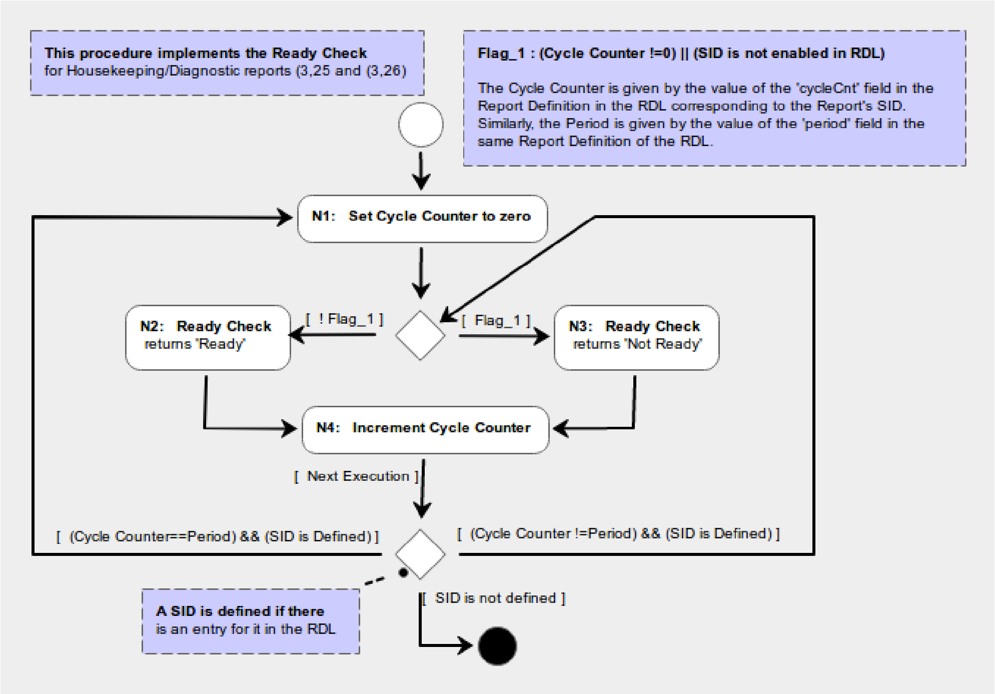

This procedure implements the Ready Check
for Housekeeping/Diagnostic reports (3,25 and (3,26)
Flag_1 : (Cycle Counter !=0) || (SID is not enabled in RDL)
The Cycle Counter is given by the value of the 'cycleCnt' field in the Report Definition in the RDL corresponding to the Report's SID. Similarly, the Period is given by the value of the 'period' field in the same Report Definition of the RDL.
N1: Set Cycle Counter to zero
[! Flag_1]
[Flag_1]
N2: Ready Check returns 'Ready'
N3: Ready Check returns 'Not Ready'
N4: Increment Cycle Counter
[Next Execution]
[(Cycle Counter==Period) && (SID is Defined)]
[(Cycle Counter !=Period) && (SID is Defined)]
[SID is not defined]
A SID is defined if there is an entry for it in the RDL

Fig. 9.5: Ready Check of Report HkRep

10 Event Reporting Service

The service type of the Event Reporting Service is 5. The PUS Extension of the CORDET Framework supports this service in full.

The event reporting service provides the capability to report event-like occurrences and to control the generation of event reports by enabling and disabling individual event identifiers.

The PUS recognizes four levels of event reports and associates to each level a service sub-type. Thus, for instance, all event reports of level 1 are carried by reports of type (5,1) and all event reports of level 2 are carried by event reports of type (5,2).

All event reports have the same behaviour irrespective of their level. The PUS Extension of the CORDET Framework one component for each event reporting level.

Event reports may carry data. The Event Identifier (EID) determines the format of the data associated to an event report. The PUS Extension accordingly treats the event identifier as a discriminant. The range of discriminants and the data associated to each discriminant are adaptation points which must be defined at application level. Some event identifiers are pre-defined at framework level. They are listed in appendix A.

Applications use event reports as follows:

- When an application encounters a situation which should trigger the generation of an event report, it retrieves an event component from the OutFactory. This component will encapsulate the event report. The event level and the event identifier are specified when the event component is retrieved from the factory because the ’make’ function of the OutFactory takes as an argument the type and the sub-type of the event report.
- The application configures the component with the destination and with any data which are attached to the event
- The application loads the event component in the OutLoader. From this point onward, the event report is processed by the CORDET Framework infrastructure:
  - If the identifier of the event is enabled, then the event report will eventually be sent to its destination;
  - If the identifier of the event is not enabled, then the event report will be discarded.

Note that the configuration of the event reports must be done by the application (as opposed to being delegated to the Update Action of the component which implements the event report). This ensures that the event report configuration reflects the state of the system at the time the event report is created (as opposed to the time when the event report is sent out).

Event identifiers can be enabled and disabled. The PUS Extension uses the report enable mechanism of the OutRegistry component to manage the enable status of event reports. This implies that, by default, all event identifiers are enabled. If an application needs some event identifiers to be disabled by default, it must disable them during the application initialization phase (adaptation point P-S5-3).

In accordance with clause 6.5.4b of reference [PS-SP], all event reports have the same destination and this destination must be statically defined. The destination is accordingly treated as a configuration parameter for the PUS Extension of the CORDET Framework.

10.1 Service 5 Report and Command Definition

The tables in this section formally specify the service 5 commands and reports by specifying how the actions, checks and attributes of generic out-going commands and reports are specialized for service 5 (see section 7). The following remarks apply:

- The update of observables which are related to the occurrence of an event is done by the Enable Check of component EvtRep. This is appropriate because this action is executed every time an application creates an event report. The update of observables which are related to the generation of report is done by the Update Action of component EvtRep. This is appropriate because this action is only executed when an event report is actually issued by an application.
- Service 5 reports are generated as soon as the condition which triggered them occur and hence their ready check always returns 'ready'
- Service 5 reports are 'one-off' reports and hence their repeat check always returns 'no repeat'

Table 10.1: Specification of EvtRep1 Component

Name	EvtRep1(5,1)
Description	Informative event report
Parameters	Event Identifier (EID) acting as discriminant followed by event-specific parameters
Discriminant	Event Identifier
Destination	The destination is set by the application when it creates the event report. For event reports generated by the PUS Extension, it is set to EVT_DEST.
Enable Check	Update service 5 observable nOfDetectedEvt x ('x' is the event severity level) and then retrieve the enable status from the Out-Registry as a function of the report type, sub-type and discriminant
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Update service 5 observables: nOfGenEvtRep x, lastEvtEid i, lastEvtTime x ('x' is the event severity level). Note that the event parameters are set by the application which creates the event report at the time it creates it.

Table 10.2: Specification of EvtRep2 Component

Name	EvtRep2(5,2)
Description	Low severity event report
Parameters	Event Identifier (EID) acting as discriminant followed by event-specific parameters
Discriminant	Event Identifier
Destination	The destination is set by the application when it creates the event report. For event reports generated by the PUS Extension, it is set to EVT_DEST.
Enable Check	Update service 5 observable nOfDetectedEvt x ('x' is the event severity level) and then retrieve the enable status from the Out-Registry as a function of the report type, sub-type and discriminant
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Update service 5 observables: nOfGenEvtRep x, lastEvtEid i, lastEvtTime x ('x' is the event severity level). Note that the event parameters are set by the application which creates the event report at the time it creates it.

Table 10.3: Specification of EvtRep3 Component

Name	EvtRep3(5,3)
Description	Medium severity event report
Parameters	Event Identifier (EID) acting as discriminant followed by event-specific parameters
Discriminant	Event Identifier
Destination	The destination is set by the application when it creates the event report. For event reports generated by the PUS Extension, it is set to EVT_DEST.
Enable Check	Update service 5 observable nOfDetectedEvt x ('x' is the event severity level) and then retrieve the enable status from the OutRegistry as a function of the report type, sub-type and discriminant
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Update service 5 observables: nOfGenEvtRep x, lastEvtEid i, lastEvtTime x ('x' is the event severity level). Note that the event parameters are set by the application which creates the event report at the time it creates it.

Table 10.4: Specification of EvtRep4 Component

Name	EvtRep4(5,4)
Description	High severity event report
Parameters	Event Identifier (EID) acting as discriminant followed by event-specific parameters
Discriminant	Event Identifier
Destination	The destination is set by the application when it creates the event report. For event reports generated by the PUS Extension, it is set to EVT_DEST.
Enable Check	Update service 5 observable nOfDetectedEvt x ('x' is the event severity level) and then retrieve the enable status from the OutRegistry as a function of the report type, sub-type and discriminant
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Update service 5 observables: nOfGenEvtRep x, lastEvtEid i, lastEvtTime x ('x' is the event severity level). Note that the event parameters are set by the application which creates the event report at the time it creates it.

Table 10.5: Specification of EvtEnbCmd Component

Name	EvtEnbCmd(5,5)
Description	Command to enable generation of a list of event identifiers
Parameters	List of event identifiers to be enabled
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Default implementation (set action outcome to 'success')
Progress Action	The event identifiers are processed in sequence and in the order in which they are stored in the event report. Each event identifier is processed in an execution cycle. Each execution cycle counts as a progress step. At each execution, the progress action performs the following actions: (a) If the event identifier is illegal, then: the illegal EID is loaded into verFailData and the Success Outcome is set to VER_ILL_EID (b) If the event identified is legal, then: its enable status is set to 'enabled'; the value of nDisabledEid x ('x' is the severity level of the EID) is decremented; and the Success Outcome of the action is set to 'success' (c) The Completion Outcome of the action is set to 'completed' if all event identifiers carried by the command have been processed; otherwise it is set to 'not completed'. The enable status of the event identifier is stored in the OutRegistry component of the Cordet Framework.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 10.6: Specification of EvtDisCmd Component

Name	EvtDisCmd(5,6)
Description	Command to disable generation of a list of event identifiers
Parameters	List of event identifiers to be disabled
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of Multi-EID Command of figure 10.1

Name	EvtDisCmd(5,6)
Progress Action	The event identifiers are processed in sequence and in the order in which they are stored in the event report. Each event identiifier is processed in an execution cycle. Each execution cycle counts as a progress step. At each execution, the progress action performs the following actions: (a) If the event identifier is illegal, then: the illegal EID is loaded into verFailData and the Success Outcome is set to VER_ILL_EID (b) If the event identified is legal, then: its enable status is set to 'disabled'; the value of nDisabledEid x ('x' is the severity level of the EID) is incremented; and the Success Outcome of the action is set to 'success' (c) The Completion Outcome of the action is set to 'completed' if all event identifiers carried by the command have been processed; otherwise it is set to 'not completed'. The enable status of the event identifier is stored in the OutRegistry component of the Cordet Framework.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 10.7: Specification of EvtRepDisCmd Component

Name	EvtRepDisCmd(5,7)
Description	This command triggers the generation of a (5,8) report holding the list of disabled event identifiers
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	(a) Compute the number N of (5,8) reports required to hold all the disabled event identifiers. (b) Retrieve N reports of type (5,8) from the OutFactory (c) Set the action outcome to 'success' if all retrievals are successful; otherwise, generate error report OUTFACTORY FAILED and set the action outcome to VER_REP_CR_FD. NB: The maximum number of (5,8) reports required in response to a single (5,7) commands is given by framework configuration parameter EVT_MAX_N5S8. If N is greater than EVT_MAX_N58, the behaviour of the framework is undefined.
Progress Action	Configure the (5,8) reports with a destination equal to the source of the (5,7) command, load them into the OutLoader, and set the action outcome to 'success'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 10.8: Specification of EvtDisRep Component

Name	EvtDisRep(5,8)
Description	Report generated in response to a (5,7) command carrying the list of disabled Event Identifiers
Parameters	The list of disabled event identifiers
Discriminant	None
Destination	The destination is set equal to the source of the (5,7) command which triggers the report
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Load the list of disabled event identifiers. First, the event identifiers of severity level 1 are loaded in order of increasing identifier. Then, the event identifiers of severity level 2 are loaded in order of increasing identifier. And so on for severity levels 3 and 4. If one single (5,8) report is not sufficient to hold all disabled event identifiers, then the event identifiers are loaded in successive (5,8) reports which are triggered by the same (5,7) command.

10.2 Service 5 Constants

The service 5 constants are listed in table 10.9.

Table 10.9: Constants for Event Reporting Service

Name	Description
EVT_N_ID	Number of event identifiers supported by the application

10.3 Service 5 Observables

The service 5 observables consist of counters and flags which are updated by the service commands. They are listed in table 10.10.

Table 10.10: Observables for Event Reporting Service

Name	Description
isEidEnabled	Enable status of the event identifiers
lastEvtEid_1	Event identifier of the last generated level 1 event report
lastEvtEid_2	Event identifier of the last generated level 2 event report
lastEvtEid_3	Event identifier of the last generated level 3 event report
lastEvtEid_4	Event identifier of the last generated level 4event report
lastEvtTime_1	Time when the last level 1 event report was generated
lastEvtTime_2	Time when the last level 2 event report was generated
lastEvtTime_3	Time when the last level 3 event report was generated
lastEvtTime_4	Time when the last level 4 event report was generated
nOfDetectedEvts_1	Number of detected occurrences of level 1 events
nOfDetectedEvts_2	Number of detected occurrences of level 2 events
nOfDetectedEvts_3	Number of detected occurrences of level 3 events
nOfDetectedEvts_4	Number of detected occurrences of level 4 events
nOfDisabledEid_1	Number of event identifiers of level 1 which are disabled
nOfDisabledEid_2	Number of event identifiers of level 2 which are disabled
nOfDisabledEid_3	Number of event identifiers of level 3 which are disabled
nOfDisabledEid_4	Number of event identifiers of level 4 which are disabled
nOfGenEvtRep_1	Number of generated level 1 event reports
nOfGenEvtRep_2	Number of generated level 2 event reports
nOfGenEvtRep_3	Number of generated level 3 event reports
nOfGenEvtRep_4	Number of generated level 4 event reports

10.4 Service 5 Requirements

The table in this section lists requirements for the event reporting service.

Table 10.11: Requirements for Service 5 (Event Reporting Service)

Req. ID	Requirement Text
P-S5-1/S	*The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 13.11*
P-S5-2/C	*Applications shall be responsible for configuring the EvtRep component with the event parameters at the point where the EvtRep component is created*
P-S5-3/C	*Applications shall be responsible for configuring the OutRegistry component to selectively disable event reports whose default enable status is: 'disabled'*
P-S5-4/S	*For each event report it pre-defines, the PUS Extension of the CORDET Framework shall provide a Generate Pre-Defined Event Function which takes as parameters the event type, subtype, discriminant and the event parameters*
P-S5-5/S	*The Generate Pre-Defined Event Function shall: retrieve an OutComponent to encapsulate the event report from the OutFactory, configure it with its parameters and load it in the OutLoader*
P-S5-6/S	*If the OutComponent retrieval from the OutFactory fails, the Generate Pre-Defined Event Function shall generate an error report of type OUTFACTORY_FAIL*
P-S5-7/S	*The PUS Extension of the CORDET Framework shall support the commands and reports specified in tables* **??** *to* **??**

11 Time-Based Scheduling Service

The service type of the Time-Based Scheduling Service is 11. The PUS Extension of the CORDET Framework supports this service only in part: the management of sub-schedules and groups and the associated commands and reports are not yet specified.

11.1 Time-Based Schedule (TBS)

The Time-Based Scheduling Service controls the Time-Based Schedule Execution Function. This function allows time-tagged requests to be pre-loaded in an application and to be released when their time of execution comes due. The pre-loaded requests are held in the *Time-Based Schedule* or TBS. The TBS consists of a list of up to SCD_N_TBA *Time-Based Activities* or TBAs. Within the TBS, each TBA is identified by an integer in the range 1 to SCD_N_TBA. Each TBA is defined by the attributes listed in table 11.1.

The entries in the TBS are arranged in a random order which is determined by the order in which the TBAs are loaded in the application. The TBS is implemented as a linked list where each item knows its "successor" (attribute `nextTba`) and its "predecessor" (attribute `prevTba`). If B is the successor TBA of A, then this means that the release time of B is later than the release time of A and that there is no other entry in the TBS with a release time between A and B. A similarl definition applies to the predecessor TBA. These attributes allow the TBAs to be navigated in the order in which they are due for release. Additionally, and also in order to facilitate the scanning of the TBAs according to the their order of release time, variable `firstTba` is maintained. This variable points to the TBA with the earliest release time (i.e. to the first TBA which is due for execution).

Initially, the TBS is empty. A slot in the TBS is empty if its release time is equal to zero. The TBS is filled up through (11,4) commands. Each such command loads a TBA in the TBS. The TBS can be reset (all its entries are deleted) with command (11,3). Individual TBAs can be deleted using command (11,4) and deletion by filtering criterium can be done with command (11,5).

The (11,4) command to load a new TBA in the TBS carries one or more instructions and each instruction carries one activity request. An activity request consists of a command which is embedded in its usual packet format within the (11,4) instruction. When the (11,4) command is processed, the commands embedded within its instructions are extracted from it and are then processed as follows:

- If the destination of the embedded command is not the host application, or if the group to which the command is assigned does not exist or is full, or if its release time is smaller than the current time (plus the time margin), then the instruction containing the embedded command is rejected and a (1,4) report will be generated for it.
- If the destination of the embedded command is the host application, then an attempt is made to create an InCommand component encapsulating it (the InCommand is requested from the InFactory).
- If the attempt to create the InCommand fails (either because its type or sub-type or discriminant are illegal), then the instruction containing the embedded command is rejected and a (1,4) report will be generated for it.
- If the attempt to create the InCommand succeeds, then the InCommand is attached to the TBA and the TBA is stored in the TBS.

- When the release time of the TBA becomes due (the release time is one of the parameters of the (11,4) instruction), it is checked whether the InCommand is in state CONFIGURED. If this is not the case, then the command is deemed to have failed its acceptance check and it is rejected with a (1,2) report.
- If the InCommand is in state CONFIGURED, it is loaded in an InManager which will then process it like an ordinary (non-scheduled) command. The selection of the InManager is an adaptation point of the PUS Extension of the CORDET Framework.

Note that the processing logic outlined above is the same as is applied by the InLoader to non-scheduled incoming commands. In other words: non-scheduled incoming commands and scheduled commands are processed according to the same logic but, for non-scheduled commands, this logic is implemented in the InLoader component; for scheduled commands, it is instead implemented in the time-based schedule execution function. More specifically, this is implemented in the following components:

- The checks on the legality of the time-scheduled command are done in the Start Action of the (11,4) command (see table **??** and in particular figure 11.1)
- The insertion of the TBA in the TBS is done by the Progress Action of the (11,4) command (see table **??**)
- The extraction of the TBA from the TBS and the loading of its embedded command in an InManager is done by the Time-Based Schedule Execution Procedure of figure 11.7

Command (11,5) is the antagonist of command (11,4): it can be used to delete one or more TBAs from the TBS. A TBA to be deleted are identified by the "request identifier", namely a triplet holding the source identifier, the APID and the source sequence count of the command embedded in the TBA,

The time-based schedule execution function can be enabled and disabled through commands (11,1) and (11,2). Initially, by default, the function is disabled. When the function is disabled, none of the TBAs in the TBS are processed.

11.2 Sub-Schedules

Within the TBS, up to SCD_N_SUB_TBS *sub-schedules* may be defined. Each sub-schedule is identified by an integer in the range 1 to SCD_N_SUB_TBS and, to each TBA, the attribute `subSchedId` is associated which identifies the sub-schedule to which the TBA belongs (i.e. all TBAs belong to one and exactly one sub-schedule).

A sub-schedule has two attributes: flag `isSubSchedEnabled` which determines whether the sub-schedule is enabled and integer `nOfTbaInSubSched` which holds the number of TBAs in the sub-schedule. In [PS-SP], sub-schedules are created and deleted dynamically. Sub-schedule S is created when a command (11,4) asks for one or more TBAs to be inserted in the sub-schedule S. The sub-schedule is deleted when the last of its TBAs is deleted from the TBS (because its release time has been reached). In the PUS Extension, the data structures for the sub-schedules are all statically defined and therefore the creation status of a sub-schedule is determined by the value of `nOfTbaInSubSched`: the sub-schedule is deemed to be created when the attribute is greater than zero (the sub-schedule is not empty).

Commands (11,20) and (11,21) can be used to, respectively, enable and disable one or more time-based sub-schedules.

The sub-schedule 1 is the default sub-schedule. An application which does not need sub-schedules should set SCD_N_SUB_TBS to 1. Since sub-schedules are disabled by default, an application which does not support sub-schedules should also enable the default sub-schedule as part of its initialization.

11.3 Groups

The TBAs in the TBS may be assigned to groups. Up to SCD_N_GROUP groups may be defined. Each group is identified by an integer in the range 1 to SCD_N_GROUP and, to each TBA, the attribute groupId is associated which identifies the group to which the TBA belongs (i.e. all TBAs belong to one and exactly one group).

A group has three attributes: flag isGroupEnabled which determines whether the group is enabled, flag isGroupInUse which determines whether the group is in use, and integer nOfTbaInGroup which holds the number of TBAs in the group. In [PS-SP], groups are created and deleted dynamically. Group G is created through command (11,22) and deleted through command (11,23). In the PUS Extension, the data structures for the sub-schedules are all statically defined and therefore the creation status of a sub-schedule is determined by the value of the isGroupInUse flag which is set/unset in response to the (11,22) and (11,23) commands. Similarly, commands (11,24) and (11,25) toggle the enabled status of a group which is currently in use.

The status of the groups which are currently in use is reported by report (11,27) which is triggered by command (11,26). The configuration of service 11 must be such that the status of all groups can fit within one single (11,27) report.

The group 1 is the default group. An application which does not need groups should set SCD_N_GROUPS to 1 and should set its InUse and Enabled flags to true as part of the application initialization.

Table 11.1: Attributes of Time-Based Activity

Name	Description	Constraint
relTime	The release time of the time-based activity or zero if this slot in the TBS is empty	A valid on-board time value
nextTba	The identifier within the TBS of the time-based activity with the next release time or zero if this is the last (fartherst in the future) entry in the TBS	Integer in range 0..SCD_N_TBA
prevTba	The identifier within the TBS of the time-based activity with the previous release time or zero if this is the first (earliest) entry in the TBS	Integer in range 0..SCD_N_TBA
subSchedId	The identifier of the sub-schedule to which the TBA belongs	Integer in range 1..SCD_N_SUB_-TBS
groupId	The identifier of the group to which the TBA belongs	Integer in range 1..SCD_N_GROUP
cmd	The InCommand holding the command associated to the TBA	A valid InCommand pointer

11.4 Service 11 Report and Command Definition

Tables **??** to **??** formally specify the service 11 commands and reports by specifying how the actions, checks and attributes of generic out-going commands and reports are specialized for this service (see section 7).

Table 11.2: Specification of ScdEnbTbsCmd Component

Name	ScdEnbTbsCmd(11,1)
Description	Command to enable the time-based schedule execution function
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Default implementation (set action outcome to 'success')
Progress Action	Set the enable status of the time-based schedule execution function to: enabled and start the Time-Based Schedule Execution Procedure of figure 11.7
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 11.3: Specification of ScdDisTbsCmd Component

Name	ScdDisTbsCmd(11,2)
Description	Command to disable the time-based schedule execution function
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Default implementation (set action outcome to 'success')
Progress Action	Set the enable status of the time-based schedule execution function to: disabled and stop the Time-Based Schedule Execution Procedure of figure 11.7
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 11.4: Specification of ScdResTbsCmd Component

Name	ScdResTbsCmd(11,3)
Description	Command to reset the time-based schedule
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Default implementation (set action outcome to 'success')
Progress Action	(a) Set the enabled status of the time-based schedule execution function to: disabled. (b) Clear all entries in the time-based schedule (TBS). An entry in the TBS is cleared by setting its release time attribute to zero. (c) Delete all sub-schedules and set the number of in use sub-schedules (nOfInUseSubSched) to zero. A sub-schedule is deleted by setting its inUse flag to false. (d) Delete all schedule groups and set the number of in use groups (nOfInUseGroup) to zero. A group is deleted by setting its inUse flag to false.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 11.5: Specification of ScdInsTbaCmd Component

Name	ScdInsTbaCmd(11,4)
Description	Command to insert one or more time-based activities (TBAs) into the time-based schedule (TBS)
Parameters	The sub-schedule to which the TBAs must be added and, for each TBA, the group to which the TBA belongs, its release time and the command which implements the activity
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of (11,4) Command of figure 11.1. This procedure first checks that the sub-schedule identifier has a legal value and then it checks the TBAs in the command. A TBA is rejected if its group identifier is illegal, or if its release time is smaller than the current time plus the time margin, or if the TBS is already full, or if there are insufficient resources to create an InCommand component to encapsulate the command embedded in the activity.
Progress Action	For all the activities in the command which have been accepted by the Start Action, the following is done: (a) The TBS is scanned and, when a free slot is found, the activity is loaded in the free slot (b) If the release time of the TBA pointed at by firstTba is larger than the release time of the new TBA, then the value of firstTba is updated to point to the newly inserted TBA (c) The nextTba and prevTba pointers of the newly inserted TBA and of its previous and next TBA are updated to keep the consistency of the TBS (d) The number of scheduled activities (nOfTba) is incremented by 1 (e) The number of activities in the sub-schedule (nOfTbaInSubSched) to which the newly inserted TBA belongs is incremented by 1 (f) The number of activities in the group (nOfTbaInGroup) to which the newly inserted TBA belongs is incremented by 1 (g) If the sub-schedule to which the newly inserted TBA belongs was empty, then the number of non-empty sub-schedules (nOfSubSched) is incremented by one After all actvities have been processed, the action outcome is set to: 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 11.6: Specification of ScdDelTbaCmd Component

Name	ScdDelTbaCmd(11,5)
Description	Command to delete one or more time-based activities (TBAs) from the time-based schedule (TBS)
Parameters	The number of activities to be deleted and the list of identifiers of the activities to be deleted. Each such identifier is made up of: the identifier of the source, the APID and the sequence count of the request embedded in the activity to be deleted.
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of (11,22) Command of figure 11.4. This procedure iterates over the list of group identifiers in the command and rejects those which are out-of-limit or which are already in use.
Progress Action	For each group identifier which has been accepted by the Start Action, the following is done: (a) The group identifier is marked as in use (its InUse flag is set to true) (b) The enable status of the group identifier is set in accordance with the enable status parameter in the command After all identifiers have been processed, the action outcome is set to: 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 11.7: Specification of ScdEnbSubSchedCmd Component

Name	ScdEnbSubSchedCmd(11,20)
Description	Command to enable one or more time-based sub-schedules
Parameters	The number of sub-schedules to be enabled followed by the list of identifiers of the sub-schedules to be enabled
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure of figure 11.3. This procedure checks all the sub-schedule identifiers in the command and rejects those which are invalid (i.e. either outside the range: 1..SCD_N_SUB_TBS or pointing at an empty sub-schuedule).
Progress Action	For all the sub-schedule identifiers which have passed the Start Check,set their isSubSchedEnabled attribute to false.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 11.8: Specification of ScdDisSubSchedCmd Component

Name	ScdDisSubSchedCmd(11,21)
Description	Command to disable one or more time-based sub-schedules
Parameters	The number of sub-schedules to be disabled followed by the list of identifiers of the sub-schedules to be disabled
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure of figure 11.3. This procedure checks all the sub-schedule identifiers in the command and rejects those which are invalid (i.e. either outside the range: 1..SCD_N_SUB_TBS or pointing at an empty sub-schuedule).
Progress Action	For all the sub-schedule identifiers which have passed the Start Check,set their isSubSchedEnabled attribute to false.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 11.9: Specification of ScdCreGrpCmd Component

Name	ScdCreGrpCmd(11,22)
Description	Command to create one or more scheduling groups
Parameters	The number of groups to be created and, for each group to be created, its identifier and its initial enable status
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of (11,22) Command of figure 11.4. This procedure iterates over the list of group identifiers in the command and rejects those which are out-of-limit or which are already in use.
Progress Action	For each group identifier which has been accepted by the Start Action, the following is done: (a) The group identifier is marked as in use (its InUse flag is set to true) (b) The enable status of the group identifier is set in accordance with the enable status parameter in the command After all identifiers have been processed, the action outcome is set to: 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 11.10: Specification of ScdDelGrpCmd Component

Name	ScdDelGrpCmd(11,23)
Description	Command to delete one or more scheduling groups
Parameters	The number of groups to be delete and the list of their identifiers
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of (11,23) Command of figure 11.5. This procedure iterates over the list of group identifiers (or, if the number of groups to be deleted is equal to zero, over all groups currently in use) and rejects those whose identifiers is out-of-limits or which have activities associated to them.
Progress Action	For all group identifiers accepted by the Start Action, the following is done: the group is deleted by setting its InUse flag to false. After all identifiers have been processed, the action outcome is set to: 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 11.11: Specification of ScdEnbGrpCmd Component

Name	ScdEnbGrpCmd(11,24)
Description	Command to enable one or more scheduling groups
Parameters	The number of groups to be enabled and the list of their identifiers
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of (11,24) and (11,25) Command of figure 11.6. This procedure iterates over the list of group identifiers in the command and rejects those which are not in use.
Progress Action	For all group identifiers accepted by the Start Action, the following is done: the isGroupEnabled flag of the group is set to 'Enabled'. After all identifiers have been processed, the action outcome is set to 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 11.12: Specification of ScdDisGrpCmd Component

Name	ScdDisGrpCmd(11,25)
Description	Command to disable one or more scheduling groups
Parameters	The number of groups to be disabled and the list of their identifiers
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of (11,24) and (11,25) Command of figure 11.6. This procedure iterates over the list of group identifiers in the command and rejects those which are not in use.
Progress Action	For all group identifiers accepted by the Start Action, the following is done: the isGroupEnabled flag of the group is set to 'Disabled'. After all identifiers have been processed, the action outcome is set to 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 11.13: Specification of ScdRepGrpCmd Component

Name	ScdRepGrpCmd(11,26)
Description	Command to trigger the generation of a (11,27) report carrying the status of the scheduling groups
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Retrieve a (11,27) report from the OutFactory. If the retrieval fails, generate error report OUTFACTORY_FAIL and set action outcome to 'failure'. Otherwise, set action outcome to 'success'.
Progress Action	Configure the (11,27) report created by the Start Action and load it in the OutLoader. Set the action outcome to 'success' if the load operation is successful and to 'failed' otherwise.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 11.14: Specification of ScdGrpRep Component

Name	ScdGrpRep(11,27)
Description	Report generated in response to a (11,26) command to report the status of the scheduling groups
Parameters	The number of currently used scheduling groups and, for each, the identifier and the enable status
Discriminant	None
Destination	TThe source of the (11,26) command which triggered the generation of the report
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Collect the information about the currently used scheduling groups

11.5 Service 11 Constants

The service 11 constants are listed in table 11.15.

Table 11.15: Constants for Time-Based Service

Name	Description
`SCD_N_TBA`	Number of time-based scheduled activities
`SCD_N_SUB_TBS`	Number of time-based sub-schedules
`SCD_N_GROUP`	Number of time-based schedule groups

11.6 Service 11 Observables and Parameters

The service 11 observables and parameters are listed in table 11.16.

The attributes of the Time-Based Activities (TBAs) in the Time-Based Schedule (TBS) are visible through service 11 reports and are therefore not defined as "observables" in the data pool.

Table 11.16: Observables and Parameters for Time-Based Scheduling Service

Name	Kind	Description	Multiplicity
`isSubSchedEnabled`	par	Enable status of a sub-schedule	SCD_N_SUB_TBS
`isTbsEnabled`	par	Enable status of time-based schedule	NULL
`nOfTbaInGroup`	par	Number of TBAs in group	SCD_N_GROUP
`nOfTbaInSubSched`	par	Number of TBAs in sub-schedule	SCD_N_SUB_TBS
`timeMargin`	par	Time margin for time-based scheduling service	NULL
`firstTba`	var	Identifier of next time-based activity due for release	NULL
`isGroupEnabled`	var	Enabled flag for time-based schedule group	SCD_N_GROUP
`isGroupInUse`	var	InUse flag for time-based schedule group	SCD_N_GROUP
`nOfGroup`	var	Number of non-empty groups	NULL
`nOfSubSched`	var	Number of non-empty sub-schedules	NULL
`nOfTba`	var	Number of currently defined time-based activities (TBAs)	NULL

11.7 Service 11 Requirements

The table in this section lists requirements for the time-based scheduling service.

Table 11.17: Requirements for Service 11 (Time-Based Scheduling Service)

Req. ID	**Requirement Text**
P-S11-1/S	*The PUS Extension of the CORDET Framework shall implement a Time-Based Schedule (TBS) consisting of SCD_N_TBA Time-Based Activities (TBAs) with the attributes defined in table 11.1*
P-S11-2/S	*The PUS Extension of the CORDET Framework shall provide a Time-Based Execution Procedure implementing the behaviour shown in figure 11.7*
P-S11-3/C	*Applications shall be responsible for periodically executing the Time-Based Execution Procedure of figure 11.7*
P-S11-4/C	*Applications which do not need sub-schedules shall: (a) set SCD_N_SUB_-TBS to 1 and (b) enable the first sub-schedule as part of their initialization*
P-S11-5/C	*Applications which do not need groups shall, as part of their initialization, (a) set SCD_N_GROUP to 1, and (b) set the InUse and Enabled flag of the first group to true*
P-S11-6/S	*The PUS Extension of the CORDET Framework shall support the commands and reports specified in tables* ?? *to* ??
P-S11-7/C	*Service 11 shall be configured such that a single (11,27) report can hold the status of SCD_N_GROUPS groups*

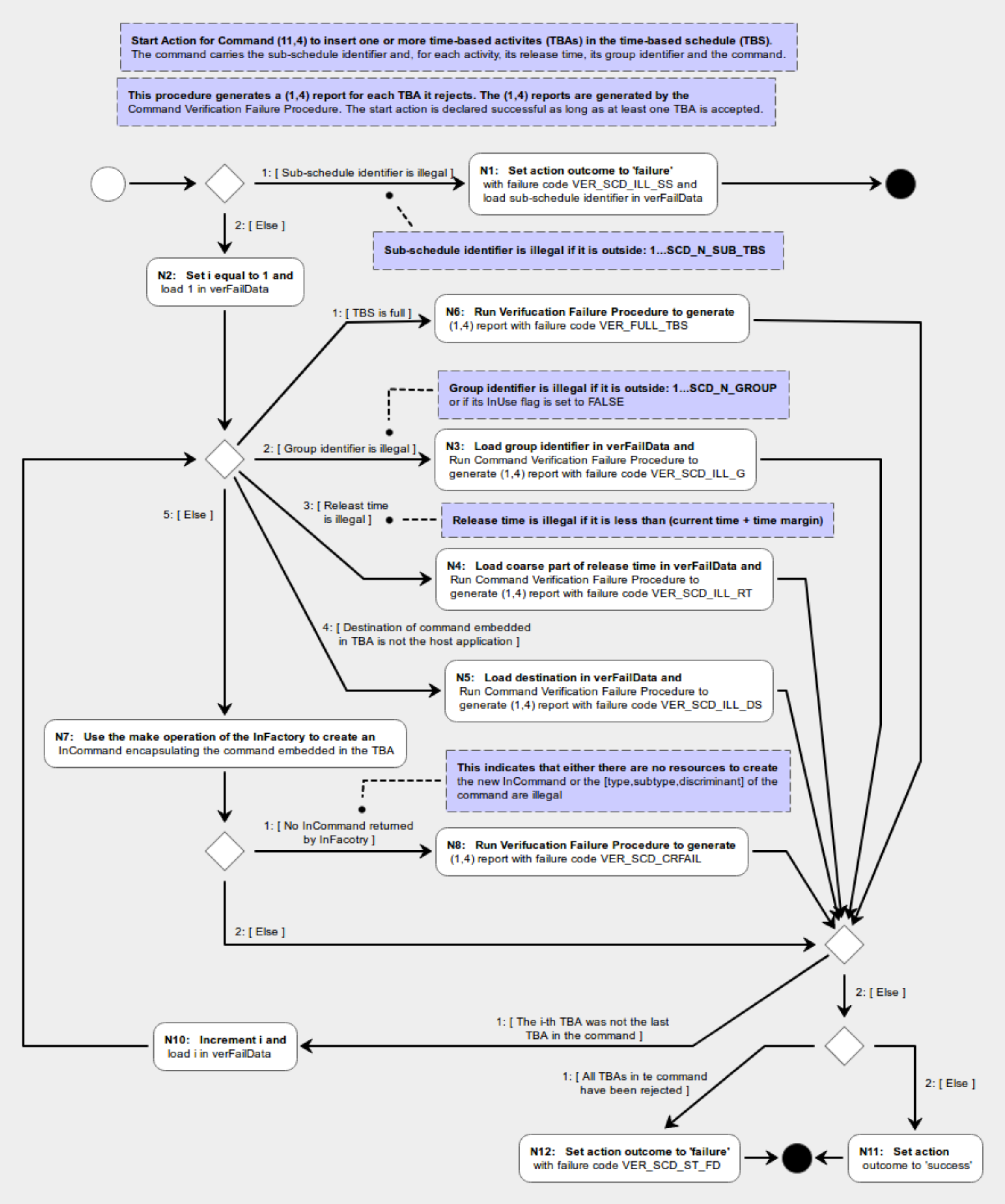

Start Action for Command (11,4) to insert one or more time-based activites (TBAs) in the time-based schedule (TBS).
The command carries the sub-schedule identifier and, for each activity, its release time, its group identifier and the command.
This procedure generates a (1,4) report for each TBA it rejects. The (1,4) reports are generated by the
Command Verification Failure Procedure. The start action is declared successful as long as at least one TBA is accepted.
1: [Sub-schedule identifier is illegal]
N1: Set action outcome to 'failure' with failure code VER_SCD_ILL_SS and load sub-schedule identifier in verFailData
Sub-schedule identifier is illegal if it is outside: 1...SCD_N_SUB_TBS
2: [Else]
N2: Set i equal to 1 and load 1 in verFailData
1: [TBS is full]
N6: Run Verifucation Failure Procedure to generate (1,4) report with failure code VER_FULL_TBS
Group identifier is illegal if it is outside: 1...SCD_N_GROUP or if its InUse flag is set to FALSE
2: [Group identifier is illegal]
N3: Load group identifier in verFailData and Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_SCD_ILL_G
5: [Else]
3: [Releast time is illegal]
Release time is illegal if it is less than (current time + time margin)
N4: Load coarse part of release time in verFailData and Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_SCD_ILL_RT
4: [Destination of command embedded in TBA is not the host application]
N5: Load destination in verFailData and Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_SCD_ILL_DS
N7: Use the make operation of the InFactory to create an InCommand encapsulating the command embedded in the TBA
This indicates that either there are no resources to create the new InCommand or the [type,subtype,discriminant] of the command are illegal
1: [No InCommand returned by InFacotry]
N8: Run Verification Failure Procedure to generate (1,4) report with failure code VER_SCD_CRFAIL
2: [Else]
2: [Else]
1: [The i-th TBA was not the last TBA in the command]
N10: Increment i and load i in verFailData
1: [All TBAs in te command have been rejected]
2: [Else]
N12: Set action outcome to 'failure' with failure code VER_SCD_ST_FD
N11: Set action outcome to 'success'

Fig. 11.1: Start Action of (11,4) Command Procedure

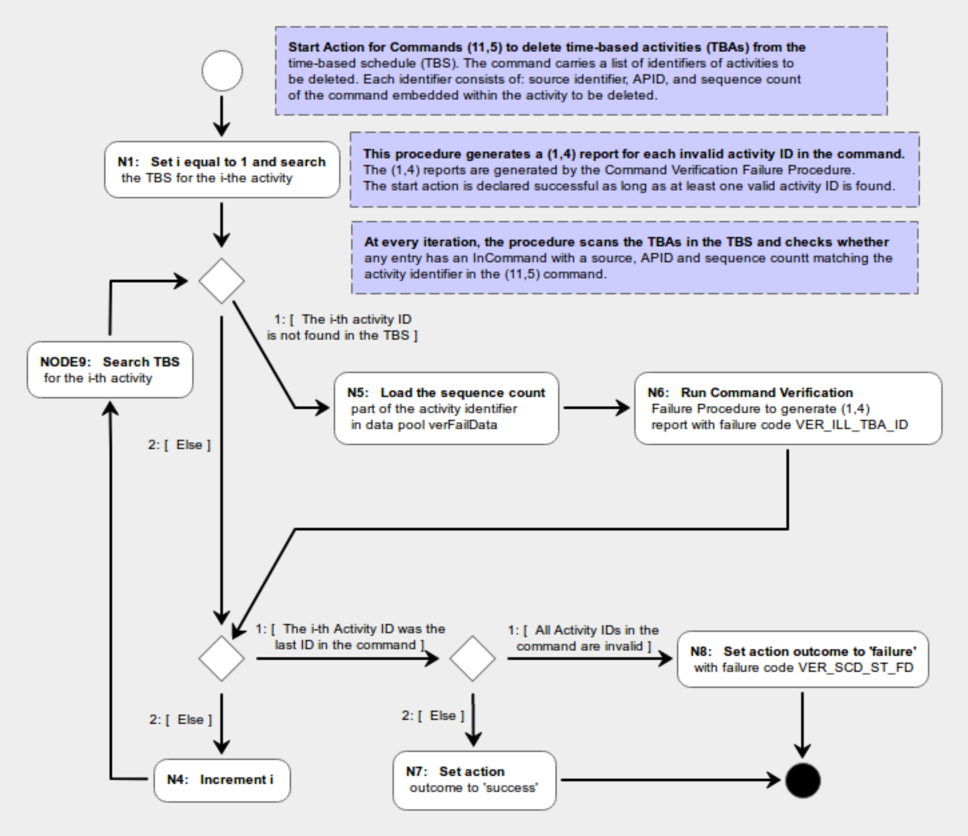
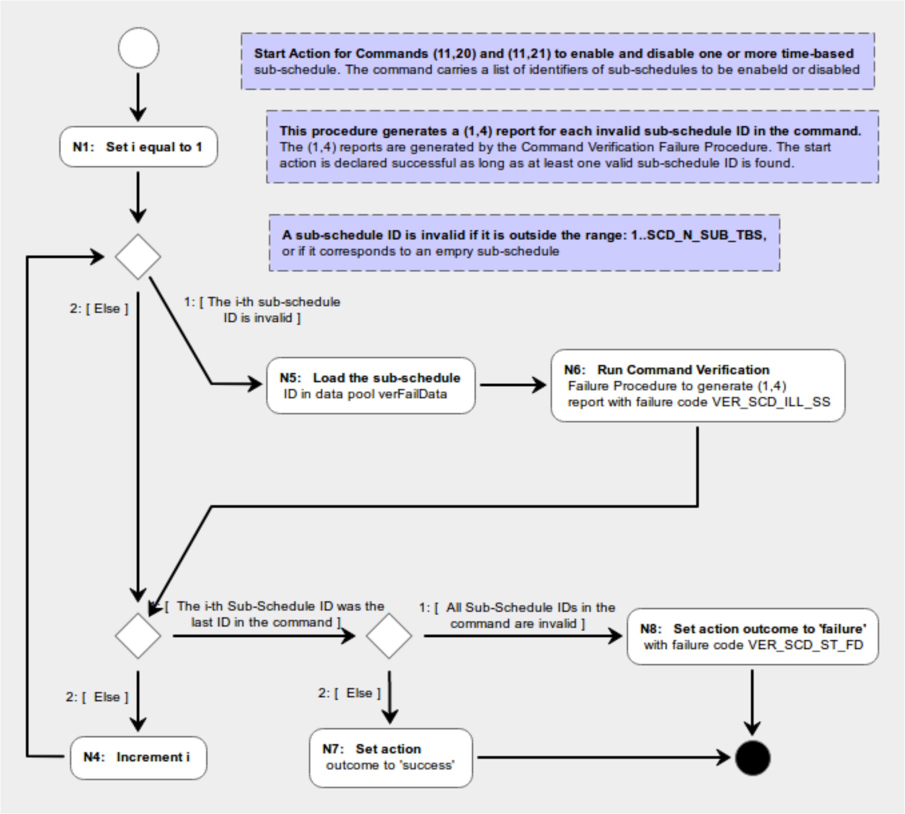

Start Action for Commands (11,5) to delete time-based activities (TBAs) from the time-based schedule (TBS). The command carries a list of identifiers of activities to be deleted. Each identifier consists of: source identifier, APID, and sequence count of the command embedded within the activity to be deleted.
This procedure generates a (1,4) report for each invalid activity ID in the command. The (1,4) reports are generated by the Command Verification Failure Procedure. The start action is declared successful as long as at least one valid activity ID is found.
At every iteration, the procedure scans the TBAs in the TBS and checks whether any entry has an InCommand with a source, APID and sequence countt matching the activity identifier in the (11,5) command.
N1: Set i equal to 1 and search the TBS for the i-the activity
NODE9: Search TBS for the i-th activity
1: [The i-th activity ID is not found in the TBS]
2: [Else]
N5: Load the sequence count part of the activity identifier in data pool verFailData
N6: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_TBA_ID
1: [The i-th Activity ID was the last ID in the command]
1: [All Activity IDs in the command are invalid]
N8: Set action outcome to 'failure' with failure code VER_SCD_ST_FD
2: [Else]
2: [Else]
N4: Increment i
N7: Set action outcome to 'success'

Fig. 11.2: Start Action of (11,5) Command Procedure



Start Action for Commands (11,20) and (11,21) to enable and disable one or more time-based sub-schedule. The command carries a list of identifiers of sub-schedules to be enabeld or disabled

This procedure generates a (1,4) report for each invalid sub-schedule ID in the command. The (1,4) reports are generated by the Command Verification Failure Procedure. The start action is declared successful as long as at least one valid sub-schedule ID is found.

A sub-schedule ID is invalid if it is outside the range: 1..SCD_N_SUB_TBS, or if it corresponds to an empry sub-schedule

N1: Set i equal to 1
2: [Else]
1: [The i-th sub-schedule ID is invalid]
N5: Load the sub-schedule ID in data pool verFailData
N6: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_SCD_ILL_SS
1: [The i-th Sub-Schedule ID was the last ID in the command]
1: [All Sub-Schedule IDs in the command are invalid]
N8: Set action outcome to 'failure' with failure code VER_SCD_ST_FD
2: [Else]
2: [Else]
N4: Increment i
N7: Set action outcome to 'success'

Fig. 11.3: Start Action of (11,20) and (11,21) Commands Procedure

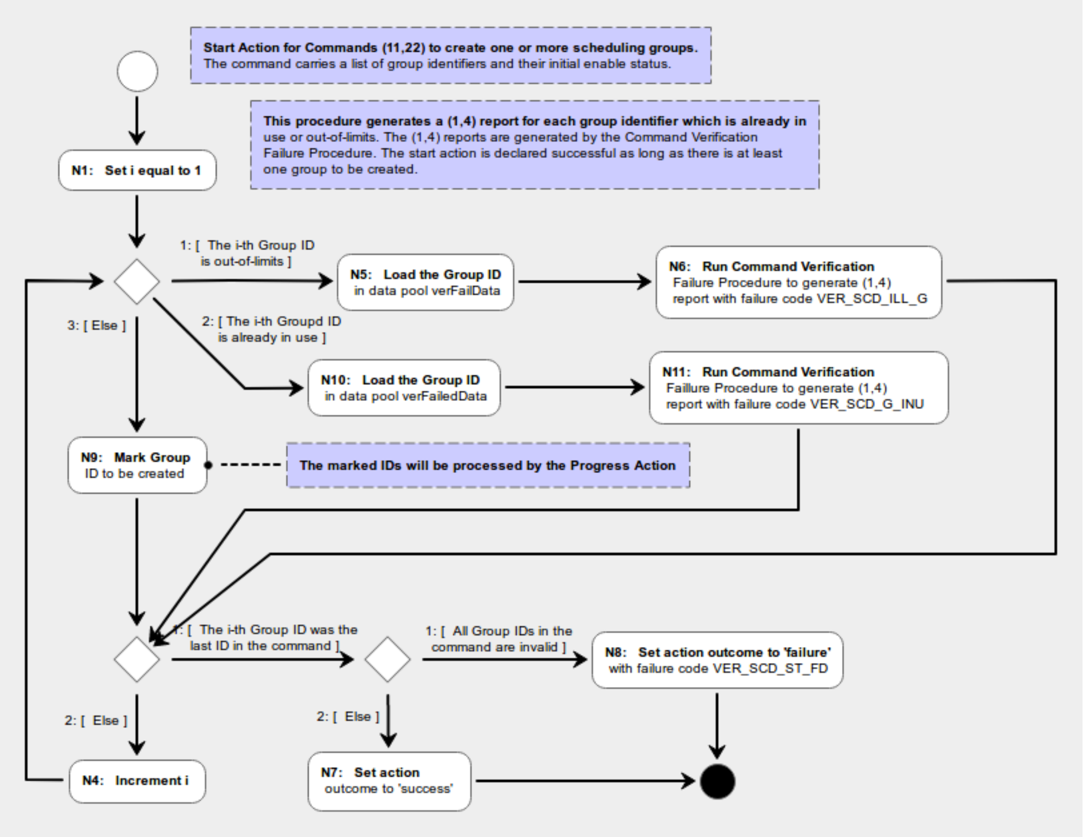
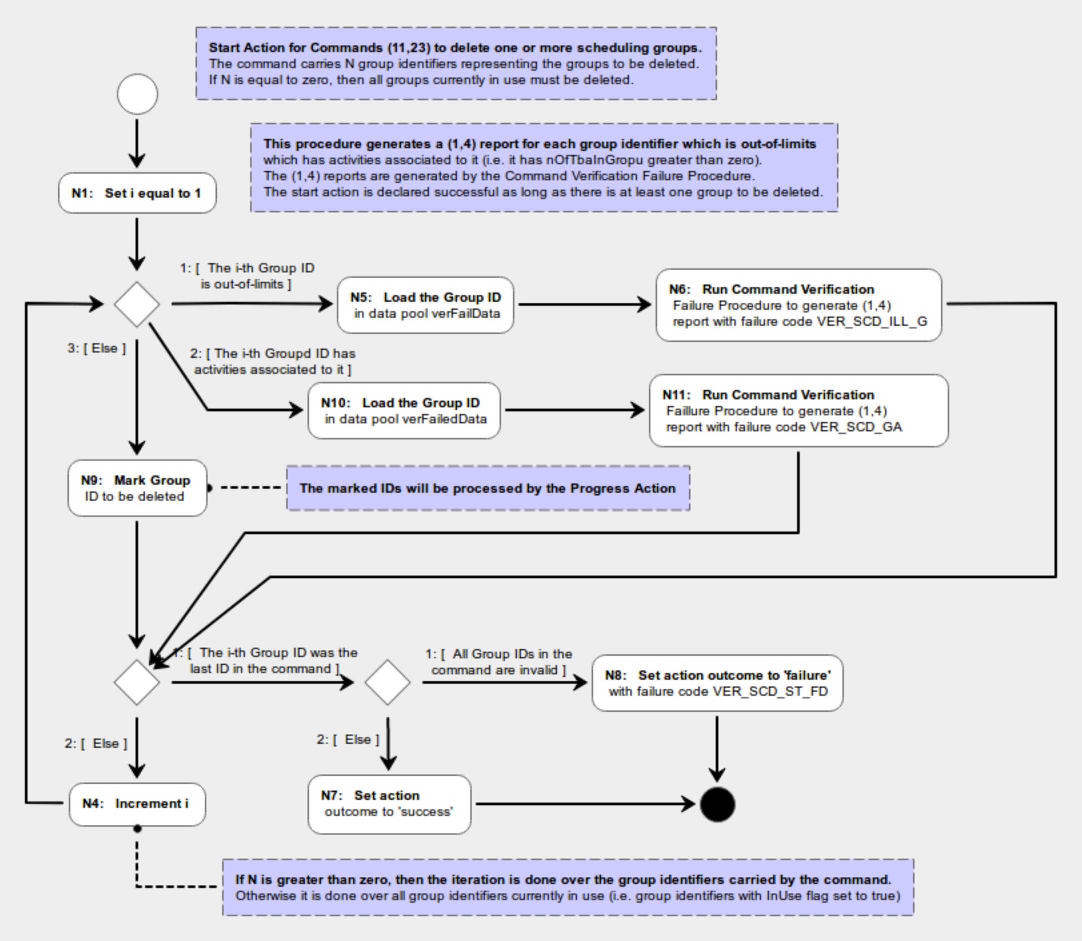

Start Action for Commands (11,22) to create one or more scheduling groups.
The command carries a list of group identifiers and their initial enable status.
This procedure generates a (1,4) report for each group identifier which is already in use or out-of-limits. The (1,4) reports are generated by the Command Verification Failure Procedure. The start action is declared successful as long as there is at least one group to be created.
N1: Set i equal to 1
1: [The i-th Group ID is out-of-limits]
N5: Load the Group ID in data pool verFailData
N6: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_SCD_ILL_G
3: [Else]
2: [The i-th Groupd ID is already in use]
N10: Load the Group ID in data pool verFailedData
N11: Run Command Verification Faillure Procedure to generate (1,4) report with failure code VER_SCD_G_INU
N9: Mark Group ID to be created
The marked IDs will be processed by the Progress Action
1: [The i-th Group ID was the last ID in the command]
1: [All Group IDs in the command are invalid]
N8: Set action outcome to 'failure' with failure code VER_SCD_ST_FD
2: [Else]
2: [Else]
N4: Increment i
N7: Set action outcome to 'success'

Fig. 11.4: Start Action of (11,22) Command Procedure



Start Action for Commands (11,23) to delete one or more scheduling groups.
The command carries N group identifiers representing the groups to be deleted.
If N is equal to zero, then all groups currently in use must be deleted.

This procedure generates a (1,4) report for each group identifier which is out-of-limits
which has activities associated to it (i.e. it has nOfTbaInGropu greater than zero).
The (1,4) reports are generated by the Command Verification Failure Procedure.
The start action is declared successful as long as there is at least one group to be deleted.

The marked IDs will be processed by the Progress Action

If N is greater than zero, then the iteration is done over the group identifiers carried by the command.
Otherwise it is done over all group identifiers currently in use (i.e. group identifiers with InUse flag set to true)

N1: Set i equal to 1
1: [The i-th Group ID is out-of-limits]
N5: Load the Group ID in data pool verFailData
N6: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_SCD_ILL_G
3: [Else]
2: [The i-th Groupd ID has activities associated to it]
N10: Load the Group ID in data pool verFailedData
N11: Run Command Verification Faillure Procedure to generate (1,4) report with failure code VER_SCD_GA
N9: Mark Group ID to be deleted
1: [The i-th Group ID was the last ID in the command]
1: [All Group IDs in the command are invalid]
N8: Set action outcome to 'failure' with failure code VER_SCD_ST_FD
2: [Else]
2: [Else]
N4: Increment i
N7: Set action outcome to 'success'

Fig. 11.5: Start Action of (11,23) Command Procedure

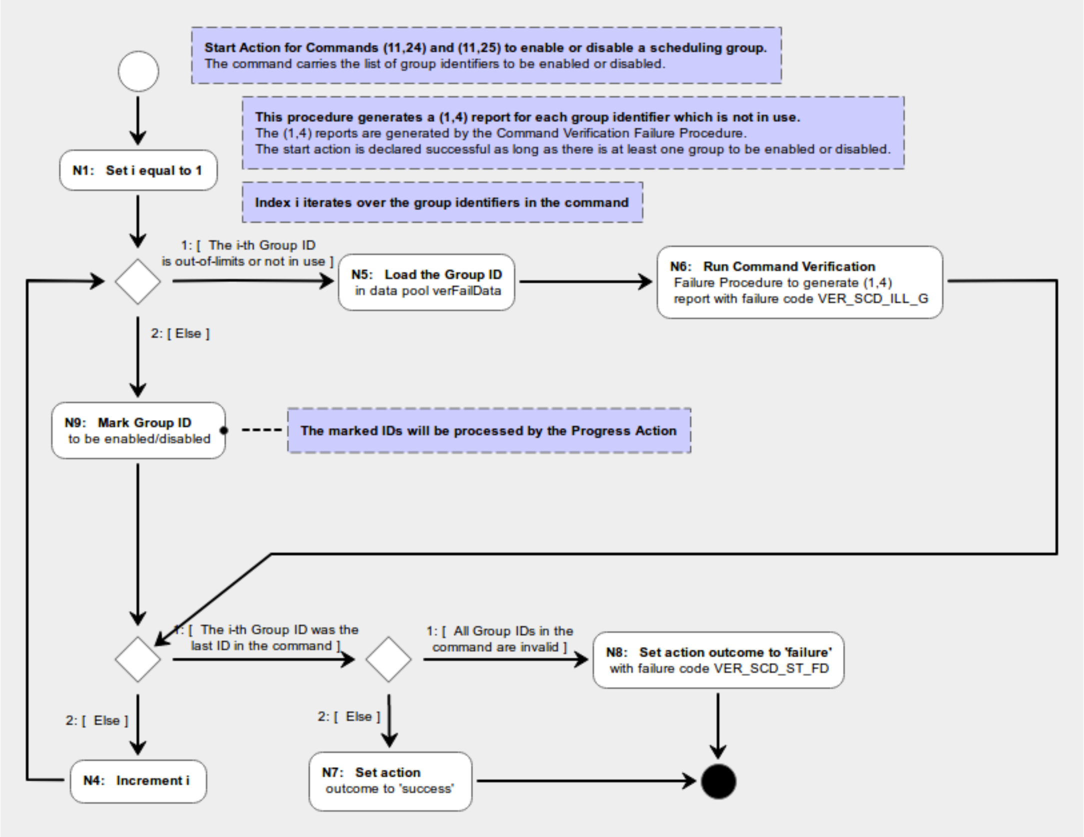

Start Action for Commands (11,24) and (11,25) to enable or disable a scheduling group.
The command carries the list of group identifiers to be enabled or disabled.
This procedure generates a (1,4) report for each group identifier which is not in use.
The (1,4) reports are generated by the Command Verification Failure Procedure.
The start action is declared successful as long as there is at least one group to be enabled or disabled.
N1: Set i equal to 1
Index i iterates over the group identifiers in the command
1: [The i-th Group ID is out-of-limits or not in use]
N5: Load the Group ID in data pool verFailData
N6: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_SCD_ILL_G
2: [Else]
N9: Mark Group ID to be enabled/disabled
The marked IDs will be processed by the Progress Action
1: [The i-th Group ID was the last ID in the command]
1: [All Group IDs in the command are invalid]
N8: Set action outcome to 'failure' with failure code VER_SCD_ST_FD
2: [Else]
2: [Else]
N4: Increment i
N7: Set action outcome to 'success'

Fig. 11.6: Start Action of (11,24) and (11,25) Command Procedure

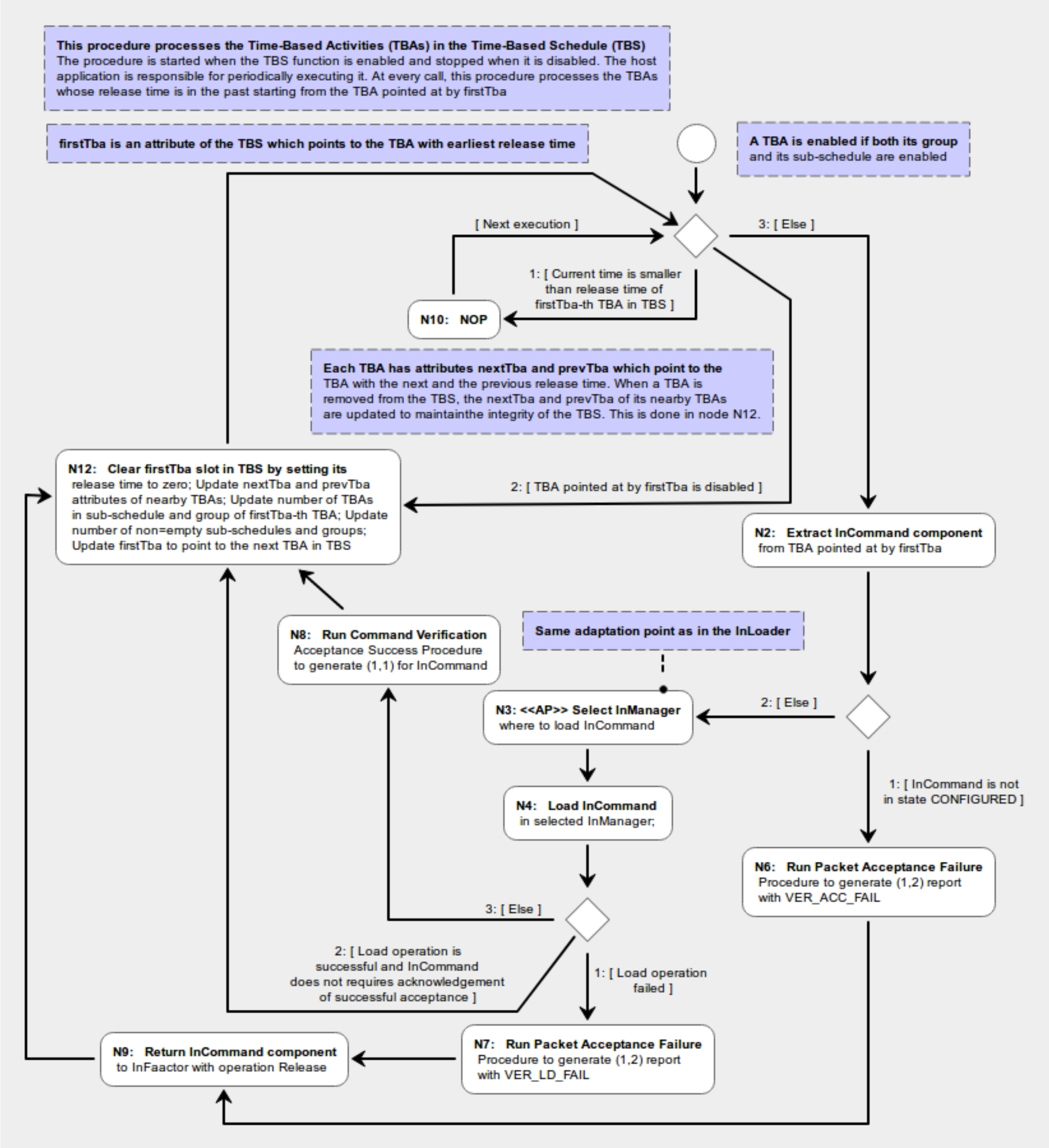

This procedure processes the Time-Based Activities (TBAs) in the Time-Based Schedule (TBS)
The procedure is started when the TBS function is enabled and stopped when it is disabled. The host application is responsible for periodically executing it. At every call, this procedure processes the TBAs whose release time is in the past starting from the TBA pointed at by firstTba
firstTba is an attribute of the TBS which points to the TBA with earliest release time
A TBA is enabled if both its group and its sub-schedule are enabled
[Next execution]
3: [Else]
1: [Current time is smaller than release time of firstTba-th TBA in TBS]
N10: NOP
Each TBA has attributes nextTba and prevTba which point to the TBA with the next and the previous release time. When a TBA is removed from the TBS, the nextTba and prevTba of its nearby TBAs are updated to maintainthe integrity of the TBS. This is done in node N12.
N12: Clear firstTba slot in TBS by setting its release time to zero; Update nextTba and prevTba attributes of nearby TBAs; Update number of TBAs in sub-schedule and group of firstTba-th TBA; Update number of non=empty sub-schedules and groups; Update firstTba to point to the next TBA in TBS
2: [TBA pointed at by firstTba is disabled]
N2: Extract InCommand component from TBA pointed at by firstTba
N8: Run Command Verification Acceptance Success Procedure to generate (1,1) for InCommand
Same adaptation point as in the InLoader
N3: <<AP>> Select InManager where to load InCommand
2: [Else]
1: [InCommand is not in state CONFIGURED]
N4: Load InCommand in selected InManager;
N6: Run Packet Acceptance Failure Procedure to generate (1,2) report with VER_ACC_FAIL
3: [Else]
2: [Load operation is successful and InCommand does not requires acknowledgement of successful acceptance]
1: [Load operation failed]
N9: Return InCommand component to InFaactor with operation Release
N7: Run Packet Acceptance Failure Procedure to generate (1,2) report with VER_LD_FAIL

Fig. 11.7: Time-Based Schedule Execution Procedure

12 On-Board Monitoring Service

The service type of the On-Board Monitoring Service is 12. The PUS Extension of the CORDET Framework supports this service only in part.

12.1 Parameter Monitoring Sub-Service

The parameter monitoring subservice controls the *parameter monitoring function*. This function monitors the values of a set of data items in the data pool[5]. Each monitored value is checked periodically to verify whether it conforms to a certain pattern of behaviour (e.g. whether it remains within certain limits). In some cases, the check is done uncondionally while in other cases it is done only if a validity condition is satisfied. Violations of the expected pattern of behaviour are reported through service 5 events.

The behaviour of the parameter monitoring function is described by the Parameter Monitoring Procedure of figure 12.1. The procedure is started when the parameter monitoring function becomes enabled and it is stopped when it becomes disabled. Thus, the enable status of the parameter monitoring function is given by the status of the Parameter Monitoring Procedure.

The Parameter Monitoring Procedure should be executed cyclically with a period of MON_-PER by the host application. The period MON_PER is the unit of time for all parameter monitoring actions in the sense that parameters are monitored periodically with a period which is a multiple of MON_PER.

Every time it is executed, the Parameter Monitoring Procedure processes a *parameter monitoring definition list* (PMDL). The PMDL consists of up to MON_N_PMON *parameter monitors*. Each parameter monitor defines a monitoring action for a data pool item. The same data pool item may be the object of several parameter monitors in the PMDL. Each parameter monitor has an identifier which is an integer in the range: [1..(MON_N_PMON].

Conditional checking is supported for all parameter monitors. To each parameter monitoring, the following items are associated:

- A validity parameter (a data pool item given by identifier `valDataItemId`)
- A bit-mask `valMask`
- An expected value `valExpVal`

The parameter monitoring action is only performed if the bit-wise AND of the bit-mask with the validity parameter is equal to the expected value. When a parameter is invalid, its checking status is set to INVALID. Note that, where desired, unconditional checking is achieved by setting the bit-mask and the expected value to zero.

Service 12 offers the means to report the content of the PMDL and to alter its content by adding or deleting parameter monitors from it.

A parameter monitor is characterized by the attributes listed in table 12.1. The first item is the identifier of the data pool item whose value the parameter monitor checks.

[5] It is recalled that the PUS uses the term 'parameter' to designate any data pool item. The PUS Extension of the CORDET Framework, instead, uses the term 'parameter' to designate a data pool item whose value is under the control of the external user of the application (e.g. the ground) and uses the term 'variable' to designate a data pool item whose value is under the control of the application itself (see section 5). In this section, the term 'parameter' is mostly used in the sense of the PUS.

The monitoring action is performed by a *Monitor Procedure*. Thus, for instance, there may be a Monitor Procedure which verifies that a parameter has a pre-defined value or there may be another procedure which verifies that a parameter remains within pre-defined limits. Attributes `monPrType` and `monPrId` identify, respectively, the type of the monitor procedure (e.g. Limit Monitoring Procedure or Delta Value Monitoring Procedure) and the specific parameter procedure instance which is used in the parameter monitor. Attribute `monPrRetVal` is the most recent return value of the Monitor Procedure and might, for instance, be equal to MON_ABOVE if the monitored value is above its upper limit or MON_NOT_EXP if the monitored parameter does not have the expected value.

Attribute `monPrPrevRetVal` is set to INVALID when a Parameter Monitor or the Monitoring Function are enabled and subsequently holds the return value of the Monitor Procedure at the previous execution. This attribute is used by the Parameter Monitoring Procedure to establish the number of consecutive times that the Monitor Procedure has the same return value (e.g. the number of consecutive times that a parameter is found to be above its upper limit).

After it is established that the Monitor Procedure has returned the same value `repNmb` times, this return value becomes the new checking status of the parameter monitor. The checking status is held in attribute `checkStatus`. Its range of values is given in table 12.2. When the checking status is updated, the following acions are executed:

- An entry is made in the Check Transition List (see section 12.1.2)
- The procedure to process the Check Transition List is run (see section 12.1.2)
- If an event is associated to the parameter monitor (i.e. if field `evtId` in table 12.1 is different from zero), then the Generate Prefined Event Function is called to generate the event
- If the parameter monitor is part of one or more functional monitors (i.e. if there non-zero entries in `fMonList` in table 12.1), then the functional monitors are notified

The violation events are: EVT_MON_*. The ground can choose whether or not an event is associated to a monitor but, once the type of monitor and the type of the monitored parameter (integer or real) is specified, the event type is also defined.

Parameter monitors can be individually enabled and disabled. When a parameter monitor becomes disabled, its Parameter Monitor Procedure is stopped. When a parameter monitor becomes enabled, its Parameter Monitor Procedure is started. Thus, the enable status of a parameter monitor is given by the started/stopped state of its Parameter Monitor Procedure.

The overall enable status of a parameter monitor can be controlled at global level (the Parameter Monitoring Procedure is started/stopped which enables/disables the entire monitoring function) or at local level (a specific Parameter Monitor State Machine and Procedure are started/stopped).

The Parameter Monitoring Procedure is executed cyclically. Individual parameter monitors may either be executed every time the procedure is executed or they may be executed only every N executions of the procedure. The value of N (a positive integer) is the period of the parameter monitor and is stored in attribute `per`. Attribute `perCnt` iterates in the range [0..(`per`-1)] and is used to keep track of the execution period of the parameter monitor.

The PUS Extension of the CORDET Framework provides the following commands to control the Parameter Monitoring Function:

- Commands (12,5) and (12,6) enable and disable the Parameter Monitoring Function
- Commands (12,1) and (12,2) enable and disable indivual parameter monitors
- Command (12,5) adds new parameter monitoring definitions to the PMDL
- Command (12,6) deletes some or all the parameter monitoring definitions in the PMDL
- Command (12,7) modifies one or more parameter monitoring definitions
- Command (12,8) triggers the generation of report (12,9) which reports the content of all or part of the PMDL
- Command (12,13) triggers the generation of report (12,14) which reports the status of all or some of the parameter monitors in the PMDL

Table 12.1: Attributes of Parameter Monitor

Name	Description	Constraint
`dataItemId`	Identifier of the data item monitored by the parameter monitor	Integer in range: 1..HK_MAX_ID
`monPrId`	Identifier of the Monitor Procedure which checks the parameter value	See section 12.1.1
`monPrType`	Identifier of the Monitor Procedure type which checks the parameter value	See section 12.1.1
`monPrRetVal`	Most recent return value of the Monitor Procedure	See table 12.2
`monPrPrevRetVal`	Previous return value of the Monitor Procedure (or INVALID after the monitoring procedure or the monitoring function has been enabled)	See table 12.2
`checkStatus`	Checking status of monitored parameter	See table 12.2
`per`	The monitoring period for the parameter monitor expressed as an integer multiple of the minimum monitoring period MON_-PER	Positive integer
`perCnt`	The phase counter	Integer in range: 0..(`per`-1)
`repNmb`	The repetition number for the monitoring check	Positive integer
`repCnt`	The repetition counter for the monitoring check	Integer in range: 0..(`repNmb`-1)
`evtId`	The identifier of the event to be generated if the parameter monitor detects a limit violation or zero if no event is to be generated	Zero or valid event identifier
`fMonList`	The list of MON_N_FPMON identifiers of the functional monitor to which the parameter monitor belongs (or zero if the parameter monitor does not belong to a functional monitor)	Integer in range: 0..MON_N_FMON
`valDataItemId`	Identifier of data item used for validity check	Integer in range: 1..HK_MAX_ID

Name	Description	Constraint
valMask	Mask used for validity check	Unsigned integer
valExpVal	Expected value for validity check	Unsigned integer

12.1.1 Monitor Procedures

The monitor procedures are responsible for checking the values of the monitored parameters and for determining whether or not they are nominal. The definition of the monitor procedures is an adaptation point for which the PUS Extension pre-defines the following options:

- Limit Check Monitor Procedure: verifies whether the value of the monitored parameter is within a pre-defined interval. See figure 12.2.
- Expected Value Monitor Procedure: verifies whether the monitored parameter has pre-defined value. See figure 12.3.
- Delta Check Monitor Procedure: verifies whether the difference between the current and previous value of the monitored parameter is within a pre-defined interval. See figure 12.4.

The PUS Standard [PS=SP] asks for the delta check to be performed on a mean value computed over a variable number of consecutive samples. The Delta Check Monitor Procedure uses a first-order moving average process. This is nearly equivalent to the mean value and is much simpler to implement.

To each parameter monitor, one instance of a monitor procedure is associated. The following limits apply to the number of monitor procedures of each type:

- MON_N_LIM: maximum number of monitor procedures of limit check type
- MON_N_EXP: maximum number of monitor procedures of expected value type
- MON_N_DEL: maximum number of monitor procedures of delta value type

It is an implementation-level decision whether the above procedures are "split by type" with separate version for different syntactical types of the parameter to be checked (e.g. one version for real-values parameters and another version for integer-valued parameters).

A monitor procedure is started when the parameter monitor is enabled (either because the entire parameter monitoring function is enabled or because the monitor itself is enabled) and may then be executed every time the parameter monitor is executed. At each execution, the procedure returns an outcome. If the procedure has found the parameter value to be nominal, it returns: MON_VALID. If, instead, it has found the parameter value to be non-nominal, it returns some other value. The range of return values other than MON_VALID is specific to each monitor procedure. Table 12.2 lists the potential outcomes of the three pre-defined monitor procedures. Applications must extend this range if they define new monitor procedures.

An application may define and load some parameter monitors as part of its initialization. In that case, the application is also responsible for starting the associated monitor procedures. In the case of parameter monitors which are defined dynamically using command (12,5), their monitor procedures are started by the command's progress action. Monitor procedures are also started and stopped when a parameter monitor is enabled and disabled through commands (12,1) and (12,2). In the case of parameter monitors which are modified dynamically using command (12,7), their monitor procedures are re-started by first stopping them and then starting them.

Table 12.2: Return Values of Monitor Procedure Execution

Name	Description
`MON_VALID`	Parameter is valid
`MON_NOT_EXP`	Parameter does not have the expected value
`MON_ABOVE`	Parameter value is above its upper limit
`MON_BELOW`	Parameter value is below its lower limit
`MON_DEL_ABOVE`	Parameter delta-value (difference between succesve values) is above its upper limit
`MON_DEL_BELOW`	Parameter delta-value (difference between succesve values) is below its lower limit

12.1.2 Check Transition List

The Check Transition List (CTL) is a data structure where the monitoring violations are accumulated. It consists of MON_N_CTL entries where each entry has the attributes listed in table 12.3.

Entries are added to the CTL by the Parameter Monitoring Procedure (see figure 12.1) when it wishes to report a monitoring violation. For this purpose, the framework provides an operation to add an entry to the CTL. When the first entry is added to the CTL, variable `ctlTimeFirstEntry` is updated with the current time.

The CTL is processed by the CTL Procedure of figure 12.5. This procedure checks whether either of the following conditions holds:

- The CTL is full (it contains at least MON_N_CTL entries)
- The CTL has not been flushed for longer than `maxRepDelay` monitoring cycles of duration MON_PER

If either condition is satisfied, the procedure generates the MonCheckTransRep report (12,12) to send the current content of the CTL to the service 12 user. The procedure then clears the CTL and sets `ctlTimeFirstEntry` to a value very far into the future.

The maximum reporting delay `maxRepDelay` can be modified with the `MonChgTransDelCmd` command (12,3). A report of the out-of-limits transitions in the CTL can also be triggered through the MonRepOOLCmd command (12,10) and is reported through the MonRepOOL-Rep report (12,9).

The destination of the (12,12) report is the "service 12 user". This is either pre-defined or it is the source of the most recent (12,15) command which enabled the parameter monitoring function (see section 12.5).

Table 12.3: Attributes of a Check Transition

Name	Description	Constraint
`dataItemId`	Identifier of the data item where the monitoring violation was detected	Integer in range: 1..HK_MAX_ID

Name	Description	Constraint
`monId`	Identifier of the Parameter Monitor which detected the violation	Integer in range: 1..MON_N_PMON
`monPrType`	Identifier of the type of the Monitor Procedure which detected the violation	See section 12.1.1
`expValChkMask`	In the case of an Expected Value Monitor violation, the expected value check mask	See section 12.1.1
`parVal`	The parameter value which triggered the violation	n.a.
`parValLim`	The parameter value limit which triggered the violation	n.a.
`checkStatus`	Checking status which triggered the violation	See table 12.2
`prevCheckStatus`	Checking status in the cycle before the violation was detected	A CUC time value

12.2 Functional Monitoring Sub-Service

The functional monitoring subservice controls the *functional monitoring function*. This function monitors the status of the parameter monitors.

The state of the functional monitoring function is held by the a *functional monitoring definition list* (FMDL). The FMDL consists of up to MON_N_FMON *functional monitors*. Each functional monitor acts as a listener for up to MON_N_PFMON parameter monitors: when one of these parameter monitors changes its checking status, the functional monitor is notified. The notification is sent out by the Parameter Monitoring Procedure. The functional monitoring function processes a notification by running the Functional Monitor Notification Procedure of figure 12.6.

Each functional monitor has an identifier which is an integer in the range: [1..MON_N_FMON]. Table tab:fmonAtt lists the attributes of a functional monitor in the FMDL.

Attribute `pmonIdList` holds the list of parameter monitors associated to the functional monitor. The list has a statically defined size of MON_N_FMON. The number of non-zero entries in the list is given by attribute `nOfMon`. Attribute `minFailNumber` holds the minimum failing number for the functional monitor: the monitor is declared to have failed only if at least `minFailNumber` of its `nOfMon` parameter monitors have detected a monitoring violation.

Attribute `checkStatus` holds the checking status for the functional monitor which is one of the values listed in table 12.5. The checking status is initialized to UNCHECKED when the functional monitor is created or when it is enabled and is then updated every time the functional monitor is notified.

Attribute `isProtected` is a boolean flag which indicates whether the functional monitor is protected.

Conditional checking is supported for all functional monitors. For this purpose, to each functional monitor, the following items are associated:

- A validity parameter (a data pool item given by identifier `valDataItemId`)

- A bit-mask `valMask`
- An expected value `valExpVal`

The monitoring action is only performed if the bit-wise AND of the bit-mask with the validity parameter is equal to the expected value. When a functional monitor is invalid, its checking status is set to INVALID. Note that, where desired, unconditional checking is achieved by setting the bit-mask and the expected value to zero.

Service 12 offers the means to report the content of the FMDL and to alter its content by adding or deleting parameter monitors from it.

If a functional monitor declares a failure, the pre-defined event EVT_FMON_FAIL is raised. This event carries the identifiers of the `nOfMon` parameter monitors and their current checking status.

Table 12.4: Attributes of a Functional Monitor

Name	**Description**	**Constraint**
`pmonIdList`	List of MON_N_PFMON identifiers of parameter monitors in the functional monitor	Integers in range: 0..MON_N_PMON
`nOfMon`	Number of parameter monitors defined in the functional monitor (number of non-zero entries in `pmonIdList`	Integers in range: 0..MON_N_-PFMON
`minFailNumber`	Minimum parameter monitor failing number	Integer in range: 1..`nOfMon`
`checkStatus`	Checking status of functional monitor	See table 12.5
`isProtected`	The flag indicating whether the functional monitor is protected	Boolean flag
`isEnabled`	The flag indicating whether the functional monitor is enabled	Boolean flag
`valDataItemId`	Identifier of data item used for validity check	Integer in range: 1..HK_MAX_ID
`valMask`	Mask used for validity check	Unsigned integer
`valExpVal`	Expected value for validity check	Unsigned integer

Table 12.5: Checking Statuses of a Functional Monitor

Name	**Description**
`MON_UNCHECKED`	Functional monitor has not yet been notified since it was last enabled
`MON_INVALID`	The validity condition for the functional monitor is not satisfied
`MON_RUNNING`	The number of parameter monitors in the functional monitor which reporterd a monitoring violation is below the minimum failing number
`MON_FAILED`	The number of parameter monitors in the functional monitor which reported a monitoring violation is greater than or eqaul to the minimum failing number

12.3 Service 12 Report and Command Definition

Tables **??** to **??** formally specify the service 12 commands and reports by specifying how the actions, checks and attributes of generic out-going commands and reports are specialized for this service (see section 7).

Table 12.6: Specification of MonEnbParMonDefCmd Component

Name	MonEnbParMonDefCmd(12,1)
Description	Command to enable one or more monitoring definitions
Parameters	The identifiers of the monitoring definitions to be enabled
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of Multi-Parameter Monitor Commands of figure 12.7
Progress Action	For every parameter monitor identifier in the command which has not been rejected by the Start Action: reset its repetition counter (attribute repCnt) and start its Monitor Procedure. Increment the data pool variable representing the number of enabled parameter monitors by the number of enabled parameter monitors. Set the action outcome to 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.7: Specification of MonDisParMonDefCmd Component

Name	MonDisParMonDefCmd(12,2)
Description	Command to disable one or more monitoring definitions
Parameters	The identifiers of the monitoring definitions to be disabled
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of Multi-Parameter Monitor Commands of figure 12.7
Progress Action	For every valid Parameter Monitor Identifier in the command: stop its Monitor Procedure and set its checking status (attribute checkStatus) to UNCHECKED. Decrement the data pool variable representing the number of enabled parameter monitors by the number of disabled parameter monitors. Set the action outcome to 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.8: Specification of MonChgTransDelCmd Component

Name	MonChgTransDelCmd(12,3)
Description	Command to change the maximum delay after which the content of the check transition list (CTL) is reported through a (12,12) report
Parameters	The new value of the maximum transition reporting delay
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Set action outcome to 'success' if the argument of the command (the new maximum reporting delay) is a positive integer; otherwise, set the outcome to 'failure'
Progress Action	Update the maximum report delay in the data pool with the value in the command
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.9: Specification of MonDelAllParMonCmd Component

Name	MonDelAllParMonCmd(12,4)
Description	Command to delete all parameter monitoring definitions
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Set action outcome to 'success' if the parameter monitoring function is disabled and if none of the currently defined parameter monitors is attached to a functional monitor which is protected; otherwise set the action outcome to 'failed
Progress Action	Delete all entries from the Parameter Monitoring Definition List (PMDL) and delete all entries from the Check Transition List (CTL). Set the data pool variable representing the number of remaining available PMDL entries equal to the size of the PMDL. Set the action outcome to 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.10: Specification of MonAddParMonDefCmd Component

Name	MonAddParMonDefCmd(12,5)
Description	Command to add one or more parameter definitions
Parameters	The parameter definitions to be added. Each parameter definition consists of parameter monitor identifier, identifier of parameter to be monitored, description of validity check, repetition counter, description of monitoring check (including identifiers of events to be generated in case of monitoring violation)
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of (12,5) Command of figure 12.8
Progress Action	For all parameter monitor definitions which have been accepted by the start action: add the definition to the Parameter Monitor Definition List (PMDL), set the checking status of the new parameter monitor to ’unchecked’, reset its repetition counter and phase counter to zero. Decrement the data pool variable representing the number of remaining available entries in the PMDL by the number of added parameter monitors. Set the action outcome to ’completed’.
Termination Action	Default implementation (set action outcome to ’success’)
Abort Action	Default implementation (set action outcome to ’success’)

Table 12.11: Specification of MonDelParMonDefCmd Component

Name	MonDelParMonDefCmd(12,6)
Description	Command to delete one or more parameter monitoring definitions
Parameters	The identifiers of the parameter monitors to be deleted
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of (12,6) Command of figure 12.9
Progress Action	For all parameter monitor identifiers which have been accepted by the Start Action: delete the parameter monitor from the Parameter Monitor Definition List (PMDL). Increment the data pool variable representing the number of remaining available PMDL entries by the number of deleted parameter monitoring definitions. Set the action outcome to ’completed’.
Termination Action	Default implementation (set action outcome to ’success’)
Abort Action	Default implementation (set action outcome to ’success’)

Table 12.12: Specification of MonModParMonDefCmd Component

Name	MonModParMonDefCmd(12,7)
Description	Command to modify one or more parameter definitions
Parameters	The modified parameter definitions. Each modified parameter definition consists of identifier of parameter monitor, identifier of parameter to be monitored, repetition counter, description of monitoring check (including identifiers of events to be generated in case of monitoring violation)
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action for (12,7) Command of figure 12.10
Progress Action	For all the parameter monitors which have been accepted by the Start Action: modify the parameter monitor definition in the PMDL according to the command parameters, set the check status to 'unchecked', reset the repetition counter and the phase counter to zero
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.13: Specification of MonRepParMonDefCmd Component

Name	MonRepParMonDefCmd(12,8)
Description	This command triggers the generation of a (12,9) report carrying one or more parameter monitor definitions
Parameters	The identifiers of the parameter monitors whose definitions are to be reported
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the Start Action of (12,8) Command Procedure of figure 12.11
Progress Action	Configure the (12,9) reports created by the Start Action and load them in the OutLoader. Set the action outcome to 'success' if the load operation is successful and to 'failed' otherwise.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.14: Specification of MonRepParMonDefRep Component

Name	MonRepParMonDefRep(12,9)
Description	Report generated in response to a (12,8) command to report one or more monitoring definitions.
Parameters	The maximum transition reporting delay, and the description of all requested parameter monitors. Each parameter monitor description consists of: parameter monitor identifier, identifier of monitored data item, description of validity condition of parameter monitor (identifier of validity data item, mask and expected value), monitoring interval, monitoring status, repetition number, check type and check-dependent data
Discriminant	None
Destination	The source of the (12,8) command which triggered the generation of the report
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 12.15: Specification of MonRepOutOfLimitsCmd Component

Name	MonRepOutOfLimitsCmd(12,10)
Description	This command triggers the generation of a (12,11) report holding the parameter monitors which are out of limits
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the Start Action of (12,10) Command Procedure of figure 12.11
Progress Action	Attempt to load the (12,11) report created by the Start Manager in the OutLoader. If the load operation is successful, set the action outcome to 'completed'. Otherwise, release the (12,11) report and set the action outcome to 'failed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.16: Specification of MonRepOutOfLimitsRep Component

Name	MonRepOutOfLimitsRep(12,11)
Description	Report generated in response to a (12,10) command carrying the parameter monitors which are out of limits
Parameters	The description of the monitors which are out of limits. Each description consists of: parameter monitor identifier, identifier of monitored data item, check type, current parameter value, value of crossed limit, previous and current checking status, time when the monitoring violation occurred.
Discriminant	None
Destination	The source of the (12,10) command which triggers the generation of the report
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 12.17: Specification of MonCheckTransRep Component

Name	MonCheckTransRep(12,12)
Description	Report carrying the content of the Check Transition List (CTL).
Parameters	The entries in the Check Transition List.
Discriminant	None
Destination	The user of the parameter monitoring function (either a predefined application or the source of the most recent command to enable the parameter monitoring function).
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 12.18: Specification of MonRepParMonStatCmd Component

Name	MonRepParMonStatCmd(12,13)
Description	This command triggers the generation of a (12,14) report carrying the status of all parameter monitors
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Attempt to retrieve an OutComponent of type (12,14) from the OutFactory with a size adequate to hold the status of all currently defined parameter monitors in the PMDL and set the action outcome to 'success' if the operation is successful. If, instead, the OutFactory fails to return the requested OutComponent generate error report OUTFACTORY_FAIL
Progress Action	Configure the OutComponent retrieved by the Start Action with the status of all parameter monitors currently defined in the PMD and load it in the OutLoader. Set action outcome to 'success' if the load operation was successful and to 'failed' otherwise.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.19: Specification of MonRepParMonStatRep Component

Name	MonRepParMonStatRep(12,14)
Description	Report generated in response to a (12,13) report carrying the status of all currently defined parameter monitors
Parameters	The checking status of all parameter monitors currently defined in the PDML
Discriminant	None
Destination	The source of the (12,13) command which triggered the generation of the report
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 12.20: Specification of MonEnbParMonFuncCmd Component

Name	MonEnbParMonFuncCmd(12,15)
Description	Command to enable the monitoring function
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Set action outcome to 'success' if the Monitoring Function is disabled (i.e. if the Monitoring Function Procedure is stopped).
Progress Action	Start the Monitoring Function Procedure. Set service 12 parameter ServUser equal to the source of this command. Set the enable status of the Parameter Monitoring Function in the data pool to: 'enabled'. Set the action outcome to: 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.21: Specification of MonDisParMonFuncCmd Component

Name	MonDisParMonFuncCmd(12,16)
Description	Command to disable the parameter monitoring function
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Set the action outcome to 'failure' if the Functional Monitoring Function is supported by the application and is enabled. Otherwise set the action outcome to 'success'.
Progress Action	Stop the Parameter Monitoring Procedure. Set the enable status of the Parameter Monitoring Function in the data pool to: 'disabled'. Set the action outcome to: 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.22: Specification of MonEnbFuncMonCmd Component

Name	MonEnbFuncMonCmd(12,17)
Description	Command to enable the functional monitoring function
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Set the action outcome to 'failed' if the Parameter Monitoring Function is 'disabled'; otherwise set the action outcome to 'success'.
Progress Action	Set the enable status of the Functional Monitoring Function in the data pool to 'enabled'. Set the action outcome to 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.23: Specification of MonDisFuncMonCmd Component

Name	MonDisFuncMonCmd(12,18)
Description	Command to disable the functional monitoring function
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Default implementation (set action outcome to 'success')
Progress Action	Set the enable status of the Functional Monitoring Function in the data pool to 'disabled'. Set the action outcome to 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.24: Specification of MonEnbFuncMonDefCmd Component

Name	MonEnbFuncMonDefCmd(12,19)
Description	Command to enable one or more functional monitoring definitions
Parameters	The identifiers of the functional monitors to be enabled
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of Multi-Functional Monitor Commands of figure 12.16
Progress Action	For all the valid functional monitor identifiers: enable the functional monitor by setting its isEnabled field in the FDML to true. Set action outcome to 'completed'. Increment the data pool variable representing the number of enabled functional monitors by the number of enabled functional monitors. Set the action outcome to 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.25: Specification of MonDisFuncMonDefCmd Component

Name	MonDisFuncMonDefCmd(12,20)
Description	Command to disable one ore more functional monitoring definitions
Parameters	The identifiers of the functional monitors to be disabled
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of Multi-Functional Monitor Commands of figure 12.16
Progress Action	For all the valid functional monitor identifiers: enable the functional monitor by setting its isEnabled field in the FDML to true. Set action outcome to 'completed'. Decrement the data pool variable representing the number of enabled functional monitors by the number of disabled functional monitors. Set the action outcome to 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.26: Specification of MonProtFuncMonDefCmd Component

Name	MonProtFuncMonDefCmd(12,21)
Description	Command to protect one or more functional monitoring definitions
Parameters	The identifiers of the functional monitors to be protected
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of Multi-Functional Monitor Commands of figure 12.16
Progress Action	For each functional monitor which has been accepted for execution by the Start Action: set its status to protected in the FMDL. Set the action outcome to 'completed'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.27: Specification of MonUnprotFuncMonDefCmd Component

Name	MonUnprotFuncMonDefCmd(12,22)
Description	Command to unprotect one or more functional monitoring definitions
Parameters	The identifiers of the functional monitors to be unprotected
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the Start Action of Multi-Functional Monitor Command Procedure of figure 12.16 and set action outcome to 'success'
Progress Action	For all the valid functional monitor identifiers in the command: unprotect the functional monitor by setting its isProtected flag in the FDML to true. Set action outcome to 'success'.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.28: Specification of MonAddFuncMonDefCmd Component

Name	MonAddFuncMonDefCmd(12,23)
Description	Command to add one or more functional monitoring definitions
Parameters	The description of the functional monitors to be added. Each description consists of: identifier, description of check validity condition (identifier of validity data item. mask, expected value), the event definition identifier, minimum failing number, list of identifiers of parameter monitors to be associated to the functional monitor.
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the Start Action of (12,23) Command Procedure of figure 12.13.
Progress Action	For each functional monitor identifier accepted for execution by the Start Action: add the functional monitor definition to the FMDL, set its checking status to ’unchecked’, set its enable status to ’disabled’, and set its protected status to ’unprotected’. Decrement the data pool variable representing the number of remaining available functional monitors by the number of added functional monitors. Set the action outcome to ’completed’.
Termination Action	Default implementation (set action outcome to ’success’)
Abort Action	Default implementation (set action outcome to ’success’)

Table 12.29: Specification of MonDelFuncMonDefCmd Component

Name	MonDelFuncMonDefCmd(12,24)
Description	Command to delete one or more functional monitoring definitions to the FMDL
Parameters	The identifiers of the functional monitors to be deleted
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the Command (12,24) Start Action Procedure of figure 12.14
Progress Action	For all functional monitors which have been accepted for execution by the Start Action: remove the functional monitor from the FMDL Increment the data pool variable representing the number of remaining available functional monitors by the number of deleted functional monitors. Set the action outcome to ’completed’.
Termination Action	Default implementation (set action outcome to ’success’)
Abort Action	Default implementation (set action outcome to ’success’)

Table 12.30: Specification of MonRepFuncMonDefCmd Component

Name	MonRepFuncMonDefCmd(12,25)
Description	This command triggers the generation of a (12,26) report carrying the definition of one or more functional monitors
Parameters	The identifiers of the functional monitors whose definition is to be reported
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the Start Action of (12,25) Command Procedure of figure 12.15
Progress Action	Configure the (12,26) reports created by the Start Action and load them in the OutLoader. Set the action outcome to 'success' if the load operation is successful and to "failed' otherwise.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.31: Specification of MonRepFuncMonDefRep Component

Name	MonRepFuncMonDefRep(12,26)
Description	Report generated in response to a (12,25) command to carry the definition of some or all functional monitoring definitions
Parameters	The description of the functional monitors. Each description consists of: identifier, description of check validity condition (identifier of validity data item. mask, expected value), the protection status, the checking status, the event definition identifier, minimum failing number, list of identifiers of parameter monitors associated to the functional monitor.
Discriminant	None
Destination	The source of the (12,25) command which triggered the generation of the report
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 12.32: Specification of MonRepFuncMonStatCmd Component

Name	MonRepFuncMonStatCmd(12,27)
Description	This command triggers the generation of a (12,28) report carrying the status of all functional monitors
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Attempt to retrieve an OutComponent of type (12,28) from the OutFactory with a size adequate to hold the status of all currently defined functional monitors in the FMDL and set the action outcome to 'success' if the operation is successful. If, instead, the OutFactory fails to return the requested OutComponent generate error report OUTFACTORY_FAIL
Progress Action	Configure the OutComponent retrieved by the Start Action with the status of all functional monitors currently defined in the FMDL and load it in the OutLoader. Set action outcome to 'success' if the load operation was successful and to 'failed' otherwise.
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 12.33: Specification of MonRepFuncMonStatRep Component

Name	MonRepFuncMonStatRep(12,28)
Description	Report generated in response to a (12,27) command carrying the status of all currently defined functional monitors
Parameters	The checking status of all functional monitors currently defined in the PDML
Discriminant	None
Destination	The source of the (12,27) command which triggered the generation of the report
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

12.4 Service 12 Constants

The service 12 constants are listed in table 12.34.

Table 12.34: Constants for Monitoring Service

Name	Description
MON_N_PMON	Maximum number of entries in the Parameter Monitoring Definition List PMDL (maximum number of parameter monitors in the application)
MON_N_LIM	Maximum number of parameter monitors with a limit check
MON_N_EXP	Maximum number of parameter monitors with an expected value check

Name	Description
MON_N_DEL	Maximum number of parameter monitors with a delta value check
MON_PER	Minimum monitoring period
MON_N_CLST	Maximum number of entries in the Check Transition List
MON_N_FMON	Maximum number of entries in the Functional Monitoring Definition List FMDL (maximum number of functional monitors in the application)
MON_N_PFMON	Maximum number of parameter monitors in a functional monitor
MON_NFPMON	Maximum number of functional monitors to which the same parameter monitor may belong
MON_FPER	Period of execution of the Functional Monitoring Procedure expressed as a multiple of MON_PER

12.5 Service 12 Observables and Parameters

The service 12 observables and parameters are listed in table 12.35. The initial values of the variables carrying the number of available and enabled monitors depends on the number parameter and functional monitors which are pre-defined at initialization time (which is an adaptation point) and must therefore be set by the application as part of its initialization.

The attributes of the parameter and functional monitors in the PMDL and FMDL are visible through service 12 reports and are therefore not defined as "observables" in the data pool.

The service 12 configuration is defined by the parameter holding the identifier of the service 12 user. This is either a pre-defined value or it is the source of the most recent (12,15) command to enable the Parameter Monitoring Function.

Table 12.35: Observables and Parameters for On-Board Monitoring Service

Name	Kind	Description
maxRepDelay	par	Maximum reporting delay
servUser	par	Identifier of the application acting as user of the on-board monitoring service
ctlRepDelay	var	Maximum reporting delay for the CTL in multiples of MON_PER
ctlTimeFirstEntry	var	Time when first entry has been added to the CTL
funcMonEnbStatus	var	Functional monitoring enable status
nmbAvailFuncMon	var	Number of available functional monitors in the FMDL
nmbAvailParMon	var	Number of available parameter monitors in the PMDL
nmbEnbFuncMon	var	Number of enabled functional monitors in the FMDL
nmbEnbParMon	var	Number of enabled parameter monitors in the PMDL

Name	Kind	Description
`parMonEnbStatus`	var	Enable state of parameter monitoring function

12.6 Service 12 Adaptation Points

The table in this section lists the adaptation points for the On-Board Monitoring Service. The first adaptation point covers the range of monitor procedures. Those supported by default by the PUS Extension of the CORDET Framework are those defined by the PUS.

The second and third adaptation points cover the initialization of the PMDL and FMDL. Both lists are empty by default but applications are free to pre-define both parameter monitors and functional monitors. In this case, it becomes their responsibility to populate the PMDL and FMDL with the predefined items. This should normally be done as part of the application initialization.

Table 12.36: Adaptation Points for Service 12 (On-Board Monitoring Service)

AP ID	Adaptation Point	Close-Out Value
P-S12-1	Definition of Parameter Monitoring Procedures (New AP)	Three parameter monitoring procedures are defined (limit check, expected value and delta check)
P-S12-2	Definition of Parameter Monitoring Definition List (PMDL) (New AP)	By default, all parameter monitors are empty
P-S12-3	Definition of Functional Monitoring Definition List (FMDL) (New AP)	By default, all parameter monitors are empty

12.7 Service 12 Requirements

The table in this section lists the requirements for the On-Board Monitoring Service

Table 12.37: Requirements for Service 12 (On-Board Monitoring Service)

Req. ID	Requirement Text
P-S12-1/S	*The PUS Extension of the CORDET Framework shall provide a Monitoring Function Procedure implementing the behaviour shown in figure 12.1*
P-S12-2/S	*The PUS Extension of the CORDET Framework shall implement a Parameter Monitor Definition List (PMDL) consisting of MON_N_PMON Parameter Monitor Definitions with the fields defined in table 12.1*
P-S12-3/S	*The PUS Extension of the CORDET Framework shall implement a Check Transition List (CTL) consisting of MON_N_CLST Check Transitions with the fields defined in table 12.3*
P-S12-4/S	*The PUS Extension of the CORDET Framework shall provide the Limit Check Monitoring Procedure implementing the behaviour shown in figure 12.2*
P-S12-5/S	*The PUS Extension of the CORDET Framework shall provide the Expected Value Monitoring Procedure implementing the behaviour shown in figure 12.3*
P-S12-6/S	*The PUS Extension of the CORDET Framework shall provide the Delta Value Monitoring Procedure implementing the behaviour shown in figure 12.4*

Req. ID	Requirement Text
P-S12-7/S	*The PUS Extension of the CORDET Framework shall provide the Process CTL Procedure implementing the behaviour shown in figure 12.5*
P-S12-8/S	*The PUS Extension of the CORDET Framework shall provide the Functional Monitor Notification Procedure implementing the behaviour shown in figure 12.6*
P-S12-9/C	*Applications shall be responsible for initializing the data pool items nmbAvailParMon and nmbEnbParMon giving the number of available and enabled parameter monitors in the PMDL*
P-S12-10/C	*As part of their initialization, applications shall be responsible for configuring and starting the monitor procedures associated to the pre-defined parameter monitors*
P-S12-11/C	*Applications shall be responsible for initializing the data pool items nmbAvailFuncMon and nmbEnbFuncMon giving the number of available and enabled functional monitors in the FMDL*
P-S12-12/C	*During their initialization, applications shall start the Monitoring Function Procedure*
P-S12-13/C	*During normal operation, applications shall cyclically execute the Monitoring Function Procedure with a period of MON_PER*
P-S12-14/S	*The PUS Extension of the CORDET Framework shall support the commands and reports specified in tables* ?? *to* ??

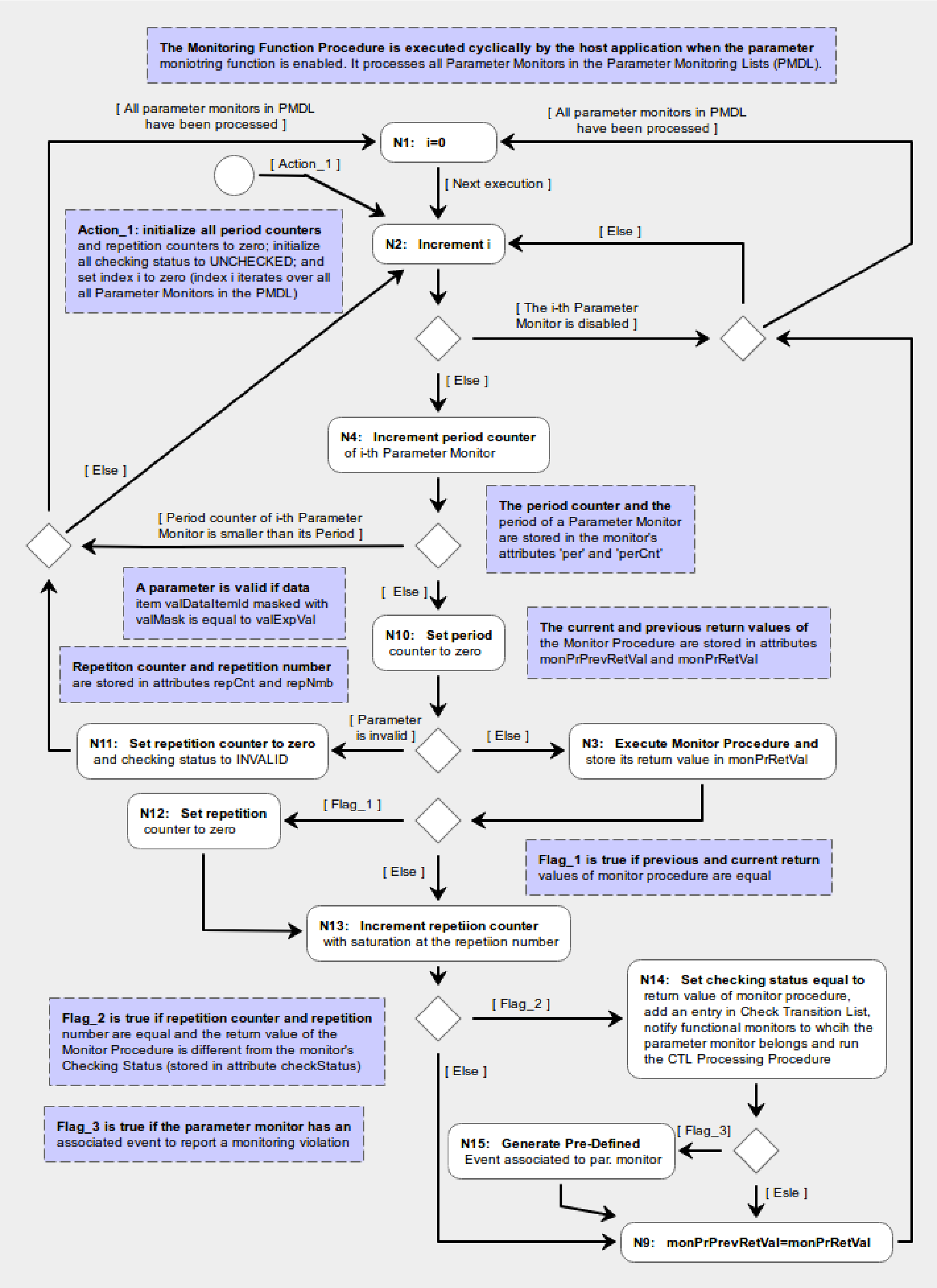

The Monitoring Function Procedure is executed cyclically by the host application when the parameter moniotring function is enabled. It processes all Parameter Monitors in the Parameter Monitoring Lists (PMDL).
[All parameter monitors in PMDL have been processed]
N1: i=0
[All parameter monitors in PMDL have been processed]
[Action_1]
[Next execution]
Action_1: initialize all period counters and repetition counters to zero; initialize all checking status to UNCHECKED; and set index i to zero (index i iterates over all all Parameter Monitors in the PMDL)
N2: Increment i
[Else]
[The i-th Parameter Monitor is disabled]
[Else]
N4: Increment period counter of i-th Parameter Monitor
[Else]
[Period counter of i-th Parameter Monitor is smaller than its Period]
The period counter and the period of a Parameter Monitor are stored in the monitor's attributes 'per' and 'perCnt'
A parameter is valid if data item valDataItemId masked with valMask is equal to valExpVal
[Else]
N10: Set period counter to zero
The current and previous return values of the Monitor Procedure are stored in attributes monPrPrevRetVal and monPrRetVal
Repetiton counter and repetition number are stored in attributes repCnt and repNmb
[Parameter is invalid]
[Else]
N11: Set repetition counter to zero and checking status to INVALID
N3: Execute Monitor Procedure and store its return value in monPrRetVal
N12: Set repetition counter to zero
[Flag_1]
[Else]
Flag_1 is true if previous and current return values of monitor procedure are equal
N13: Increment repetiion counter with saturation at the repetiion number
Flag_2 is true if repetition counter and repetition number are equal and the return value of the Monitor Procedure is different from the monitor's Checking Status (stored in attribute checkStatus)
[Flag_2]
[Else]
N14: Set checking status equal to return value of monitor procedure, add an entry in Check Transition List, notify functional monitors to whcih the parameter monitor belongs and run the CTL Processing Procedure
Flag_3 is true if the parameter monitor has an associated event to report a monitoring violation
N15: Generate Pre-Defined Event associated to par. monitor
[Flag_3]
[Esle]
N9: monPrPrevRetVal=monPrRetVal

Fig. 12.1: Parameter Monitoring Procedure

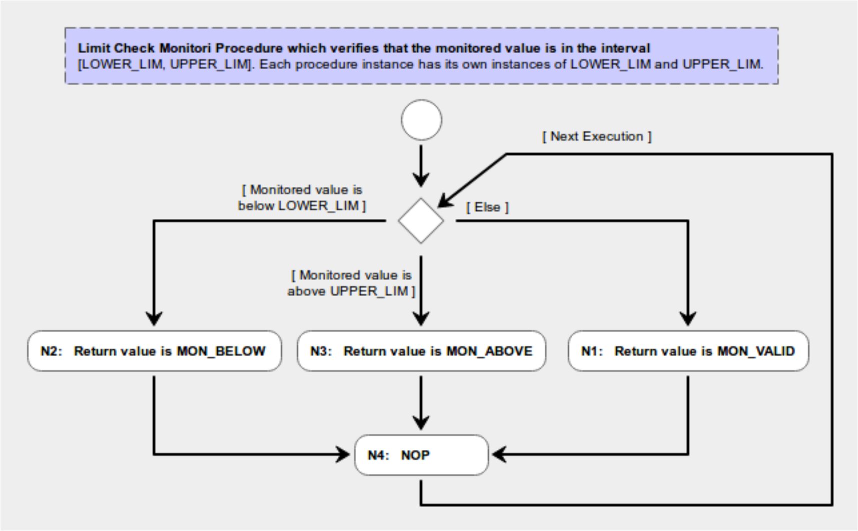

Limit Check Monitori Procedure which verifies that the monitored value is in the interval [LOWER_LIM, UPPER_LIM]. Each procedure instance has its own instances of LOWER_LIM and UPPER_LIM.
[Next Execution]
[Monitored value is below LOWER_LIM]
[Else]
[Monitored value is above UPPER_LIM]
N2: Return value is MON_BELOW
N3: Return value is MON_ABOVE
N1: Return value is MON_VALID
N4: NOP

Fig. 12.2: Limit Check Monitor Procedure

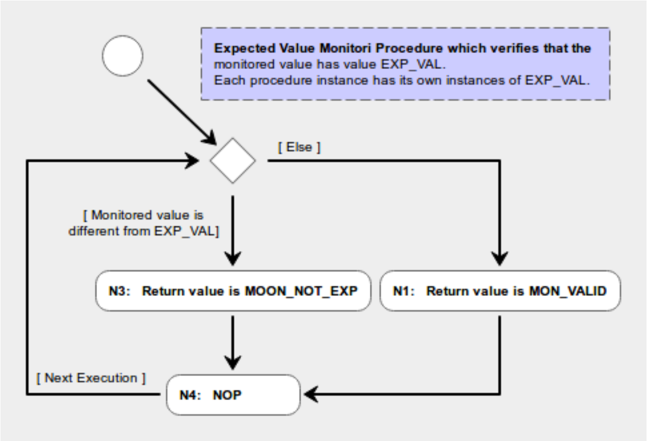

Expected Value Monitori Procedure which verifies that the monitored value has value EXP_VAL.
Each procedure instance has its own instances of EXP_VAL.
[Else]
[Monitored value is different from EXP_VAL]
N3: Return value is MOON_NOT_EXP
N1: Return value is MON_VALID
[Next Execution]
N4: NOP

Fig. 12.3: Expected Value Monitor Procedure

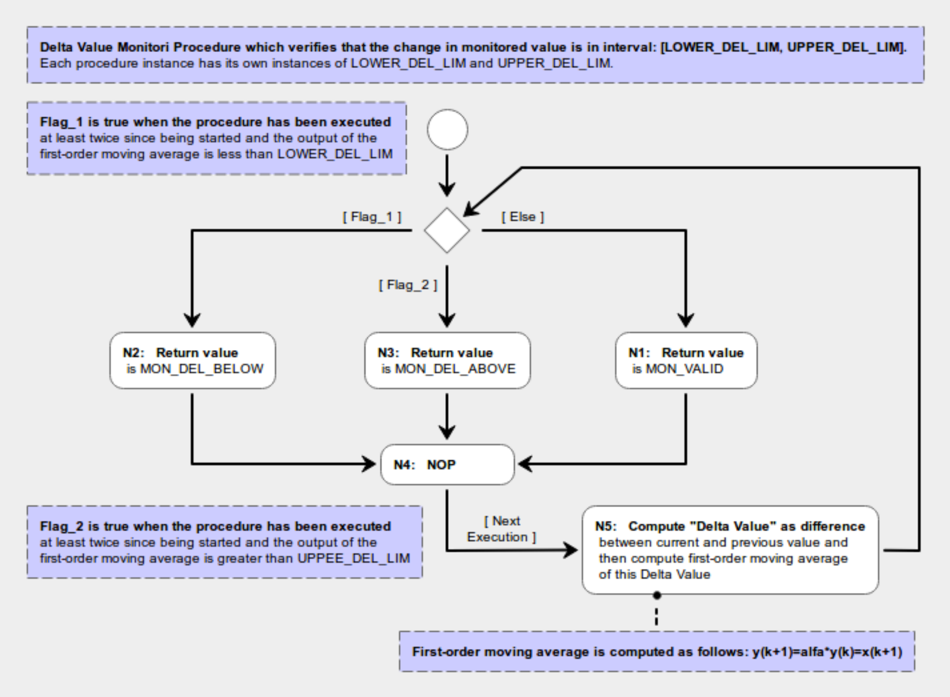

Delta Value Monitori Procedure which verifies that the change in monitored value is in interval: [LOWER_DEL_LIM, UPPER_DEL_LIM].
Each procedure instance has its own instances of LOWER_DEL_LIM and UPPER_DEL_LIM.
Flag_1 is true when the procedure has been executed at least twice since being started and the output of the first-order moving average is less than LOWER_DEL_LIM
[Flag_1]
[Else]
[Flag_2]
N2: Return value is MON_DEL_BELOW
N3: Return value is MON_DEL_ABOVE
N1: Return value is MON_VALID
N4: NOP
Flag_2 is true when the procedure has been executed at least twice since being started and the output of the first-order moving average is greater than UPPEE_DEL_LIM
[Next Execution]
N5: Compute "Delta Value" as difference between current and previous value and then compute first-order moving average of this Delta Value
First-order moving average is computed as follows: y(k+1)=alfa*y(k)=x(k+1)

Fig. 12.4: Delta Value Monitor Procedure

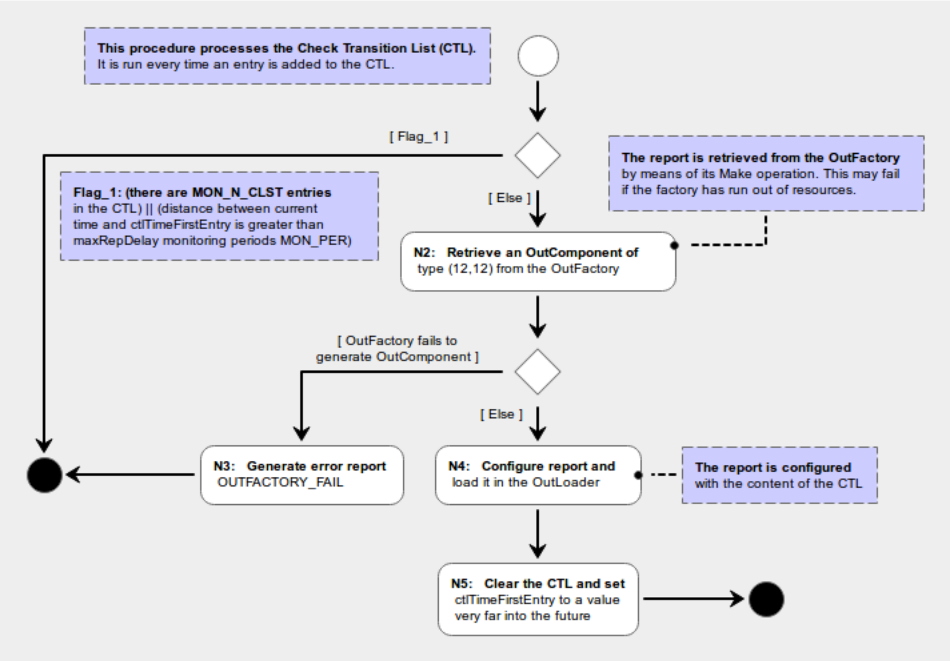

This procedure processes the Check Transition List (CTL).
It is run every time an entry is added to the CTL.
[Flag_1]
The report is retrieved from the OutFactory by means of its Make operation. This may fail if the factory has run out of resources.
Flag_1: (there are MON_N_CLST entries in the CTL) || (distance between current time and ctlTimeFirstEntry is greater than maxRepDelay monitoring periods MON_PER)
[Else]
N2: Retrieve an OutComponent of type (12,12) from the OutFactory
[OutFactory fails to generate OutComponent]
[Else]
N3: Generate error report OUTFACTORY_FAIL
N4: Configure report and load it in the OutLoader
The report is configured with the content of the CTL
N5: Clear the CTL and set ctlTimeFirstEntry to a value very far into the future

Fig. 12.5: CTL Processing Procedure

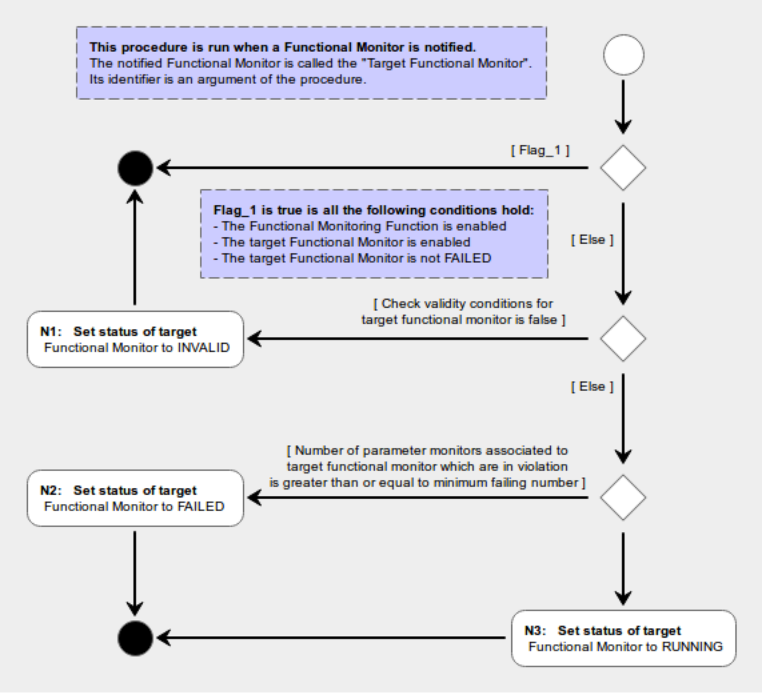
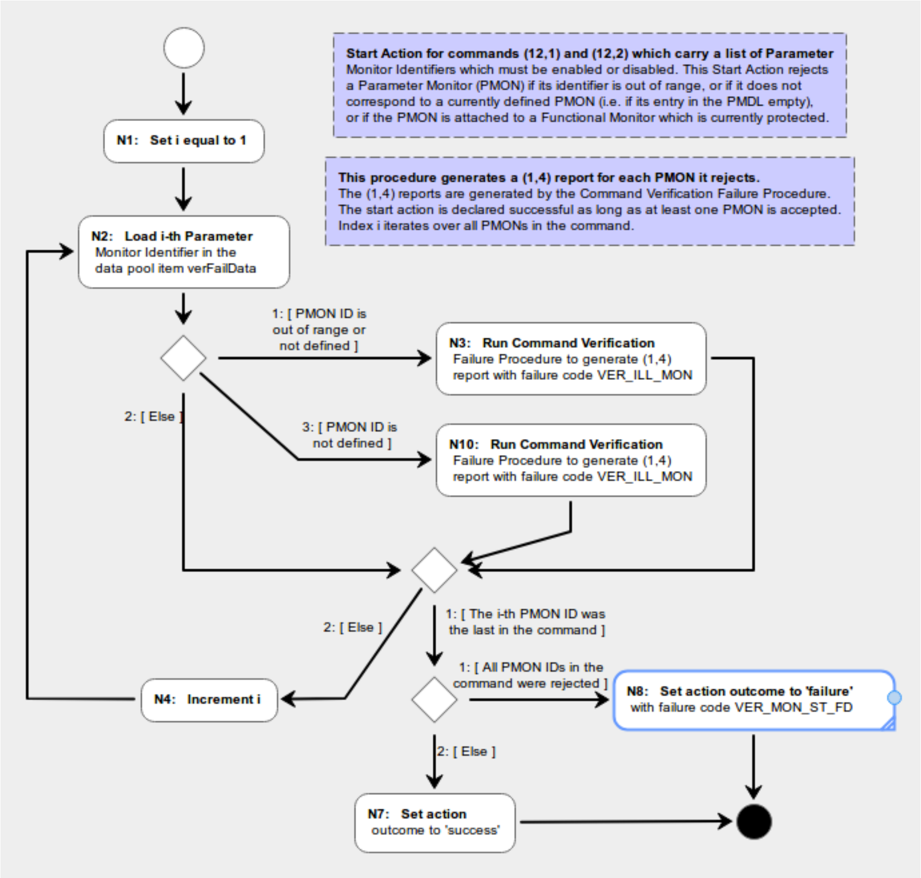

This procedure is run when a Functional Monitor is notified.
The notified Functional Monitor is called the "Target Functional Monitor".
Its identifier is an argument of the procedure.
[Flag_1]
Flag_1 is true is all the following conditions hold:
- The Functional Monitoring Function is enabled
- The target Functional Monitor is enabled
- The target Functional Monitor is not FAILED
[Else]
[Check validity conditions for target functional monitor is false]
N1: Set status of target Functional Monitor to INVALID
[Else]
[Number of parameter monitors associated to target functional monitor which are in violation is greater than or equal to minimum failing number]
N2: Set status of target Functional Monitor to FAILED
N3: Set status of target Functional Monitor to RUNNING

Fig. 12.6: Functional Monitor Notification Procedure



Start Action for commands (12,1) and (12,2) which carry a list of Parameter Monitor Identifiers which must be enabled or disabled. This Start Action rejects a Parameter Monitor (PMON) if its identifier is out of range, or if it does not correspond to a currently defined PMON (i.e. if its entry in the PMDL empty), or if the PMON is attached to a Functional Monitor which is currently protected.

This procedure generates a (1,4) report for each PMON it rejects. The (1,4) reports are generated by the Command Verification Failure Procedure. The start action is declared successful as long as at least one PMON is accepted. Index i iterates over all PMONs in the command.

N1: Set i equal to 1
N2: Load i-th Parameter Monitor Identifier in the data pool item verFailData
1: [PMON ID is out of range or not defined]
N3: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_MON
2: [Else]
3: [PMON ID is not defined]
N10: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_MON
2: [Else]
1: [The i-th PMON ID was the last in the command]
1: [All PMON IDs in the command were rejected]
N8: Set action outcome to 'failure' with failure code VER_MON_ST_FD
N4: Increment i
2: [Else]
N7: Set action outcome to 'success'

Fig. 12.7: Start Action of (12,1) and (12,2) Command Procedure

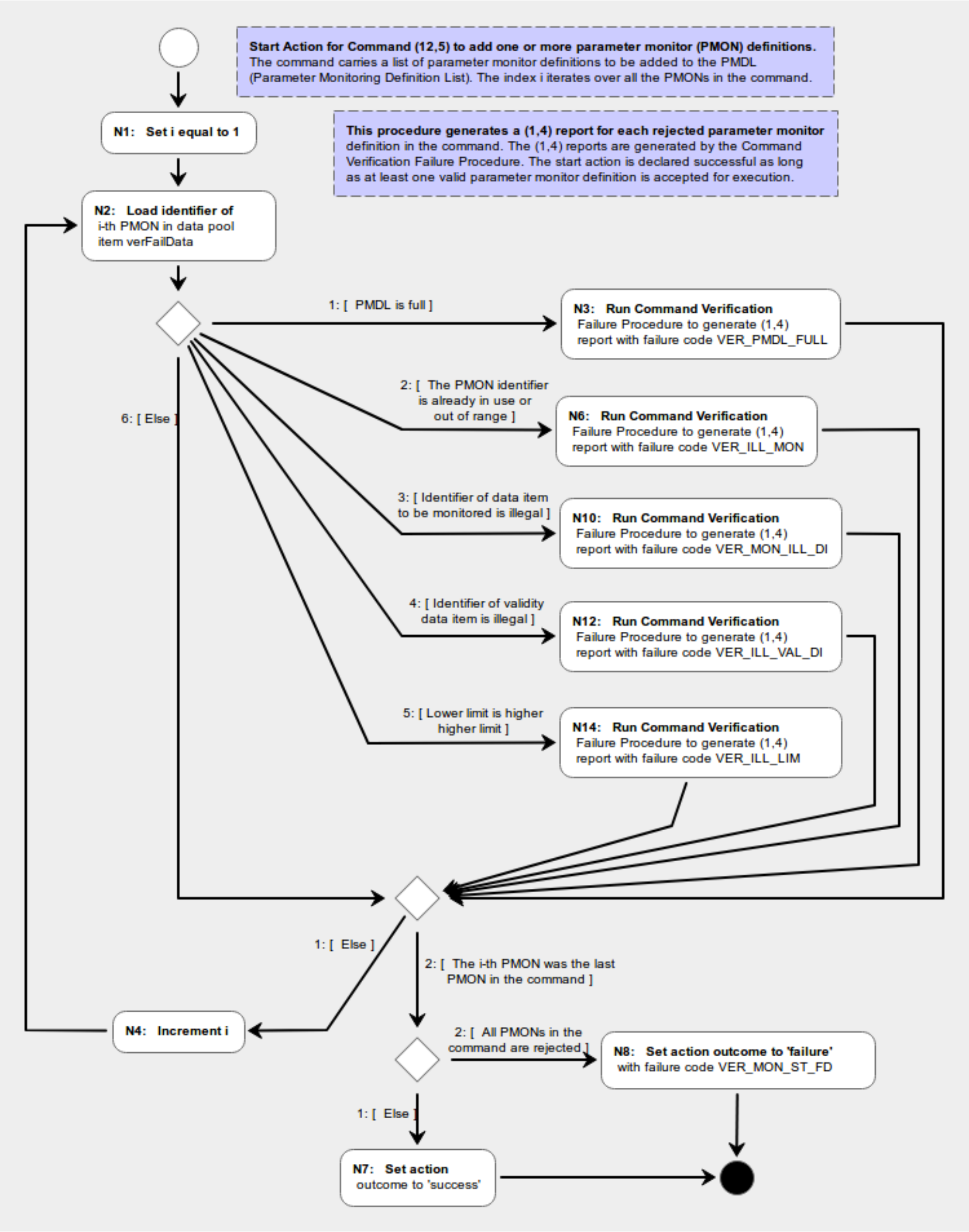

Start Action for Command (12,5) to add one or more parameter monitor (PMON) definitions. The command carries a list of parameter monitor definitions to be added to the PMDL (Parameter Monitoring Definition List). The index i iterates over all the PMONs in the command.
This procedure generates a (1,4) report for each rejected parameter monitor definition in the command. The (1,4) reports are generated by the Command Verification Failure Procedure. The start action is declared successful as long as at least one valid parameter monitor definition is accepted for execution.
N1: Set i equal to 1
N2: Load identifier of i-th PMON in data pool item verFailData
1: [PMDL is full]
N3: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_PMDL_FULL
2: [The PMON identifier is already in use or out of range]
N6: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_MON
6: [Else]
3: [Identifier of data item to be monitored is illegal]
N10: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_MON_ILL_DI
4: [Identifier of validity data item is illegal]
N12: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_VAL_DI
5: [Lower limit is higher higher limit]
N14: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_LIM
1: [Else]
2: [The i-th PMON was the last PMON in the command]
N4: Increment i
2: [All PMONs in the command are rejected]
N8: Set action outcome to 'failure' with failure code VER_MON_ST_FD
1: [Else]
N7: Set action outcome to 'success'

Fig. 12.8: Start Action of (12,5) Command Procedure

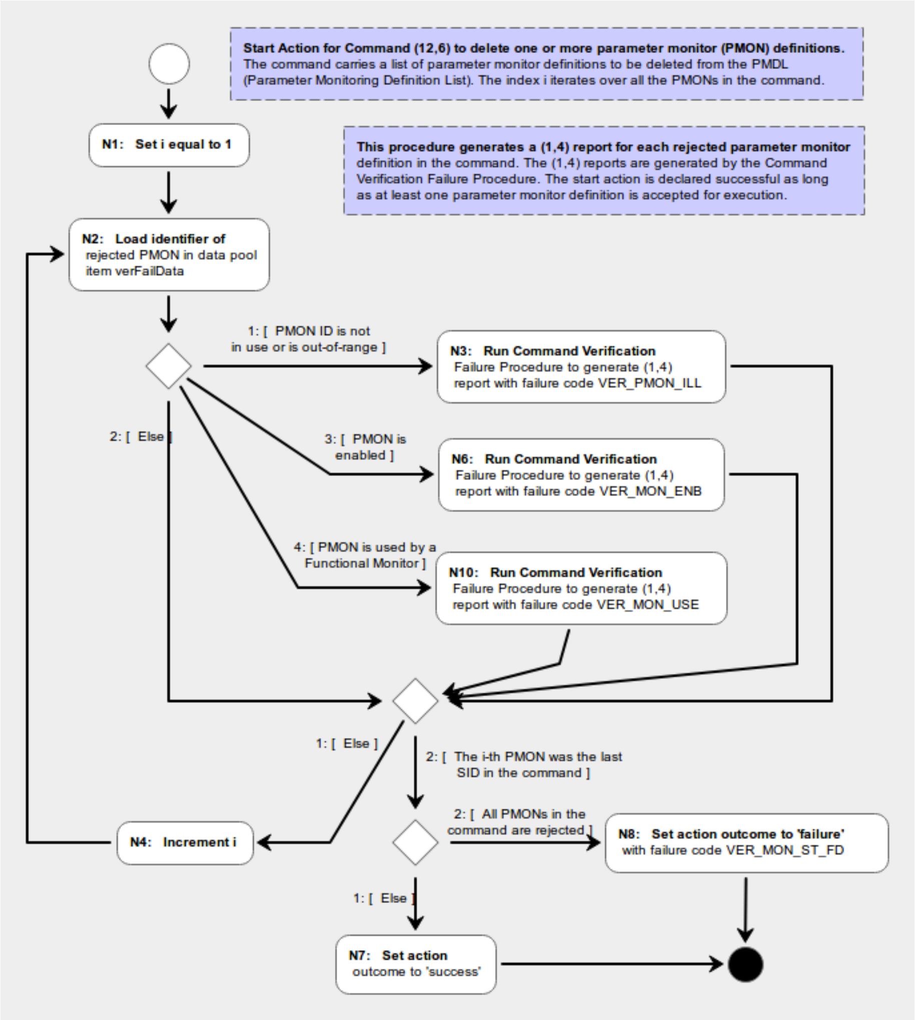

Start Action for Command (12,6) to delete one or more parameter monitor (PMON) definitions. The command carries a list of parameter monitor definitions to be deleted from the PMDL (Parameter Monitoring Definition List). The index i iterates over all the PMONs in the command.
This procedure generates a (1,4) report for each rejected parameter monitor definition in the command. The (1,4) reports are generated by the Command Verification Failure Procedure. The start action is declared successful as long as at least one parameter monitor definition is accepted for execution.
N1: Set i equal to 1
N2: Load identifier of rejected PMON in data pool item verFailData
1: [PMON ID is not in use or is out-of-range]
N3: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_PMON_ILL
2: [Else]
3: [PMON is enabled]
N6: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_MON_ENB
4: [PMON is used by a Functional Monitor]
N10: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_MON_USE
1: [Else]
2: [The i-th PMON was the last SID in the command]
N4: Increment i
2: [All PMONs in the command are rejected]
N8: Set action outcome to 'failure' with failure code VER_MON_ST_FD
1: [Else]
N7: Set action outcome to 'success'

Fig. 12.9: Start Action of (12,6) Command Procedure

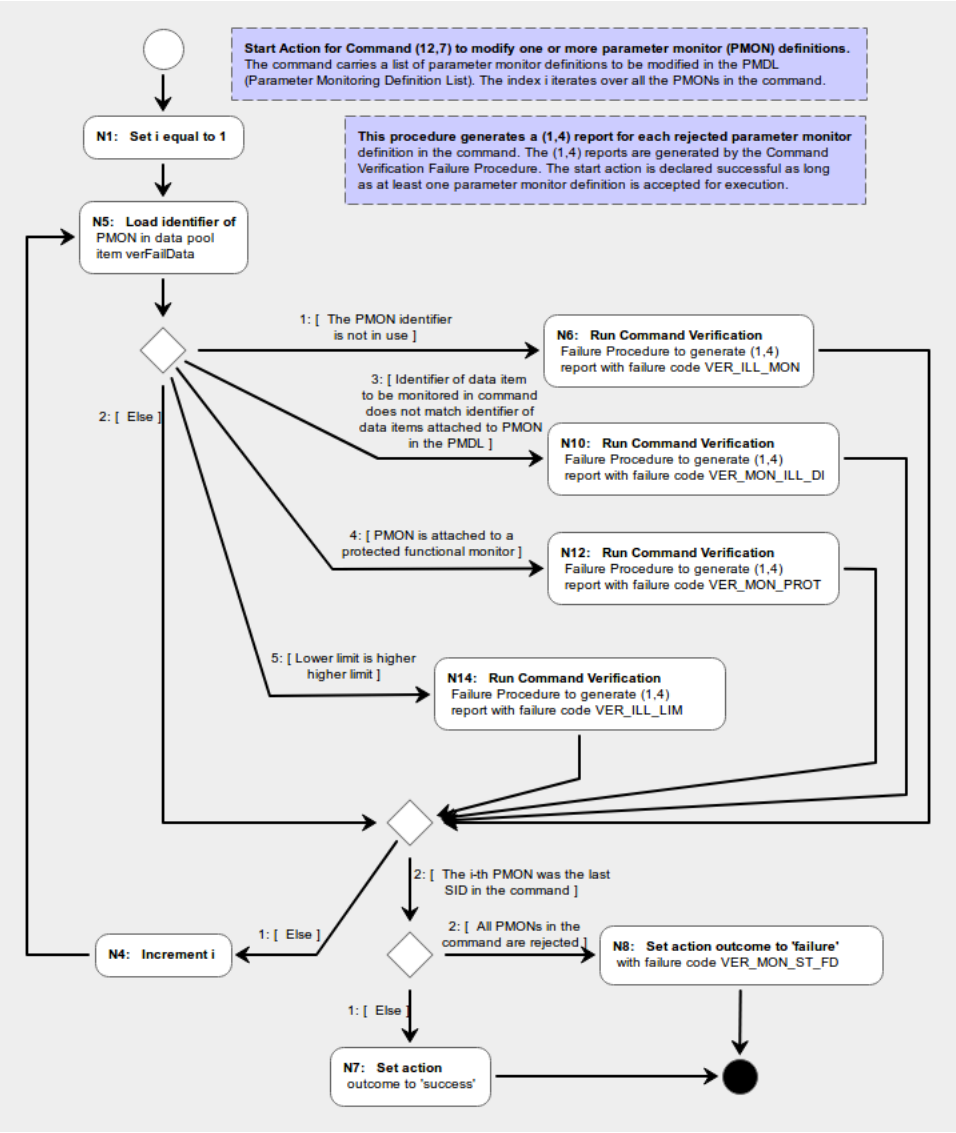

Start Action for Command (12,7) to modify one or more parameter monitor (PMON) definitions.
The command carries a list of parameter monitor definitions to be modified in the PMDL (Parameter Monitoring Definition List). The index i iterates over all the PMONs in the command.
This procedure generates a (1,4) report for each rejected parameter monitor definition in the command. The (1,4) reports are generated by the Command Verification Failure Procedure. The start action is declared successful as long as at least one parameter monitor definition is accepted for execution.
N1: Set i equal to 1
N5: Load identifier of PMON in data pool item verFailData
1: [The PMON identifier is not in use]
N6: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_MON
3: [Identifier of data item to be monitored in command does not match identifier of data items attached to PMON in the PMDL]
N10: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_MON_ILL_DI
2: [Else]
4: [PMON is attached to a protected functional monitor]
N12: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_MON_PROT
5: [Lower limit is higher higher limit]
N14: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_LIM
2: [The i-th PMON was the last SID in the command]
1: [Else]
N4: Increment i
2: [All PMONs in the command are rejected]
N8: Set action outcome to 'failure' with failure code VER_MON_ST_FD
1: [Else]
N7: Set action outcome to 'success'

Fig. 12.10: Start Action of (12,7) Command Procedure

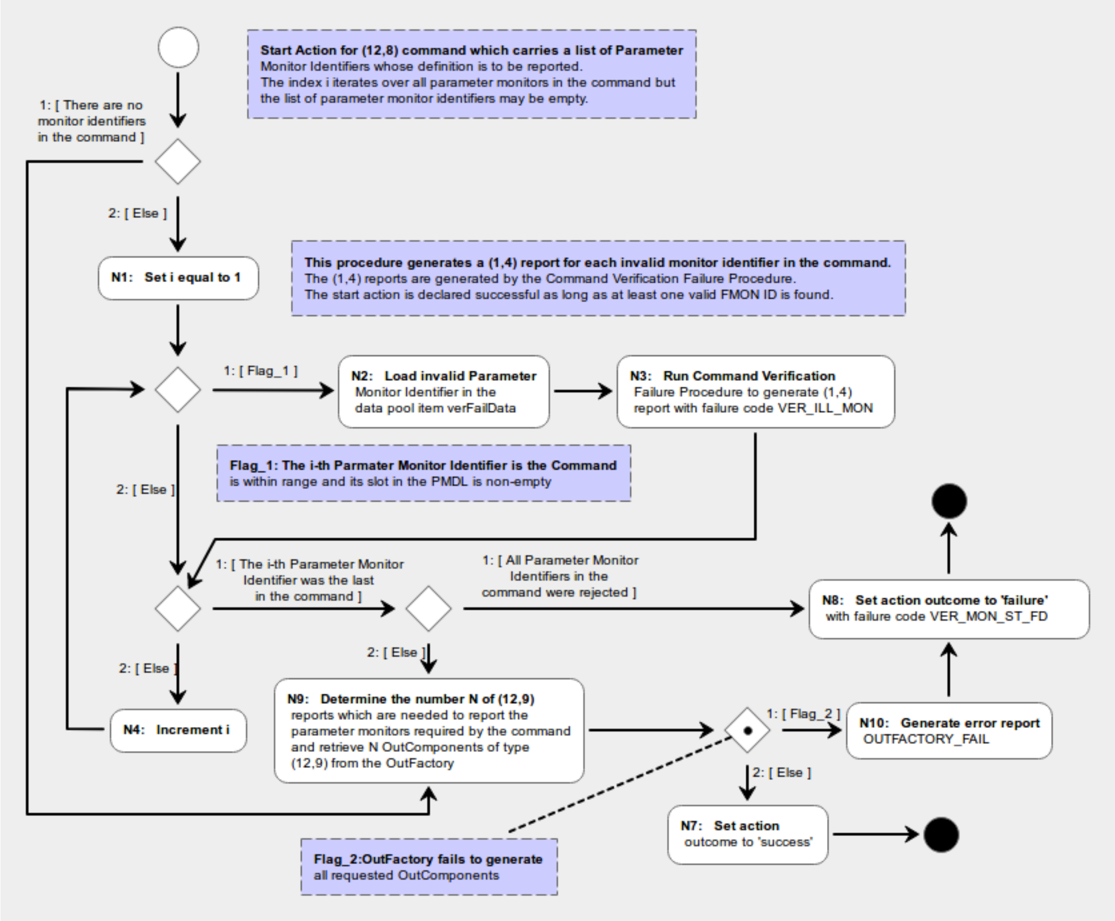

Start Action for (12,8) command which carries a list of Parameter Monitor Identifiers whose definition is to be reported.
The index i iterates over all parameter monitors in the command but the list of parameter monitor identifiers may be empty.
1: [There are no monitor identifiers in the command]
2: [Else]
N1: Set i equal to 1
This procedure generates a (1,4) report for each invalid monitor identifier in the command.
The (1,4) reports are generated by the Command Verification Failure Procedure.
The start action is declared successful as long as at least one valid FMON ID is found.
1: [Flag_1]
N2: Load invalid Parameter Monitor Identifier in the data pool item verFailData
N3: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_MON
Flag_1: The i-th Parmater Monitor Identifier is the Command is within range and its slot in the PMDL is non-empty
2: [Else]
1: [The i-th Parameter Monitor Identifier was the last in the command]
1: [All Parameter Monitor Identifiers in the command were rejected]
N8: Set action outcome to 'failure' with failure code VER_MON_ST_FD
2: [Else]
2: [Else]
N4: Increment i
N9: Determine the number N of (12,9) reports which are needed to report the parameter monitors required by the command and retrieve N OutComponents of type (12,9) from the OutFactory
1: [Flag_2]
N10: Generate error report OUTFACTORY_FAIL
2: [Else]
N7: Set action outcome to 'success'
Flag_2:OutFactory fails to generate all requested OutComponents

Fig. 12.11: Start Action of (12,8) Command Procedure

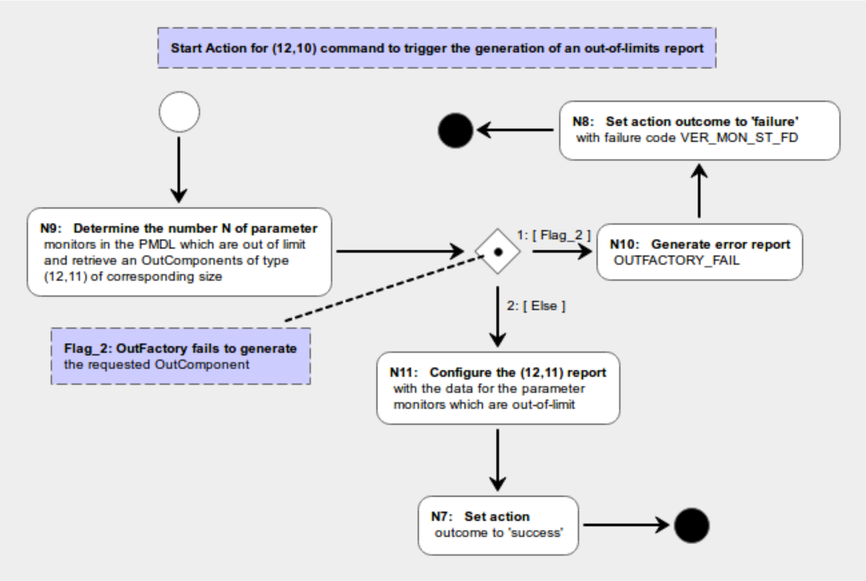
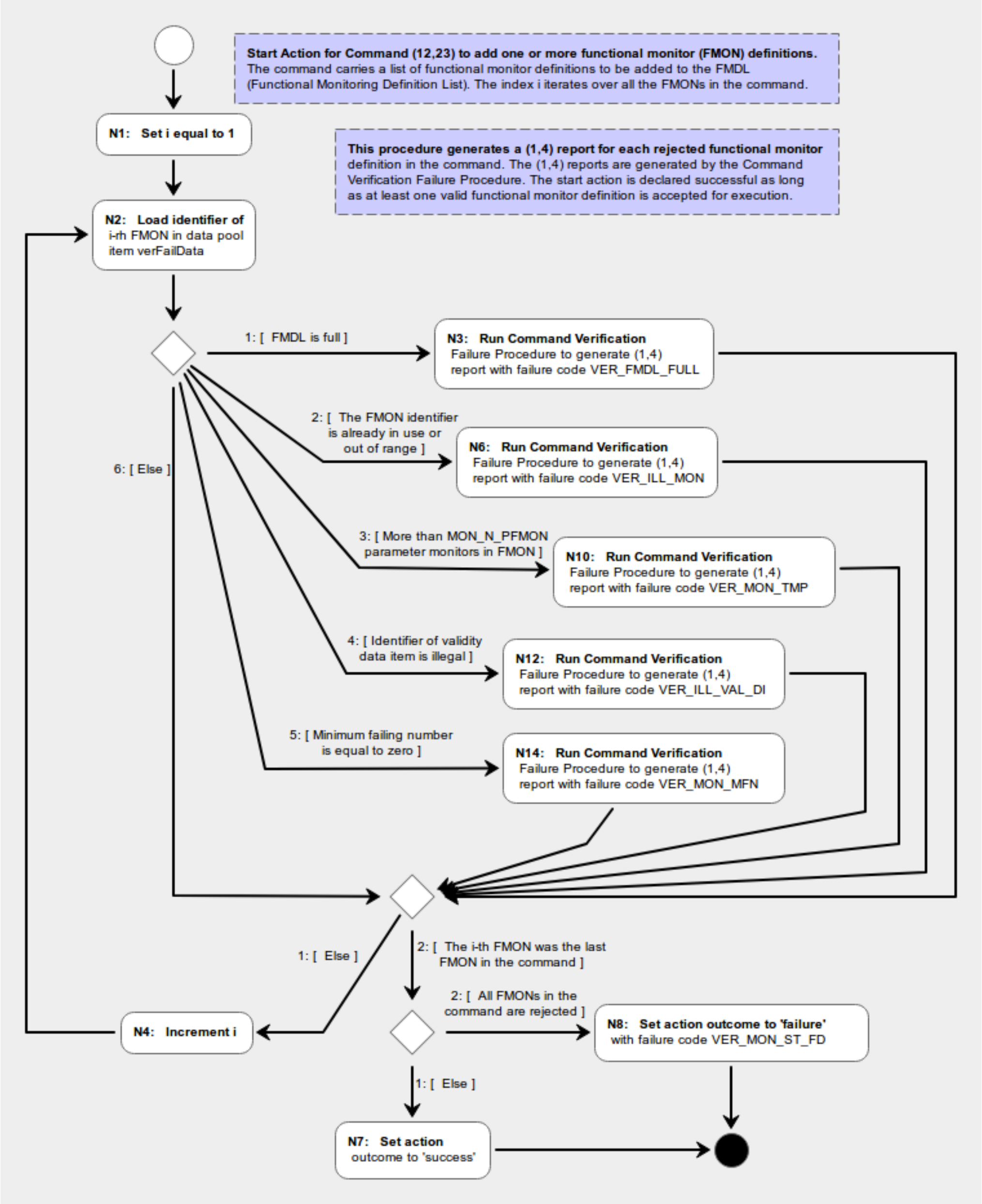

Start Action for (12,10) command to trigger the generation of an out-of-limits report
N8: Set action outcome to 'failure' with failure code VER_MON_ST_FD
N9: Determine the number N of parameter monitors in the PMDL which are out of limit and retrieve an OutComponents of type (12,11) of corresponding size
1: [Flag_2]
N10: Generate error report OUTFACTORY_FAIL
2: [Else]
Flag_2: OutFactory fails to generate the requested OutComponent
N11: Configure the (12,11) report with the data for the parameter monitors which are out-of-limit
N7: Set action outcome to 'success'

Fig. 12.12: Start Action of (12,10) Command Procedure


Start Action for Command (12,23) to add one or more functional monitor (FMON) definitions. The command carries a list of functional monitor definitions to be added to the FMDL (Functional Monitoring Definition List). The index i iterates over all the FMONs in the command.

This procedure generates a (1,4) report for each rejected functional monitor definition in the command. The (1,4) reports are generated by the Command Verification Failure Procedure. The start action is declared successful as long as at least one valid functional monitor definition is accepted for execution.

N1: Set i equal to 1
N2: Load identifier of i-th FMON in data pool item verFailData
1: [FMDL is full]
N3: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_FMDL_FULL
2: [The FMON identifier is already in use or out of range]
N6: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_MON
6: [Else]
3: [More than MON_N_PFMON parameter monitors in FMON]
N10: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_MON_TMP
4: [Identifier of validity data item is illegal]
N12: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_VAL_DI
5: [Minimum failing number is equal to zero]
N14: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_MON_MFN
1: [Else]
2: [The i-th FMON was the last FMON in the command]
2: [All FMONs in the command are rejected]
N4: Increment i
N8: Set action outcome to 'failure' with failure code VER_MON_ST_FD
1: [Else]
N7: Set action outcome to 'success'

Fig. 12.13: Start Action of (12,23) Command Procedure

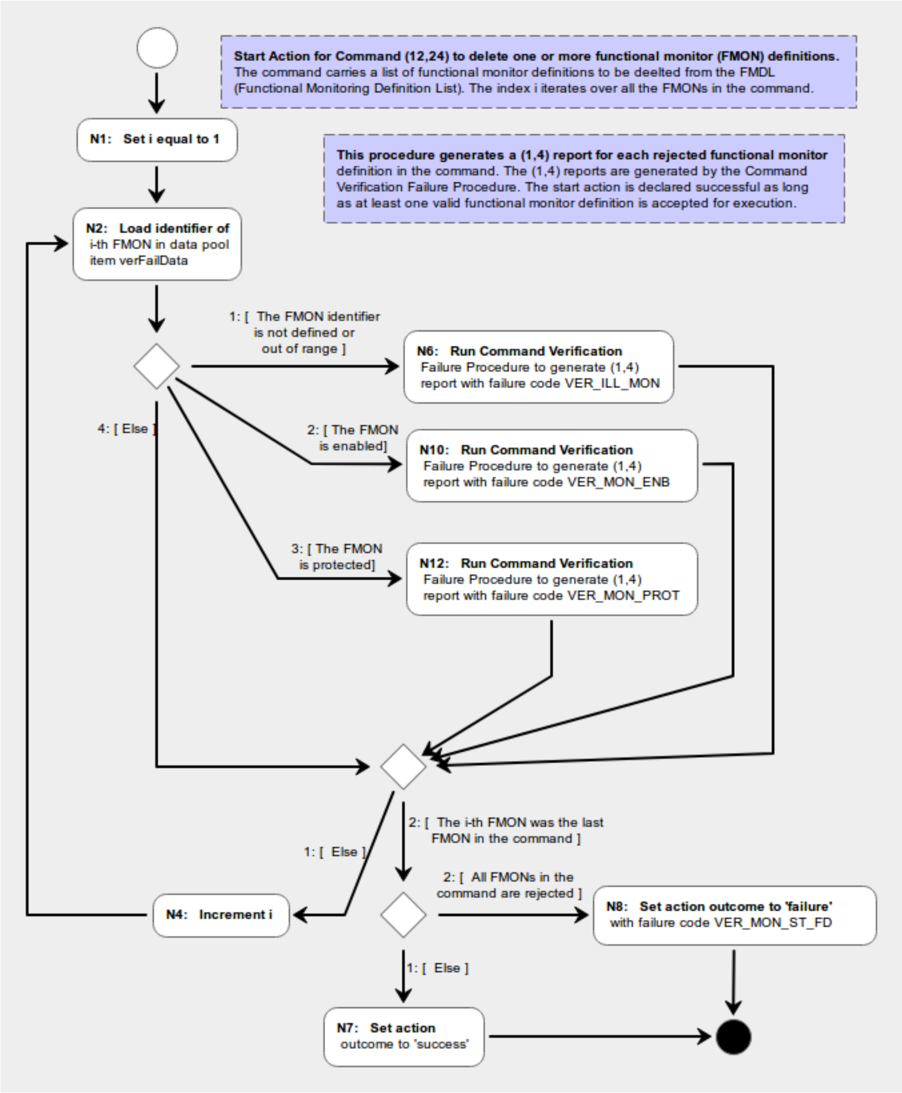

Start Action for Command (12,24) to delete one or more functional monitor (FMON) definitions.
The command carries a list of functional monitor definitions to be deelted from the FMDL (Functional Monitoring Definition List). The index i iterates over all the FMONs in the command.
N1: Set i equal to 1
This procedure generates a (1,4) report for each rejected functional monitor definition in the command. The (1,4) reports are generated by the Command Verification Failure Procedure. The start action is declared successful as long as at least one valid functional monitor definition is accepted for execution.
N2: Load identifier of i-th FMON in data pool item verFailData
1: [The FMON identifier is not defined or out of range]
N6: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_MON
4: [Else]
2: [The FMON is enabled]
N10: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_MON_ENB
3: [The FMON is protected]
N12: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_MON_PROT
2: [The i-th FMON was the last FMON in the command]
1: [Else]
2: [All FMONs in the command are rejected]
N4: Increment i
N8: Set action outcome to 'failure' with failure code VER_MON_ST_FD
1: [Else]
N7: Set action outcome to 'success'

Fig. 12.14: Start Action of (12,24) Command Procedure

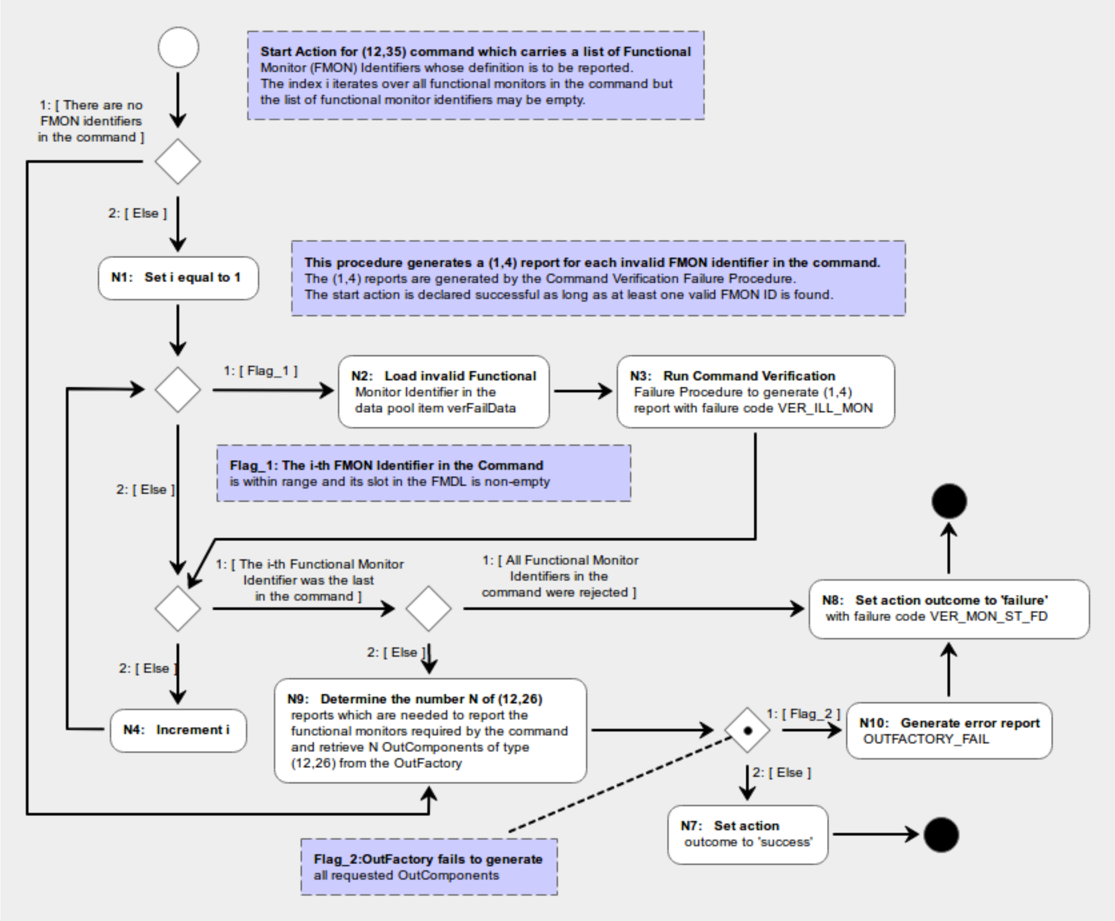

Start Action for (12,35) command which carries a list of Functional Monitor (FMON) Identifiers whose definition is to be reported.
The index i iterates over all functional monitors in the command but the list of functional monitor identifiers may be empty.
1: [There are no FMON identifiers in the command]
2: [Else]
N1: Set i equal to 1
This procedure generates a (1,4) report for each invalid FMON identifier in the command.
The (1,4) reports are generated by the Command Verification Failure Procedure.
The start action is declared successful as long as at least one valid FMON ID is found.
1: [Flag_1]
N2: Load invalid Functional Monitor Identifier in the data pool item verFailData
N3: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_MON
2: [Else]
Flag_1: The i-th FMON Identifier in the Command is within range and its slot in the FMDL is non-empty
1: [The i-th Functional Monitor Identifier was the last in the command]
1: [All Functional Monitor Identifiers in the command were rejected]
N8: Set action outcome to 'failure' with failure code VER_MON_ST_FD
2: [Else]
2: [Else]
N4: Increment i
N9: Determine the number N of (12,26) reports which are needed to report the functional monitors required by the command and retrieve N OutComponents of type (12,26) from the OutFactory
1: [Flag_2]
N10: Generate error report OUTFACTORY_FAIL
2: [Else]
N7: Set action outcome to 'success'
Flag_2:OutFactory fails to generate all requested OutComponents

Fig. 12.15: Start Action of (12,25) Command Procedure

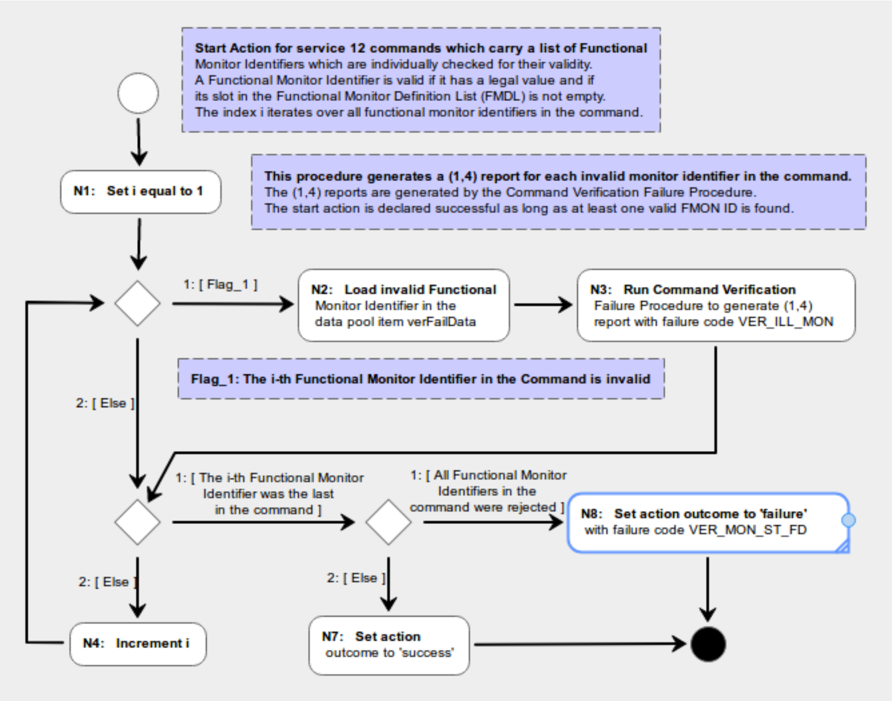

Start Action for service 12 commands which carry a list of Functional Monitor Identifiers which are individually checked for their validity.
A Functional Monitor Identifier is valid if it has a legal value and if its slot in the Functional Monitor Definition List (FMDL) is not empty.
The index i iterates over all functional monitor identifiers in the command.
N1: Set i equal to 1
This procedure generates a (1,4) report for each invalid monitor identifier in the command.
The (1,4) reports are generated by the Command Verification Failure Procedure.
The start action is declared successful as long as at least one valid FMON ID is found.
1: [Flag_1]
N2: Load invalid Functional Monitor Identifier in the data pool item verFailData
N3: Run Command Verification Failure Procedure to generate (1,4) report with failure code VER_ILL_MON
Flag_1: The i-th Functional Monitor Identifier in the Command is invalid
2: [Else]
1: [The i-th Functional Monitor Identifier was the last in the command]
1: [All Functional Monitor Identifiers in the command were rejected]
N8: Set action outcome to 'failure' with failure code VER_MON_ST_FD
2: [Else]
2: [Else]
N4: Increment i
N7: Set action outcome to 'success'

Fig. 12.16: Start Action of Multi-Functional Monitor Command Procedure

13 Large Packet Transfer Service

The service type of the Large Packet Transfer Service is 13. The PUS Extension of the CORDET Framework supports this service in full and extends the packet down-link sub-service with two private commands.

The Large Packet Transfer Service provides the capabilities to perform a *down-transfer* (namely a transfer of large packet from the service provider to the service user) and an *up-transfer* (namely a transfer of a large packet from the service user to the service provider).

The service is built around the concepts of *Large Packet Transfer Buffer* or LPT Buffer and *Large Packet Transfer State Machine* or LPT State Machine.

13.1 LPT Buffers

In the case of down-transfers, the LPT Buffer is the memory area within the host application where the out-going large packet is stored. The host application is responsible for loading the data to be down-transferred into this area and service 13 is responsible for splitting the data in this area into reports which are sent to their destination in sequence.

Similarly, in the case of up-transfers, the LPT Buffer is the memory area to which the incoming large packet is stored. Service 13 is responsible for processing the sequence of incoming commands which carry the large packet and for storing their content into the LPT Buffer. The host application is responsible for collecting the large packet from the LPT Buffer.

The number of LPT Buffers in an application is statically defined and is equalt to LPT_-N_BUF. Each LPT Buffer has an identifier which is an integer in the range 0 to (LPT_-N_BUF-1). To each LPT Buffer, the following attributes are associated:

- `lptDest`: the destination of the down-transfer originating from the LPT Buffer (only meaningful for LPT Buffers which act as sources of a down-transfer)
- `lptSrc`: the source of the up-transfer in the LPT Buffer (only meaningful for LPT Buffers which act as destinations of an up-transfer)
- `lptSize`: the size of the large packet in the LPT Buffer, namely the amount of data to be down-transferred (for LPT Buffers which act as sources of a down-transfer) or the amount of up-transferred data (for LPT Buffers which act as destinations of an up-transfer)
- `lptRemSize`: the amount of data still to be down-transferred in the currently on-going down-transfer from the LPT Buffer (only meaningful for LPT Buffers which act as sources of a down-transfer)
- `partSize`: the part size for the up- or down-transfer, namely the size of transfer data in a single service 13 report (down-transfer) or in a single service 13 command (up-transfer)
- `lptTimeOut`: the time-out for service 13 commands (only meaningful for LPT Buffers which act as targets for an up-transfer)
- `lptTime`: the time when the last service 13 up-transfer command has been received (only meaningful for LPT Buffers which act as targets for an up-transfer)
- `largeMsgTransId`: the identifier of the large packet transfer which has the LPT Buffer as source (down-transfer) or as destination (up-transfer)

- partSeqNmb: the part sequence number for the currently on-going down-transfer from the LPT Buffer or up-transfer to the LPT Buffer
- lptFailCode: the failure code for an up-transfer to the LPT Buffer which was aborted (only meaningful for LPT Buffers which act as destinations of an up-transfer).

Different LPT Buffers may have different values of lptDest or partSize. This allows the application designers to allocate LPT Buffers to different destinations (e.g. one LPT Buffer is used for large transfers to/from the ground and another LPT Buffer is used for large transfers to/from other destinations). Or, it allows them to use different LPT Buffers for different large packet sizes. Also, lptDest and partSize are data pool parameters (see section ??). Their values can therefore be adjusted by the service 13 user. It is the user responsibility to avoid changing the value of lptDest or partSize for a given LPT Buffer while a transfer to or from that buffer is under way.

In accordance with [PS-SP], each large packet transfer has a unique *Large Message Transaction Identifier*. This identifier is stored in variable largeMsgTransId. Its value is set as follows:

- At application initialization time, the value of largeMsgTransId for the i-th LPT Buffer is initialized to: i.
- When a down-transfer from an LPT Buffer is started, then the value of its Large Message Transaction Identifier is set to the value of the largeMsgTransId variable for the LPT Buffer
- When a down-transfer from an LPT Buffer is terminated, then the value of the buffer's largeMsgTransId variable is incremented by LPT_N_BUF.
- When an up-transfer to an LPT Buffer is started, then the value of the buffer's largeMsgTransId variable is set equal to the value of the transfer's Large Message Transaction Identifier
- When an up-transfer to an LPT Buffer is terminated, then the value of the buffer's largeMsgTransId variable is incremented by LPT_N_BUF.

To illustrate, suppose that the second LPT Buffer is used exclusively as a source for down-transfers and that there are 10 LPT Buffers (i.e. LPT_N_BUF is equal to 10). In this case, the first down-transfer from the LPT Buffer has a Large Message Transaction Identifier equal to 1; the second one has a Large Message Transfer Identifier equal to 11; the third one has a Large Message Transaction Identifier equal to 21; etc. The general idea is that, as long as an LPT Buffer is only used for down-transfers, then its Large Message Transaction Identifier can be used to identify the LPT Buffer from which the down-transfer originates because the LPT Buffer identifier is equal to: (largeMsgTransId MOD LPT_N_BUF).

A similar rule holds for up-transfers. Service 13 uses the Large Message Transaction Identifier of service 13 commands to decide to which LPT Buffer the transfer is to be directed. The identifier of the up-transfer is given by the Large Message Transaction Identifier modulus LPT_N_BUF.

Variable lptFailCode holds the reason for an abortion of an up-transfer. Three values are possible:

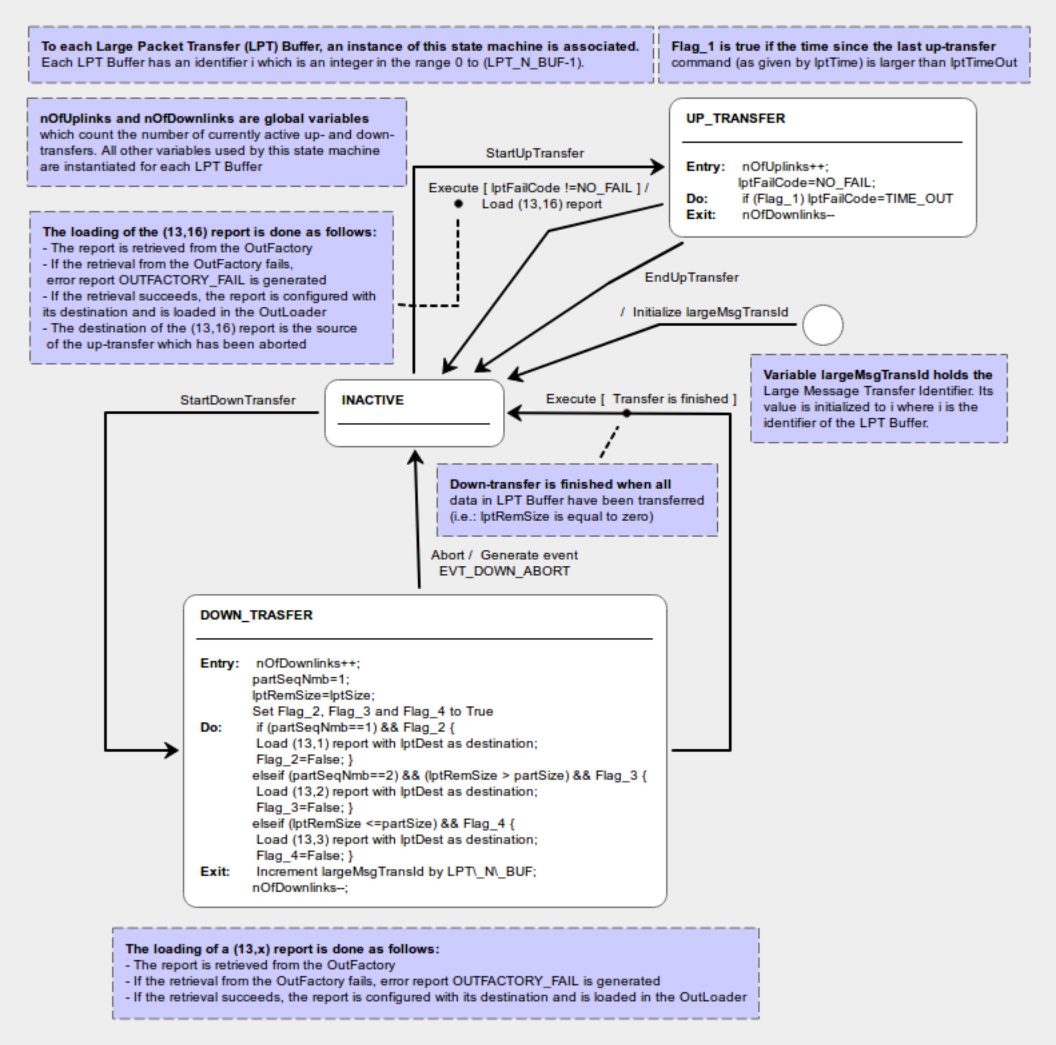
- NO_FAIL: default value in the absence of any failures
- PART_NMB_ERR: the first up-transfer command had a part sequence number different from 1 or a subsequent up-transfer command had a part sequence number which

was out-of-order.

- TIME_OUT: an up-transfer command has been received later than `lptTimeOut` since the previous up-transfer command

13.2 The LPT State Machine

The LPT State Machine controls a down- or up-transfer. Its diagram is shown in figure 13.1.



To each Large Packet Transfer (LPT) Buffer, an instance of this state machine is associated. Each LPT Buffer has an identifier i which is an integer in the range 0 to (LPT_N_BUF-1).

Flag_1 is true if the time since the last up-transfer command (as given by lptTime) is larger than lptTimeOut

nOfUplinks and nOfDownlinks are global variables which count the number of currently active up- and down-transfers. All other variables used by this state machine are instantiated for each LPT Buffer

The loading of the (13,16) report is done as follows:
- The report is retrieved from the OutFactory
- If the retrieval from the OutFactory fails, error report OUTFACTORY_FAIL is generated
- If the retrieval succeeds, the report is configured with its destination and is loaded in the OutLoader
- The destination of the (13,16) report is the source of the up-transfer which has been aborted

```
UP_TRANSFER

Entry:  nOfUplinks++;
        lptFailCode=NO_FAIL;
Do:     if (Flag_1) lptFailCode=TIME_OUT
Exit:   nOfDownlinks--
```

Variable largeMsgTransId holds the Large Message Transfer Identifier. Its value is initialized to i where i is the identifier of the LPT Buffer.

Down-transfer is finished when all data in LPT Buffer have been transferred (i.e.: lptRemSize is equal to zero)

```
DOWN_TRASFER

Entry:  nOfDownlinks++;
        partSeqNmb=1;
        lptRemSize=lptSize;
        Set Flag_2, Flag_3 and Flag_4 to True
Do:     if (partSeqNmb==1) && Flag_2 {
         Load (13,1) report with lptDest as destination;
         Flag_2=False; }
        elseif (partSeqNmb==2) && (lptRemSize > partSize) && Flag_3 {
         Load (13,2) report with lptDest as destination;
         Flag_3=False; }
        elseif (lptRemSize <=partSize) && Flag_4 {
         Load (13,3) report with lptDest as destination;
         Flag_4=False; }
Exit:   Increment largeMsgTransId by LPT\_N\_BUF;
        nOfDownlinks--;
```

The loading of a (13,x) report is done as follows:
- The report is retrieved from the OutFactory
- If the retrieval from the OutFactory fails, error report OUTFACTORY_FAIL is generated
- If the retrieval succeeds, the report is configured with its destination and is loaded in the OutLoader

Fig. 13.1: Large Packet Transfer (LPT) State Machine

To each LPT Buffer, one instance of the LPT State Machine is associated. When no transfer to or from the LPT Buffer is under way, the state machine is in state INACTIVE.

13.2.1 Management of Down-Transfers

A down-transfer is started by sending command *StartDownTransfer* to the LPT state machine. In response to this command, the state machine makes a transition to state DOWN_-

TRANSFER. The service 13 logic assumes that, at entry into this state, the LPT Buffer has been loaded with the large packet to be down-transferred and that the amount of data to be down-transferred are stored in variable `lptSize`.

The do-action of the state DOWN_TRANSFER is responsible for allocating and loading the down-transfer reports (13,1), (13,2) and (13,3).

The collection of the data from the LPT Buffer is done by the Update Action of the service 13 reports. The Update Action is also responsible for updating the `partSeqNmb` and the `lptRemSize` variables: every time a service 13 report is executed, `partSeqNmb` is incremented by 1 and `lptRemSize` is decremented by `partSize`. Hence, the normal flow of actions at the starts of a down-transfer (i.e. when the LPT State Machine enters state DOWN_TRANSFER) is as follows:

1. The LPT State Machine is executed and the do-action of state DOWN_TRANSFER creates the OutComponent encapsulating the (13,1) report and loads it in the OutLoader which in turn loads it in the OutManager
2. The OutManager is executed and this causes the OutComponent encapsulating the (13,1) report to be executed.
3. The Update Action of the OutComponent encapsulating the (13,1) report is executed and this causes the first part of down-transfer data to be collected from the LPT Buffer, `partSeqNmb` to be incremented by 1, and `lptRemSize` to be decremented by `partSize`.
4. The (13,1) report is handed over to the middleware for eventual transfer to its destination.
5. The (13,1) report is a one-shot report and it is therefore released after being executed once.

After the first part of the service 13 down-transfer has been processed, the intermediate parts are processed according to the following logic:

1. The LPT State Machine is executed and the do-action of state DOWN_TRANSFER creates the OutComponent encapsulating the (13,2) report and loads it in the OutLoader which in turn loads it in the OutManager
2. The OutManager is executed and this causes the OutComponent encapsulating the (13,2) report to be executed.
3. The Update Action of the OutComponent encapsulating the (13,2) report is executed and this causes the next part of down-transfer data to be collected from the LPT Buffer, `partSeqNmb` to be incremented by 1, and `lptRemSize` to be decremented by `partSize`.
4. The (13,2) report is handed over to the middleware for eventual transfer to its destination.
5. The (13,2) report is a repeat report which remains pending for as long as `lptRemSize` is greater than `partSize`. Hence, steps 2 to 5 are repeated multiple times until the last intermediate part of the down-transfer is processed.

When `lptRemSize` has become smaller than `partSize`, the last part of the down-transfer is processed according to the following logic:

1. The LPT State Machine is executed and the do-action of state DOWN_TRANSFER creates the OutComponent encapsulating the (13,3) report and loads it in the

OutLoader which in turn loads it in the OutManager

2. The OutManager is executed and this causes the OutComponent encapsulating the (13,3) report to be executed.
3. The Update Action of the OutComponent encapsulating the (13,3) report is executed and this causes the last part of data to be collected from the LPT Buffer and `lptRemSize` to be decremented by `partSize`.
4. The (13,3) report is handed over to the middleware for eventual transfer to its destination.
5. The (13,3) report is a one-shot report and it is therefore released after it is executed once.
6. The value of `lptRemSize` is now zero or negative and this causes the LPT State Machine to make a transition back to state INACTIVE. This marks the end of the down-transfer.

Command *StartDownTransfer* may either originate from the host application (if the host application autonomously decides to start a down-transfer) or it may originate from the private command (13,129). The latter command is provided by the framework to let the user trigger a down-transfer. It takes as an argument the identifier of the LPT Buffer (an integer in the range 1 to LPT_N_BUF) from which the down-transfer is to be started.

A down-transfer may be terminated prematurely with command `Abort`. This causes a transition from DOWN_TRANSFER to INACTIVE and the generation of event report EVT_DOWN_ABORT.

Command *Abort* may either originate from the host application (if the host application autonomously decides to abort a down-transfer) or it may originate from the private command (13,130). The latter command is provided by the framework to let the user abort an on-going down-transfer. The command takes as an argument the Large Message Transfer Identifier of the down-transfer.

Finally, the LPT State Machine is responsible for managing the nOfDownlinks variable which represents the number of currently on-going down-transfers.

13.2.2 Management of Up-Transfers

An up-transfer is started by sending command *StartUpTransfer* to the state machine. In response to this command, the state machine makes a transition to state UP_TRANSFER where it remains until either command `EndUpTransfer` brings the state machine back to state INACTIVE or the up-transfer is aborted.

Command `StartUpTransfer` originates from the (13,9) command which marks the start of an up-transfer. Command `EndUpTransfer` originates from the (13,11) command which marks the end of the up-transfer.

An up-transfer is aborted if variable `failCode` holding the failure code becomes set. At entry into state UP_TRANSFER, this variable is set to NO_FAIL (nominal value in the absence of failures). This value may change in two ways:

- If there is a time-out, namely if the time elapsed since the last up-transfer command exceeds `lptTimeOut`, or
- If the an up-transfer command is received with a part sequence number which is out-of-sequence.

The first condition is evaluated by the do-action of state UP_TRANSFER. This implies that the resolution of the time-out check is given by the period with which the LPT State Machine is executed. The second condition is evaluated by the start action of the (13,10) and (13,11) commands.

The Progress Action of the up-transfer commands is responsible for copying the data from the command to the LPT Buffer and for updating the variables associated to the LPT Buffer. After the up-transfer is terminated and the LPT State Machine is back in state INACTIVE, application is responsible for processing the data in the LPT Buffer. Note that there is no mechanism to prevent another up-transfer from being started while the application is still busy processing the data in the LPT Buffer. Avoinding this kind of conflicts is a user responsibility.

Finally, the LPT State Machine is responsible for managing the nOfUplinks variable which represents the number of currently on-going up-transfers.

13.3 Service 13 Report and Command Definition

Tables ?? to ?? formally specify the service 13 commands and reports by specifying how the actions, checks and attributes of generic out-going commands and reports are specialized for service 13 (see section 7).

Table 13.1: Specification of LptDownFirstRep Component

Name	LptDownFirstRep(13,1)
Description	Report carrying the first part of a down-transfer
Parameters	Large message transaction identifier, part sequence number and transfer data
Discriminant	None
Destination	The destination is loaded from parameter lptDest of the LPT Buffer holding the Large Packet to be transferred. This is determined as follows. If the down-transfer is autonomously started by the host application, then its destination is determined by the host application itself. If, instead, the down-transfer is triggered by a (13,129) command, then its destination is the same as the source of the (13,129) command.
Enable Check	Report is enabled if the LPT State Machine is in state DOWN_-TRANSFER
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	The following actions are performed: (a) Load the first part of the large packet from the LPT Buffer (b) Set the transaction identifier equal to largeMsgTransId (c) Set the part number equal to partSeqNmb (d) Increment partSeqNmb; and decrement lptRemSize by partSize (e) Set the action outcome to: 'completed'

Table 13.2: Specification of LptDownInterRep Component

Name	LptDownInterRep(13,2)
Description	Report carrying an intermediate part of a down-transfer
Parameters	Large message transaction identifier, part sequence number and transfer data
Discriminant	Large Message Trans. Identifier
Destination	The destination is loaded from parameter lptDest of the LPT Buffer holding the Large Packet to be transferred. It is determined in the same way as the destination of the (13,1) report which started the down-transfer.
Enable Check	Report is enabled if the LPT State Machine is in state DOWN_-TRANSFER
Ready Check	Default implementation (report is always ready)
Repeat Check	Report is repeated as long as lptRemSize is greater than partSize
Update Action	The following actions are performed: (a) Load the next part of the large packet from the LPT Buffer (b) Set the transaction identifier equal to largeMsgTransId (c) Set the part number equal to partSeqNmb (d) Increase partSeqNmb (e) Decrement lptRemSize by partSize (f) Set the action outcome to: 'completed'

Table 13.3: Specification of LptDownLastRep Component

Name	LptDownLastRep(13,3)
Description	Report carrying the last part of a down-transfer
Parameters	Large message transaction identifier, part sequence number and transfer data
Discriminant	Large Message Trans. Identifier
Destination	The destination is loaded from parameter lptDest of the LPT Buffer holding the Large Packet to be transferred. It is determined in the same way as the destination of the (13,1) report which started the down-transfer.
Enable Check	Report is enabled if the LPT State Machine is in state DOWN_-TRANSFER
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	The following actions are performed: (a) Load the last part of the large packet from the LPT Buffer (b) Set the transaction identifier equal to largeMsgTransId (c) Set the partnumber equal to partSeqNmb (d) Set the action outcome to: 'completed'

Table 13.4: Specification of LptUpFirstCmd Component

Name	LptUpFirstCmd(13,9)
Description	Command to carry the first part of an up-transfer
Parameters	Large message transaction identifier, part sequence number and part data for up-transfer
Discriminant	Large Message Trans. Identifier
Ready Check	Default implementation (command is always ready)
Start Action	The following actions are performed: (a) Determine the identifier of the LPT Buffer for the up-transfer by computing: (x MOD LPT_N_BUF) where ‘x’ is the Large Message Transaction Identifier. (b) Set action outcome to ’success’ if the Part Sequence Number is equal to 1 and the LPT State Machine is in state INACTIVE; otherwise set the action outcome to ‘failure’
Progress Action	The following actions are performed: (a) Send command StartUpTransfer to LPT State Machine (b) Copy the up-transfer data to LPT Buffer and set lptSize to be equal to the amout of copied data (c) Set lptTime to the current time; set partSeqNmb to 1 (d) Set lptSrc to the source of the command (e) Set the action outcome to: ’completed’
Termination Action	Default implementation (set action outcome to ’success’)
Abort Action	Default implementation (set action outcome to ’success’)

Table 13.5: Specification of LptUpInterCmd Component

Name	LptUpInterCmd(13,10)
Description	Command to carry an intermediate part of an up-transfer
Parameters	Large message transaction identifier, part sequence number and part data for up-transfer
Discriminant	Large Message Trans. Identifier
Ready Check	Default implementation (command is always ready)
Start Action	Run the Procedure Up-Transfer Start Action of figure 13.2.
Progress Action	The following actions are performed: (a) Copy the up-transfer data to LPT Buffer and increment lptSize by the amount of copied data (b) Set lptTime to the current time (c) Set patSeqNmb to the part sequence number carried by the command
Termination Action	Default implementation (set action outcome to ’success’)
Abort Action	Default implementation (set action outcome to ’success’)

Table 13.6: Specification of LptUpLastCmd Component

Name	LptUpLastCmd(13,11)
Description	Command to carry the last part of an up-transfer
Parameters	Large message transaction identifier, part sequence number and part data for up-transfer
Discriminant	Large Message Trans. Identifier
Ready Check	Default implementation (command is always ready)
Start Action	Run the Procedure Up-Transfer Start Action of figure 13.2.
Progress Action	The following actions are performed: (a) Copy the lptSize up-transfer data to LPT Buffer and increment lptSize by the amount of copied data (b) Set current time (c) Set patSeqNmb to the part sequence number carried by the command (d) Send EndUpTransfer command to LPT State Machine (e) Set action outcome to: 'completed'
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 13.7: Specification of LptUpAbortRep Component

Name	LptUpAbortRep(13,16)
Description	Report to notify the abortion of an up-transfer
Parameters	Large message transaction identifier and failure reason
Discriminant	Large Message Trans. Identifier
Destination	The destination is the same as the source of the up-transfer being interrupted
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	The large message transaction identifier is taken from parameter largeMsgTransId and the failure reason is read from variable lptFailCode.

Table 13.8: Specification of LptStartDownCmd Component

Name	LptStartDownCmd(13,129)
Description	Command to start a down-transfer
Parameters	Large message transaction identifier
Discriminant	Large Message Trans. Identifier
Ready Check	Default implementation (command is always ready)
Start Action	The following actions are performed: (a) Determine the identifier of the LPT Buffer for the up-transfer by computing: (x MOD LPT_N_BUF) where ‘x’ is the Large Message Transaction Identifier (b) Set action outcome to ’success’ if the LPT State Machine is in state INACTIVE; otherwise set the action outcome to ‘failure’
Progress Action	Send command StartDownTransfer to the LPT State Machine
Termination Action	Default implementation (set action outcome to ’success’)
Abort Action	Default implementation (set action outcome to ’success’)

Table 13.9: Specification of LptAbortDownCmd Component

Name	LptAbortDownCmd(13,130)
Description	Command to abort a down-transfer
Parameters	Large message transaction identifier
Discriminant	Large Message Trans. Identifier
Ready Check	Default implementation (command is always ready)
Start Action	The following actions are performed: (a) Determine the identifier of the LPT Buffer for the up-transfer by computing: (x MOD LPT_N_BUF) where ‘x’ is the Large Message Transaction Identifier (b) Set action outcome to ’success’ if the LPT State Machine is in state DOWN_TRANSFER; otherwise set the action outcome to ‘failure’
Progress Action	Send command Abort to the LPT State Machine
Termination Action	Default implementation (set action outcome to ’success’)
Abort Action	Default implementation (set action outcome to ’success’)

13.4 Service 13 Constants

The service 13 constants are listed in table 13.10.

Table 13.10: Constants for Large Packet Transfer Service

Name	Description
LPT_N_BUF	Number of Large Packet Transfer Buffers available for down-or up-transfer of large packets

13.5 Service 13 Observables and Parameters

The service 13 observables and parameters are listed in table 13.11.

Table 13.11: Observables and Parameters for LPT Service

Name	Kind	Description	Multiplicity
lptDest	par	Destination of transfer from LPT Buffer	LPT_N_BUF
lptTimeOut	par	Time-out for up-tramsfer to LPT Buffer	LPT_N_BUF
partSize	par	Part size for transfers to/from LPT Buffer	LPT_N_BUF
lptRemSize	var	Remaining size of a large packet in the LPT Buffer (part of the large packet not yet down-transferred)	LPT_N_BUF
lptSize	var	Size of large packet in the LPT Buffer	LPT_N_BUF
lptSrc	var	Source of the large packet up-transfer to the LPT Buffer	LPT_N_BUF
lptTime	var	Time when last up-transfer command to the LPT Buffer was received	LPT_N_BUF
nOfDownlinks	var	Number of on-going down-transfers	NULL
nOfUplinks	var	Number of on-going up-transfers	NULL
partSeqNmb	var	Part sequence number for the up/down/transfer to/from the LPT Buffer	LPT_N_BUF

13.6 Service 13 Adaptation Points

The PUS Extension of the CORDET Framework defines service 13 in full with the exception of the mechanism to access the LPT Buffers. The location of these buffers and the means to access them are application-specific and the framework accordingly defines an Adaptation Point to access these buffers. In a simple case, the adaptation point might take the form of a function which takes the identifier of an LPT Buffer as an argument and whcih returns the start address and size of the buffer itself.

Table 13.12: Adaptation Points for Service 13 (Large Packet Transfer Service)

AP ID	Adaptation Point	Close-Out Value
P-S13-1	Operation to access the i-th LPT Buffer (New AP)	No default defined at framework level

13.7 Service 13 Requirements

The table in this section lists requirements for the test service.

Table 13.13: Requirements for Service 13 (Large Packet Transfer Service)

Req. ID	Requirement Text
P-S13-1/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, an LptDownFirstRep component to encapsulate a report of type (13,1)*
P-S13-2/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, an LptInterFirstRep component to encapsulate a report of type (13,2)*
P-S13-3/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, an LptLastFirstRep component to encapsulate a report of type (13,3)*
P-S13-4/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, an LptUpFirstCmd component to encapsulate a command of type (13,9)*
P-S13-5/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, an LptUpInterCmd component to encapsulate a command of type (13,10)*
P-S13-6/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, an LptUpLastCmd component to encapsulate a command of type (13,11)*
P-S13-7/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, an LptUpAbortRep component to encapsulate a report of type (13,16)*
P-S13-8/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, an LptStartDownCmd component to encapsulate a command of type (13,129)*
P-S13-9/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, an LptAbortDownCmd component to encapsulate a command of type (13,130)*
P-S13-10/A	*The LptDownFirstRep component shall close the OutComponent adaptation points as indicated in table* ??
P-S13-11/A	*The LptDownInterRep component shall close the OutComponent adaptation points as indicated in table* ??
P-S13-12/A	*The LptDownLastRep component shall close the OutComponent adaptation points as indicated in table* ??
P-S13-13/A	*The LptUpFirstCmd component shall close the InCommand adaptation points as indicated in table* ??
P-S13-14/A	*The LptUpInterCmd component shall close the InCommand adaptation points as indicated in table* ??
P-S13-15/A	*The LptUpLastCmd component shall close the InCommand adaptation points as indicated in table* ??
P-S13-16/A	*The LptUpAbortRep component shall close the OutComponent adaptation points as indicated in table* ??

Req. ID	Requirement Text
P-S13-17/A	*The LptStartDownCmd component shall close the InCommand adaptation points as indicated in table ??*
P-S13-18/A	*The LptAbortDownCmd component shall close the InCommand adaptation points as indicated in table ??*
P-S13-19/S	*The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 13.11*
P-S13-20/S	*The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the parameters listed in table ??*
P-S13-21/C	*The part size or destination of a large packet transfer shall not be updated while a transfer is under way*
P-S13-22/C	*The initiator of a down-transfer shall ensure that, prior to starting a down-transfer, all data are available in the LPT Buffer and* `lptSize` *is initialized to the amount of data to be down-transferred*
P-S13-23/C	*The user shall not start an up-transfer to an LPT Buffer which is being processed by the application*

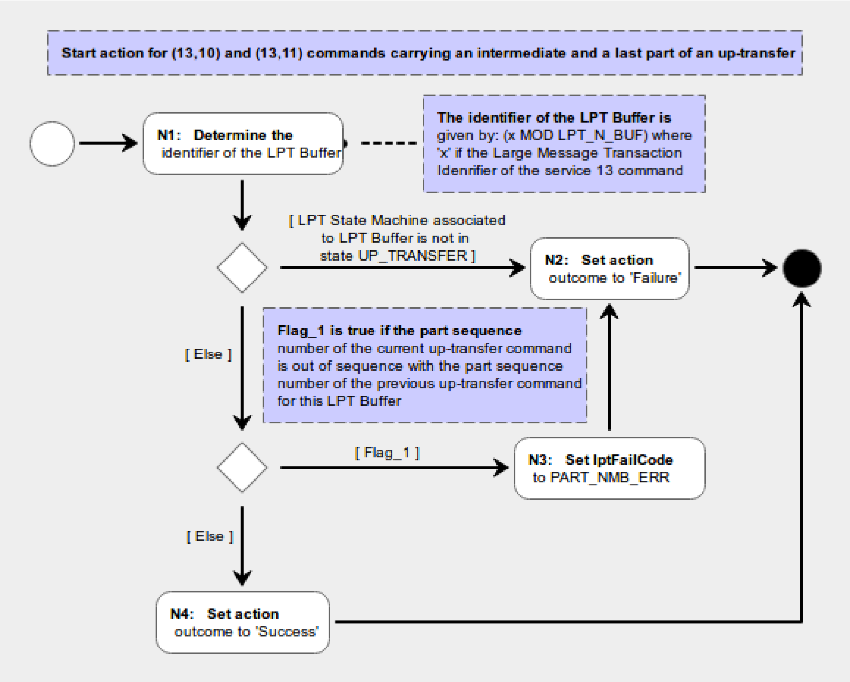

Start action for (13,10) and (13,11) commands carrying an intermediate and a last part of an up-transfer
N1: Determine the identifier of the LPT Buffer
The identifier of the LPT Buffer is given by: (x MOD LPT_N_BUF) where 'x' if the Large Message Transaction Idenrifier of the service 13 command
[LPT State Machine associated to LPT Buffer is not in state UP_TRANSFER]
N2: Set action outcome to 'Failure'
[Else]
Flag_1 is true if the part sequence number of the current up-transfer command is out of sequence with the part sequence number of the previous up-transfer command for this LPT Buffer
[Flag_1]
N3: Set lptFailCode to PART_NMB_ERR
[Else]
N4: Set action outcome to 'Success'

Fig. 13.2: Up-Transfer Start Action

14 Test Service

The service type of the Test Service is 17. The PUS Extension of the CORDET Framework supports this service in full.

The Test Service provides the capability to perform two kinds of connections tests: the *Are-You-Alive Test* and the *On-Board Connection Test*.

The Are-You-Alive test is like a ping test: an external user sends a command of type (17,1) to the application and the application responds by sending to the user a (17,2) report. Neither the (17,1) command nor the (17,2) report carry any parameters.

In the On-Board-Connection Test, an external user sends a command of type (17,3) to application A asking it to perform a connection test with some other application B. Application B is specified through a parameter carried by the (17,3) command.

The way the connection test is performed is not specified by the PUS. The PUS Extension of the CORDET Framework implements it as an Are-You-Alive Test from application A to application B. If this Are-You-Alive Test is successful, application A generates a (17,4) report to its user. The Are-You-Alive Test is declared successful if a (17,2) report from application B is received within time `AreYouAliveTimeOut` from the sending of the (17,1) command.

14.1 Service 17 Command and Report Definition

In the CORDET Framework an out-going report is encapsulated in an OutComponent component and an incoming command is encapsulated in an InCommand component. The framework extension accordingly offers the following components to implement the two commands and the two reports of service 17:

- Component AreYouAliveCmd implements command (17,1)
- Component AreYouAliveRep implements report (17,2)
- Component OnBoardConnectCmd implements command (17,3)
- Component OnBoardConnectRep implements report (17,4)

These components are defined by the way they close the adaptation points of the OutComponent and InCommand. This is defined formally in tables **??** to **??** but the main points are as follows.

The AreYouAliveCmd commmand implements a Progress Action which creates and loads the AreYouAliveRep report. The report destination is the same as the source of the AreYouAliveCmd command. Thus, the processing of the AreYouAliveCmd command consists in sending an AreYouAliveRep to the source of the AreYouAliveCmd. The AreYouAliveCmd commmand is always accepted and it is always started, executed and terminated successfully.

The OnBoardConnectCmd command is always accepted. The command carries as its single parameter the identifier of the application with which the connection test must be performed. The Start Action of the command verifies the legality of this application identifier. In order to establish its legality, service 17 maintains parameter `onBoardConnectDestLst` to hold the list of legal targets for the On-Board-Connection test. If the application identifier carried by the OnBoardConnectCmd command is not included in this list, its Start Action is deemed to have failed. The Start Action of the OnBoardConnectCmd command is shown in figure

14.1 as an activity diagram.

If, instead, the legality of the target application identifier is confirmed, the Start Action sends an AreYouAliveCmd command to the target application. Normally, the target application should respond by sending it an AreYouAliveRep report. If the expected response (the AreYouAliveRep report) is not received within time `areYouAliveTimeOut`, the command is deemed to have failed its execution.

The mechanism through which the AreYouAliveRep report notifies the OnBoardConnectCmd command of its arrival is as follows:

- The service 17 maintains integer variable `areYouAliveSrc`
- The Start Action of the OnBoardConnectCmd command resets `areYouAliveSrc` to zero
- The Update Action of the incoming report AreYouAliveRep loads its source in variable `areYouAliveSrc`
- The Progress Action of the OnBoardConnectCmd command only declares the command to have successfully terminated if, within time-out `areYouAliveTimeOut`, it finds `areYouAliveSrc` equal to the identifier of the application with which the connection test is done

One implication of this mechanism is that only one On-Board-Connection Test may be active at a given time (i.e. the user should only send a new OnBoardConnectCmd command to an application after execution of the previous OnBoardConnectCmd command has completed). This constraint is not enforced by the framework and is under the responsibility of the user of the service.

The time-out parameter `areYouAliveTimeOut` is the same for all target applications. There is, in other words, an underlying assumption that the response time of all target applications is similar and that there is therefore no need to maintain separate time-outs for each target application. If this assumption is not satisfied, the user must update the value of `areYouAliveTimeOut` before starting an On-Board-Connection Test.

Tables **??** to **??** formally specify the service 17 commands and reports by specifying how the actions, checks and attributes of generic out-going commands and reports are specialized for service 17 (see section 7). The following considerations apply to the service 17 commands and reports:

- The service 17 commands execute in 'one-shot' mode and therefore do not generate progress reports.
- Service 17 reports are generated unconditionally and hence their enable check always returns 'report enabled'.
- Service 17 reports are generated as soon as the condition which triggered them occur and hence their ready check always returns 'ready'
- Service 17 reports are 'one-off' reports and hence their repeat check always returns 'no repeat'

Table 14.1: Specification of TstAreYouAliveCmd Component

Name	TstAreYouAliveCmd(17,1)
Description	Command to perform and Are-You-Alive Connection Test
Parameters	None
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Retrieve (17,2) report from OutFactory and set action outcome to 'success' if the retrieval succeeds. If the retrieval fails, generate error report OUTFACTORY FAILED and set outcome of Start Action to 'failed' with failure code VER_REP_CR_FD
Progress Action	Configure the (17,2) report with a destination equal to the source of the (17,1), load it in the OutLoader, and then set the action outcome to 'completed'
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

Table 14.2: Specification of TstAreYouAliveRep Component

Name	TstAreYouAliveRep(17,2)
Description	Report generated in response to a (17,1) command requesting an Are-You-Alive Connection Test
Parameters	None
Discriminant	None
Destination	The source of the (17,1) command which triggered the report
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

Table 14.3: Specification of TstConnectCmd Component

Name	TstConnectCmd(17,3)
Description	Command to perform and On-Board Connection Test.
Parameters	Identifier of application with which the connection test is done
Discriminant	None
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of (17,3) Command of figure 14.1)
Progress Action	Run the procedure Progress Action of (17,3) Command of figure 14.1)
Termination Action	Set action outcome to 'success' if the (17,4) report was issued and to 'failure' otherwise with failure code VER_TST_TO
Abort Action	Default implementation (set action outcome to 'success')

Table 14.4: Specification of TstConnectRep Component

Name	TstConnectRep(17,4)
Description	Report generated in response to a (17,3) command requesting a On-Board Connection Test
Parameters	Identifier of application with which the connection test was done
Discriminant	None
Destination	The source of the (17,3) command which triggered the report
Enable Check	Default implementation (report is always enabled)
Ready Check	Default implementation (report is always ready)
Repeat Check	Default implementation (report is not repeated)
Update Action	Default implementation (do nothing)

14.2 Service 17 Constants

The service 17 constants are listed in table 14.5.

Table 14.5: Constants for Test Service

Name	Description
`LPT_N_BUF`	Number of Large Packet Transfer Buffers available for down-or up-transfer of large packets

14.3 Service 17 Observables and Parameters

The service 17 observables and parameters are listed in table 13.11.

Table 14.6: Observables and Parameters for Test Service

Name	Kind	Description
`AreYouAliveTimeOut`	par	Time-out for the Are-You-Alive Test initiated in response to an On-Board Connection Test
`OnBoardConnectDestLst`	par	Identifiers of target applications for an On-Board-Connection Test
`AreYouAliveSrc`	var	Source of the latest (17,2) report received in response to a (17,1) command triggered by a (17,3) command
`AreYouAliveStart`	var	Time when the Are-You-Alive Test is started in response to an On-Board Connection Test
`OnBoardConnectDest`	var	Destination of the (17,1) triggered by a (17,3) command

14.4 Service 17 Requirements

The table in this section lists requirements for the test service.

Table 14.7: Requirements for Service 17 (Test Service)

Req. ID	**Requirement Text**
P-S17-1/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, an AreYouAliveCmd component to encapsulate a (17,1) command*
P-S17-2/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, an AreYouAliveRep component to encapsulate a (17,2) report*
P-S17-3/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, an OnBoardConnectCmd component to encapsulate a (17,3) command*
P-S17-4/S	*The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, an OnBoardConnectCmd component to encapsulate a (17,4) report*
P-S17-5/A	*The AreYouAliveCmd component shall close the InCommand adaptation points as indicated in table* ??

Req. ID	Requirement Text
P-S17-6/A	*The AreYouAliveRep component shall close the OutComponent adaptation points as indicated in table* ??
P-S17-7/A	*The OnBoardConnectCmd component shall close the InCommand adaptation points as indicated in table* ??
P-S17-8/A	*The OnBoardConnectRep component shall close the OutComponent adaptation points as indicatedin table* ??
P-S17-9/C	*An application shall not be sent a (17,3) command before execution of the previous (17,3) command has completed*
P-S17-10/S	*The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 14.6*
P-S17-11/S	*The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the parameters listed in table 14.6*

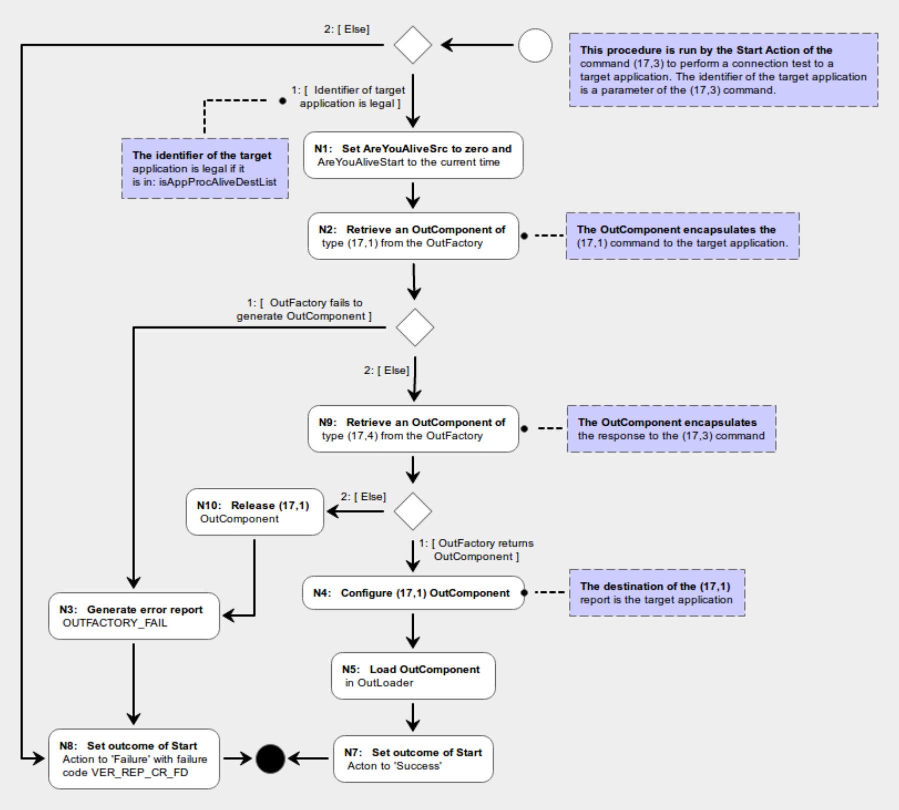

2: [Else]
This procedure is run by the Start Action of the command (17,3) to perform a connection test to a target application. The identifier of the target application is a parameter of the (17,3) command.
1: [Identifier of target application is legal]
The identifier of the target application is legal if it is in: isAppProcAliveDestList
N1: Set AreYouAliveSrc to zero and AreYouAliveStart to the current time
N2: Retrieve an OutComponent of type (17,1) from the OutFactory
The OutComponent encapsulates the (17,1) command to the target application.
1: [OutFactory fails to generate OutComponent]
2: [Else]
N9: Retrieve an OutComponent of type (17,4) from the OutFactory
The OutComponent encapsulates the response to the (17,3) command
N10: Release (17,1) OutComponent
2: [Else]
1: [OutFactory returns OutComponent]
N4: Configure (17,1) OutComponent
The destination of the (17,1) report is the target application
N3: Generate error report OUTFACTORY_FAIL
N5: Load OutComponent in OutLoader
N8: Set outcome of Start Action to 'Failure' with failure code VER_REP_CR_FD
N7: Set outcome of Start Acton to 'Success'

Fig. 14.1: Start Action of OnBoardConnectCmd (17,3) Command

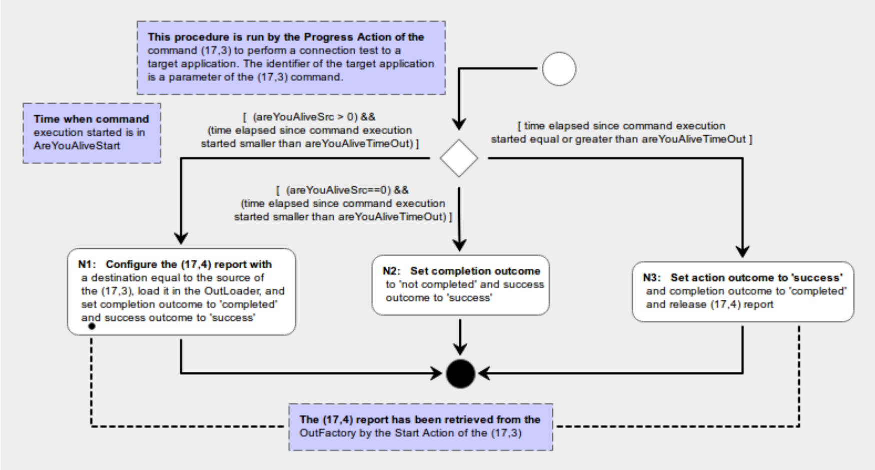

This procedure is run by the Progress Action of the command (17,3) to perform a connection test to a target application. The identifier of the target application is a parameter of the (17,3) command.
Time when command execution started is in AreYouAliveStart
[(areYouAliveSrc > 0) && (time elapsed since command execution started smaller than areYouAliveTimeOut)]
[time elapsed since command execution started equal or greater than areYouAliveTimeOut]
[(areYouAliveSrc==0) && (time elapsed since command execution started smaller than areYouAliveTimeOut)]
N1: Configure the (17,4) report with a destination equal to the source of the (17,3), load it in the OutLoader, and set completion outcome to 'completed' and success outcome to 'success'
N2: Set completion outcome to 'not completed' and success outcome to 'success'
N3: Set action outcome to 'success' and completion outcome to 'completed' and release (17,4) report
The (17,4) report has been retrieved from the OutFactory by the Start Action of the (17,3)

Fig. 14.2: Progress Action of OnBoardConnectCmd (17,3) Command

15 Event Action Service

The specification of this service is still TBD.

A Pre-Defined Event Reports

The table in this section lists all the service 5 event reports which are generated by components of the PUS Extension of the CORDET Framework. For each event report, the following information is provided:

- The name of the event report
- The description of the event report
- The parameters carried by the event report

Table A.1: Event Reports

Name	Description	Parameters
EVT_DOWN_ABORT	Generated by an LPT State Machine when a down-transfer is aborted	LPT State Machine Identifier
EVT_UP_ABORT	Generated by an LPT State Machine when an up-transfer is aborted	LPT State Machine Identifier
EVT_MON_LIM_R	Generated when a Limit Check Monitoring Procedure has detected an invalid parameter value of real type	Identifier of parameter monitor and of monitored data item, sub-status of parameter monitor and last value of data item
EVT_MON_LIM_I	Generated when a Limit Check Monitoring Procedure has detected an invalid parameter value of integer type	Identifier of parameter monitor and of monitored data item, sub-status of parameter monitor and last value of data item
EVT_MON_EXP	Generated when a Expected Value Monitoring Procedure has detected an invalid parameter value of integer type	Identifier of parameter monitor and of monitored data item, sub-status of parameter monitor and last value of data item
EVT_MON_DEL_R	Generated when a Delta Check Monitoring Procedure has detected an invalid parameter value of real type	Identifier of parameter monitor and of monitored data item, sub-status of parameter monitor and last value of data item
EVT_MON_DEL_I	Generated when a Delta Check Monitoring Procedure has detected an invalid parameter value of integer type	Identifier of parameter monitor and of monitored data item, sub-status of parameter monitor and last value of data item
EVT_FMON_FAIL	Generated when a functional monitor has declared a failure	Identifiers of parameter monitors associated to the functional monitors and of their checking status
EVT_CLST_FULL	Generated when the Monitoring Function Procedure tries to add an entry to the Check Transition List but the list is full	None
EVT_DUMMY_1	Dummy level 1 event used for testing purposes	One dummy parameter
EVT_DUMMY_2	Dummy level 2 event used for testing purposes	None
EVT_DUMMY_3	Dummy level 3 event used for testing purposes	One dummy parameter
EVT_DUMMY_4	Dummy level 4 event used for testing purposes	None

B Error Reports

The table in this section lists all the error reports which are generated by the PUS Extension of the CORDET Framework. For each error report, the following information is provided:

- The name of the error report
- The description of the error report
- The parameters carried by the error report

Table B.1: Error Reports

Name	Description	Parameters
`INLOADER_ACC_FAIL`	Generated by InLoader when creation of an InReport for an incoming report has failed	Packet identifier of report and identifier of reason for the creation failure
`INLOADER_INV_DEST`	Generated by InLoader when it receives a report with an invalid destination	Type, subtype, discriminant and destination of the report with the invalid destination
`OUTFACTORY_FAIL`	Generated when an attempt to retrieve a report from the OutFactory has failed	Type, subtype and discriminant of the report whose generation failed
`SNDPCKT_INV_DEST`	Generated by Send Packet Procedure when it finds an invalid destination in an OutComponent	Type, subtype and discriminant of the report with the invalid destination and invalid destination of the report

C Request Verification Failure Codes

Request verification failure reports of service 1 carry a failure code. The table in this section lists all the failure codes supported by the PUS Extension of the CORDET Framework. Failure reports carry parameters. Some of these parameters are common to all failure reports but the Failure Verification Data is code-specific (see section 8.1). This is defined in the rightmost column of the table.

Table C.1: Request Verification Failure Codes

Name	Description	Ver. Failure Data
VER_CMD_INV_DEST	Failure code for all (1,10) reports	None
VER_REP_CR_FD	Failure code for start actions when they unsuccessfully attempt to create a new report from the OutFactory	None
VER_OUTLOADER_FD	Failure code for start actions when the Load operation in the OutLoader has failed	None
VER_SID_IN_USE	A (3,1) or (3,2) command attempted to create a new report with a SID which is already in use	The SID in use
VER_FULL_RDL	A (3,1) or (3,2) command attempted to create a new report at a time when the RDL is already full	None
VER_RDL_CONSTR	A (3,1) or (3,2) command attempted to create a new report which configuration data which violate an RDL constraint of table 9.1	The identifier of the violated constraint
VER_DUPL_DI	A (3,1) or (3,2) command carried the same data item identifier twice	The duplicated data item identifier
VER_ILL_SID	A service 3 command had an invalid SID	The invalid SID
VER_ENABLED_SID	A (3,3) or (3,4) command attempted to enabled a SID which is already enabled	The enabled SID
VER_SID_START_FD	All the instructions in a service 3 command have been rejected	None
VER_FACT_PRGR_FD	The progress action of a multi-instruction service 3 command has failed to retrieve a report from the OutFactory	The SID for which the retrieval from the OutFactory was attempted
VER_ILL_EID	The start action of a service 5 command has encountered an illegal Event Identifier (EID)	The illegal EID
VER_EID_ST_FD	All the instructions in a service 5 command have been rejected	None
VER_ILL_MON	A Parameter or Functional Monitor Identifier in a service 12 command is out-of-range or not defined	The rejected Parameter or Functional Monitor Identifier
VER_MON_START_FD	All the instructions in a service 12 command have been rejected	None
VER_PMDL_FULL	A service 12 command has found the Parameter Monitor Definition List (PMDL) full	The parameter monitor identifier for which the violation was found
VER_MON_ILL_DI	A service 12 command has found the data item identifier of the parameter to be monitored illegal	The parameter monitor identifier for which the violation was found

Name	Description	Ver. Failure Data
VER_MON_PROT	A service 12 command as found a parameter monitor which belongs to a protected functional monitor	The parameter monitor identifier for which the violation was found
VER_MON_ENB	A service 12 command has found a parameter or functional monitor which is enabled	The parameter or functional monitor identifier for which the violation was found
VER_MON_USE	A service 12 command has found a parameter monitor which is used by a functional monitor	The parameter monitor identifier for which the violation was found
VER_FMDL_FULL	A service 12 command has found a Functional Monitor Definition List (FMDL) full	The functional monitor identifier for which the violation was found
VER_MON_TMP	A service 12 command has found too many parameter monitors in a functional monitor	The functional monitor identifier for which the violation was found
VER_MON_MFN	A service 12 command has found a value of minimum failing number equal to zero	The functional monitor identifier for which the violation was found
VER_SCD_ILL_SS	Failure code for start action of service 11 command when it finds an illegal sub-schedule identifier	The sub-schedule identifier
VER_FULL_TBS	A service 11 command found the Time-Based Schedule (TBS) full	Identifier of the request within the command where the error occurred
VER_SCD_ILL_G	A service 11 command found an illegal schedule group identifier	The illegal group identifier
VER_SCD_ILL_RT	A service 11 command found an illegal release time	Coarse part of illegal release time
VER_SCD_ILL_DS	A service 11 command found an illegal destination for an scheduled command	Illegal destination
VER_SCD_CRFAIL	A service 11 command was unable to create an InCommand for a scheduled command (either due to lack of resources or due to illegal command type)	Identifier of the request within the command where the error occurred
VER_SCD_ST_FD	All instructions in a service 11 command have been rejected	None
VER_ILL_ACT_ID	Command (11,5) was unable to find an activity identifier in the TBS	The sequence count part of the activity ID
VER_TST_TO	The time-out of the (17,3) command has triggered	None

Name	Description	Ver. Failure Data
VER_CRE_FD	The InLoader has failed to create an InCommand to hold an incoming command	None
VER_CMD_LD_FD	The InLoader has failed to load an InCommand component into its InManager	None

D PUS Requirements Compliance Matrix

The table in this section presents the level of compliance achieved by the PUS Extension of the CORDET Framework to the PUS requirements of AD-1. The first three columns give the identifier, the title and the text of the PUS requirement. The fourth column gives the compliance status which can be one of the following:

C1 The requirement is directly implemented by the PUS Extension of the CORDET Framework or by the CORDET Framework itself (i.e. applications instantiated from the framework are guaranteed to be compliant with the requirement)

C2 The requirement may be implemented by applications instantiated from the PUS Extension of the CORDET Framework (i.e. applications instantiated from the framework may be made be compliant with the requirement)

NC The requirement is not compatible with the PUS Extension of the CORDET Framework (i.e. applications instantiated from the framework cannot be compliant with the requirement)

NA The requirement is not covered by the PUS Extension of the CORDET Framework

In several cases, the compliance level is declared to be 'C1/C2' when part of the requirement is implemented by the PUS Extension of the CORDET Framework and part is left to the application developers.

The fourth column in the table provides a discussion of the level of compliance and, wherever possible, the following additional information is provided:

C1 Traceability to the framework requirements implementing the PUS requirement

C2 Traceability to the adaptation point(s) where application developers can insert their own requirements to achieve compliance

NC Justification for non-compliance

NA Explanation of the reason for the non-applicability of the requirement

Only requirements in sections 5 to 7 of the PUS are covered. Requirements in section 8 merely state the layout of the standard commands and reports. Compliance to these requirements is uncontroversial and is guaranteed in all cases. Requirements in section 9 are not relevant to the PUS Extension of the CORDET Framework and are therefore ignored.

Table D.1: Mapping of PUS Requirements to CORDET Requirement

N	Title	Requirement	C	Justification
5.3.1a	General	Each service type shall be uniquely identified by exactly one service type name.	C1/C2	The service type names and identifiers of pre-defined services are taken from the PUS and the service types names and identifiers of other services are set by the application developers at adaptation point ICM-18 for incoming commands and OCM-7 for out-going reports.
b		Each service type shall be uniquely identified by exactly one service type identifier that is an unsigned integer greater than or equal to 1, and less than or equal to 255.	C1/C2	See justification of first requirement in this clause
c		Each standard service type shall have a service type identifier less than or equal to 127.	C1/C2	See justification of first requirement in this clause
d		Each mission specific service type shall be associated with a service type identifier greater than or equal to 128.	C1/C2	See justification of first requirement in this clause
5.3.2a	Subservice Type	Each service type shall define at least one subservice type.	C1/C2	For pre-defined services, the PUS is followed and at least one sub-service is defined. For other services, adaptation points ICM-19 for incoming commands and OCM-8 for out-going reports imply definition of a sub-service for each service.
b		Each subservice type shall be defined by exactly one service type.	C1/C2	See justification of first requirement in this clause
c		Each subservice type shall be uniquely identified by exactly one subservice type name.	C1/C2	See justification of first requirement in this clause

N	Title	Requirement	C	Justification
d		For each subservice type, whether the realization of that subservice type is implicitly required for each realization of the service type or required by tailoring shall be declared when specifying that subservice type.	C1	For pre-defined services, dependencies between sub-services are identified and formulated as use constraint requirements.
e		For each subservice type, whether multiple realizations of that subservice type are allowed within a single service shall be declared when specifying that subservice type.	NA	This requirement does not concern the implementation of the services and is therefore outside the scope of the PUS Extension of the CORDET Framework (TBC)
f		For each subservice type, the observables shall be declared when specifying that subservice type.	C1	A list of observables is provided for each pre-defined service offered by the PUS Extension of the CORDET Framework
5.3.3.1a	Message Type	Each message type shall be uniquely identified by exactly one message type name.	C1	The CORDET Framework implements a message as a command or report exchanged between applications and identifies the type of a message through the triplet [type, sub-type, discriminant]. See section 4 of CORDET Framework Definition Document.
b		Each message type shall be uniquely identified by exactly one message type identifier.	C1	See justification of first requirement in this clause
c		Each message type identifier shall be composed of: 1. the service type identifier of the service type that contains that message type; 2. a message subtype identifier that uniquely identifies that message type within that service type.	C1	See justification of first requirement in this clause

N	Title	Requirement	C	Justification
d		Each message subtype identifier shall be an unsigned integer greater than or equal to 1, and less than or equal to 255.	C1/C2	For pre-defined services, the command and report types are taken over from the PUS. For other services, they are under the control of the application developer through adaptation points ICM-18, ICM-19, OCM-7 and OCM-8.
e		Each standard message type identifier shall have a message subtype identifier less than or equal to 127.	C1/C2	See justification of requirement e in this clause.
f		Each mission specific message type that belongs to a standard service type shall have a service subtype identifier greater than or equal to 128.	C1/C2	See justification of requirement e in this clause.
g		Each message type shall either be: 1. a request type, or 2. a report type.	C1	See justification of first requirement in this clause
5.3.3.2a	Request Type	Each request type shall define one or more instruction types.	C1/C2	For pre-defined services, the PUS definition is followed. For application-dependent services, the user is responsible for providing the information requested in this requirement.
b		Each instruction type shall be defined for exactly one request type.	C1/C2	See justification of first requirement in this clause
c		Each instruction type shall be uniquely identified by exactly one instruction type name.	C1/C2	See justification of first requirement in this clause

N	Title	Requirement	C	Justification
d		For each request type and for each instruction type of that request type, whether that request type provides a single instruction slot or multiple instruction slots for that instruction type shall be declared when specifying that request type.	C2	For requests which may include multiple instructions, the PUS Extension does not impose any upper boundary on the number of instructions in a command. Any such upper boundary must be imposed by the user (e.g. in the command database).
e		For each request type that contains several instruction types, the allowed combinations of instruction types that can be used in a request of that request type shall be declared when specifying that request type.	C1	For all services pre-defined by the PUS Extension, when a request instance may contain multiple instructions, then those instructions are all of the same type (TBC).
f		For each instruction type, the instruction arguments used by that instruction type, their definition and their ordering within the instruction type shall be declared when specifying that instruction type.	C1/C2	The complete layout of a request must be defined as part of the definition of a command (see adaptation points OCM-12 and ICM-21)
g		For each request type that provides multiple instruction slots, if that request type constrains the scope of the instructions that can be issued within a request of that type, the argument or set of arguments of the related instruction types that define that scope shall be grouped together in the definition of the request type.	TBC	This requirement is not understood. It presumably refers to a situation where a multi-instruction request carries parameters which apply to all instructions in the request. The requirement states that those parameters must be grouped together. If this interpretation is correct, then the services pre-defined by the PUS Extension are compliant.
h		For each request type, the definition of the request arguments provided by that request type, their definition and their ordering within the request type shall be declared when specifying that request type.	C1/C2	See justification requirement f in this clause.

N	Title	Requirement	C	Justification
5.3.3.3a	Report Type	Each report type shall either be: 1. a data report type, 2. a verification report type, or 3. an event report type.	C1/C2	For pre-defined services, the report types are taken over from the PUS. For other services, the report type is under the control of the application developer through adaptation points OCM-*.
b		Each report type shall define exactly one notification type.	C1/C2	In the CORDET Framework, notifications are implicitly defined within reports
c		Each notification type shall be defined for exactly one report type.	C1/C2	See justification of first requirement in this clause
d		Each notification type shall be uniquely identified by exactly one notification type name.	C1/C2	See justification of first requirement in this clause
e		For each report type and for each notification type of that report type, whether that report type provides a single notification slot or multiple notification slots for that notification type shall be declared when specifying that report type.	C1/C2	See justification of first requirement in this clause
5.3.4a	Capability Type	Each subservice type shall define at least one capability type.	C1/C2	The capability types are defined implicitly when a service is defined. For the pre-defined services, the PUS Extension follows the PUS.
b		For each capability type defined by a subservice type, the applicability constraints of that capability type shall be declared when specifying that subservice type.	C2	The CORDET Framework does not enforce any compatibility constraints. These must be enforced by users during the instantiation process
5.3.5.1a	Transaction Type	Each transaction type shall be defined by exactly one capability type.	NA	This requirement does not concern the implementation of the services and it therefore has no impact on the PUS Extension of the CORDET Framework

N	Title	Requirement	C	Justification
b		Each transaction type shall either be: 1. a request related transaction type, 2. an autonomous data reporting transaction type, or 3. an event reporting transaction type.	NA	See justification of previous requirement.
5.3.5.2.1a	Request related transaction type	Each request related transaction type shall involve exactly one request type.	C1	The CORDET Framework only defines individual commands and report. The PUS Extension implicitly defines transactions when it specifies links between a command and the reports it triggers or when it specified the conditions under which data or event reports are generated. Compliance with the requirement is guaranteed for the services pre-defined by the PUS Extension which follow the PUS.
b		Each request type shall be involved in exactly one request related transaction type.	C1	See justification of previous requirement.
5.3.5.2.2a	Response Type	Each request type shall be linked to at most one data report type.	C1	See justification of first requirement of clause 5.3.5.2.1
b		For each instruction type that is linked to a notification type, whether a realization of that instruction type can cause the generation of multiple notifications shall be declared when specifying that instruction type.	C1	See justification of first requirement of clause 5.3.5.2.1
5.3.5.2.3a	Execution verification profile	For each request type, the pre-conditions to verify prior to starting the execution of each request of that type shall be declared when specifying that request type.	C1	The condition to start execution of a command are verified in the command's Start Action (adaptation points ICM-8)

N	Title	Requirement	C	Justification
b		For each instruction type, the pre-conditions to verify prior to starting the execution of each instruction of that type shall be declared when specifying that instruction type.	C1	The CORDET Framework does not directly implement the concept of instructions. Instructions are therefore implicitly embedded within commands. Verification of their execution pre-conditions can be done either as part of a command's Start Action (adaptation point ICM-8) or as part of the the command's Progress Action (adaptation point ICM-9). For the commands pre-defined by the PUS Extension, the first option has been selected and the pre-conditions for execution of a command are verified as part of that command's Start Action.
c		For each request type that provides a multiple instruction slots capability, whether the subservice verifies the suitability of all instructions contained within each request of that type before authorizing the start of execution of that request shall be declared when specifying that request type.	C1/C2	For the services PUS Extension, the rules stated in the PUS are followed. For other services, users choose between these two options when they Implement the Start Action of a command.
d		For each instruction type, the conditions to verify during the execution of each instruction of that type shall be declared when specifying that instruction type.	C1/C2	See justification of previous two requirements
e		For each instruction type, the post-conditions to verify at the end of the execution of each instruction of that type shall be declared when specifying that instruction type.	C1	The post-conditions of an instruction can be verified either in the Progress Action (adaptation point ICM-9) or in the Termination Action (adaptation point ICM-10). For commands pre-defined by the PUS Extension, the second option has been chosen.

N	Title	Requirement	C	Justification
f		For each request type, the post-conditions to verify at the end of the execution of each request of that type shall be declared when specifying that request type.	C1	The post-conditions of a request must be verified in the Termination Action of a command (adaptation point ICM-10).
g		For each request type, the execution verification profile used to report the start, progress and completion of execution of each request of that type shall be declared when specifying that request type.	C1	The execution verification profile of a request is specified when the Start Action, Progress Action and Termination Action of a command are specified (adaptation points ICM-8, 9 and 10). Adaptation point ICM12 to 17 can be used to specify how the notifications of the verification outcomes should be handled.
h		Each progress of execution notification shall provide the means to uniquely identify the instruction that progress of execution is notified.	C1	The progress of execution notifications are generated through calls to the Operation to Report Progress Success for InCommand and the Operation to Report Progress Failed for InCommand (adaptation points ICM-14 and 15). These operations take the command identifier and the execution step identifier as arguments. The latter can be used to identify the instruction which failed or succeeded.
I		For each instruction type, the functionality that the subservice performs when executing an instruction of that type shall be declared when specifying that instruction type.	C1	The functionality executed when a command is executed is defined by the Progress Action of the command which holds the instruction (adaptation points ICM-9). This action therefore implements both the request-level and instruction level actions.

N	Title	Requirement	C	Justification
j		For each request type, the request-specific functionality that the subservice performs when executing a request of that type shall be declared when specifying that request type.	C1	See previous requirement
5.3.5.3a	Autonomous data reporting transaction type	Each autonomous data reporting transaction type shall involve exactly one data report type.	C1	The CORDET Framework does not enforce this constraint. For the services in the PUS Extension, the rules of the PUS are followed and the constraint is therefore satisfied.
b		Each data report type shall be involved in at most one autonomous data reporting transaction type.	C1	See justification of first requirement in this clause
5.3.5.4a	Event reporting transaction type	Each event reporting transaction type shall involve exactly one event report type.	C1	The CORDET Framework does not enforce this constraint. For the services in the PUS Extension, the rules of the PUS are followed and the constraint is therefore satisfied.
b		Each event report type shall be involved in exactly one event reporting transaction type.	C1	See justification of first requirement in this clause
5.3.6	Tailoring the generic service type abstraction level	Tailoring the generic service type abstraction level shall consist of: 1. adding mission-specific service types; 2. adding mission-specific subservice types; 3. adding mission-specific capability types; 4. adding mission-specific message types.	C2	The CORDET Framework allows new service types and sub-types to be added through adaptation points OCM-* and ICM-*. For each new service, mission-specific capabilities and messages can be associated. Capability and message types are defined implicitly through the definition of the service types and sub-types.
5.4.2.1a	Application process	Each application process shall either be: 1. an on-board application process, or 2. a ground application process.	C1	The way PUS-style application processes are implemented in the CORDET Framework is discussed in section 3.5 of the CORDET User Manual

N	Title	Requirement	C	Justification
b		Each application process that hosts at least one subservice provider shall be identified by an application process identifier that is unique across the system that hosts that subservice provider.	C2	Applications can customize the factory components which create the packets representing commands and reports (adaptation points FAC-1) such that they fill in the header information in the packets in accordance with their allocation of APIDs.
c		Each application process identifier shall be an unsigned integer that is less than or equal to 2046.	C2	See justification of previous requirement.
d		Each application process that hosts at least one subservice user shall be identified by an application process user identifier that is unique within the context of the overall space system.	C1	The application process user identifier of a service user is the source of commands issued by that service user and the destination of reports received by that service user. This can be mapped to the concept of command source and report destination (see section 4 of the CORDET Framework Definition Document).
e		Each application process user identifier shall be an unsigned integer that is greater than or equal to 0, and less than or equal to 65535.	C1	See justification of previous requirement.
f		For each report that it generates, each on-board application process shall time tag that report using the on-board reference time.	C1	The time-stamp of out-going components is set by the Send Packet Procedure of the OutComponent of the CORDET Framework (see section 6.1.1 of the CORDET Framework Definition Document).

N	Title	Requirement	C	Justification
g		For each application process, whether that application process time tags the reports before collecting the values of the constituting parameters or after shall be declared when specifying that application process.	C1	In the CORDET Framework, the time-stamp of a report represents the time when an application makes a request to issue that report (this is after the report data have been collected). See section 4.2.1 of the CORDET Framework Definition Document.
h		For each application process, whether that application process provides the capability to report the status of the on-board time reference used when time tagging reports shall be declared when specifying that application process.	NA	The CORDET Framework defines an interface for acquiring the current time (see adaptation point C2-TIM-1 in [CR-UM]) but it does not include an interface for acquiring the status of the on-board time reference. This capability, if required, must be provided entirely at application level.
I		For each application process, whether that application process provides the capability to count the type of generated messages per destination and report the corresponding message type counter shall be declared when specifying that application process.	C1	The OutStream components maintain counters of out-going commands and reports sent to their destination (there is one OutStream for each destination). See section 5.2.1 of the CORDET Framework Definition Document. The framework however does not, by default, provide the capability to count the number of messages of a given type sent to a given destination.
j		Each application process that provides the capability to count the type of generated messages per destination and report the corresponding message type counter shall maintain, per destination, a counter for each message type that it generates.	C1	See justification of previous requirement.

N	Title	Requirement	C	Justification
5.4.3.2a	On-board parameter	Each on-board parameter shall be identified by exactly one on-board parameter identifier that is unique across the entire spacecraft.	C2	The PUS Extension of the CORDET Framework maps on-board parameters to the Data Items in the Data Pool Component. The application developer is responsible for defining the Data Items (see adaptation point DP-7) and this includes the allocation of their identifiers.
b		The set of on-board parameter minimum sampling intervals used to access the on-board parameters shall be declared when specifying the spacecraft architecture.	C2	The PUS Extension of the CORDET Framework does not enforce a minimum sampling time. This mus be enforced by the application. Note that the definition of service 3 includes the definition of a minimum collection period for housekeeping reports (HK_COLLECT_PER)
c		Each on-board parameter shall be associated to exactly one on-board parameter minimum sampling interval.	C2	See justification of requirement b in this clause
d		All on-board parameters accessed by an application process shall be associated to the same on-board parameter minimum sampling interval.	C2	See justification of requirement b in this clause
5.4.3.3.1a	On-board memory	Each on-board memory shall be identified by exactly one on-board memory identifier.	C2	The on-board memory identifiers and the characteristics of the on-board memories are defined as part of the instantiation of service 6. See adaptation points TBD.
b		At any time, each on-board memory identifier shall uniquely identify exactly one on-board memory that is unique across the entire spacecraft.	C2	See justification of first requirement in this clause

N	Title	Requirement	C	Justification
c		For each on-board memory, the following characteristics of that memory shall be declared when specifying that memory: 1. the memory access alignment constraint; 2. the memory size, in bytes; 3. the allowed operations; 4. the addressing scheme.	C2	See justification of first requirement in this clause
d		When declaring the characteristics of an on-board memory, the allowed operations shall be one of the following: 1. 'read only'; 2. 'read and write'; 3. 'write only'.	C2	See justification of first requirement in this clause
e		For each on-board memory, whether scrubbing that memory is supported shall be declared when specifying that memory.	C2	See justification of first requirement in this clause
f		For each on-board memory, whether write protecting that memory is supported shall be declared when specifying that memory.	C2	See justification of first requirement in this clause
5.4.3.3.2a	Addressing Scheme	For each on-board memory, whether an absolute addressing scheme for that memory is exposed in the space to ground interface shall be declared when specifying that memory.	C2	See justification of first requirement in the previous clause
b		Absolute addressing implies that the memory addresses and related offsets shall be expressed in bytes.	C2	See justification of first requirement in the previous clause

N	Title	Requirement	C	Justification
c		For each on-board memory, whether a base plus offset addressing scheme for that memory is exposed in the space to ground interface shall be declared when specifying that memory.	C2	See justification of first requirement in the previous clause
d		Base plus offset addressing implies that the base references when expressed as an absolute address and related offsets shall be expressed in bytes.	C2	See justification of first requirement in the previous clause
5.4.3.4a	Virtual channel	The list of virtual channels defined for downlinking reports and their characteristics shall be declared when specifying the space to ground interface.	NA	This requirement does not concern the implementation of the services and is therefore outside the scope of the PUS Extension of the CORDET Framework
b		For each virtual channel defined for downlinking reports, the virtual channel identifier used to refer to that virtual channel shall be declared when specifying that virtual channel.	NA	This requirement does not concern the implementation of the services and is therefore outside the scope of the PUS Extension of the CORDET Framework
5.4.4	Checksum algorithm	For each checksum algorithm used on-board, the list of subservice providers that use that checksum algorithm shall be declared when specifying the spacecraft architecture.	C2	In the PUS Extension of the CORDET Framework, the checksumming of commands and reports is not explicitly modelled and it must be provided by the application in order to compute the CRC field of commands and reports.
5.4.5a	On-board file system	Each on-board file system shall be identified by exactly one on-board file system identifier that is unique across the entire spacecraft.	C2	The on-board file system and their characteristics are defined as part of the instantiation of service 23. See adaptation points TBD.

N	Title	Requirement	C	Justification
b		Each object in an on-board file system shall be uniquely identified by an object path that is the combination of a repository path and an object name.	C2	See justification of first requirement in this clause
c		For each on-board file system, whether that file system supports files with unbounded size shall be declared when specifying that file system.	C2	See justification of first requirement in this clause
v		The set of file attributes supported by each on-board file system shall be declared when specifying that file system.	C2	See justification of first requirement in this clause
e		For each on-board file system, whether that file system provides the capability to lock files shall be declared when specifying file system.	C2	See justification of first requirement in this clause
f		An on-board file system shall not be accessed by more than one file management service.	C2	See justification of first requirement in this clause
5.4.6a	Service	Each service shall be of exactly one service type.	C1	To each CORDET Service, one single type attribute is assigned (see section 4.1.1 of the CORDET Framework Definition Document)
b		For each subservice type whose realization is implicitly required, each service of the related service type shall provide at least one subservice of that subservice type.	C2	The PUS Extension of the CORDET Framework does not enforce this requirement. It is up to the user to instantiate services which are implicitly required.
c		For each subservice type whose realization is required by tailoring and for each service of the service type that defines that subservice type, whether the realization of that subservice type is required for that service shall be declared when specifying that service.	C2	See justification of requirement b in this clause

N	Title	Requirement	C	Justification
d		For each subservice type that allows multiple realizations within a single service, each realization of that subservice type shall be declared when specifying that service.	C2	See justification of requirement b in this clause
e		The service topology of the overall space system shall be declared when specifying the space system architecture.	C1	The service topology of a CORDET System is defined by several adaptation points and application developers are required to fill in all framework adaptation points (or accept their default implementation) as part of the application instantiation process
5.4.7.1a	Subservice	Each subservice shall be of exactly one subservice type.	NA	This and the next requirement are definitions rather than requirement.
b		Each subservice shall belong to exactly one service.	NA	See justification of first requirement in this clause
5.4.7.2.1a	Subservice Entity	Each subservice entity shall belong to exactly one subservice.	NA	This and the next two requirements are definitions rather than requirement.
b		Each subservice entity shall be hosted by exactly one application process.	NA	See justification of first requirement in the previous clause
c		Each subservice entity shall be either a subservice user or a subservice provider.	NA	See justification of first requirement in the previous clause
5.4.7.2.2a	Subservice Provider	Each subservice shall provide exactly one subservice provider.	NA	This requirement is a definition rather than a requirement
5.4.7.2.3a	Subservice User	Each subservice shall provide at least one subservice user.	NA	This requirement is a definition rather than a requirement
5.4.8a	Capability	Each subservice shall provide at least one subservice capability.	NA	This requirement is a definition rather than a requirement

N	Title	Requirement	C	Justification
5.4.8b		For each subservice and for each capability type defined by the corresponding subservice type, the inclusion of the related capability in that subservice shall comply with the applicability constraints of that capability type.	TBD	This requirement is not understood
5.4.9a	Failed progress of execution	For each request type for which a failed progress of execution can be reported, whether the corresponding failed progress of execution notifications are reported within failed progress of execution verification reports or as part of the completion of execution verification report for the related requests shall be declared when specifying the request type related subservice.	C1	The CORDET Framework supports both options but the option whereby the failed progress of execution is reported through a Failed Progress of Execution Notification is the most natural and it is the one which is selected by default in the PUS Extension.
5.4.10a	Transactions	Each subservice shall provide the means to manage all transactions that it initiates according to the mission operational requirements.	C1/C2	The CORDET Framework provides the means to manage incoming and out-going reports and commands. The PUS Extension implements the transaction rules mandated by the PUS.
b		Each transaction shall be initiated and maintained by exactly one subservice.	C1	See justification of previous requirement.
5.4.11.1a	Message	Each message shall be of a single message type.	C1	message is either a report or a request and its type is defined by the pair [service type, service sub-type]. The CORDET Framework directly supports the concepts of service types and sub-types and assigned one single type/sub-type pair to each message.
5.4.11.2.1a	Request	Each request shall be generated by exactly one subservice user.	C1	The CORDET Framework allows a command to have one single source.

N	Title	Requirement	C	Justification
b		Each request shall be addressed to exactly one subservice provider.	C1	The CORDET Framework allows a command to have only one single destination.
c		Each request shall be uniquely identified by a request identifier that is the combination of: 1. a source identifier that corresponds to the application process user identifier of the application process that hosts the subservice user that generates that request; 2. a destination identifier that corresponds to the combination of the application process identifier of the application process that hosts the subservice provider that is responsible for executing that request and the system identifier of the system that hosts that application process; 3. a sequence count or request name that is produced by the application process that hosts the subservice user.	C1	CORDET Commands carry dentifiers of both their source and destination and a source sequence counter (see section 4.1 of the CORDET Framework Definition Document)
d		Each request shall be of exactly one request type.	C1	The type of a request is given by the pair [service type, service sub-type]. The CORDET Framework directly supports both the concept of service type and of service sub-type.
e		Each request whose request type provides a single instruction slot shall contain exactly one instruction that is of an instruction type defined for that request type.	C1	The PUS Extension defines request and instruction types in accordance with the PUS

N	Title	Requirement	C	Justification
f		Each request whose request type provides multiple instruction slots shall contain an ordered list of one or more instructions, each one being of an instruction type defined for that request type.	C1	The PUS Extension defines request and instruction types in accordance with the PUS
5.4.11.2.2a	Acknowledgement	Each request shall contain: 1. a flag indicating whether the reporting of the successful acceptance of that request by the destination application process is requested; 2. a flag indicating whether the reporting of the successful start of execution of that request by the destination application process is requested; 3. a flag indicating whether the reporting of the successful progresses of execution of that request by the destination application process is requested; 4. a flag indicating whether the reporting of the successful completion of execution of that request by the destination application process is requested.	C1	CORDET commands carry four acknowledgement flags which determined which of the four stages of their life-cycle (acceptance, start, progress, and termination) are acknowledged (see section 4.1 of the CORDET Framework Definition Document)

N	Title	Requirement	C	Justification
5.4.11.2.3a	Request execution verification	For each request that it receives, the subservice provider in charge of the execution of that request shall, in sequence: 1. if the pre-conditions for the execution of that request are not fulfilled: (a) notify the execution reporting subservice of its parent application process of the failed start of execution; (b) stop processing that request; 2. if the pre-conditions for the execution of that request are fulfilled, notify the execution reporting subservice of its parent application process of the successful start of execution; 3. for each step, if any: (a) verify the execution conditions of that step, if any; (b) if the execution conditions of that step are not fulfilled, notify the execution reporting subservice of its parent application process of the failed progress of execution of that step; (c) if the step's execution conditions are fulfilled, notify the execution reporting subservice of its parent application process of the successful progress of execution of that step; at the end of the execution of that request: newline (a) verify the post-conditions of execution, if any; (b) if any step execution has failed or if the post-conditions of execution are not fulfilled, notify the execution reporting subservice of its parent application process of the failed completion of execution and stop processing that request; (c) if the post-conditions of execution are fulfilled, notify the execution reporting subservice of its parent application process of the successful completion of execution;	C1	The life-cycle of a CORDET command in a service provider is defined in section 4.1 of the CORDET Framework Definition Document. As requested by this requirement, start, progress and completion of execution of a command are checked and notification may be sent out in response to these checks (see adaptation points ICM-12 to 17). However, failure of a progress step leads to termination of execution of the command. In such a case (failure of a progress step), the originator of the request is notified with one single failure report indicating the failure of the progress and, by implication, also the failure of the command completion. This requirement only concerns reporting of verification outcomes for commands. The PUS is silent about the conditions under which the outcome of instruction-level verifications should be reported. In this respect, the PUS Framework takes the approach that, for instructions, only execution failures are reported and that they are reported unconditionally.

N	Title	Requirement	C	Justification
5.4.11.3.1a	Report	Each report shall be generated by exactly one subservice provider.	C1	In the CORDET Framework, both reports and commands have one single source
b		Each report shall be addressed to exactly one subservice user.	C1	In the CORDET Framework, both reports and commands have one single destination
c		Each report shall be uniquely identified by a report identifier that is the combination of: 1. a source identifier that is the application process identifier of the application process that hosts the subservice provider that generates that report; 2. a destination identifier that corresponds to the application process user identifier of the application process that hosts the subservice user that is responsible for processing that report; 3. a source sequence count that is produced by the application process that hosts the subservice provider.	C1	CORDET reports carry identifiers of both their source and destination and a source sequence counter (see section 4.2 of the CORDET Framework Definition Document)
d		Each report shall be of exactly one report type.	C1	The type of a CORDET report is given by the pair [type, sub-type].
e		Each report whose report type provides a single notification slot shall contain exactly one notification that is of a notification type defined for that report type.	C1	The PUS Extension defines report and notification types in accordance with the PUS

N	Title	Requirement	C	Justification
f		Each report whose report type provides multiple notification slots shall contain an ordered list of one or more notifications, where: 1. all notifications in the list are of the same notification type, and 2. that notification type is one of those defined for that report type.	C1	The PUS Extension defines report and notification types in accordance with the PUS
5.4.11.3.2a	Response	The destination of any response shall be the source of the corresponding request.	C1/C2	For pre-defined services, the PUS is followed. For application-specific services, this requirement must be enforced by application developers when they close adaptation point OCM-9.
		If a request implies the generation of a response that exceeds the length that can be carried in a telemetry packet of the maximum packet size of the CCSDS space packet protocol, that request shall be rejected.	C1/C2	For pre-defined services, the PUS is followed. For application-specific services, this requirement must be enforced by application developers when they define the Start Action for commands (see adaptation point ICM-12).
5.4.11.3.3a	Data Report	For each data report that can be generated in an autonomous data reporting transaction, the destination of the data report in that case shall be declared when specifying the related subservice.	C1/C2	For pre-defined services, the PUS is followed. For application-specific services, this requirement must be enforced by application developers when they close adaptation point OCM-9.

N	Title	Requirement	C	Justification
5.4.12a	Building the space system architecture	Deploying the service topology of an overall space system should consist of: 1. specifying new implementations of PUS services by instantiating the service types and related components; 2. assessing the adequacy of reusing existing service implementations: (a) ensuring their compliance to the PUS standard services; (b) verifying their compliance to the overall system constraints.	NA	This requirement does not concern the implementation of the services and is therefore outside the scope of the PUS Extension of the CORDET Framework

N	Title	Requirement	C	Justification
6.1.2.1a	Request Verification	Each request verification service shall contain at least one of the following: 1. one or more routing and reporting subservices, 2. one or more acceptance and reporting subservices, 3. one or more execution reporting subservices.	C1	The PUS Extension of the CORDET Framework provides adaptation points which allow an application to deploy: (a) one routing and reporting subservice (this is part of the InLoader component which, in the CORDET Framework, is responsible for routing incoming packets, see adaptation points ILD-9, 11 and 12 in [CR-SP]); (b) one acceptance and reporting sub-service (this is part of the InLoader Load Command/Report Procedure which, in the CORDET Framework is responsible for the acceptance of incoming packets, see adaptation points ILD-14 and 15); and (c) one execution reporting sub-service for each command type (since, in the CORDET Framework, execution checks and reporting of their outcomes are done in the InCommand components which encapsulate incoming commands, see adaptation points ICM-12 to 17). The PUS Extension closes these adaptation points to provide an implementation which conforms to the requirements of service 1 in the PUS.
6.1.2.2.1a	Destination of verification reports	For each verification report that it generates, the application process shall address that report to the application process that hosts the subservice user that has generated the corresponding request.	C1	See close-out of adaptation point P-S1-10

N	Title	Requirement	C	Justification
6.1.2.2.2a	Application process that routes requests	Each application process that is involved in routing requests shall host exactly one routing and reporting subservice.	C1	See justification of clause 6.1.2.1a
6.1.2.2.3a	Application process that executes requests	Each application process that hosts one or more subservices that execute requests shall host: 1. exactly one acceptance and reporting subservice; 2. at most one execution reporting subservice.	C1	See justification of clause 6.1.2.1a.
6.1.3.1.1a	Application Process	The list of application processes that the routing and reporting subservice addresses shall be declared when specifying the spacecraft architecture.	C2	This list is implicitly declared when, during the framework instantiation process, the adaptation point ILD-11 of the CORDET Framework is closed.
6.1.3.2a	Routing verification of a request	The routing and reporting subservice shall provide the capability to perform routing verification for the requests that it receives.	C1	In the CORDET Framework, routing of incoming packets is done by the InLoader component (see [CR-SP]). This component also verifies the validity of the command destination and routing information.
b		The list of routing verification checks that the routing and reporting subservice performs shall be declared when specifying that subservice.	C1	In the CORDET Framework, routing of incoming packets is done by the InLoader component (see [CR-SP]) which performs one single routing check to verify the validity of an incoming command or report. This check is implemented by closing adaptation point ILD-11.

N	Title	Requirement	C	Justification
c		For each request that it receives, the routing and reporting subservice shall: 1. perform the routing verification checks on that request; 2. determine, based on the output of those checks, whether the routing verification of that request has succeeded or failed.	C1	In the CORDET Framework, routing verification is performed by the InLoader. The routing check is implemented by closing adaptation point ILD-11 and the reporting of a routing failure is implemented by closing adaptation point ILD-12. The PUS Extension offers a component which closes adaptation point ILD-12 by generating a (1,10) report.
6.1.3.3a	Reporting Failed Routing	The routing and reporting subservice shall provide the capability to report the failed routing of requests.	C1	This capability is provided by the VerFailedRoutingRep component of the PUS Extension of the CORDET Framework
		Each failed routing verification report shall contain exactly one failed routing notification.	C1	See definition of VerFailedRoutingRep component of the PUS Extension of the CORDET Framework.
		Each failed routing notification shall contain: 1. the identifier of the request that failed the routing verification; 2. the failure notice made of: (a) a failure code; auxiliary data, if any, used to explain the reason for the failed routing.	C1	The specification of the (1,10) report provided by the PUS Extension follows the PUS. The auxiliary information is specified in the Packet Rerouting Failure Procedure.
		The list of failure codes defined for failed routing notifications shall be declared when specifying the routing and reporting subservice.	C1	The list of failure codes and their auxiliary data for service 1 reports is specified in appendix B of the PUS Specification Document
		For each failure code defined for failed routing notifications, the associated auxiliary data shall be declared when specifying the routing and reporting subservice.		See previous requirement

N	Title	Requirement	C	Justification
		For each request that fails its routing verification, the routing and reporting subservice shall: 1. generate a single failed routing notification and associated report for that request; 2. discard that request.	C1	See definition of Packet Rerouting Failure Procedure
6.1.4.1a	Acceptance verification of a request	The acceptance and reporting subservice shall provide the capability to perform acceptance verification for a request that it receives.	C1	In the CORDET Framework, the acceptance verification is performed by the InLoader Load Command/Report Procedure. Incoming commands are encapsulated in components of type InCommand. Each such component offers a Configuration Check (see adaptation point ICM-3) where the acceptance verification check is implemented. For the commands provided by the PUS Extension of the CORDET Framework, the verification check is specified at adaptation point P-PCR-21.
b		The list of acceptance verification checks that the acceptance and reporting subservice performs during the acceptance verification of a request shall be declared when specifying that subservice.	C2	For all commands supported by the PUS Extension of the CORDET Framework, the InLoader Load Command/Report procedure which performs two acceptance checks: (a) check of the legality of the command type, and (b) check of available resources for the command in the host applications. The PUS Extension adds two more acceptance checks (see close-out of adaptation point ILD-13): (c) check of the command checksum and (d) check of the command length.

N	Title	Requirement	C	Justification
c		For each request that it receives, the acceptance and reporting subservice shall: 1. perform the acceptance verification checks on that request; 2. determine, based on the output of those checks, whether the acceptance verification of that request has succeeded or failed.	C1	In the CORDET Framework, the acceptance check for an incoming command is split into two parts: the first part is done by the InLoader and the second part of done by the InCommand component. See also justification to previous requirement.
6.1.4.2a	Reporting Successful Acceptance	The acceptance and reporting subservice shall provide the capability to report the successful acceptance verification of requests.	C1	See Operation to Report Acceptance Success (Adaptation Point ILD-13)
b		Each successful acceptance verification report shall contain exactly one successful acceptance notification.	C1	See definition of InLoader component: the operation to Report Acceptance Success is called once for each incoming command which passes its acceptance check
c		Each successful acceptance notification shall contain: 1. the identifier of the request that successfully passed the acceptance verification.	C1	The specification of the content of the (1,1) reports offered by the PUS Extension of the CORDET Framework follows the PUS
d		For each request that successfully passes its acceptance verification, the acceptance and reporting subservice shall: 1. if the successful acceptance reporting is requested, generate a single successful acceptance notification and associated report for that request.	C1	See definition of InLoader component in the CORDET Framework.
6.1.4.3a	Reporting failed acceptance	The acceptance and reporting subservice shall provide the capability to report the failed acceptance of requests.	C1	The PUS Extension of the CORDET Framework implements service 1 in full (see requirements P-S1-*).

N	Title	Requirement	C	Justification
b		Each failed acceptance verification report shall contain exactly one failed acceptance notification.	C1	Each service 1 report provided by the PUS Extension of the CORDET Framework covers one single command failure.
c		Each failed acceptance notification shall contain: 1. the identifier of the request that failed the acceptance verification; 2. the failure notice made of: (a) a failure code; (b) auxiliary data, if any, used to explain the reason for the failed acceptance.	C1	The specification of the service 1 reports offered by the PUS Extension of the CORDET Framework follows the PUS
d		The list of failure codes defined for failed acceptance notifications shall be declared when specifying the acceptance and reporting subservice.	C1/C2	The failure codes defined at the level of the PUS Extension of the CORDET Framework are defined in requirement P-S1-13 but applications may define additional failure codes.
e		For each failure code defined for failed acceptance notifications, the associated auxiliary data shall be declared when specifying the acceptance and reporting subservice.	C1	For each acceptance failure report, one single item of auxiliary data may be defined. For the failure codes supported by the PUS Extension of the CORDET Framework, these are specified in requirement P-S1-13.
f		For each request that fails its acceptance verification, the acceptance and reporting subservice shall: 1. generate a single failed acceptance notification and associated report for that request; 2. discard that request.	C1	The generation of the failure notification and the discarding of requests which fail their acceptance check is done by the InLoader component of the CORDET Framework.

N	Title	Requirement	C	Justification
6.1.5.1.1a	Reporting successful start of execution	The execution reporting subservice shall provide the capability to generate the successful start of execution verification reports.	C1	The PUS Extension of the CORDET Framework implements service 1 in full (see requirements P-S1-*).
b		For each successful start of execution notification that it receives, the execution reporting subservice shall: 1. if the successful start of execution reporting is requested, generate a single successful start of execution verification report containing that notification.	C1	See definition of report component VerStartSucc. Note that the processing of a incoming command can result in at most one single Successful Start of Execution Notification.
6.1.5.1.2a	Reporting failed start of execution	The execution reporting subservice shall provide the capability to generate the failed start of execution verification reports.	C1	The PUS Extension of the CORDET Framework implements service 1 in full (see requirements P-S1-*).
b		For each failed start of execution notification that it receives, the execution reporting subservice shall: 1. generate a single failed start of execution verification report containing that notification.	C1	See definition of report component VerStartFailed. Note that a command whose start of execution fails is discarded (i.e. the command terminates with the generation of a (1,4) report). Note also that the processing of a incoming command can result in at most one single Failed Start of Execution Notification.
6.1.5.2.1a	Reporting successful progress of execution	The execution reporting subservice shall provide the capability to generate the successful progress of execution verification reports.	C1	The PUS Extension of the CORDET Framework implements service 1 in full (see requirements P-S1-*).

N	Title	Requirement	C	Justification
b		For each successful progress of execution notification that it receives, the execution reporting subservice shall: 1. if the successful progress of execution reporting is requested, generate a single successful progress of execution verification report containing that notification.	C1/C2	See definition of report component VerPrgrSucc. The definition of the progress steps is under the responsibility of applications (see adaptation point P-S1-7)
6.1.5.2.2a	Reporting failed progress of execution	The execution reporting subservice shall provide the capability to generate the failed progress of execution verification reports.	C1	The PUS Extension of the CORDET Framework implements service 1 in full (see requirements P-S1-*).
b		For each failed progress of execution notification that it receives, the execution reporting subservice shall: 1. if the application process that hosts the execution reporting subservice is configured for the corresponding request type to report the failed progress of execution notifications in failed progress of execution verification reports, generate a single failed progress of execution verification report containing that notification.	C1	See definition of report component VerPrgrFailed. Note that a command whose progress of execution fails is discarded (i.e. the command terminates with the generation of a (1,6) report).
6.1.5.3.1a	Reporting successful completion of execution	The execution reporting subservice shall provide the capability to generate the successful completion of execution verification reports.	C1	The PUS Extension of the CORDET Framework implements service 1 in full (see requirements P-S1-*).

N	Title	Requirement	C	Justification
b		For each successful completion of execution notification that it receives, the execution reporting subservice shall: 1. if the successful completion of execution reporting is requested, generate a single successful completion of execution verification report containing that notification.	C1	See definition of report component VerTermSucc.
6.1.5.3.2a	Reporting failed completion of execution	The execution reporting subservice shall provide the capability to generate the failed completion of execution verification reports.	C1	The PUS Extension of the CORDET Framework implements service 1 in full (see requirements P-S1-*).
b		For each failed completion of execution notification that it receives, the execution reporting subservice shall: 1. generate a single failed completion of execution verification report containing that notification.	C1	See definition of report component VerTermFailed.

N	Title	Requirement	C	Justification
c		For each failed completion of execution notification that is accompanied of failed progress of executions notifications to be reported as part of the completion of execution verification report, the execution reporting subservice shall include those failed progress of execution notifications in the failed completion of execution notification.	TBD	By default, in the PUS Extension of the CORDET Framework, the failure of progress of execution is reported through a (1,6) report and the generation of the (1,6) report excludes the generation of the (1,8) report. See justification of compliance to clause 5.4.9a. NB: The practical implications of this requirement are not understood. Suppose that we have a situation where a command has generated five Failed-Progress-Of-Execution notifications and that it has been agreed that these must be included in the Failed-Completiong-Of-Execution notification. I assume that this means that the TM(1,8) for this command will have to somehow include the five failure codes (and any associated auxiliary data) for the five Failed-Progress-Of-Execution notifications. But how can this be done in view of the fact that the layout of the TM(1,8) in clause 8.1.2.8 only includes one single Failure Notice?
6.2	Device Access	Definition of service 2	n.a.	This service is not yet supported by the PUS Extension of the CORDET Framework
6.3.2.1.1a	Housekeeping reporting subservice	Each housekeeping service shall contain at least one housekeeping reporting subservice.	C1	The PUS Extension of the CORDET Framework includes support for the reporting subservice

N	Title	Requirement	C	Justification
6.3.2.1.2a	Diagnostic reporting subservice	Each housekeeping service shall contain zero or more diagnostic reporting subservices.	C1	The PUS Extension of the CORDET Framework includes support for the diagnostic subservice
6.3.2.1.3a	Parameter functional reporting configuration subservice	Each housekeeping service shall contain at most one parameter functional reporting configuration subservice.	n.a.	The PUS Extension of the CORDET Framework does not support the parameter functional configuration subservice
6.3.2.2.1a	Housekeeping reporting subservice	Each application process shall host at most one housekeeping reporting subservice provider.	C1	The PUS Extension of the CORDET Framework supports one housekeeping reporting subservice per application
6.3.2.2.2a	Diagnostic reporting subservice	Each application process shall host at most one diagnostic reporting subservice provider.	C2	The PUS Extension of the CORDET Framework supports one diagnostic reporting subservice per application
6.3.2.2.3a	Parameter functional reporting configuration subservice	Each application process shall host at most one parameter functional reporting configuration subservice provider.	n.a.	The PUS Extension of the CORDET Framework does not support the parameter functional configuration subservice
6.3.3.1	Parameter accessibility	The housekeeping reporting subservice shall be able to collect and report the sampled values of each on-board parameter that is accessible to the application process that hosts that subservice.	C1	The housekeeping reports report the values of the data items in the data pool which contain all application parameters and variables

N	Title	Requirement	C	Justification
6.3.3.2a	Housekeeping parameter report structure	The on-board resources allocated to the housekeeping reporting subservice to host the housekeeping parameter report structures shall be declared when specifying that subservice.	C2	The on-board resource dedicated to the housekeeping service is the data pool whose content must be defined by the user during the framework instantiation process (adaptation point P-DP-7). Additionally, the application developer must size the Report Definition List (RDL) data structure by defining the values of the constants HK_*.
b		The on-board resources allocated to the contemporaneous evaluation of housekeeping parameter report structures used by the housekeeping reporting subservice shall be declared when specifying that subservice.	C2	See previous requirement

N	Title	Requirement	C	Justification
c		Each housekeeping parameter report structure shall consist of: 1. a housekeeping parameter report structure identifier; 2. the collection interval used to generate the corresponding reports; 3. an ordered list of zero or more simply commutated parameters; 4. an ordered list of zero or more super commutated parameter sets, each set consisting of: (a) the number of sampled values to report for each parameter of that set, and (b) the ordered list of one or more parameters contained within that set; if the housekeeping reporting subservice provides the capability for managing the periodic generation of housekeeping parameter reports, a status indicating whether the periodic generation action of the corresponding housekeeping parameter reports is enabled or disabled.	C1	See specification of Report Definition List (RDL)
6.3.3.3a	Housekeeping parameter report structure	The housekeeping reporting subservice shall provide the capability for generating housekeeping parameter reports.	C1	The PUS Extension of the CORDET Framework supports reports (3,25)
b		Each housekeeping parameter report shall contain exactly one housekeeping parameter notification.	C1	See definition of hkRep component

N	Title	Requirement	C	Justification
c		Each housekeeping parameter notification shall contain: 1. the housekeeping parameter report structure identifier; 2. in the specified order for simply commutated parameters, a single sampled value for each simply commutated parameter; 3. in the specified order for super commutated parameter sets, for each super commutated parameter set: (a) the 'super commutated sample repetition number' sets of sampled values.	C1	See definition of hkRep component
d		For each housekeeping parameter report structure for which periodic generation is enabled, the housekeeping reporting subservice shall generate a corresponding housekeeping parameter report periodically, according to the collection interval specified for that definition.	C1/C2	See definition of HkRep component encapsulating a housekeeping report. Users are responsible for allocating the instances of this component to OutManager components which are executed with a frequency corresponding to the report's collection period.
e		For each housekeeping parameter report structure for which periodic generation is enabled, the housekeeping reporting subservice shall collect one sampled value for each simply commutated parameter during the collection interval specified for the corresponding housekeeping parameter report structure.	C1	See definition of hkRep component

N	Title	Requirement	C	Justification
f		For each housekeeping parameter report structure for which periodic generation is enabled, the housekeeping reporting subservice shall collect all sampled values for each super commutated parameter during the collection interval specified for the corresponding housekeeping parameter report structure, in accordance with a sub-period equal to the collection interval divided by the corresponding 'super commutated sample repetition number'.	C1/C2	The framework provides the Sampling Buffer as a data structure to hold super-commutated data items but the user is responsible for filling it with the sampled values of the super-commutated data items (see requirement P-S3-6)
6.3.3.4.1a	Enable the periodic generation of housekeeping parameter reports	The housekeeping reporting subservice capability to enable the periodic generation of housekeeping parameter reports shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports command (3,5)
b		Each request to enable the periodic generation of housekeeping parameter reports shall contain one or more instructions to enable the periodic generation of a housekeeping parameter report.	C1	See definition of HkEnable component
c		Each instruction to enable the periodic generation of a housekeeping parameter report shall contain: 1. the housekeeping parameter report structure identifier to enable.	C1	See definition of HkEnable component
d		The housekeeping reporting subservice shall reject any instruction to enable the periodic generation of a housekeeping parameter report if: 1. that instruction refers to a housekeeping parameter report structure that is unknown.	C1	See definition of Start Action of HkEnable component

N	Title	Requirement	C	Justification
e		For each instruction to enable the periodic generation of a housekeeping parameter report that it rejects, the housekeeping reporting subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of HkEnable component
f		The housekeeping reporting subservice shall process any valid instruction that is contained within a request to enable the periodic generation of housekeeping parameter reports regardless of the presence of faulty instructions.	C1	See definition of Progress Action of HkEnable component
g		For each valid instruction to enable the periodic generation of a housekeeping parameter report, the housekeeping reporting subservice shall: 1. set the periodic generation action status of that housekeeping parameter report structure to 'enabled'.	C1	See definition of Progress Action of HkEnable component
6.3.3.4.2a	Disable the periodic generation of housekeeping parameter reports	The housekeeping reporting subservice shall provide the capability to disable the periodic generation of housekeeping parameter reports if the capability to enable the periodic generation of housekeeping parameter reports is provided by that subservice.	C1	The PUS Extension of the CORDET Framework supports command (3,6)
b		Each request to disable the periodic generation of housekeeping parameter reports shall contain one or more instructions to disable the periodic generation of a housekeeping parameter report.	C1	See definition of HkDisable component

N	Title	Requirement	C	Justification
c		Each instruction to disable the periodic generation of a housekeeping parameter report shall contain: 1. the housekeeping parameter report structure identifier to disable.	C1	See definition of HkDisable component
d		The housekeeping reporting subservice shall reject any instruction to disable the periodic generation of a housekeeping parameter report if: 1. that instruction refers to a housekeeping parameter report structure that is unknown.	C1	See definition of Start Action of HkDisable component
e		For each instruction to disable the periodic generation of a housekeeping parameter report that it rejects, the housekeeping reporting subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of HkDisable component
f		The housekeeping reporting subservice shall process any valid instruction that is contained within a request to disable the periodic generation of housekeeping parameter reports regardless of the presence of faulty instructions.	C1	See definition of Progress Action of HkDisable component
g		For each valid instruction to disable the periodic generation of a housekeeping parameter report, the housekeeping reporting subservice shall: 1. set the periodic generation action status of that housekeeping parameter report structure to 'disabled'.	C1	See definition of Progress Action of HkDisable component

N	Title	Requirement	C	Justification
6.3.3.5.1a	Create a housekeeping parameter report structure	The housekeeping reporting subservice capability to create a housekeeping parameter report structure shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports command (3,1)
b		Each request to create a housekeeping parameter report structure shall contain exactly one instruction to create a housekeeping parameter report structure.	C1	See definition of HkCreate component
c		Each instruction to create a housekeeping parameter report structure shall contain: 1. the housekeeping parameter report structure identifier to create; 2. the collection interval; 3. the list of simply commutated parameters in the required order; 4. the list of super commutated parameter sets in the required order.	C1	See definition of HkCreate component
d		The housekeeping reporting subservice shall reject any request to create a housekeeping parameter report structure if any of the following conditions occurs: 1. that request contains an instruction that refers to a housekeeping parameter report structure that is already in use; 2. the same parameter is identified more than once in that request; 3. the resources allocated to the hosting of housekeeping parameter report structures are exceeded.	C1	See definition of Start Action of HkCreate component

N	Title	Requirement	C	Justification
e		For each request to create a housekeeping parameter report structure that is rejected, the housekeeping reporting subservice shall generate a failed start of execution notification.	C1	See definition of Start Action of HkCreate component
f		For each valid instruction to create a housekeeping parameter report structure, the housekeeping reporting subservice shall: 1. create that definition; 2. set its periodic generation action status to ’disabled’.	C1	See definition of Progress Action of HkCreate component
6.3.3.5.2a	Delete housekeeping parameter report structures	The housekeeping reporting subservice shall provide the capability to delete housekeeping parameter report structures if the capability to create a housekeeping report definition is provided by that subservice.	C1	The PUS Extension of the CORDET Framework supports command (3,3)
b		Each request to delete housekeeping parameter report structures shall contain one or more instructions to delete a housekeeping parameter report structure.	C1	See definition of HkDelete component
c		Each instruction to delete a housekeeping parameter report structure shall contain: 1. the housekeeping parameter report structure identifier to delete.	C1	See definition of HkDelete component

N	Title	Requirement	C	Justification
d		The housekeeping reporting subservice shall reject any instruction to delete a housekeeping parameter report structure if any of the following conditions occurs: 1. that instruction refers to a housekeeping parameter report structure that is unknown; 2. that instruction refers to a housekeeping parameter report structure whose periodic generation action status is 'enabled'.	C1	See definition of Start Action of HkDelete component
e		For each instruction to delete a housekeeping parameter report structure that it rejects, the housekeeping reporting subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of HkDelete component
f		The housekeeping reporting subservice shall process any valid instruction that is contained within a request to delete housekeeping parameter report structures regardless of the presence of faulty instructions.	C1	See definition of Start Action of Progress Action of HkDelete component
g		For each valid instruction to delete a housekeeping parameter report structure, the housekeeping reporting subservice shall: 1. delete the housekeeping parameter report structure referred to by that instruction.	C1	See definition of Start Action of Progress Action of HkDelete component
6.3.3.6a	Report housekeeping parameter report structures	The housekeeping reporting subservice capability to report housekeeping parameter report structures shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports both command (3,9) and report (3,10)

N	Title	Requirement	C	Justification
b		Each request to report housekeeping parameter report structures shall contain one or more instructions to report a housekeeping parameter report structure.	C1	See definition of HkRepStructCmd component
c		Each instruction to report a housekeeping parameter report structure shall contain: 1. the housekeeping parameter report structure identifier to report.	C1	See definition of HkRepStructCmd component
d		The housekeeping reporting subservice shall reject any instruction to report a housekeeping parameter report structure if: 1. that instruction refers to a housekeeping parameter report structure that is unknown.	C1	See definition of Start Action of HkRepStructCmd component
e		For each instruction to report a housekeeping parameter report structure that it rejects, the housekeeping reporting subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of HkRepStructCmd component
f		The housekeeping reporting subservice shall process any valid instruction that is contained within a request to report housekeeping parameter report structures regardless of the presence of faulty instructions.	C1	See definition of Progress Action of HkRepStructCmd component

N	Title	Requirement	C	Justification
g		For each valid instruction to report a housekeeping parameter report structure, the housekeeping reporting subservice shall generate a single housekeeping parameter report structure report that contains exactly one housekeeping parameter report structure notification that includes: 1. the housekeeping parameter report structure identifier; 2. If the housekeeping reporting subservice provides the capability for managing the periodic generation of housekeeping parameter reports, the periodic generation action status; 3. the collection interval; 4. the ordered list of simply commutated parameters; 5. the ordered list of super commutated parameter sets. 86	C1	See definition of Progress Action of HkRepStructCmd component and definition of HkRepStructRep component. With respect to point 2, it is noted that the periodic generation of housekeeping reports is supported by the service 3 implementation of the PUS Extension of the CORDET Framework
6.3.3.7a	Generate a one shot report for housekeeping parameter report structures	The housekeeping reporting subservice capability to generate a one shot report for housekeeping parameter report structures shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports both command (3,27) and report (3,25)
b		Each request to generate a one shot report for housekeeping parameter report structures shall contain one or more instructions to generate a one shot report for a housekeeping parameter report structure.	C1	See definition of HkOneShotRep component

N	Title	Requirement	C	Justification
c		Each instruction to generate a one shot report for a housekeeping parameter report structure shall contain: 1. the housekeeping parameter report structure identifier of the report to generate.	C1	See definition of HkOneShotRep component
d		The housekeeping reporting subservice shall reject any instruction to generate a one shot report for a housekeeping parameter report structure if: 1. that instruction refers to a housekeeping parameter report structure that is unknown.	C1	See definition of Start Action of HkOneShotRep component
e		For each instruction to generate a one shot report for a housekeeping parameter report structure that it rejects, the housekeeping reporting subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of HkOneShotRep component
f		The housekeeping reporting subservice shall process any valid instruction that is contained within a request to generate a one shot report for housekeeping parameter report structures regardless of the presence of faulty instructions.	C1	See definition of Progress Action of HkOneShotRep component
g		For each valid instruction to generate a one shot report for a housekeeping parameter report structure, the housekeeping reporting subservice shall generate a single housekeeping parameter report.	C1	See definition of Progress Action of HkOneShotRep component

N	Title	Requirement	C	Justification
6.3.3.8	Append parameters to a housekeeping parameter report structure		n.a.	This capability is not supported by the PUS Extension of the CORDET Framework
6.3.3.9	Modify the collection interval of housekeeping parameter report structures		n.a.	This capability is not yet supported by the PUS Extension of the CORDET Framework
6.3.310	Report the periodic generation properties of housekeeping parameter report structures		n.a.	This capability is not yet supported by the PUS Extension of the CORDET Framework
6.3.4	Diagnostic reporting subservice		C1/C2	The components offerd by the PUS Extension to implement housekeeping-related capabilities also implement the corresponding diagnostic-related capability. Hence, the level of compliance to the diagnostic reporting subservice requirements is the same as the level of compliance to the homologous housekeeping reporting sub-service requirements.
6.3.5	Parameter functional reporting configuration subservice		n.a.	This subservice is not supported by the PUS Extension of the CORDET Framework

N	Title	Requirement	C	Justification
6.4	Parameter Statistics Reporting	Definition of service 4	n.a.	This service is not supported by the PUS Extension of the CORDET Framework
6.5	Event Reporting	Definition of service 5		
6.5.2.1.1a	Event reporting subservice	Each event reporting service shall contain at least one event reporting subservice.	C1	The PUS Extension of the CORDET Framework supports service 5 in full
6.5.2.2a	Application process	Each application process shall host at most one event reporting subservice provider.	C1	An application instantiated from the CORDET Framework can only provide one instance of a service of a given type and of its sub-services
6.5.3a	Event Definition	The list of events that can be detected by the event reporting subservice shall be declared when specifying that subservice.	C2	The set of events that can be reported by the service 5 implementation of the PUS Extension of the CORDET Framework is an adaptation point (see table **??**)
b		For each event that can be detected by the event reporting subservice, the event definition used to report on the occurrences of that event, the related event severity level, the event definition identifier and, if any, auxiliary data shall be declared when specifying that subservice.	C2	Application developers must provide this information in order to define the EvtRep components which encapsulate their event reports (see table **??**)
c		Each event definition shall be uniquely identified by the combination of the identifier of the application process that hosts the event reporting subservice provider that is in charge to report on the occurrences of the associated event and an event definition identifier.	C2	Both the definition of the APIDs and of the EIDs are under the responsibility of application developers
6.5.4a	Event Reporting	The event reporting subservice shall provide the capability to generate event reports.	C1	See definition of the EvtRep component in table **??**

N	Title	Requirement	C	Justification
b		The destination of the event reports generated by the event reporting subservice shall be declared when specifying that subservice.	C2	The destination of an event report must be specified by an application at run-time when the event is configured. See definition of EvtRep component in table **??**
c		If the event reporting subservice supports the capability for controlling the generation of event reports specified in clause 6.5.5, that subservice shall generate an event notification whenever it detects the occurrence of an event associated to an event definition for which event report generation is enabled.	C1	The PUS Extension of the CORDET Framework provides the capability to enable and disable event reports (see definition of EvtEnableCmd and EvtDisableCmd components)
d		If the event reporting subservice does not support the capability for controlling the generation of event reports specified in clause 6.5.5, that subservice shall generate an event notification whenever it detects the occurrence of an event.	n.a.	See previous requirement
e		Each event notification shall contain: 1. the event definition identifier of the associated event definition; 2. the auxiliary data associated to that event definition, if any.	C1	See definition of the EvtRep component
f		For each event notification that it generates, the event reporting subservice shall generate an event report of the related event severity level, which contains that notification.	C1	The event notification is encapsulated in the EvtRep component. The processing of this component by the CORDET Framework results in the generation of the corresponding event report

N	Title	Requirement	C	Justification
6.5.5.1a	Event report generation status	For each event that can be detected by the event reporting subservice, the subservice shall maintain a status indicating whether the event report generation for that event is enabled or disabled.	C1	The CORDET Framework provides the capability to track the enable status of any out-going command or report (see OutRegistry component in [CR-SP])
b		For each event that can be detected by the event reporting subservice, the initial enabled or disabled event report generation status shall be declared when specifying that subservice.	C1/C2	The default enable status of all out-going commands or reports in the CORDET Framework is: 'enabled'. A different initial value can be achieved by configuring the OutRegistry during the application initialization phase.
6.5.5.2a	Enable the report generation of event definitions	The event reporting subservice capability to enable the report generation of event definitions shall be declared when specifying that subservice.	C1	The PUS Extension supports command (5,5)
b		Each request to enable the report generation of event definitions shall contain one or more instructions to enable the report generation of an event definition.	C1	See definition of the EvtEnableCmd component
c		Each instruction to enable the report generation of an event definition shall contain: 1. the event definition identifier of the event definition to enable.	C1	See definition of the EvtEnableCmd component
d		The event reporting subservice shall reject any instruction to enable the report generation of an event definition if: 1. that instruction refers to an unknown event definition.	C1	See definition of Start Action of EvtEnableCmd component

N	Title	Requirement	C	Justification
e		For each instruction to enable the report generation of an event definition that it rejects, the event reporting subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of EvtEnableCmd component
f		The event reporting subservice shall process any valid instruction that is contained within a request to enable the report generation of event definitions regardless of the presence of faulty instructions.	C1	See definition of the EvtEnableCmd component
g		For each valid instruction to enable the report generation of an event definition, the event reporting subservice shall: set the event report generation status of the event definition to enabled.	C1	See definition of Progress Action of EvtEnableCmd component
6.5.5.3a	Disable the report generation of event definitions	The event reporting subservice shall provide the capability to disable the report generation of event definitions if the capability to enable the report generation of event definitions is provided by that subservice.	C1	The PUS Extension supports command (5,6)
b		Each request to disable the report generation of event definitions shall contain one or more instructions to disable the report generation of an event definition.	C1	See definition of the EvtDisableCmd component
c		Each instruction to disable the report generation of an event definition shall contain: 1. the event definition identifier of the event definition to disable.	C1	See definition of the EvtDisableCmd component

N	Title	Requirement	C	Justification
d		The event reporting subservice shall reject any instruction to disable the report generation of an event definition if: 1. that instruction refers to an unknown event definition.	C1	See definition of Start Action of EvtDisableCmd component
e		For each instruction to disable the report generation of an event definition that it rejects, the event reporting subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of EvtDisableCmd component
f		The event reporting subservice shall process any valid instruction that is contained within a request to disable the report generation of event definitions regardless of the presence of faulty instructions.	C1	See definition of the EvtDisableCmd component
g		For each valid instruction to disable the report generation of an event definition, the event reporting subservice shall: set the event report generation status of the event definition to disabled.	C1	See definition of Progress Action of EvtDisableCmd component
6.5.5.4a	Report the list of disabled event definitions	The event reporting subservice capability to report the list of disabled event definitions shall be declared when specifying that subservice.	C1	The PUS Extension supports command (5,7)
b		Each request to report the list of disabled event definitions shall contain exactly one instruction to report the list of disabled event definitions.	C1	See definition of the EvtRepDisabledCmd component

N	Title	Requirement	C	Justification
c		For each valid instruction to report the list of disabled event definitions, the event reporting subservice shall: generate, for each event definition whose event report generation status is disabled, a single disabled event definition notification that includes:the related event definition identifier.	C1	See definition of the EvtRepDisabledCmd component
d		For each valid request to report the list of disabled event definitions, the event reporting subservice shall generate a single disabled event definitions list report that includes all related disabled event definition notifications.	C1	See definition of the EvtRepDisabledCmd and EvtRepDisabledRep components
6.5.6	Subservice observables	The following observables shall be defined for the event reporting subservice: 1. per severity level: (a) the accumulated number of detected event occurrences, (b) the number of event definitions whose event report generation status is disabled, (c) the accumulated number of generated event reports, (d) the event definition identifier of the last generated event Report, (e) the generation time of the last event report.	C1	See definition of Observable Data Items associated to service 5
6.6	Memory Management	Definition of service 6	n.a.	This service is not yet supported by the PUS Extension of the CORDET Framework
6.8	Function Management	Definition of service 8	n.a.	This service is not yet supported by the PUS Extension of the CORDET Framework

N	Title	Requirement	C	Justification
6.9	Time Management	Definition of service 9	n.a.	This service is not yet supported by the PUS Extension of the CORDET Framework
6.11	Time-Base Scheduling	Definition of service 11		
6.11.2.1.1a	Time-based scheduling subservice	Each time-based scheduling service shall contain at least one time-based scheduling subservice.	C1	The PUS Extension of the CORDET Framework supports the time-based scheduling sub-service and it allows one instance of the service to be deployed in an application. The service only contains one instance of its sub-service.
6.11.2.2a	Application process	Each application process shall host at most one time-based scheduling subservice provider.	C1	See statement of compliance to previous clause
6.11.3.1a	Application process	The list of application processes that can be addressed by the time-based scheduling subservice when releasing requests shall be declared when specifying that subservice.	C1	TBD
6.11.4.1a	Capability	Whether the time-based scheduling subservice supports the capability for managing sub-schedules shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports the sub-schedule concept
b		Whether the time-based scheduling subservice supports the capability for managing groups specified shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports the group concept

N	Title	Requirement	C	Justification
6.11.4.2a	General	Each scheduled activity definition shall consist of: 1. the request; 2. the release time of that request; 3. if sub-schedules are supported, the identifier of the sub-schedule to which that scheduled activity is associated; 4. if groups are supported, the identifier of the group to which that scheduled activity is associated.	C1	See definition of attributes of a time-based activity or TBA
b		Each scheduled activity definition shall be identified by a scheduled activity identifier that corresponds to the identifier of the request contained in that definition.	C1	A scheduled activity is identified by an integer in the range from 1 to SCD_N_TBA. The identifier of the request in the TBA (source, destination and source sequence counter of the request) can be reconstructed from the information in the TBA which includes a pointer to the InCommand encapsulating the request.
c		The maximum number of scheduled activity definitions that the time-based scheduling subservice can insert within the time-based schedule and contemporaneously process at any time shall be declared when specifying that subservice.	C2	This is given by SCD_N_TBA
d		The time margin that the time-based scheduling subservice uses when inserting activities in the time-based schedule or time-shifting activities shall be declared when specifying that subservice.	C1	The time-margin is one of the parameters associated to the time-based scheduling service

N	Title	Requirement	C	Justification
e		The maximum delta time between the release time specified in a scheduled activity definition and the real release time of the related request shall be declared when specifying that subservice.	C2	This delta-time depends on how frequently the TBS is processed by the host application and on how frequently commands and processed by the host application. Both items are defined by the application designer as part of the instantiation process of the PUS Extension of the CORDET Framework
6.11.4.3.1a	Status	The time-based scheduling subservice shall maintain a status indicating whether the overall time-based schedule execution function is enabled or disabled.	C1	The PUS Extension of the CORDET Framework supports the (11,1) and (11,2) commands and defines the enable status of the TBS as a service 11 observable
b		When starting the time-based scheduling subservice, the time-based schedule execution function status shall be set to disabled.	C1	The default enable status of the TBS is: disabled
6.11.4.3.2a	Enable the time-based schedule execution function	The time-based scheduling subservice shall provide the capability to enable the time-based schedule execution function.	C1	The PUS Extension of the CORDET Framework supports command (11,1)
b		Each request to enable the time-based schedule execution function shall contain exactly one instruction to enable the time-based schedule execution function.	C1	See definition of component ScdEnbTbs
6.11.4.3.3a	Disable the time-based schedule execution function	The time-based scheduling subservice shall provide the capability to disable the time-based schedule execution function.	C1	The PUS Extension of the CORDET Framework supports command (11,2)
b		Each request to disable the time-based schedule execution function shall contain exactly one instruction to disable the time-based schedule execution function.	C1	See definition of component ScdDisTbs

N	Title	Requirement	C	Justification
6.11.4.4a	Reset the time-based schedule	The time-based scheduling subservice shall provide the capability to reset the time-based schedule.	C1	The PUS Extension of the CORDET Framework supports command (11,3)
b		Each request to reset the time-based schedule shall contain exactly one instruction to reset the time-based schedule.	C1	See definition of component ScdResTbs
c		"For each valid instruction to reset the time-based schedule, the time-based scheduling subservice shall: 1. set the time-based schedule execution function status to ""disabled""; 2. delete all scheduled activities from the schedule; 3. if sub-schedules are supported, delete all sub-schedules; 4. if groups are supported, enable all groups."	C1	See definition of progress action of component ScdResTbs
6.11.4.5a	Insert activities into the time-based schedule	The time-based scheduling subservice shall provide the capability to insert activities into the time-based schedule.	C1	The PUS Extension of the CORDET Framework supports command (11,4)
b		Each request to insert activities into the time-based schedule shall contain: 1. if sub-schedules are supported, the sub-schedule identifier; 2. one or more instructions to insert an activity into the time-based schedule.	C1	See definition of component ScdInsTbs. Note that sub-schedules are supported.

N	Title	Requirement	C	Justification
c		The time-based scheduling subservice shall reject any request to insert activities into the time-based schedule if: 1. that request implies the creation of a new sub-schedule but the maximum number of sub-schedules that can be contemporaneously managed is already reached.	C1	See definition of Start Action of command ScdInsTba.
d		For each request to insert activities into the time-based schedule that is rejected, the time-based scheduling subservice shall generate a failed start of execution notification.	C1	See definition of Start Action of command ScdInsTba.
e		Each instruction to insert an activity into the time-based schedule shall contain: 1. if groups are supported, the group identifier associated to the new scheduled activity; 2. the release time of that new scheduled activity; 3. the request associated to that new scheduled activity.	C1	See definition of component ScdInsTbs. Note that groups are supported.
f		The list of verification checks that the time-based scheduling subservice shall perform on the requests associated to the new scheduled activities shall be declared when specifying that subservice.	C1	See definition of Start Action of command ScdInsTba.

N	Title	Requirement	C	Justification
g		The time-based scheduling subservice shall reject any instruction to insert an activity into the time-based schedule if any of the following conditions occurs: 1. the activity cannot be added since the maximum number of scheduled activities that can be contemporaneously processed is already reached; 2. the release time of the activity is earlier than the time obtained by adding the time-based schedule time margin to the current time; 3. that instruction refers to a group that is unknown; 4. the request contained in that instruction fails any of the verification checks.	C1	See definition of Start Action of command ScdInsTba.
h		For each instruction to insert an activity into the time-based schedule that it rejects, the time-based scheduling subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of command ScdInsTba.
I		The time-based scheduling subservice shall process any valid instruction that is contained within a request to insert activities into the time-based schedule regardless of the presence of faulty instructions.	C1	See definition of Start Action of command ScdInsTba.
j		For each valid request to insert activities into the time-based schedule, the time-based scheduling subservice shall: 1. if sub-schedules are supported and the sub-schedule specified in that request is unknown: (a) create that sub-schedule; (b) set its status to disabled.	C1	See definition of Progress Action of command ScdInsTba. Note that, in the context of the PUS Extension, to create a sub-schedule means to set is inUse flag to true.

N	Title	Requirement	C	Justification
l		For each valid instruction to insert an activity into the time-based schedule, the time-based scheduling subservice shall: 1. create a new scheduled activity in the schedule; 2. place the request specified in that instruction into the new scheduled activity; 3. set the release time of the new scheduled activity to the release time specified in that instruction; 4. if sub-schedules are supported, associate the new scheduled activity to the sub-schedule specified in that instruction; 5. if groups are supported, associate the new scheduled activity to the group specified in that instruction.	C1	See definition of Progress Action of command ScdInsTba.
6.11.4.6a	Schedule execution logic	The time-based schedule execution process shall process the scheduled activities in the order of their release times.	C1	See definition of Time-Based Schedule Execution Procedure
b		"The time-based schedule execution process shall consider a scheduled activity is disabled if any of the following conditions occurs: 1. the time-based schedule execution function status is ""disabled""; 2. that scheduled activity is associated to a disabled sub-schedule; 3. that scheduled activity is associated to a disabled group"	C1	See definition of Time-Based Schedule Execution Procedure

N	Title	Requirement	C	Justification
c		For each scheduled activity whose release time is reached, the time-based schedule execution process shall, in sequence: 1. if that scheduled activity is not disabled, release the related request; 2. delete that scheduled activity from the schedule; 3. if that scheduled activity was the last scheduled activity of a sub-schedule, delete the sub-schedule	C1	See definition of Time-Based Schedule Execution Procedure
6.11.5.1a	Time-based sub-schedules	The maximum number of sub-schedules that the time-based scheduling subservice can contemporaneously manage shall be declared when specifying that subservice.	C1	See definition of observables associated to service 11.
		For each sub-schedule, the time-based scheduling subservice shall maintain a status indicating whether the schedule execution function for that sub-schedule is enabled or disabled.	C1	See definition of observables associated to service 11.
6.11.5.2.1a	Enable time-based sub-schedules	The time-based scheduling subservice capability to enable time-based sub-schedules shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports command (11,20)
b		Each request to enable time-based sub-schedules shall contain: 1. one or more instructions to enable a time-based sub-schedule, or 2. exactly one instruction to enable all time-based sub-schedules.	C1	See definition of component ScdEnbSubSched
c		Each instruction to enable a time-based sub-schedule shall contain: 1. the identifier of the sub-schedule to enable.	C1	See definition of component ScdEnbSubSched

N	Title	Requirement	C	Justification
d		The time-based scheduling subservice shall reject any instruction to enable a time-based sub-schedule if: 1.that instruction refers to an unknown sub-schedule.	C1	See definition of Start Action of command ScdEnbSubSched. A sub-schedule identifier is regarded as unknown if it is illegal or if the associated sub-schedule is empty (which means that it is not being used)
e		For each instruction to enable a time-based sub-schedule that it rejects, the time-based scheduling subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of command ScdEnbSubSched.
f		The time-based scheduling subservice shall process any valid instruction that is contained within a request to enable time-based sub-schedules regardless of the presence of faulty instructions.	C1	See definition of Progress Action of command ScdEnbSubSched.
g		For each valid instruction to enable a time-based sub-schedule, the time-based scheduling subservice shall: 1. set the status of that sub-schedule to enabled.	C1	See definition of Progress Action of command ScdEnbSubSched.
h		For each valid instruction to enable all time-based sub-schedules, the time-based scheduling subservice shall: 1. for each sub-schedule maintained by the subservice, set its status to enabled.	C1	See definition of Progress Action of command ScdEnbSubSched.
6.11.5.2.2a	Disable time-based sub-schedules	The time-based scheduling subservice capability to disable time-based sub-schedules shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports command (11,21)

N	Title	Requirement	C	Justification
b		Each request to disable time-based sub-schedules shall contain: 1. one or more instructions to disable a time-based sub-schedule, or 2. exactly one instruction to disable all time-based sub-schedules.	C1	See definition of component ScdDisSubSched
c		Each instruction to disable a time-based sub-schedule shall contain: 1. the identifier of the sub-schedule to disable.	C1	See definition of component ScdDisSubSched
d		The time-based scheduling subservice shall reject any instruction to disable a time-based sub-schedule if: 1.that instruction refers to an unknown sub-schedule.	C1	See definition of Start Action of command ScdDisSubSched. A sub-schedule identifier is regarded as unknown if it is illegal or if the associated sub-schedule is empty (which means that it is not being used)
e		For each instruction to disable a time-based sub-schedule that it rejects, the time-based scheduling subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of command ScdDisSubSched.
f		The time-based scheduling subservice shall process any valid instruction that is contained within a request to disable time-based sub-schedules regardless of the presence of faulty instructions.	C1	See definition of Progress Action of command ScdDisSubSched.
g		For each valid instruction to disable a time-based sub-schedule, the time-based scheduling subservice shall: 1. set the status of that sub-schedule to disable.	C1	See definition of Progress Action of command ScdDisSubSched.

N	Title	Requirement	C	Justification
h		For each valid instruction to disable all time-based sub-schedules, the time-based scheduling subservice shall: 1. for each sub-schedule maintained by the subservice, set its status to disable.	C1	See definition of Progress Action of command ScdDisSubSched.
6.11.5.2.3	Report the status of each time-based sub-schedule			This capability is not yet supported by the PUS Extension of the CORDET Framework
6.11.6.1a	Time-base scheduling groups	The maximum number of groups that the time-based scheduling subservice can contemporaneously manage shall be declared when specifying that subservice.	C1	See definition of constants associated to service 11. The value of the constants is defined when the service is instantiated.
b		For each group, the time-based scheduling subservice shall maintain a status indicating whether the schedule execution function for that group is enabled or disabled.	C1	See definition of observables associated to service 11
6.11.6.2.1a	Create time-based scheduling groups	The time-based scheduling subservice capability to create time-based scheduling groups shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports command (11,22)
b		Each request to create time-based scheduling groups shall contain one or more instructions to create a time-based scheduling group.	C1	See definition of command ScdCreGrp
c		Each instruction to create a time-based scheduling group shall contain: 1. the identifier of the group; 2. the group status at creation time.	C1	See definition of command ScdCreGrp

N	Title	Requirement	C	Justification
d		The time-based scheduling subservice shall reject any instruction to create a time-based scheduling group if any of the following conditions occurs: 1. that instruction refers to an already existing group; 2. the maximum number of groups that can be contemporaneously managed is already reached.	C1	See definition of Start Action of command ScdCreGrp
e		For each instruction to create a time-based scheduling group that it rejects, the time-based scheduling subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of command ScdCreGrp
f		The time-based scheduling subservice shall process any valid instruction that is contained within a request to create time-based scheduling groups regardless of the presence of faulty instructions.	C1	See definition of Progress Action of command ScdCreGrp
g		For each valid instruction to create a time-based scheduling group, the time-based scheduling subservice shall: 1. add the group identifier to the list of groups maintained by that sub-service; 2. set the group status to the value specified in the instruction.	C1	See definition of Progress Action of command ScdCreGrp
6.11.6.2.2a	Delete time-based scheduling groups	The time-based scheduling subservice shall provide the capability to delete time-based scheduling groups if the capability to create time-based scheduling groups is provided by that subservice.	C1	The PUS Extension of the CORDET Framework supports command (11,23)

N	Title	Requirement	C	Justification
b		Each request to delete time-based scheduling groups shall contain: 1. one or more instructions to delete a time-based scheduling group, or 2. exactly one instruction to delete all time-based scheduling groups.	C1	See definition of command ScdDelGrp
c		Each instruction to delete a time-based scheduling group shall contain: the identifier of the group to be deleted	C1	See definition of command ScdDelGrp
d		The time-based scheduling subservice shall reject any instruction to delete a time-based scheduling group if any of the following conditions occurs: 1. that instruction refers to a group that does not exist; 2. that instruction refers to a group that has associated activities.	C1	See definition of Start Action of command ScdDelGrp
e		For each instruction to delete a time-based scheduling group that it rejects, the time-based scheduling subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of command ScdDelGrp
f		The time-based scheduling subservice shall process any valid instruction that is contained within a request to delete time-based scheduling groups regardless of the presence of faulty instructions.	C1	See definition of Start Action of command ScdDelGrp
g		For each valid instruction to delete a time-based scheduling group, the time-based scheduling subservice shall: 1.delete the group identifier from the list of groups maintained by that subservice.	C1	See definition of Progress Action of command ScdDelGrp

N	Title	Requirement	C	Justification
h		For each valid instruction to delete all time-based scheduling groups, the time-based scheduling subservice shall: 1. for each group that has no associated activity, delete the identifier of that group; 2. for each group that has associated activities, generate a failed execution notification for that group.	C1	See definition of Progress Action of command ScdDelGrp
6.11.6.3.1a	Enable time-based scheduling groups	The time-based scheduling subservice shall provide the capability to enable time-based scheduling groups if the capability to create time-based scheduling groups is provided by that subservice.	C1	The PUS Extension of the CORDET Framework supports command (11,24)
b		Each request to enable time-based scheduling groups shall contain: 1. one or more instructions to enable a time-based scheduling group, or 2. exactly one instruction to enable all time-based scheduling groups.	C1	See definition of command ScdEnbGrp
c		Each instruction to enable a time-based scheduling group shall contain: 1. the identifier of the group to enable.	C1	See definition of command ScdEnbGrp
d		The time-based scheduling subservice shall reject any instruction to enable a time-based scheduling group if: 1. that instruction refers to an unknown group.	C1	See definition of Start Action of command ScdEnbGrp
e		For each instruction to enable a time-based scheduling group that it rejects, the time-based scheduling subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of command ScdEnbGrp

N	Title	Requirement	C	Justification
f		The time-based scheduling subservice shall process any valid instruction that is contained within a request to enable time-based scheduling groups regardless of the presence of faulty instructions.	C1	See definition of Progress Action of command ScdEnbGrp
g		For each valid instruction to enable a time-based scheduling group, the time-based scheduling subservice shall: 1. set the status of that group to enabled.	C1	See definition of Progress Action of command ScdEnbGrp
6.11.6.3.2a	Disable time-based scheduling groups	The time-based scheduling subservice shall provide the capability to disable time-based scheduling groups if the capability to create time-based scheduling groups is provided by that subservice.	C1	The PUS Extension of the CORDET Framework supports command (11,25)
b		Each request to enable time-based scheduling groups shall contain: 1. one or more instructions to disable a time-based scheduling group, or 2. exactly one instruction to disable all time-based scheduling groups.	C1	See definition of command ScdDisGrp
c		Each instruction to disable a time-based scheduling group shall contain: 1. the identifier of the group to disable.	C1	See definition of command ScdDisGrp
d		The time-based scheduling subservice shall reject any instruction to disable a time-based scheduling group if: 1. that instruction refers to an unknown group.	C1	See definition of Start Action of command ScdDisGrp

N	Title	Requirement	C	Justification
e		For each instruction to disable a time-based scheduling group that it rejects, the time-based scheduling subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of command ScdDisGrp
f		The time-based scheduling subservice shall process any valid instruction that is contained within a request to disable time-based scheduling groups regardless of the presence of faulty instructions.	C1	See definition of Progress Action of command ScdDisGrp
g		For each valid instruction to disable a time-based scheduling group, the time-based scheduling subservice shall: 1. set the status of that group to disabled.	C1	See definition of Progress Action of command ScdDisGrp
6.11.6.3.3a	Report the status of each time-based scheduling group	The time-based scheduling subservice capability to report the status of each time-based scheduling group shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports command (11,26) and report (11,27)
b		Each request to report the status of each time-based scheduling group shall contain exactly one instruction to report the status of each time-based scheduling group.	C1	See definition of command ScdRepGrp
c		For each valid instruction to report the status of each time-based scheduling group, the time-based scheduling subservice shall: generate, for each group managed by the time-based scheduling subservice, a single time-based scheduling group status notification that includes:(a) the group identifier; (b) its status.	C1	See definition of Progress Action of command ScdRepGrp and definition of report ScdGrpRep

N	Title	Requirement	C	Justification
d		For each valid request to report the status of each time-based scheduling group, the time-based scheduling subservice shall generate a single time-based scheduling group status report that includes all related time-based scheduling group status notifications.	C1	See definition of Progress Action of command ScdRepGrp and definition of report ScdGrpRep
6.11.7	Reports of time-based scheduled activities			This capability is not yet supported by the PUS Extension of the CORDET Framework
6.11.8.1a	Time-shift all scheduled activities	The time-based scheduling subservice capability to time-shift all scheduled activities shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports command (11,15)
b		Each request to time-shift all scheduled activities shall contain exactly one instruction to time-shift all scheduled activities.	C1	See definition of component ScdTimeShiftTbs
c		Each instruction to time-shift all scheduled activities shall contain: a time offset, positive or negative, to add to the release time of all scheduled activities.	C1	See definition of component ScdTimeShiftTbs
d		The time-based scheduling subservice shall reject any request to time-shift all scheduled activities if: the time obtained by adding the time offset to the release time of the earliest activity contained within the time-based schedule is earlier than the time obtained by adding the time-based schedule time margin to the current time.	C1	See definition of Start Action of component ScdTimeShiftTbs

N	Title	Requirement	C	Justification
e		For each request to time-shift all scheduled activities that is rejected, the time-based scheduling subservice shall generate a failed start of execution notification.	C1	See definition of Start Action of component ScdTimeShiftTbs
f		For each valid instruction to time-shift all scheduled activities, the time-based scheduling subservice shall: for each scheduled activity contained within the time-based schedule: set the release time of that scheduled activity to the sum of the current release time of that activity and the time offset.	C1	See definition of Progress Action of component ScdTimeShiftTbs
6.11.8.2.	Summary-report all time-based scheduled activities			This capability is not yet supported by the PUS Extension of the CORDET Framework
6.11.8.3	Detail-report all time-based scheduled activities			This capability is not yet supported by the PUS Extension of the CORDET Framework
6.11.9	General	Whether the time-based scheduling subservice supports the identification of scheduled activities by request identifier shall be declared when specifying that subservice	C1	This capability is supported. The request identifier is made up of the triplet: request identifier, APID, source sequence counter. One of the attributes of a TBA is the InCommand which encapsulates the TBA's request. The three elements of the request identifier can be retrieved from the InCommand.
6.11.9.1a	Delete time-based scheduled activities identified by request identifier	The time-based scheduling subservice capability to delete time-based scheduled activities identified by request identifier shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports command (11,5)

N	Title	Requirement	C	Justification
b		Each request to delete time-based scheduled activities identified by request identifier shall contain one or more instructions to delete a time-based scheduled activity identified by request identifier.	C1	See definition of component ScdDelTba
c		Each instruction to delete a time-based scheduled activity identified by request identifier shall contain: the identifier of the scheduled activity to delete.	C1	See definition of component ScdDelTba
d		The time-based scheduling subservice shall reject any instruction to delete a time-based scheduled activity identified by request identifier if: that instruction contains a request identifier is unknown.	C1	See definition of component ScdDelTba
e		For each instruction to delete a time-based scheduled activity identified by request identifier that it rejects, the time-based scheduling subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of ScdDelTba
f		The time-based scheduling subservice shall process any valid instruction that is contained within a request to delete time-based scheduled activities identified by request identifier regardless of the presence of faulty instructions.	C1	See definition of Progress Action of ScdDelTba

N	Title	Requirement	C	Justification
g		For each valid instruction to delete a time-based scheduled activity identified by request identifier, the time-based scheduling subservice shall: 1. delete the scheduled activity corresponding to the request identifier; 2. if that scheduled activity was the last scheduled activity of a sub-schedule, delete the sub-schedule.	C1	See definition of Progress Action of ScdDelTba
6.11.9.3	Time-shift scheduled activities identified by request Identifier			This capability is not yet supported by the PUS Extension of the CORDET Framework
6.11.9.4	Summary-report time-based scheduled activities identified by request identifier			This capability is not yet supported by the PUS Extension of the CORDET Framework
6.11.9.5	Detail-report time-based scheduled activities identified by request identifier			This capability is not yet supported by the PUS Extension of the CORDET Framework
6.11.10	Managing the time-based scheduled activities identified by a filter			This capability is not yet supported by the PUS Extension of the CORDET Framework

N	Title	Requirement	C	Justification
6.1.1.11a	System Observables	The following observables shall be defined for the time-based scheduling subservice: 1. the time-based schedule execution function status (enabled or disabled); 2. the current number of scheduled activities in the time-based schedule; 3. if sub-schedules are supported, the current number of sub-schedules; 4. if groups are supported, the current number of groups.	C1	See definition of observables associated to service 11. The number of sub-schedules and the number of groups are implemented as the numbers of non-empty sub-schedules and the number of non-empty groups.
6.12	On-Board Monitoring	Definition of service 12		
6.12.2.1.1a	Parameter Monitoring Subservice	Each on-board monitoring service shall contain exactly one parameter monitoring subservice.	C1	The PUS Extension of the CORDET Framework supports both sub-services of the On-Board Monitoring service and it allows one instance of the service to be deployed in an application. The service only contains one instance of each of its two sub-services.
6.12.2.1.2a	Functional Monitoring Subservice	Each on-board monitoring service shall contain at most one functional monitoring subservice.	C1	See statement of compliance to previous clause
6.12.2.2a	Application process	For each on-board monitoring service that contains both, a parameter monitoring subservice and a functional monitoring subservice, the two subservice providers of that service shall be hosted by the same application process.	C2	In the CORDET Framework, the allocation of sub-services to application processes is done during the framework instantiation process when the application developers define the groups (see section 4.2 of [PX-SP]

N	Title	Requirement	C	Justification
6.12.2.3.1a	Service	Each on-board monitoring service shall be associated to exactly one event reporting subservice.	C1	The PUS Extension of the CORDET Framework supports one on-board monitoring service and one event reporting subservice. The former service is associated to the latter subservice.
b		The event reporting subservice that is associated to the on-board monitoring service shall be declared when specifying that on-board monitoring service.	C1	See statement of compliance to previous clause
6.12.3.1	Parameter accessibility	The parameter monitoring subservice shall be able to monitor all on- board parameters that are accessible to the application process that hosts the subservice.	C1	The parameter monitoring subservice has access to all data pool parameters and variables
6.12.3.2.1a	Minimum capability	The parameter monitoring subservice shall support the evaluation of the following minimum check types: 1. Limit-check, 2. Expected-value-check	C1	Both check types are supported. See requirements S13-3 and 4.
b		When performing a limit-check, the parameter monitoring subservice shall: 1. check that the value of a parameter lies within a pair of limit values; 2. declare the check successful when the value of the parameter is less than or equal to the high limit value and greater than or equal to the low limit value.	C1	See definition of Limit Check Monitoring Procedure

N	Title	Requirement	C	Justification
c		When performing an expected-value-check, the parameter monitoring subservice shall: 1. check that the value resulting from applying a bit mask to a parameter is equal to the expected value; 2. declare the check successful when these two values are equal.	C1	See definition of Expected Value Monitoring Procedure
6.12.3.2.2a	Additional capability	The parameter monitoring subservice may support the evaluation of the delta-check type.	C1	This check is supported. See requirement S13-5
b		Whether the parameter monitoring subservice supports the delta-check type shall be declared when specifying that subservice.	C1	This check is supported. See requirement S13-5
c		When performing a delta-check, the parameter monitoring subservice shall: 1. calculate the delta value between two consecutive values of a parameter; 2. declare the check successful when the delta value is less than or equal to the high threshold value and greater than or equal to the low threshold value.	C1	See definition of Delta Value Monitoring Procedure
6.12.3.3a	Parameter monitoring definition	The maximum number of parameter monitoring definitions that the parameter monitoring subservice can contemporaneously evaluate at any time shall be declared when specifying that subservice.	C2	This number is given by the sum of constants MON_N_PMON which must be set at framework instantiation time
b		The parameter monitoring subservice shall provide the capability to process several parameter monitoring definitions for the same on-board parameter.	C1	See definition of Parameter Monitoring Definition List

N	Title	Requirement	C	Justification
c		Whether the parameter monitoring subservice supports conditional checking of parameter monitoring definitions shall be declared when specifying that subservice.	C1	Conditional checking is supported for each monitored parameter (see definition of PMDL)
d		Whether the parameter monitoring subservice uses a single, subservice-specific monitoring interval for all parameter monitoring definitions or uses a definition-specific monitoring interval for each parameter monitoring definition shall be declared when specifying that subservice.	C1	To each parameter monitor, a dedicated monitoring period expressed as a multiple of MON_PER is associated. See definition of parameter monitor attributes
e		If the parameter monitoring subservice uses a subservice-specific monitoring interval, that monitoring interval shall be declared when specifying that subservice.	C1	See previous requirement
f		Monitoring intervals shall be expressed in 'on-board parameter minimum sampling interval' units.	C1	See statement of compliance to previous two clauses
g		Each parameter monitoring definition shall contain: 1. the identifier of the parameter monitoring definition; 2. the identifier of the on-board parameter to monitor; 3. if the parameter monitoring subservice supports the conditional checking of parameter monitoring definitions, a check validity condition that yielding false prevents the check being performed; 4. if the parameter monitoring subservice uses definition-specific monitoring intervals, a monitoring interval; 5. a check definition.	C1	See definition of parameter monitor attributes in PMDL

N	Title	Requirement	C	Justification
h		Each check validity condition shall contain:1. the identifier of an on-board parameter to use as a validity parameter; 2. a bit-mask; 3. an expected value.	C1	See definition of parameter monitor attributes in PMDL
I		When computing the check validity condition, the parameter monitoring subservice shall: 1. perform a bitwise-and between the bit-mask and the sampled value of the validity parameter; 2. declare the condition true when the masked value equals the expected value.	C1	See definition of Monitoring Function Procedure
j		Each check definition shall contain: . . .	C1	See definition of parameter monitor attributes and of Monitoring Function Procedures (but sub-clause (5) is unclear and compliance is TBD)
6.12.3.4a	Statuses	The parameter monitoring subservice shall maintain a status indicating whether the overall parameter monitoring function is enabled or disabled.	C1	The parameter monitoring function is disabled if the Monitoring Function Procedure is stopped. The status of the procedure is an observable data item.
b		When starting the parameter monitoring subservice, the overall parameter monitoring function status shall be set to 'enabled'.	C2	The starting of the Parameter Monitoring Procedure (and hence the enabling of the parameter monitoring function) is the responsibility of the host application (see use constraint S12-8)
c		For each parameter monitoring definition, the parameter monitoring subservice shall maintain a status indicating whether that parameter monitoring definition is enabled or disabled.	C1	A parameter monitoring is enabled if its Parameter Monitor Procedure is started.

N	Title	Requirement	C	Justification
d		For each parameter monitoring definition, the parameter monitoring subservice shall maintain a status indicating the established status of the checks performed on the monitored parameter.	C1	See definition of parameter monitor attributes in PMDL
6.12.3.5.1a	Enable the parameter monitoring function	The parameter monitoring subservice shall provide the capability to enable the parameter monitoring function.	C1	The PUS Framework supports command (12,15)
		Each request to enable the parameter monitoring function shall contain exactly one instruction to enable the parameter monitoring function.	C1	See definition of component MonEnbMonFncCmd
		For each valid instruction to enable the parameter monitoring function, the parameter monitoring subservice shall: 1. set the PMON function status to 'enabled'; 2. for each parameter monitoring definition that is enabled: 3 (a) set its PMON checking status to 'unchecked'; (b) reset the repetition counter; start the parameter monitoring process.	C1	See progress action of component MonEnbMonFncCmd
6.12.3.5.2a	Disable the parameter monitoring function	The parameter monitoring subservice shall provide the capability to disable the parameter monitoring function.	C1	The PUS Framework supports command (12,16)

N	Title	Requirement	C	Justification
c		The parameter monitoring subservice shall reject any instruction to disable the parameter monitoring function if: 1. the on-board monitoring service includes a functional monitoring subservice whose functional monitoring function is enabled.	C1	See definition of Start Action of MonDisMonFncCmd component
d		For each request to disable the parameter monitoring function that is rejected, the parameter monitoring subservice shall generate a failed start of execution notification.	C1	See definition of Start Action of MonDisMonFncCmd component
e		For each valid instruction to disable the parameter monitoring function, the parameter monitoring subservice shall: 1. set the PMON function status to 'disabled'; 2. stop the parameter monitoring process.	C1	See definition of Progress Action of MonDisMonFncCmd component
6.12.3.6.1a	Enable parameter monitoring definitions	The parameter monitoring subservice shall provide the capability to enable parameter monitoring definitions.	C1	The PUS Extension of the CORDET Framework supports command (12,1)
b		Each request to enable parameter monitoring definitions shall contain one or more instructions to enable a parameter monitoring definition.	C1	See definition of component MonEnbParMonCmd
c		Each instruction to enable a parameter monitoring definition shall contain: 1. the identifier of the parameter monitoring definition.	C1	See definition of component MonEnbParMonCmd

N	Title	Requirement	C	Justification
d		The parameter monitoring subservice shall reject any instruction to enable a parameter monitoring definition if any of the following conditions occurs: 1. that instruction refers to a parameter monitoring definition identifier that is not in the PMON list; 2. that instruction refers to a parameter monitoring definition that is used by a protected functional monitoring definition.	C1	See definition of Start Action of command MonEnbParMonCmd
e		For each instruction to enable a parameter monitoring definition that it rejects, the parameter monitoring subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of command MonEnbParMonCmd
f		The parameter monitoring subservice shall process any valid instruction that is contained within a request to enable parameter monitoring definitions regardless of the presence of faulty instructions.	C1	See definition of Start Action of command MonEnbParMonCmd
g		For each valid instruction to enable a parameter monitoring definition, the parameter monitoring subservice shall: 1. reset the repetition counter of that parameter monitoring definition; 2. set the PMON status of that parameter monitoring definition to 'enabled'.	C1	See definition of Progress Action of command MonEnbParMonCmd
6.12.3.6.2a	Disable parameter monitoring definitions	The parameter monitoring subservice shall provide the capability to disable parameter monitoring definitions.	C1	The PUS Extension of the CORDET Framework supports command (12,2)

N	Title	Requirement	C	Justification
b		Each request to disable parameter monitoring definitions shall contain one or more instructions to disable a parameter monitoring definition.	C1	See definition of component MonDisParMonCmd
c		Each instruction to disable a parameter monitoring definition shall contain: 1. the identifier of the parameter monitoring definition.	C1	See definition of component MonDisParMonCmd
d		The parameter monitoring subservice shall reject any instruction to disable a parameter monitoring definition if any of the following conditions occurs: 1. that instruction refers to a parameter monitoring definition identifier that is not in the PMON list; 2. that instruction refers to a parameter monitoring definition that is used by a protected functional monitoring definition.	C1	See definition of Start Action of command MonDisParMonCmd
e		For each instruction to disable a parameter monitoring definition that it rejects, the parameter monitoring subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of command MonDisParMonCmd
f		The parameter monitoring subservice shall process any valid instruction that is contained within a request to disable parameter monitoring definitions regardless of the presence of faulty instructions.	C1	See definition of Start Action of command MonDisParMonCmd

N	Title	Requirement	C	Justification
g		For each valid instruction to disable a parameter monitoring definition, the parameter monitoring subservice shall: 1. set the PMON status of the parameter monitoring definition to 'disabled'; 2. set the PMON checking status of the parameter monitoring definition to 'unchecked'.	C1	See definition of Progress Action of command MonDisParMonCmd
6.12.3.6.3a	Parameter monitoring process	If the PMON function status is 'disabled', the parameter monitoring subservice shall not perform the parameter monitoring process for any parameter monitoring definitions.	C1	If the parameter monitoring function is disabled, the Monitoring Function Procedure is in the stopped state and it therefore does not execute any action
b		If the PMON status of a parameter monitoring definition is disabled, the parameter monitoring subservice shall not perform the parameter monitoring process for that definition.	C1	See definition of Monitoring Function Procedure

N	Title	Requirement	C	Justification
c		When performing the parameter monitoring process for a parameter monitoring definition, at the end of the monitoring interval, the parameter monitoring subservice shall, in sequence: 1. if the subservice supports the conditional checking of parameter monitoring definitions, compute the check validity condition; 2. if the computed check validity condition yields false: 3. (a) set the PMON checking status to 'invalid'; (b) reset the repetition counter of that parameter monitoring definition; if the subservice does not support the conditional checking of parameter monitoring definitions, or if the check validity condition yields true: (a) perform the check specified by the check definition, using a newly sampled value of the monitored parameter; (b) if the specified 'repetition number' of consecutive checks of the monitored parameter have all produced the same checking status output, establish a new PMON checking status;	C1	See definition of Monitoring Function Procedure
d		When a new PMON checking status is established, if that status differs from the previous PMON checking status, the parameter monitoring subservice shall: (a) record a check transition by adding that transition to the check transition list; (b) if an event definition is associated to that transition, raise the corresponding event.	C1	See definition of Monitoring Function Procedure

N	Title	Requirement	C	Justification
e		When a new PMON checking status is established for an expected-value-check, the parameter monitoring subservice shall set the PMON checking status to: 1. ’unexpected value’ if the specified ’repetition number’ of consecutive checks were declared unsuccessful; 2. ’expected value’, if the specified ’repetition number’ consecutive checks were declared successful.	C1	See definition of Monitoring Function Procedure and of Expected Value Monitor Procedure
f		When a new PMON checking status is established for a limit-check, the parameter monitoring subservice shall set the PMON checking status to: 1. ’above high limit’, if the specified ’repetition number’ of consecutive checks were declared unsuccessful and the parameter value in each check was greater than the high limit value; 2. ’below low limit’, if the specified ’repetition number’ of consecutive checks were declared unsuccessful and the parameter value in each check was less than the low limit value; 3. ’within limits’, if the specified ’repetition number’ of consecutive checks were declared successful.	C1	See definition of Monitoring Function Procedure and of Limit Check Monitor Procedure

N	Title	Requirement	C	Justification
g		When a new PMON checking status is established for a delta-check, the parameter monitoring subservice shall set the PMON checking status to: 1. 'above high threshold', if the specified 'repetition number' of consecutive checks were declared unsuccessful and the delta value in each check was greater than the high threshold value; 2. 'below low threshold', if the specified 'repetition number' of consecutive checks were declared unsuccessful and the delta value in each check was less than the low threshold value; 3. 'within thresholds', if the specified 'repet	C1	See definition of Monitoring Function Procedure and of Delta Value Monitor Procedure
6.12.3.7a	Reporting the check transitions	The parameter monitoring subservice shall provide the capability to report the contents of the check transition list.	C1	The PUS Extension of the CORDET Framework supports the (12,12) report

N	Title	Requirement	C	Justification
b		When reporting the contents of the check transition list, the parameter monitoring subservice shall: 1. for each check transition in the check transition list, generate a check transition notification containing: (a) the identifier of the parameter monitoring definition for which the check transition is recorded; (b) the identifier of the monitored parameter; (c) the check type; (d) for an expected-value-check, the expected-value-check mask; (e) the parameter value that has caused the transition; (f) the limit crossed; (g) the PMON checking status before the transition; (h) the PMON checking status resulting from the transition; (i) the transition time; 2. generate a single check transition report containing all the generated check transition notifications; 3. remove all the reported check transitions from the check transition list.	C1	See definition of MonChkTransRep
c		The maximum number of transitions required for issuing a check transition report shall be declared when specifying the parameter monitoring subservice.	C1	This is the same as the maximum size of the Check Transition List (CTL). This is defined when the service is instantiated.

N	Title	Requirement	C	Justification
d		The parameter monitoring subservice shall report the contents of the check transition list whenever one of the following condition occurs: 1. the maximum number of transitions required for issuing a check transition report is reached; 2. at the maximum transition reporting delay after the occurrence of the first check transition recorded in the check transition list.	C1	See definition of Ready Check of MonChkTransRep
e		The maximum transition reporting delay shall be expressed in 'on-board parameter minimum sampling interval' units.	C1	See definition of constants associated to service 12. The value of the constant is defined when the service is instantiated.
f		The default maximum transition reporting delay shall be declared when specifying the parameter monitoring subservice.	C1	See definition of constants associated to service 12. The value of the constant is defined when the service is instantiated.
6.12.3.8a	Change the maximum transition reporting delay	The parameter monitoring subservice capability to change the maximum transition reporting delay shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports the (12,3) report
		Each request to change the maximum transition reporting delay shall contain exactly one instruction to change the maximum transition reporting delay	C1	See definition of component MonChgTransDelCmd
		Each instruction to change the maximum transition reporting delay shall contain: 1. the maximum transition reporting delay.	C1	See definition of component MonChgTransDelCmd

N	Title	Requirement	C	Justification
		For each valid instruction to change the maximum transition reporting delay, the parameter monitoring subservice shall: 1. set the maximum transition reporting delay to the value specified in that instruction.	C1	See progress action of MonChgTransDelCmd command
6.12.3.9.1a	Add parameter monitoring definitions	The parameter monitoring subservice capability to add parameter monitoring definitions shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports the (12,5) command (but the optional Monitoring Interval field is not supported)
b		If the capability to add parameter monitoring definitions is provided by the parameter monitoring subservice, that subservice shall provide at least one of the following capabilities: 1. the capability to delete all parameter monitoring definitions specified in clause 6.12.3.9.2; 2. the capability to delete parameter monitoring definitions specified in clause 6.12.3.9.3.	C1	Both capabilities are supported by the PUS Extension of the CORDET Framework.
c		Each request to add parameter monitoring definitions shall contain one or more instructions to add a parameter monitoring definition.	C1	See definition of component MonAddParMonCmd
d		Each instruction to add a parameter monitoring definition shall contain: 1. the contents of the parameter monitoring definition.	C1	See definition of component MonAddParMonCmd

N	Title	Requirement	C	Justification
e		The parameter monitoring subservice shall reject any instruction to add a parameter monitoring definition if any of the following conditions occurs: 1. that instruction cannot be added since the PMON list is full; 2. that instruction refers to a parameter monitoring definition identifier that is already in the PMON list; 3. that instruction refers to a parameter to monitor that is not accessible; 4. that instruction refers to a validity parameter that is not accessible; 5. that instruction refers to a limit check for which the high limit is lower than the low limit; 6. that instruction refers to a delta check for which the high threshold is lower than the low threshold.	C1	See definition of Start Action of MonAddParMonCmd but note that: (a) in additional to the rejection conditions stated in this clause, additional ones are defined by the PUS Extension; and (b) conditions 3 and 4 are always satisfied because the parameter to monitor and the validity parameters are data pool items and data pool items are always accessible.
f		For each instruction to add a parameter monitoring definition that it rejects, the parameter monitoring subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of MonAddParMonCmd
g		The parameter monitoring subservice shall process any valid instruction that is contained within a request to add parameter monitoring definitions regardless of the presence of faulty instructions.	C1	See definition of Start Action of MonAddParMonCmd

N	Title	Requirement	C	Justification
h		For each valid instruction to add a parameter monitoring definition, the parameter monitoring subservice shall: 1. add a new parameter monitoring definition to the PMON list, using data from that instruction; 2. set the PMON checking status of the new parameter monitoring definition to ’unchecked’; 3. set the PMON status of the new parameter monitoring definition to ’disabled’.	C1	See definition of Progress Action of MonAddParMonCmd
6.12.3.9.2a	Delete all parameter monitoring definitions	The parameter monitoring subservice capability to delete all parameter monitoring definitions shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports the (12,4) command
b		Each request to delete all parameter monitoring definitions shall contain exactly one instruction to delete all parameter monitoring definitions.	C1	See definition of component MonDelAllParMonCmd
c		The parameter monitoring subservice shall reject any request to delete all parameter monitoring definitions if any of the following conditions occurs: 1. the PMON list contains one or more parameter monitoring definitions that are used by the functional monitoring subservice; 2. the PMON function status is ’enabled’.	C1	See definition of Start Action of MonDelAllParMonCmd command
d		For each request to delete all parameter monitoring definitions that is rejected, the parameter monitoring subservice shall generate a failed start of execution notification.	C1	See definition of Start Action of MonDelAllParMonCmd command

N	Title	Requirement	C	Justification
e		For each valid instruction to delete all parameter monitoring definitions, the parameter monitoring subservice shall: 1. delete all entries in the PMON list; 2. delete all entries in the check transition list.	C1	See definition of Progress Action of MonDelAllParMonCmd command
6.12.3.9.3a	Delete parameter monitoring definitions	The parameter monitoring subservice capability to delete parameter monitoring definitions shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports the (12,6) command
b		Each request to delete parameter monitoring definitions shall contain one or more instructions to delete a parameter monitoring definition.	C1	See definition of component MonDelParMonCmd
c		Each instruction to delete a parameter monitoring definition shall contain: 1. the identifier of the parameter monitoring definition.	C1	See definition of component MonDelParMonCmd
d		The parameter monitoring subservice shall reject any instruction to delete a parameter monitoring definition if any of the following conditions occurs: 1. that instruction refers to a parameter monitoring definition identifier that is not in the PMON list; 2. that instruction refers to a parameter monitoring definition whose PMON status is 'enabled'; 3. that instruction refers to a parameter monitoring definition that is used by a functional monitoring definition.	C1	See definition of Start Action of MonDelParMonCmd command

N	Title	Requirement	C	Justification
e		For each instruction to delete a parameter monitoring definition that it rejects, the parameter monitoring subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of MonDelParMonCmd command
f		The parameter monitoring subservice shall process any valid instruction that is contained within a request to delete parameter monitoring definitions regardless of the presence of faulty instructions.	C1	See definition of Start Action of MonDelParMonCmd command
g		For each valid instruction to delete a parameter monitoring definition, the parameter monitoring subservice shall: 1. remove the parameter monitoring definition that is referred to by that instruction from the PMON list.	C1	See definition of Progress Action of MonDelParMonCmd command
6.12.3.9.4a	Modify parameter monitoring definitions	The parameter monitoring subservice capability to modify parameter monitoring definitions shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports the (12,7) command
b		Each request to modify parameter monitoring definitions shall contain one or more instructions to modify a parameter monitoring definition.	C1	See definition of MonModParMonCmd command

N	Title	Requirement	C	Justification
c		Each instruction to modify a parameter monitoring definition shall contain: 1. the identifier of the parameter monitoring definition; the identifier of the monitored parameter used by that parameter monitoring definition; 3. the means to modify: (a) the repetition number; (b) for a limit-check, its low limit, its high limit and the event definition identifier of each associated event; (c) for an expected-value-check, its expected-value-check mask, its expected value and the event definition identifier of its associated event; (d) for a delta-check, its low delta threshold, its high delta threshold and the event definition identifier of each associated event.	C1	See definition of MonModParMonCmd command

N	Title	Requirement	C	Justification
d		The parameter monitoring subservice shall reject any instruction to modify a parameter monitoring definition if any of the following conditions occurs: 1. that instruction refers to a parameter monitoring definition identifier that is not in the PMON list; 2. that instruction refers to a monitored parameter that is not the one used in that parameter monitoring definition; 3. that instruction refers to a limit check for which the high limit is lower than the low limit; 4. that instruction refers to a delta check for which the high threshold is lower than the low threshold; 5. that instruction refers to a parameter monitoring definition that is used by a protected functional monitoring definition.	C1	See definition of Start Action of MonModParMonCmd command
e		For each instruction to modify a parameter monitoring definition that it rejects, the parameter monitoring subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of MonModParMonCmd command
f		The parameter monitoring subservice shall process any valid instruction that is contained within a request to modify parameter monitoring definitions regardless of the presence of faulty instructions.	C1	See definition of Progress Action of MonModParMonCmd command

N	Title	Requirement	C	Justification
		For each valid instruction to modify a parameter monitoring definition, the parameter monitoring subservice shall: 1. modify the parameter monitoring definition that is referred to by that instruction, using data from that instruction; 2. set the PMON checking status of the modified parameter monitoring definition to unchecked; 3. reset the repetition counter of that parameter monitoring definition.	C1	See definition of Progress Action of MonModParMonCmd command
6.12.3.10a	Report parameter monitoring definitions	The parameter monitoring subservice capability to report parameter monitoring definitions shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports both the (12,8) command and the (12,9) report
b		Each request to report parameter monitoring definitions shall contain: 1. one or more instructions to report a parameter monitoring definition, or 2. exactly one instruction to report all parameter monitoring definitions.	C1	See definition of MonRepParMonCmd command and MonRepParMonRep report
c		Each request to report parameter monitoring definitions shall contain: 1. one or more instructions to report a parameter monitoring definition, or 2. exactly one instruction to report all parameter monitoring definitions.	C1	See definition of MonRepParMonCmd command
d		Each instruction to report a parameter monitoring definition shall contain: 1. the identifier of the parameter monitoring definition.	C1	See definition of MonRepParMonCmd command

N	Title	Requirement	C	Justification
e		The parameter monitoring subservice shall reject any instruction to report a parameter monitoring definition if: 1. that instruction refers to a parameter monitoring definition identifier that is not in the PMON list.	C1	See definition of Start Action of MonRepParMonCmd command
f		For each instruction to report a parameter monitoring definition that it rejects, the parameter monitoring subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of MonRepParMonCmd command
g		The parameter monitoring subservice shall process any valid instruction that is contained within a request to report parameter monitoring definitions regardless of the presence of faulty instructions.	C1	See definition of Progress Action of MonRepParMonCmd command
h		For each valid instruction to report a parameter monitoring definition, the parameter monitoring subservice shall generate a single parameter monitoring definition notification that includes: 1. the parameter monitoring definition that is referred to by that instruction; 2. the PMON status of that parameter monitoring definition.	C1	See definition of Progress Action of MonRepParMonCmd command and MonRepParMonRep report
I		For each valid instruction to report all parameter monitoring definitions, the parameter monitoring subservice shall generate, for each parameter monitoring definition maintained by that subservice, a single parameter monitoring definition notification.	C1	See definition of MonRepParMonCmd command and MonRepParMonRep report

N	Title	Requirement	C	Justification
j		For each valid request to report parameter monitoring definitions, the parameter monitoring subservice shall generate a single parameter monitoring definition report that contains: 1. if changing the maximum transition reporting delay is supported, the current value of that delay; 2. all related parameter monitoring definition notifications.	C1	See definition of MonRepParMonCmd command and MonRepParMonRep report
6.12.3.11a	Report the status of each parameter monitoring Definition	The parameter monitoring subservice capability to report the status of each parameter monitoring definition shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports both the (12,14) command and the (12,15) report
b		Each request to report the status of each parameter monitoring definition shall contain exactly one instruction to report the status of each parameter monitoring definition.	C1	See definition of MonRepParMonCmd command and MonRepParMonRep report
c		For each valid instruction to report the status of each parameter monitoring definition, the parameter monitoring subservice shall: 1. generate, for each parameter monitoring definition in the PMON list, a single parameter monitoring definition status notification that includes: (a) the identifier of the parameter monitoring definition; (b) its PMON status.	C1	See definition of MonRepParMonCmd command and MonRepParMonRep report

N	Title	Requirement	C	Justification
d		For each valid request to report the status of each parameter monitoring definition, the parameter monitoring subservice shall generate a single parameter monitoring definition status report that includes all related parameter monitoring definition status notifications.	C1	See definition of MonRepParMonCmd command and MonRepParMonRep report
6.12.3.12a	Report the out-of-limits	The parameter monitoring subservice capability to report the out-of-limits shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports both the (12,10) command and the (12,11) report
b		Each request to report the out-of-limits shall contain exactly one instruction to report the out-of-limits.	C1	See definition of MonRepOutOfLimitsCmd command and MonRepOutOfLimitsRep report
c		For an expected-value-check, only the following transitions shall be reported in the out-of-limits report: 1. unchecked to unexpected value; 2. invalid to unexpected value; 3. expected value to unexpected value.	C1	The Start Action of the MonRepOutOfLimitsCmd command retrieves from the CTL all the transactions which have resulted in a violation. These transactions are then reported by the MonRepOutOfLimitsRep report
d		For a limit-check, only the following transitions shall be reported in the out-of-limits report: 1. unchecked to below low limit; 2. unchecked to above high limit; 3. invalid to below low limit; 4. invalid to above high limit; 5. within limits to below low limit; 6. within limits to above high limit; 7. below low limit to above high limit; 8. above high limit to below low limit.	C1	The Start Action of the MonRepOutOfLimitsCmd command retrieves from the CTL all the transactions which have resulted in a violation. These transactions are then reported by the MonRepOutOfLimitsRep report

N	Title	Requirement	C	Justification
e		For a delta-check, only the following transitions shall be reported in the out-of-limits report: 1. unchecked to below low threshold; 2. unchecked to above high threshold; 3. invalid to below low threshold; 4. invalid to above high threshold; 5. within threshold to below high threshold; 6. within threshold to above high threshold; 7. below low threshold to above high threshold; 8. above high threshold to below low threshold.	C1	The Start Action of the MonRepOutOfLimitsCmd command retrieves from the CTL all the transactions which have resulted in a violation. These transactions are then reported by the MonRepOutOfLimitsRep report
f		For each valid instruction to report the out-of-limits, the parameter monitoring subservice shall generate: 1. for each check transition to report, a single out-of-limit notification that includes: (a) the identifier of the parameter monitoring definition for which the check transition is recorded; (b) the identifier of the monitored parameter; (c) the check type; (d) for an expected-value-check, the expected-value-check mask; (e) the parameter value that has caused the transition; (f) the limit crossed; (g) the PMON checking status before the transition; (h) the PMON checking status resulting from the transition; (i) the transition time.	C1	See definition of MonRepOutOfLimitsCmd command and MonRepOutOfLimitsRep report
g		For each valid request to report the out-of-limits, the parameter monitoring subservice shall generate a single out-of-limits report that includes all related out-of-limit notifications.	C1	See definition of MonRepOutOfLimitsCmd command and MonRepOutOfLimitsRep report

N	Title	Requirement	C	Justification
6.12.3.13	Subservice observables	The following observables shall be defined for the parameter monitoring subservice: 1. the number of remaining available entries in the parameter monitoring definition list; 2. the number of enabled parameter monitoring definitions; 3. the PMON function status.	C1	See definition of observable Data Items associated to service 12 in [PX-SP]
6.12.4.1.1a	Parameter monitoring definition	The functional monitoring subservice shall be able to observe, at any time, the PMON checking status of each parameter monitoring definition of the parameter monitoring subservice of the parent on-board Monitoring service.	C1	Capability is implicitly provided by the specification of the commands and reports which implement the functional monitoring function
6.12.4.1.2a	General	The maximum number of functional monitoring definitions that the functional monitoring subservice can contemporaneously evaluate at any time shall be declared when specifying that subservice.	C1	See definition of constants associated to service 12. The value of the constant is defined when the service is instantiated.
b		The maximum number of parameter monitoring definitions that a functional monitoring definition can refer to shall be declared when specifying the functional monitoring subservice.	C1	See definition of constants associated to service 12. The value of the constant is defined when the service is instantiated.
c		Whether the functional monitoring subservice supports conditional checking of functional monitoring definitions shall be declared when specifying that subservice.	C1	Conditional checking is supported.

N	Title	Requirement	C	Justification
d		Whether the functional monitoring subservice supports specifying, for each functional monitoring definition, the minimum number of contemporaneously violated parameter monitoring definitions that establishes a functional monitoring checking failure shall be declared when specifying that subservice.	C1	Definition of a minimum failing number is supported for each functional monitor
e		If the functional monitoring subservice does not support specifying, for each functional monitoring definition, the minimum PMON failing number, the subservice shall use a value of 1 as the minimum PMON failing number for all functional monitoring definitions.	n.a.	See previous requirement
f		Each functional monitoring definition shall contain: 1. its identifier; 2. if the functional monitoring subservice supports the conditional checking of functional monitoring definitions, a check validity condition that yielding false prevents the check being performed; 3. the event definition identifier of the event to raise; 4. if the subservice supports specifying the minimum PMON failing number, a minimum PMON failing number; 5. a set of one or more parameter monitoring definition identifiers.	C1	See definition of attributes of a functional monitor in the Functional Monitoring Definition List (FMDL) in [PX-SP]
6.12.4.1.3a	Statuses	The functional monitoring subservice shall maintain a status indicating whether the overall functional monitoring function is enabled or disabled.	C1	See definition of observables associated to service 12.

N	Title	Requirement	C	Justification
b		For each functional monitoring definition, the functional monitoring subservice shall maintain a status indicating whether that functional monitoring definition is enabled or disabled.	C1	See definition of attributes of a functional monitor in the Functional Monitoring Definition List (FMDL) in [PX-SP]
c		For each functional monitoring definition, the functional monitoring subservice shall maintain a status indicating the result of the check performed.	C1	See definition of attributes of a functional monitor in the Functional Monitoring Definition List (FMDL) in [PX-SP]
d		If the functional monitoring subservice supports the capability for protecting functional monitoring definitions, the functional monitoring subservice shall maintain, for each functional monitoring definition, a status indicating whether that functional monitoring definition is protected or unprotected.	C1	See definition of attributes of a functional monitor in the Functional Monitoring Definition List (FMDL) in [PX-SP]
6.12.4.4.1a	Enable the functional monitoring function	The functional monitoring subservice shall provide the capability to enable the functional monitoring function.	C1	The PUS Extension of the CORDET Framework supports the (12,17) command
b		Each request to enable the functional monitoring function shall contain exactly one instruction to enable the functional monitoring function.	C1	See definition of MonEnbFuncMonCmd command
c		The functional monitoring subservice shall reject any request to enable the functional monitoring function if: 1. the parameter monitoring function of the associated parameter monitoring subservice is disabled.	C1	See definition of Start Action of MonEnbFuncMonCmd command

N	Title	Requirement	C	Justification
d		For each request to enable the functional monitoring function that is rejected, the functional monitoring subservice shall generate a failed start of execution notification.	C1	See definition of Start Action of MonEnbFuncMonCmd command
e		For each valid instruction to enable the functional monitoring function, the functional monitoring subservice shall: 1. set the FMON function status to enabled; 2. for each functional monitoring definition that is enabled: (a) 3. set its FMON checking status to unchecked; start immediately the monitoring of the enabled functional monitoring definitions.	C1	See definition of Progress Action of MonEnbFuncMonCmd command
6.12.4.4.2a	Disable the functional monitoring function	The functional monitoring subservice shall provide the capability to disable the functional monitoring function.	C1	The PUS Extension of the CORDET Framework supports the (12,18) command
b		Each request to disable the functional monitoring function shall contain exactly one instruction to disable the functional monitoring function.	C1	See definition of MonDisFuncMonCmd command
c		For each valid instruction to disable the functional monitoring function, the functional monitoring subservice shall: 1. set the FMON function status to disabled. 2. stop immediately the monitoring of the functional monitoring definitions.	C1	See definition of Progress Action of MonDisFuncMonCmd command

N	Title	Requirement	C	Justification
6.12.4.5.1	Monitoring transitions	For each functional monitoring definition, whenever a new PMON checking status has been established for one of its parameter monitoring definitions, the functional monitoring subservice shall perform the following: 1. If the FMON function status is enabled and the FMON status is enabled and the current FMON checking status is not failed: (a) the check validity condition, if any, is computed; (b) If the computed check validity condition yields false, the FMON checking status is set to invalid. 2. If the FMON function status is enabled, the FMON status is enabled and the current FMON checking status is neither failed nor invalid: (a) check if the number of related parameter monitoring definitions that are contemporaneously in violation equals or exceeds the minimum PMON failing number; (b) if the check yields true, the FMON checking status is set to failed and the associated event is raised; (c) if the check yields false, the FMON checking status is set to running.	C1	See definition of Functional Monitor Notification Procedure in [PX-SP]
6.12.4.5.2a	Enable functional monitoring definitions	The functional monitoring subservice shall provide the capability to enable functional monitoring definitions.	C1	The PUS Extension of the CORDET Framework supports the (12,19) command
b		Each request to enable functional monitoring definitions shall contain one or more instructions to enable a functional monitoring definition.	C1	See definition of MonEnbFuncMonDefCmd command

N	Title	Requirement	C	Justification
c		Each instruction to enable a functional monitoring definition shall contain: 1. the identifier of the functional monitoring definition.	C1	See definition of MonEnbFuncMonDefCmd command
d		The functional monitoring subservice shall reject any instruction to enable a functional monitoring definition if: 1. that instruction refers to a functional monitoring definition identifier that is not in the FMON list.	C1	See definition of Start Action of MonEnbFuncMonDefCmd command
e		For each instruction to enable a functional monitoring definition that it rejects, the functional monitoring subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of MonEnbFuncMonDefCmd command
f		The functional monitoring subservice shall process any valid instruction that is contained within a request to enable functional monitoring definitions regardless of the presence of faulty instructions.	C1	See definition of Progress Action of MonEnbFuncMonDefCmd command
g		For each valid instruction to enable a functional monitoring definition, the functional monitoring subservice shall: 1. set the FMON status of the functional monitoring definition to enabled	C1	See definition of Progress Action of MonEnbFuncMonDefCmd command
6.12.4.5.3a	Disable functional monitoring definitions	The functional monitoring subservice shall provide the capability to disable functional monitoring definitions.	C1	The PUS Extension of the CORDET Framework supports the (12,20) command

N	Title	Requirement	C	Justification
b		Each request to disable functional monitoring definitions shall contain one or more instructions to disable a functional monitoring definition.	C1	See definition of MonDisFuncMonDefCmd command
c		Each instruction to disable a functional monitoring definition shall contain: 1. the identifier of the functional monitoring definition.	C1	See definition of MonDisFuncMonDefCmd command
d		The functional monitoring subservice shall reject any instruction to disable a functional monitoring definition if: 1. that instruction refers to a functional monitoring definition identifier that is not in the FMON list.	C1	See definition of Start Action of MonDisFuncMonDefCmd command
e		For each instruction to disable a functional monitoring definition that it rejects, the functional monitoring subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of MonDisFuncMonDefCmd command
f		The functional monitoring subservice shall process any valid instruction that is contained within a request to disable functional monitoring definitions regardless of the presence of faulty instructions.	C1	See definition of Progress Action of MonDisFuncMonDefCmd command
g		For each valid instruction to disable a functional monitoring definition, the functional monitoring subservice shall: 1. set the FMON status of the functional monitoring definition to disabled; 2. set the FMON checking status of the functional monitoring definition to unchecked.	C1	See definition of Progress Action of MonDisFuncMonDefCmd command

N	Title	Requirement	C	Justification
6.12.4.6.1a	Protect functional monitoring definitions	The functional monitoring subservice capability to protect functional monitoring definitions shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports the (12,21) command
b		Each request to protect functional monitoring definitions shall contain one or more instructions to protect a functional monitoring definition.	C1	See definition of MonProtFuncMonDefCmd command
c		Each instruction to protect a functional monitoring definition shall contain: 1. the identifier of the functional monitoring definition.	C1	See definition of MonProtFuncMonDefCmd command
d		The functional monitoring subservice shall reject any instruction to protect a functional monitoring definition if: 1. that instruction refers to a functional monitoring definition identifier that is not in the FMON list.	C1	See definition of Start Action of MonProtFuncMonDefCmd command
e		For each instruction to protect a functional monitoring definition that it rejects, the functional monitoring subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of MonProtFuncMonDefCmd command
f		The functional monitoring subservice shall process any valid instruction that is contained within a request to protect functional monitoring definitions regardless of the presence of faulty instructions.	C1	See definition of Progress Action of MonProtFuncMonDefCmd command

N	Title	Requirement	C	Justification
g		For each valid instruction to protect a functional monitoring definition, the functional monitoring subservice shall: 1. set the FMON protection status of the functional monitoring definition to protected.	C1	See definition of Progress Action of MonProtFuncMonDefCmd command
6.12.4.6.2a	Unprotect functional monitoring definitions	The functional monitoring subservice capability to unprotect functional monitoring definitions shall be provided if the capability to protect functional monitoring definitions is provided by that subservice.	C1	The PUS Extension of the CORDET Framework supports the (12,22) command
b		Each request to unprotect functional monitoring definitions shall contain one or more instructions to unprotect a functional monitoring definition.	C1	See definition of MonUnprotFuncMonDefCmd command
c		Each instruction to unprotect a functional monitoring definition shall contain: 1. the identifier of the functional monitoring definition.	C1	See definition of MonUnprotFuncMonDefCmd command
d		The functional monitoring subservice shall reject any instruction to unprotect a functional monitoring definition if: 1. that instruction refers to a functional monitoring definition identifier that is not in the FMON list.	C1	See definition of Start Action of MonUnprotFuncMonDefCmd command
e		For each instruction to unprotect a functional monitoring definition that it rejects, the functional monitoring subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of MonUnprotFuncMonDefCmd command

N	Title	Requirement	C	Justification
f		The functional monitoring subservice shall process any valid instruction that is contained within a request to unprotect functional monitoring definitions regardless of the presence of faulty instructions.	C1	See definition of Progress Action of MonUnprotFuncMonDefCmd command
g		For each valid instruction to unprotect a functional monitoring definition, the functional monitoring subservice shall: 1. set the FMON protection status of the functional monitoring definition to unprotected.	C1	See definition of Progress Action of MonUnprotFuncMonDefCmd command
6.12.4.7.1a	Add functional monitoring definitions	The functional monitoring subservice capability to add functional monitoring definitions shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports the (12,23) command
b		Each request to add functional monitoring definitions shall contain one or more instructions to add a functional monitoring definition.	C1	See definition of MonAddFuncMonDefCmd command
c		Each instruction to add a functional monitoring definition shall contain: 1. the contents of the functional monitoring definition.	C1	See definition of MonAddFuncMonDefCmd command

N	Title	Requirement	C	Justification
d		The functional monitoring subservice shall reject any request to add functional monitoring definitions if any of the following conditions occurs: 1. that request contains an instruction that refers to a functional monitoring definition identifier that is already in the FMON list; 2. that request contains more than one instruction for the same functional monitoring definition.	C1	See definition of Start Action of MonAddFuncMonDefCmd command
e		The functional monitoring subservice shall reject any instruction to add a functional monitoring definition if any of the following conditions occurs: 1. that instruction cannot be added since the FMON list is full; 2. that instruction refers to a parameter monitoring definition identifier that is not in the PMON list; 3. that instruction refers to a validity parameter that is not accessible.	C1	See definition of Start Action of MonAddFuncMonDefCmd command
f		For each request to add functional monitoring definitions that it rejects, the functional monitoring subservice shall generate the failed start of execution notification for that request.	C1	See definition of Start Action of MonAddFuncMonDefCmd command
g		For each instruction to add a functional monitoring definition that it rejects, the functional monitoring subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of MonAddFuncMonDefCmd command

N	Title	Requirement	C	Justification
h		The functional monitoring subservice shall process any valid instruction that is contained within a request to add functional monitoring definitions regardless of the presence of faulty instructions.	C1	See definition of Progress Action of MonAddFuncMonDefCmd command
I		For each valid instruction to add a functional monitoring definition, the functional monitoring subservice shall: 1. add a new functional monitoring definition to the FMON list, using data from that instruction; 2. set the FMON checking status of the new functional monitoring definition to unchecked; 3. set the FMON status of the new functional monitoring definition to disabled; 4. if the functional monitoring subservice supports the capability for protecting functional monitoring definitions, set the FMON protection status of the new functional monitoring definition to protected.	C1	See definition of Progress Action of MonAddFuncMonDefCmd command
6.12.4.7.2a	Delete functional monitoring definitions	The functional monitoring subservice shall provide the capability to delete functional monitoring definitions if the capability to add functional monitoring definitions is provided by that subservice.	C1	The PUS Extension of the CORDET Framework supports the (12,24) command
b		Each request to delete functional monitoring definitions shall contain one or more instructions to delete a functional monitoring definition.	C1	See definition of MonDelFuncMonDefCmd command

N	Title	Requirement	C	Justification
c		Each instruction to delete a functional monitoring definition shall contain: 1. the identifier of the functional monitoring definition.	C1	See definition of MonDelFuncMonDefCmd command
d		The functional monitoring subservice shall reject any instruction to delete a functional monitoring definition if any of the following conditions occurs: 1. that instruction refers to a functional monitoring definition identifier that is not in the FMON list; 2. that instruction refers to a functional monitoring definition whose FMON status is enabled; 3. that instruction refers to a functional monitoring definition whose FMON protection status is protected.	C1	See definition of Start Action of MonDelFuncMonDefCmd command
e		For each instruction to delete a functional monitoring definition that it rejects, the functional monitoring subservice shall generate the failed start of execution notification for that instruction.	C1	See definition of Start Action of MonDelFuncMonDefCmd command
f		The functional monitoring subservice shall process any valid instruction that is contained within a request to delete functional monitoring definitions regardless of the presence of faulty instructions.	C1	See definition of Progress Action of MonDelFuncMonDefCmd command
g		For each valid instruction to delete a functional monitoring definition, the functional monitoring subservice shall: 1. remove the functional monitoring definition that is referred to by that instruction from the FMON list.	C1	See definition of Progress Action of MonDelFuncMonDefCmd command

N	Title	Requirement	C	Justification
6.12.4.8a	Report functional monitoring definitions	The functional monitoring subservice capability to report functional monitoring definitions shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports the (12,25) command and the (12,26) report
b		Each request to report functional monitoring definitions shall contain: 1. one or more instructions to report a functional monitoring definition, or 2. exactly one instruction to report all functional monitoring definitions.	C1	See definition of MonRepFuncMonDefCmd command
c		Each instruction to report a functional monitoring definition shall contain: 1. the identifier of the functional monitoring definition.	C1	See definition of MonRepFuncMonDefCmd command
d		The functional monitoring subservice shall reject any instruction to report a functional monitoring definition if: 1. that instruction refers to a functional monitoring definition identifier that is not in the FMON list.	C1	See definition of Start Action of MonRepFuncMonDefCmd command
e		For each instruction to report a functional monitoring definition that it rejects, the functional monitoring subservice shall generate the failed start of execution notification for that instruction	C1	See definition of Start Action of MonRepFuncMonDefCmd command
f		The functional monitoring subservice shall process any valid instruction that is contained within a request to report functional monitoring definitions regardless of the presence of faulty instructions.	C1	See definition of Progress Action of MonRepFuncMonDefCmd command

N	Title	Requirement	C	Justification
g		For each valid instruction to report a functional monitoring definition, the functional monitoring subservice shall 1. generate a single functional monitoring definition notification that includes: (a) the content of the functional monitoring definition that is referred to by that instruction; (b) if the functional monitoring subservice supports the capability for protecting functional monitoring definitions, the FMON protection status of that functional monitoring definition; (c) the FMON status of that functional monitoring definition.	C1	See definition of Progress Action of MonRepFuncMonDefCmd command
h		For each valid instruction to report all functional monitoring definitions, the functional monitoring subservice shall: 1. for each functional monitoring definition maintained by that subservice, generate a single functional monitoring definition notification that includes: (a) the contents of that functional monitoring definition; (b) if the functional monitoring subservice supports the capability for protecting functional monitoring definitions, the FMON protection status of that functional monitoring definition; (c) the FMON status of that functional monitoring definition.	C1	See definition of Progress Action of MonRepFuncMonDefCmd command

N	Title	Requirement	C	Justification
I		For each valid request to report functional monitoring definitions, the functional monitoring subservice shall generate a single functional monitoring definition report that contains all related functional monitoring definition notifications.	C1	See definition of Progress Action of MonRepFuncMonDefCmd command and of MonRepFuncMonDefRep report
6.12.4.9a	Report the status of each functional monitoring Definition	The functional monitoring subservice capability to report the status of each functional monitoring definition shall be declared when specifying that subservice.	C1	The PUS Extension of the CORDET Framework supports the (12,26) report
b		Each request to report the status of each functional monitoring definition shall contain exactly one instruction to report the status of each functional monitoring definition.	C1	See definition of MonRepFuncMonDefRep report
c		For each valid instruction to report the status of each functional monitoring definition, the functional monitoring subservice shall: 1. generate, for each functional monitoring definition in the FMON list, a single functional monitoring definition status notification that includes: (a) the identifier of that functional monitoring definition; (b) if the functional monitoring subservice supports the capability for protecting functional monitoring definitions, its FMON protection status; (c) its FMON status; (d) its FMON checking status.	C1	See definition of MonRepFuncMonDefRep report

N	Title	Requirement	C	Justification
d		For each valid request to report the status of each functional monitoring definition, the functional monitoring subservice shall generate a single functional monitoring definition status report that includes all related functional monitoring definition status notifications.	C1	See definition of Progress Action of MonRepFuncMonDefCmd command and of MonRepFuncMonDefRep report
6.12.4.10	Subservice observables	The following observables shall be defined for the functional monitoring subservice: 1. the number of remaining available entries in the functional monitoring definition list; 2. the number of enabled functional monitoring definitions; 3. the FMON function status.	C1	See definition of service 12 observables in [PX-SP]
6.13	Large Packet Transfer	Definition of service 13		
6.13.2.1.1	Subservice	Each large packet transfer service shall contain at least one of: 1. the large packet downlink subservice; 2. the large packet uplink subservice.	C1	The PUS Extension of the CORDET Framework supports both sub-services
6.13.2.1.2	Large packet downlink subservice	Each large packet transfer service shall contain at most one large packet downlink subservice.	C1	An application instantiated from the CORDET Framework can only provide one instance of a service of a given type and of its sub-services
6.13.2.1.3	Large packet uplink subservice	Each large packet transfer service shall contain at most one large packet uplink subservice.	C1	An application instantiated from the CORDET Framework can only provide one instance of a service of a given type and of its sub-services

N	Title	Requirement	C	Justification
6.13..2.2a	Application process	Each large packet transfer subservice provider shall be hosted by exactly one application process. NOTE: This implies that when both the large packet downlink subservice and the large packet uplink subservice are supported, the sending entity of the downlink subservice and the receiving entity of the uplink subservice are both hosted by that same on-board application process.	C2	In the CORDET Framework, the allocation of sub-services to application processes is done during the framework instantiation process when the application developers define the groups (see section 4.2 of [PX-SP]
b		Each application process shall host at most one large packet transfer subservice provider.	C1	The PUS Extension of the CORDET Framework supports one large packet transfer service per application
6.13.3.1a	Configuration of large packet downlink subservice	The maximum number of large packets that can be downlinked concurrently shall be declared when specifying the large packet downlink subservice.	C2	Each down- or up-transfer locks a Large Packet Transfer Buffer for the duration of the transfer. Hence, the maximum number of simultaneously active up- and down-transfers is determined by the number Large Packet Transfer Buffers available in the host application. This is an application constant set during the framework instantiation process (see definition of Constants for Service 13 in [PX-SP]).
b		The part size used by the large packet downlink subservice to decompose large packets shall be declared when specifying that subservice.	C2	The part size is one of the parameters defined by the PUS Extension for service 13 (see definition of Parameters for Service 13 in [PX-SP])

N	Title	Requirement	C	Justification
c		The maximum time allocated to the receiving entity for receiving a subsequent downlink part report after the reception of the previous one shall be declared when specifying the large packet downlink subservice.	n.a.	The PUS Extension of the CORDET Framework does not cover the reception of down-link transfers.
6.13.3.2	Resources	The resources allocated to the sending entity of the large packet downlink subservice to process large packets shall be declared when specifying the spacecraft architecture and its operations.	C2	The only framework resources used by the large packet transfer service are the memory used to create the up- and down-transfer packets and the resources for the Large Packet Transfer Buffers. The former resources are allocated within the factory components which create the components encapsulating the packets and are therefore declared when the factories are instantiated and when their adaptation points FAC-1 and FAC-2 are closed. The latter resources are declared when the values of the service 13 constants are defined at application instantiation time. Bandwidth resources for the down- and up-transfers are provided by the middleware and they are therefore declared when the InStream and OutStream components are defined during the instantiation process.

N	Title	Requirement	C	Justification
6.13.3.3.1a	Downlink Process	The sending entity of the large packet downlink subservice shall have the capability to process each large packet that it receives. NOTE: This Standard assumes that on-board, the large packets are not duplicated. The synchronization between the source of the large packets and the large packet downlink subservice is beyond the scope of this Standard.	C1	The PUS Extension uses Large Packet Transfer Buffers (LPTBs) to hold the data corresponding to one single down- or up-transfer. There is therefore no duplication of packet data.
b		For each large packet that it processes, the sending entity of the large packet downlink subservice shall: 1. assign a unique large message transaction identifier to that large packet; 2. split the large packet into parts; 3. associate to each part, a unique part sequence number; 4. encapsulate each part into a single downlink part report.	C1	These capability are implemented by the LPT State Machine of the PUS Extension.
c		Each part report shall contain exactly one part notification made of: (a) an identifier of whether the part report contains the First part, an Intermediate part or the Last part of the large packet; (b) the large message transaction identifier; (c) the part sequence number; (d) the part itself.	C1	See definition of components LptDownFirstRep, LptDownInterRep and LptDownLastRep
d		The destination of the part reports generated by the large packet downlink subservice shall be declared when specifying the space to ground architecture.	C2	For each Large Packet Transfer Buffer, a destination is defined as a parameter of the service 13 definition in the PUS Extension (see definition of Parameters for Service 13 in [PX-SP]).

N	Title	Requirement	C	Justification
e		The sending entity of the large packet downlink subservice shall generate the part reports related to each large packet, in increasing order of the part sequence number and at the highest frequency supported under the prevailing operation constraints.	C2	The LPT State Machine which manages a down-link transfer hands over a packet to the middleware every time it is executed. The frequency of execution of the LPT State Machine is decided by the host application.
6.13.3.3.2	Accepting part reports and reconstructing large packets		n.a.	The PUS Extension of the CORDET Framework does not cover the reception of down-transfers.
6.13.3.4	Subservice observables	The following observables shall be defined for the on-board large packet downlink subservice: 1. the number of on-going downlinks; 2. the list of large message transaction identifiers associated to the on-going downlinks in an array of size corresponding to the maximum number of large packets that can be downlinked concurrently.	C1	See definition of Observable Data Items associated to service 13 in [PX-SP]
6.13.4.1a	Configuration of large packet uplink subservice	The maximum number of large packets that can be uplinked concurrently shall be declared when specifying the large packet uplink subservice.	C2	See statement of compliance to clause 6.13.3.1a.
b		The part size used by the large packet uplink subservice to decompose large packets shall be declared when specifying that subservice.	n.a.	The PUS Extension of the CORDET Framework does not cover the sending of up-transfer packets

N	Title	Requirement	C	Justification
c		The maximum time allocated to the uplink receiving entity for receiving a subsequent uplink part request after the reception of the previous one (uplink reception time-tout) shall be declared when specifying the large packet uplink subservice.	C2	The PUS Extension of the CORDET Framework does not implement any tme-out mechanism for uplink packets: uplink packets are processed when they are received. If needed, a time-out mechanism can be implemented by the host application which can use the up-transfer abort mechanism to abort the up-transfer in case of time-out violation.
6.13.4.2a	Resources	The resources allocated to the uplink receiving entity of the large packet uplink subservice to process large packets shall be declared when specifying the spacecraft architecture and its operations.	C2	See statement of compliance to clause 6.13.4.2a.
6.13.4.3.1a	Uplink Process	For each large packet that it processes, the sending entity of the large packet uplink subservice shall: 1. assign a unique large message transaction identifier to that large Packet; 2. split the large packet into parts; 3. associate to each part, a unique part sequence number; 4. encapsulate each part into a single uplink part request.	n.a.	The PUS Extension of the CORDET Framework does not cover the sending of up-transfer packets
b		Each part request shall contain: exactly one part instruction made of: (a) an identifier of whether the part request is the 'First' part, an 'Intermediate' part or the 'Last' part of the large packet; (b) the large message transaction identifier; (c) the part sequence number; (d) the part itself.	C1	See definition of the components encapsulating up-transfer packets (components LptUpFirstCmd, LptUpInterCmd and LptUpLastCmd)

N	Title	Requirement	C	Justification
c		The destination of the uplink part requests generated by the large packet uplink subservice shall be declared when specifying the space to ground architecture.	n.a.	The PUS Extension of the CORDET Framework does not cover the sending of up-transfer packets
d		The sending entity of the large packet uplink subservice shall generate the uplink part requests related to each large packet, in increasing order of part sequence number and at the highest frequency supported under the prevailing operation constraints.	n.a.	The PUS Extension of the CORDET Framework does not cover the sending of up-transfer packets
6.13.4.3.2a	Accepting uplink part requests and reconstructing large packets	The receiving entity of the large packet uplink subservice shall be able to process all uplink part requests that it receives.	C1	The PUS Extension of the CORDET Framework supports commands (13,9) to (13,11)
b		The receiving entity of the large packet uplink subservice shall initiate the uplink operation when it receives the request to uplink the first part of the large packet.	C1	Reception of a (13,9) triggers the transition of the LPT State Machine to state UP_TRANSFER. This marks the start of the reception process.
c		The receiving entity of the large packet uplink subservice shall initiate the reception timer after the successful reception of the request to uplink the first part or the request to uplink an intermediate part.	C1	See definition of LPT State Machine
d		The receiving entity of the large packet uplink subservice shall end the uplink operation when the request to uplink the last part of the large packet has successfully been received.	C1	Reception of a (13,11) triggers the transition of the LPT State Machine from state UP_TRANSFER to state INACTIVE. This marks the end of the reception process.

N	Title	Requirement	C	Justification
e		The receiving entity of the large packet uplink subservice shall abort the uplink operation when the reception timer reaches the uplink reception timeout.	C1	The expiration of the time-out triggers a transition of the LPT State Machine from state UP_TRANSFER to state INACTIVE. This marks the end of the reception process.
f		The receiving entity of the large packet uplink subservice shall abort the uplink operation when a discontinuity is detected in the uplink reception sequence.	C1	Detection of a discontinuity triggers a transition of the LPT State Machine from state UP_TRANSFER to state INACTIVE. This marks the end of the reception process.
g		For each uplink part request that is received, the receiving entity of the large packet uplink subservice shall include that part in the reconstruction process of the related large packet.	C1	See definition of Progress Action of LptUpFirstCmd, LptUpInterCmd and LptUpLastCmd components
h		Upon successful completion of the uplink operation, the receiving entity of the large packet uplink subservice shall: 1. generate that large packet for subsequent routing to its destination.	C1	This is done implicitly because the large packet is re-constructed in an LPT Buffer which is then available to the host application for further processing
I		For each large packet uplink that is aborted, the receiving entity of the large packet uplink subservice shall: 1. generate a single large packet uplink abortion notification that includes the reason of that abortion; 2. discard that large packet and the related uplink part requests.	C1	1. The PUS Extension supports report (13,16) to carry the notification of an up-transfe abort. 2. If the LPT State Machine is in state INACTIVE (which would be the case after an up-transfer has been aborted), all up-transfer commands are rejected with an acceptance check failure
6.13.4.3.3a	Large packet uplink abortion report	The receiving entity of the large packet uplink shall provide the capability to generate large packet uplink abortion reports.	C1	The PUS Extension supports report (13,16) to carry the notification of an up-transfe abort.

N	Title	Requirement	C	Justification
		Each large packet uplink abortion notification shall contain: 1. the large message transaction identifier; 2. the abortion reason.	C1	See definition of LptUpAbortRep component
6.13.4.4	Subservice Observables	The following observables shall be defined for the large packet uplink subservice: 1. the number of on-going uplinks; 2. the list of the large message transaction identifiers associated to the on-going uplinks in an array of size corresponding to the maximum number of large packets that can be uplinked Concurrently.	C1	See definition of Observable Data Items associated to service 13 in [PX-SP]
6.14	Real-Time Forwarding Control	Definition of service 14	n.a.	This service is not yet supported by the PUS Extension of the CORDET Framework
6.15	On-Board Storage and Retrieval	Definition of service 15	n.a.	This service is not yet supported by the PUS Extension of the CORDET Framework
6.17.2.1.1a	Test subservice	Each test service shall contain at least one test subservice.	C1	The PUS Extension of the CORDET Framework supports service 17 in full
6.17.2.2a	Application process	Each application process shall host at most one test subservice provider.	C1	An application instantiated from the CORDET Framework can only provide one instance of a service of a given type
6.17.3a	Perform an are-you-alive connection test	The test subservice shall provide the capability to perform an are-you-alive connection test.	C1	The PUS Extension of the CORDET Framework supports sub-types 1 and 2 of service 17
b		Each request to perform an are-you-alive connection test shall contain exactly one instruction to perform an are-you-alive connection test.	C1	Command (17,1) triggers one single are-you-alive test

N	Title	Requirement	C	Justification
c		For each valid instruction to perform an are-you-alive connection test, the test subservice shall generate a single are-you-alive connection test notification that notifies that the application process that hosts the test subservice is alive and has successfully received the request.	C1	The command (17,1) triggers generation of one single report (17,2)
d		For each valid request to perform an are-you-alive connection test, the test subservice shall generate a single are-you-alive connection test report that includes the related are-you-alive connection test notification.	C1	The command (17,1) triggers generation of one single report (17,2)
6.17.4.1a	Application process accessibility	The list of application processes for which the test subservice can perform an on-board connection testing shall be declared when specifying that subservice.	NA	This requirement does not concern the implementation of the services and is therefore outside the scope of the PUS Extension of the CORDET Framework
b		For each application process for which the test subservice can perform an on-board connection testing, the criteria for a successful on-board connection test between that application process and that service shall be declared when specifying that subservice.	NA	This requirement does not concern the implementation of the services and is therefore outside the scope of the PUS Extension of the CORDET Framework. In general, the success criterium for an are-you-alive test is that a (17,2) report be generated in response to a (17,1) command but applications may specify additional timing constraints.
6.17.4.2a	Perform an on-board connection test	The test subservice capability to perform an on-board connection test shall be declared when specifying that subservice.	NA	This requirement does not concern the implementation of the services and is therefore outside the scope of the PUS Extension of the CORDET Framework

N	Title	Requirement	C	Justification
b		Each request to perform an on-board connection test shall contain exactly one instruction to perform an on-board connection test.	C1	A command (17,3) contains triggers one single on-board connection test
c		Each instruction to perform an on-board connection test shall contain: the identifier of the application process that connection test is requested.	C1	See specification command (17,3)
d		The test subservice shall reject any request to perform an on-board connection test if: 1. that request contains an instruction that refers to an application process that is not in the list of application processes for which the test subservice can perform an on-board connection testing.	C1	The start action of command (17,3) checks the legality of the target for the connection test and declares failure if this does not match an entry in a pre-defined list of application identifiers
e		For each request to perform an on-board connection test that is rejected, the test subservice shall generate a failed start of execution notification.	C1	See statement of compliance to previous requirement

N	Title	Requirement	C	Justification
f		For each valid instruction to perform an on-board connection test, the test subservice shall: 1. perform a connection test with the application process referred to by that instruction; 2. if the criteria for a successful on-board connection test with that application process are satisfied, generate a single on-board connection test notification that includes the identifier of the application process that connection has been tested. 3. if the criteria for a successful on-board connection test with that application process are not satisfied, generate a failed completion of execution verification report.	C1	See progress action of command (17,3). The connection test is implemented as an Are-You-Alive test with the target application.
g		For each valid request to perform an on-board connection test, the test subservice shall generate a single on-board connection test report that includes the related on-board connection test notification.	C1	See progress action of command (17,3) and specification of report (17,4)
6.18	On-Board Control Procedure	Definition of service 18	n.a.	This service is not yet supported by the PUS Extension of the CORDET Framework
6.19	Event-Action	Definition of service 19	n.a.	This service is not yet supported by the PUS Extension of the CORDET Framework
6.2		Definition of service 20	n.a.	This service is not yet supported by the PUS Extension of the CORDET Framework
6.21	Request Sequencing	Definition of service 21	n.a.	This service is not yet supported by the PUS Extension of the CORDET Framework

N	Title	Requirement	C	Justification
6.22	Position-Based Scheduling	Definition of service 22	n.a.	This service is not yet supported by the PUS Extension of the CORDET Framework
6.23	File Management	Definition of service 23	n.a.	This service is not yet supported by the PUS Extension of the CORDET Framework
7.3.1a	Packet field type code	Each packet field shall be associated to a packet field code that indicates the data type of any value carried by that packet field.	C2	The definition of the attributes of commands and reports is an adaptation point of the CORDET Framework (see adaptation points OCM-12 and ICM-21 in [CR-SP]). The definition of the syntactical types of these attributes is therefore done as part of the framework instantiation process.
b		Tailoring this Standard for a mission, for each new message type defined for that mission, the packet field type code of each field of that new message type shall be declared when specifying that message type.	C2	See justification of first requirement in this clause
c		Tailoring this Standard for a mission, for each message type field that packet field format code is unknown, the packet field format code of that field shall be declared when specifying the application process that uses the related message type.	C2	See justification of first requirement in this clause
d		The PTC specified in Table 7-1 shall be used to declare the PTC of each packet field.	C2	See justification of first requirement in this clause
e		The PTC of each packet field shall be declared when specifying the structure of each packet type.	C2	See justification of first requirement in this clause
7.3.2a	Booelan	Each packet field used to carry Boolean values shall be of PTC 1.	C2	See justification of first requirement in clause 7.3.1a

N	Title	Requirement	C	Justification
b		The PFCs specified in Table 7-2 shall be used for packet fields carrying Boolean values.	C2	See justification of first requirement in clause 7.3.1a
7.3.3a	Enumerated	Each packet field used to carry Boolean values shall be of PTC 2.	C2	See justification of first requirement in clause 7.3.1a
b		The PFCs specified in Table 7-3 shall be used for packet fields carrying Boolean values.	C2	See justification of first requirement in clause 7.3.1a
7.3.4a	Unsigned Integer	Each packet field used to carry Boolean values shall be of PTC 3.	C2	See justification of first requirement in clause 7.3.1a
b		Each unsigned integer value shall be encoded with Bit 0 being the most significant bit (MSB) and Bit N1 the least significant bit (LSB).	C2	See justification of first requirement in clause 7.3.1a
c		The PFCs specified in Table 7-4 shall be used for packet fields carrying unsigned integer values.	C2	See justification of first requirement in clause 7.3.1a
7.3.5a	Signed Integer	Each packet field used to carry Boolean values shall be of PTC 4.	C2	See justification of first requirement in clause 7.3.1a
		Bit 0 of each signed integer parameter shall be used to determine the sign of the parameter value.	C2	See justification of first requirement in clause 7.3.1a
		The PFCs specified in Table 7-5 shall be used for packet fields carrying unsigned integer values.	C2	See justification of first requirement in clause 7.3.1a
7.3.6a	Real	Each packet field used to carry Boolean values shall be of PTC 5.	C2	See justification of first requirement in clause 7.3.1a
		The PFCs specified in Table 7-6 shall be used for packet fields carrying Boolean values.	C2	See justification of first requirement in clause 7.3.1a
7.3.7a	Bit-String	Each packet field used to carry Boolean values shall be of PTC 6.	C2	See justification of first requirement in clause 7.3.1a

N	Title	Requirement	C	Justification
b		The PFCs specified in Table 7-7 shall be used for packet fields carrying Boolean values.	C2	See justification of first requirement in clause 7.3.1a
c		The variable length bitstring shall have the structure specified in Figure 7-3.	C2	See justification of first requirement in clause 7.3.1a
d		For each application process that uses variable-length octet-strings, the PFC of the length field of the variable-length bit-string format shall be declared when specifying that application process.	C2	See justification of first requirement in clause 7.3.1a
e		Each spare field of a telemetry or a telecommand packet shall be of fixed-length PTC 6.	C2	See justification of first requirement in clause 7.3.1a
f		For each spare field of a telemetry or a telecommand packet, all bits of that field shall be set to zero.	C2	See justification of first requirement in clause 7.3.1a
g		For each packet field containing a fixed-length bit-string whose length is deduced, the definition used to deduce that length shall be declared when specifying the related packet field type.	C2	See justification of first requirement in clause 7.3.1a
h		For each packet field containing a fixed-length bit-string whose length is deduced, the deduction of the length shall only result from the content of one or more preceding fields of the same packet, of one or more mission constants or a combination of both.	C2	See justification of first requirement in clause 7.3.1a
7.3.8a	Octet-String	Each packet field used to carry Boolean values shall be of PTC 7.	C2	See justification of first requirement in clause 7.3.1a

N	Title	Requirement	C	Justification
b		The PFCs specified in Table 7-8 shall be used for packet fields carrying Boolean values.	C2	See justification of first requirement in clause 7.3.1a
c		The variable length octet-string shall have the structure specified in Figure 7-3.	C2	See justification of first requirement in clause 7.3.1a
d		For each application process that uses variable-length octet-strings, the PFC of the length field of the variable-length bit-string format shall be declared when specifying that application process.	C2	See justification of first requirement in clause 7.3.1a
e		For each packet field containing a fixed-length octet-string whose length is deduced, the definition used to deduce that length shall be declared when specifying the related packet field type.	C2	See justification of first requirement in clause 7.3.1a
f		For each packet field containing a fixed-length octet-string whose length is deduced, the deduction of the length shall only result from the content of one or more preceding fields of the same packet, of one or more mission constants or a combination of both.	C2	See justification of first requirement in clause 7.3.1a
7.3.9a	Character-String	Each packet field used to carry character-string values shall be of PTC 8.	C2	See justification of first requirement in clause 7.3.1a
b		The values that character-string parameters can take shall be sequences of visible characters.	C2	See justification of first requirement in clause 7.3.1a
c		The PFCs specified in Table 7-9 shall be used for packet fields carrying character-string values.	C2	See justification of first requirement in clause 7.3.1a

N	Title	Requirement	C	Justification
d		The variable length characterstring format shall have the structure specified in Figure 7-5:	C2	See justification of first requirement in clause 7.3.1a
e		For each application process that uses variable-length character-strings, the PFC of the length field of the variable-length character-string format shall be declared when specifying that application process.	C2	See justification of first requirement in clause 7.3.1a
f		For each packet field containing a fixed-length character-string whose length is deduced, the definition used to deduce that length shall be declared when specifying the related packet field type.	C2	See justification of first requirement in clause 7.3.1a
g		For each packet field containing a fixed-length character-string whose length is deduced, the deduction of the length shall only result from the content of one or more preceding fields of the same packet, of one or more mission constants or a combination of both.	C2	See justification of first requirement in clause 7.3.1a
7.3.10a	Absolute Time	Each packet field used to carry absolute time values shall be of PTC 9.	C2	See justification of first requirement in clause 7.3.1a
b		Each absolute time parameter value shall be a positive time offset that is a number of seconds and fractions of a second from a given epoch.	C2	See justification of first requirement in clause 7.3.1a
c		If the absolute time parameter has CDS format, the standard CCSDS epoch of 1958 January 1 shall be used.	C2	See justification of first requirement in clause 7.3.1a

N	Title	Requirement	C	Justification
d		The PFCs specified in Table 7-10 shall be used for packet fields carrying absolute time values.	C2	See justification of first requirement in clause 7.3.1a
7.3.11a	Relative Time	Each packet field used to carry relative time values shall be of PTC 10.	C2	See justification of first requirement in clause 7.3.1a
b		Each relative time parameter value shall be a positive or a negative time offset that is the number of seconds and fractions of a second from the occurrence time of an event whose identification can be derived from other parameters in the packet (identifying a type of on-board event) or a number of seconds and fractions of a second between two absolute times.	C2	See justification of first requirement in clause 7.3.1a
c		The PFCs specified in Table 7-11 shall be used for packet fields carrying relative time values.	C2	See justification of first requirement in clause 7.3.1a
7.3.12a	Deduced	Each packet field whose structure and format is deduced shall be of PTC 11 PFC 0.	C2	See justification of first requirement in clause 7.3.1a
b		For each packet field whose structure and format is deduced, the definition used to deduce that structure and format shall be declared when specifying the related packet field type.	C2	See justification of first requirement in clause 7.3.1a
c		For each packet field whose structure and format is deduced, the deduction of the structure and format shall only result from the content of one or more preceding fields of the same packet, of one or more mission constants or a combination of both.	C2	See justification of first requirement in clause 7.3.1a

N	Title	Requirement	C	Justification
7.313a	Packet	Each packet field used to carry packets shall be of PTC 12.	C2	See justification of first requirement in clause 7.3.1a
b		The PFCs specified in Table 7-12 shall be used for packet fields carrying packets.	C2	See justification of first requirement in clause 7.3.1a
7.4.2	The CCSDS Space Packet	Once a telecommand or a telemetry packet has been generated by an application process, no one shall update that packet.	C1	The act of generating a packet in the CORDET Framework coincides with its being executed by its OutManager. After this execution is completed, the packet is handed over to the middleware through the OutStream and can no longer be accessed by the framework infrastructure.
7.4.3.1a	Telemetry packet secondary header	With the exception of the spacecraft time packets specified in clauses 6.9.4.2 and 6.9.4.3, all telemetry packets defined in this Standard shall have a telemetry packet secondary header.	C2	The PUS Extension of the CORDET Framework specifies the existence of a number of attributes (see section 4 of [PX-SP]) but their precise definition is done during the framework instantiation process.
b		Each telemetry packet secondary header shall have the structure specified in Figure 7-7.	C1/C2	See statement of compliance to the next requirements in this clause
c		Each application process shall set the TM packet PUS version number of each telemetry packet it generates to 2.	C2	This field does not exist in the CORDET Framework but, since its value is fixed, it can be added by applications when they implement the functions which fill in the header of their telemetry packets.

N	Title	Requirement	C	Justification
d		Each application process that provides the capability to report the spacecraft time reference status used when time tagging telemetry packets shall set the spacecraft time reference status field of each telemetry packet it generates to the status of the on-board time reference used when time tagging that telemetry packet.	C2	The value of this field is provisionally assumed to be zero. This may change after service 9 has been defined (TBC).
e		Each application process that does not provide the capability to report the status of the on-board time reference used when time tagging telemetry packets shall set the spacecraft time reference status field of each telemetry packet it generates to 0.	C2	See statement of compliance to previous requirement
f		For each report that it generates, each application process shall set the message type ID field of the corresponding telemetry packet to the message type identifier of that report.	C1	The CORDET Framework pre-defines the type and sub-type attribute which, taken together, constitute the message type.
g		For each report that it generates, each application process that provides the capability to count the type of generated messages per destination and report the corresponding message type counter shall set the message type counter of the related telemetry packet to the value of the related counter.	C1	The CORDET Framework maintains counters of messages generated for each [APID,Destination] pair but it does not, by default, provide the capability to count the number of generated messages of a given type.

N	Title	Requirement	C	Justification
h		Each application process that does not provide the capability to count the type of generated messages per destination and report the corresponding message type counter shall set the message type counter field of each telemetry packet it generates to 0.	C1/C2	This capability is, by default, not provided by the CORDET Framework and hence this field can be set to zero (unless an application decides to provide the capability at its own level)
I		Each application process shall set the destination ID field of each telemetry packet it generates to the application process user identifier of the application process addressed by the related report.	C1	See mapping of destination field in section 4 of [PX-SP].
j		The PFC of the time field of telemetry packets shall be declared when specifying the time service used by the spacecraft.	C2	See statement of compliance to clause 7.3.1a.
k		Each application process shall set the time field of each telemetry packet it generates to the time tag of the related report.	C1	The time-stamp of out-going components is set by the Send Packet Procedure of the OutComponent of the CORDET Framework (see section 6.1.1 of the CORDET Framework Definition Document).
l		For each application process, the presence and bit-size of the spare field of the telemetry packet secondary header shall be declared when specifying that application process.	C2	See statement of compliance to clause 7.3.1a.
7.4.3.2a	Telemetry User Data Field	Each telemetry user data field shall have the structure specified in Figure 7-8.	C2	See statement of compliance to clause 7.3.1a.
		The telemetry padding word size used by each application process shall be declared when specifying that application process.	C2	See statement of compliance to clause 7.3.1a.

N	Title	Requirement	C	Justification
		For each telemetry packet that it generates, each application process shall ensure that the total length of that packet is an integer multiple of the padding word size declared for that application process by including a user data spare field of the minimum bit-size that results in that integer multiple.	C2	See statement of compliance to clause 7.3.1a.
		Whether checksumming telemetry packets is used shall be declared when tailoring this standard to the mission.	NA	The CORDET Framework treats check-summing as a middleware-level function and therefore does not provide an interface for computing the check-sum of a packet.
		If checksumming telemetry packets is used for the mission, the type of checksum to use, that is either the ISO standard 16-bits checksum or the CRC standard 16-bits, shall be declared when tailoring this standard to the mission.	NA	See statement of compliance to previous requirement
		If checksumming telemetry packets is used for the mission, for each telemetry packet that it generates, each application process shall: 1. calculate the checksum of that packet, and 2. set the calculated value in the packet error control field of that packet.	NA	See statement of compliance to previous requirement
7.4.4.1a	Telecommand packet secondary header	With the exception of the CPDU command packet specified in clause 9, all telecommand packets defined in this Standard shall have a telecommand packet secondary header.	C2	The PUS Extension of the CORDET Framework specifies the existence of a number of attributes (see section 4 of [PX-SP]) but their precise definition is done during the framework instantiation process.

N	Title	Requirement	C	Justification
b		Each telecommand packet secondary header shall have the structure specified in Figure 7-9.	C1/C2	See statement of compliance to the next requirements in this clause
c		For each request that it issues, each application process shall set the TC packet PUS version number to 2.	C1	This field does not exist in the CORDET Framework and the framework ignores it.
d		For each request that it issues, each application process shall set: the bit 3 of the acknowledgement flags field of the corresponding telecommand packet to: (a) 1 if the reporting of the successful acceptance of that request by the destination application process is requested (b) 0 otherwise; the bit 2 of the acknowledgement flags field of the corresponding telecommand packet to: (a) 1 if successful start of execution of that request by the destination application process is requested; (b) 0 otherwise; the bit 1 of the acknowledgement flags field of the corresponding telecommand packet to: (a) 1 if the reporting of the successful progresses of execution of that request by the destination application process is requested; (b) 0 otherwise; the bit 0 of the acknowledgement flags field of the corresponding telecommand packet to: (a) 1 if the reporting of the successful completion of execution of the related request by the destination application process is requested; (b) 0 otherwise.	C1	The CORDET Framework defines acknowledge flags for commands with the same semantics as the PUS (see section 4 of [PX-SP]).

N	Title	Requirement	C	Justification
e		For each request that it issues, each application process shall set the message type ID field of the corresponding telecommand packet to the message type identifier of that request.	C1	The CORDET Framework pre-defines the type and sub-type attribute which, taken together, constitute the message type.
f		For each request that it issues, each application process shall set the source ID field to its source identifier.	C1	See mapping of source field in section 4 of [PX-SP].
g		For each application process that issues requests, the presence and bit-size of the spare field of the telecommand packet secondary header shall be declared when specifying that application process.	C2	See statement of compliance to clause 7.3.1a.
7.4.4.2a	Telecommand User Data Field	Each telecommand user data field shall have the structure specified in Figure 7-10.	C2	See statement of compliance to clause 7.3.1a.
b		The telecommand padding word size used for each application process shall be declared when specifying that application process.	C2	See statement of compliance to clause 7.3.1a.
c		For each telecommand packet that it generates, each application process shall ensure that the total length of that packet is an integer multiple of the padding word size declared for that application process, by including a user data spare field of the minimum bit-size that results in that integer multiple.	C2	See statement of compliance to clause 7.3.1a.

N	Title	Requirement	C	Justification
d		The type of checksum to use for checksumming all telecommand packets, which is either the ISO standard 16-bits checksum or the CRC standard 16-bits checksum, shall be declared when tailoring this standard to the mission.	C2	See statement of compliance to clause 7.3.1a.
		For each telecommand packet that it generates, each application process shall: 1. calculate the checksum of that packet, and 2. set the calculated value in the packet error control field of that packet.		