

THE CORDET FRAMEWORK

PUS Extension

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

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

Written By:	Alessandro Pasetti
Checked By:	Marcel Opprecht
Document Ref.:	PP-DF-COR-0003
Issue:	Work-in-progress
Created On:	06/01/2018, at: 20:50

Contents

1	Change History	7
2	Applicable and Reference Documents	8
3	Introduction	9
3.1	Scope of CORDET Framework	9
3.2	Scope of PUS Extension of CORDET Framework	10
3.2.1	Overview of Supported Services	11
3.3	Specification Format	12
3.4	Compliance to PUS Requirements	13
4	Report and Command Attributes	17
4.1	Mapping of Discriminant Attribute	18
4.2	Mapping of Group Attribute	18
4.3	Requirements	19
5	The Data Pool Component	20
5.1	Data Pool Concepts	20
5.2	Data Pool Behaviour	20
5.3	Service Observability Concept	22
5.4	Service Parameterization Concept	22
5.5	Adaptation Points	22
5.6	Requirements	23
6	Report and Command Factories	24
6.1	Observables	24
6.2	Adaptation Points	25
6.3	Requirements	25
7	Definition of PUS Services	26
7.1	Report and Command Adaptation Points	27
7.2	Dependencies Between Services	31
7.3	Requirements	32
8	Request Verification Service	35
8.1	Service 1 Report and Command Definition	37
8.2	Service 1 Observables	39
8.3	Service 1 Adaptation Points	40
8.4	Service 1 Requirements	41
9	Housekeeping Service	45
9.1	Report Definition List (RDL)	46
9.2	Management of Super-Commutated Data Items	47
9.3	Debug Variables	48
9.4	Service 3 Report and Command Definition	48
9.5	Service 3 Constants	53
9.6	Service 3 Observables	53
9.7	Service 3 Parameters	54
9.8	Service 3 Requirements	54
10	Event Reporting Service	61

10.1 Pre-Defined Event Reports	62
10.2 Service 5 Report and Command Definition	62
10.3 Service 5 Constants	65
10.4 Service 5 Observables	65
10.5 Service 5 Requirements	66
11 On-Board Monitoring Service	68
11.1 Parameter Monitoring Sub-Service	68
11.1.1 Monitor Procedures	71
11.1.2 Check Transition List	72
11.2 Functional Monitoring Sub-Service	73
11.3 Service 12 Report and Command Definition	75
11.4 Service 12 Constants	83
11.5 Service 12 Observables	83
11.6 Service 12 Parameters	85
11.7 Service 12 Adaptation Points	85
11.8 Service 12 Requirements	85
12 Large Packet Transfer Service	99
12.1 LPT Buffers	99
12.2 The LPT State Machine	101
12.2.1 Management of Down-Transfers	101
12.2.2 Management of Up-Transfers	103
12.3 Service 13 Report and Command Definition	104
12.4 Service 13 Constants	109
12.5 Service 13 Observables	109
12.6 Service 13 Adaptation Points	110
12.7 Service 13 Parameters	110
12.8 Service 13 Requirements	111
13 Test Service	113
13.1 Service 17 Command and Report Definition	113
13.2 Service 17 Observables	116
13.3 Service 17 Parameters	117
13.4 Service 17 Requirements	117
14 Event Action Service	119
A Pre-Defined Event Reports	120
B Error Reports	122
C Request Verification Failure Codes	124
D PUS Requirements Compliance Matrix	126

List of Figures

3.1	Applications as Providers and Users of Services	9
3.2	Hierarchical Definition of Services	11
7.1	Extension of InCommand Component	26
7.2	Model of InCommand Component	27
8.1	Packet Rerouting Failure Procedure	42
8.2	Packet Acceptance Failure Procedure	42
8.3	Command Verification Success Procedure	43
8.4	Command Verification Failure Procedure	43
8.5	Command Progress Success Procedure	44
8.6	Command Progress Failure Procedure	44
9.1	Start Action of Command HkCreateCmd	56
9.2	Start Action of Command HkDeleteCmd	57
9.3	Start Action of Multi-SID Commands	58
9.4	Progress Action of Command HkRepStructCmd	59
9.5	Ready Check of Report HkRep	60
10.1	Start Action of Multi-EID Commands	67
11.1	Parameter Monitoring Procedure	87
11.2	Limit Check Monitor Procedure	88
11.3	Expected Value Monitor Procedure	88
11.4	Delta Value Monitor Procedure	89
11.5	CTL Processing Procedure	89
11.6	Functional Monitor Notification Procedure	90
11.7	Start Action of (12,1) and (12m2) Command Procedure	91
11.8	Start Action of (12,5) Command Procedure	92
11.9	Start Action of (12,6) Command Procedure	93
11.10	Start Action of (12,7) Command Procedure	94
11.11	Start Action of (12,8) Command Procedure	95
11.12	Start Action of (12,23) Command Procedure	96
11.13	Start Action of (12,24) Command Procedure	97
11.14	Start Action of Multi-Functional Monitor Command Procedure	98
12.1	Large Packet Transfer (LPT) State Machine	101
12.2	Up-Transfer Start Action	112
13.1	Start Action of OnBoardConnectCmd Command (17,3)	118
13.2	Progress Action of OnBoardConnectCmd Command (17,3)	118

List of Tables

1.1	Detailed List of Changes in Issue 0.1	7
2.1	Applicable Documents	8
2.2	Reference Documents	8
3.1	Services Supported by PUS Extension	11
3.2	Terminological Mapping PUS-CORDET	16
4.1	Mapping of CORDET Attributes to PUS Attributes	17
4.2	Requirements for Command and Report Attributes	19
5.1	Adaptation Points for Data Pool Component	22
5.2	Requirements for Data Pool Component	23
6.1	Data Items for Factory Components	24
6.2	Requirements for Factory Components	25
7.1	Adaptation Points for PUS Reports	28
7.2	Adaptation Points for PUS Commands	30
7.3	Service Dependencies	31
7.4	Requirements for Framework Extension Commands and Reports	32
7.5	Supported PUS Commands and Reports	33
8.1	Sources of Routing, Acceptance and Execution Notifications	36
8.2	Specification of ReqVerRep Component	39
8.3	Observables for Service 1 (Request Verification)	39
8.4	Adaptation Points for Service 1 (Request Verification)	40
8.5	Requirements for Service 1 (Request Verification)	41
9.1	Fields in Report Definition Data Structure	46
9.2	Specification of HkCreateCmd Component	49
9.3	Specification of HkDeleteCmd Component	49
9.4	Specification of HkEnableCmd Component	50
9.5	Specification of HkDisableCmd Component	50
9.6	Specification of HkRepStructCmd Component	51
9.7	Specification of HkRepStructRep Component	51
9.8	Specification of HkRep Component	52
9.9	Specification of HkOneShotCmd Component	52
9.10	Constants for Service 3 (Housekeeping Service)	53
9.11	Observables for Service 3 (Housekeeping Service)	53
9.12	Parameters for Service 3 (Housekeeping Service)	54
9.13	Requirements for Service 3 (Housekeeping Service)	54
10.1	Specification of EvtRep Component	62
10.2	Specification of EvtEnableCmd Component	63
10.3	Specification of EvtDisableCmd Component	63
10.4	Specification of EvtRepDisabledCmd Component	64
10.5	Specification of EvtRepDisableRep Component	64
10.6	Constants for Service 5 (Event Reporting Service)	65
10.7	Observables for Service 5 (Event Reporting Service)	65
10.8	Requirements for Service 5 (Event Reporting Service)	66
11.1	Attributes of Parameter Monitor	70
11.2	Return Values of Monitor Procedure Execution	72
11.3	Attributes of a Check Transition	72
11.4	Attributes of a Functional Monitor	74
11.5	Checking Statuses of a Functional Monitor	74
11.6	Specification of MonEnbMonDefCmd Component	75
11.7	Specification of MonDisParMonDefCmd Component	75

11.8 Specification of MonChgTransDelCmd Component	75
11.9 Specification of MonDelAllParMonCmd Component	77
11.10 Specification of MonAddParMonCmd Component	77
11.11 Specification of MonDelParMonCmd Component	77
11.12 Specification of MonModParMonCmd Component	79
11.13 Specification of MonRepParMonCmd Component	79
11.14 Specification of MonRepParMonRep Component	79
11.15 Specification of MonCheckTransRep Component	81
11.16 Specification of MonRepParMonStatCmd Component	81
11.17 Specification of MonEnbMonFuncCmd Component	82
11.18 Specification of MonDisMonFuncCmd Component	82
11.19 Specification of MonAddFuncMonDefCmd Component	82
11.20 Constants for Service 12 (On-Board Monitoring Service)	83
11.21 Observables for Service 12 (On-Board Monitoring)	83
11.22 Parameters for Service 12 (On-Board Monitoring Service)	85
11.23 Adaptation Points for Service 12 (On-Board Monitoring Service)	85
11.24 Requirements for Service 12 (On-Board Monitoring Service)	85
12.1 Specification of LptDownFirstRep Component	104
12.2 Specification of LptDownInterRep Component	105
12.3 Specification of LptDownLastRep Component	105
12.4 Specification of LptUpFirstCmd Component	106
12.5 Specification of LptUpInterCmd Component	106
12.6 Specification of LptUpLastCmd Component	107
12.7 Specification of LptUpAbortRep Component	107
12.8 Specification of LptStartDownCmd Component	108
12.9 Specification of LptAbortDownCmd Component	108
12.10 Constants for Service 13 (Large Packet Transfer Service)	109
12.11 Observables for Service 13 (Large Packet Transfer Service)	109
12.12 Adaptation Points for Service 13 (Large Packet Transfer Service)	110
12.13 Parameters for Service 3 (Large Packet Transfer Service)	110
12.14 Requirements for Service 13 (Large Packet Transfer Service)	111
13.1 Specification of AreYouAliveCmd Component	115
13.2 Specification of AreYouAliveRep Component	115
13.3 Specification of OnBoardConnectCmd Component	116
13.4 Specification of OnBoardConnectRep Component	116
13.5 Observables for Service 17 (Test Service)	116
13.6 Parameters for Service 17 (Test Service)	117
13.7 Requirements for Service 17 (Test Service)	117
A.1 Error Reports	121
B.1 Error Reports	123
C.1 Request Verification Failure Codes	124
D.1 Mapping of PUS Requirements to CORDET Requirement	127

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).

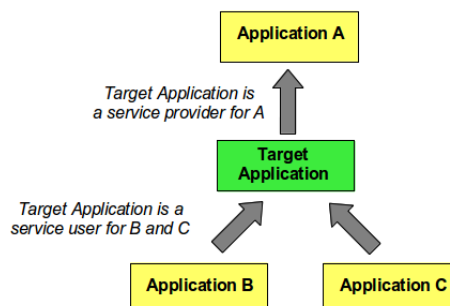


Fig. 3.1: Applications as Providers and Users of Services

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.

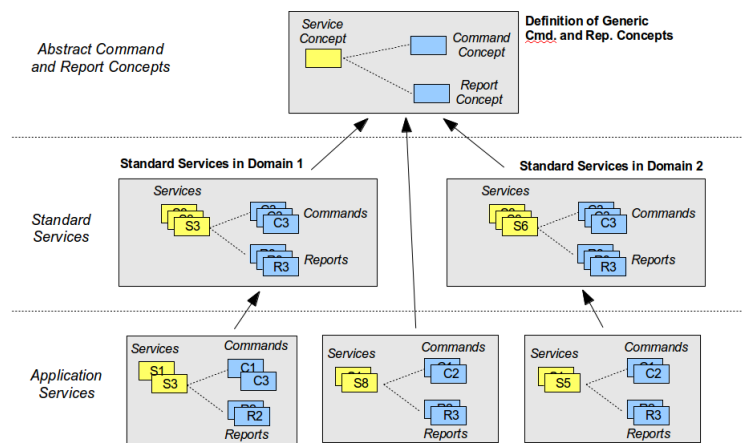


Fig. 3.2: Hierarchical Definition of Services

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	11
13	Large Packet Transfer Service	12
17	Test Service	13
19	Event Action Service	14

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.

3.4.0.1 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).

3.4.0.2 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. In this respect, the PUS Framework takes the approach that, for instructions, only execution failures are reported and that they are reported unconditionally.

3.4.0.3 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 checks¹. 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 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:

¹See, 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

- 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, the instruction-level checks are implemented within the execution actions themselves. The request-level check is considered to be successful as long as at least one instruction-level check has been passed. In accordance with the rule stated in the previous paragraph, for instructions, only execution failures are reported.

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' and the PUS stipulates that, as part of the Start-of-Execution verification, valid SIDs should be accepted for execution whereas invalid SIDs should trigger Failed-Start-of-Execution notifications which might eventually trigger the generation of (1,4) reports. The PUS Extension of the CORDET Framework responds to this requirement by specifying that the Start Action of the (3,5) command evaluates the validity of the SIDs and generates the (1,4) reports for each invalid SID. The Start Action is considered to be successful as long as at least one valid SID is found.

3.4.0.4 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.

3.4.0.5 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.

3.4.0.6 Command Abort in Case of Progress of Execution Failures

Point 3 of clause 5.4.11.2.3a implies that a Progress-of-Execution failure for a command does not necessarily result in the command being aborted. By default, the CORDET Framework assumes that a command which has encountered a progress-of-execution failure is aborted after having generated a Progress-of-Execution Failure Report (see InCommand State Machine in reference [CR-SP]). It is TBC whether this behaviour is consistent with the PUS. In any case, applications who wish to generate a Completion-of-Execution Failure Report after the Progress-of-Execution Failure Report can do so in the Abort Action associated to the command.

3.4.0.7 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.
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 it is a positive integer which is equal to the number of times that the Progress Action has been executed since the execution of the command started.
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. In most cases, the mapping is straightforward but, in the case of the discriminant and of the APID, clarifications are in order which are provided in the next two sub-sections.

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

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	<i>Components encapsulating a command or a report shall implement all attributes defined for them by the PUS</i>
P-CRA-2/S	<i>Components encapsulating a command or a report shall provide operations to access in read and write mode all their PUS-defined attributes</i>
P-CRA-3/S	<i>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</i>
P-CRA-4/C	<i>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</i>
P-CRA-5/C	<i>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</i>

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 Procedure² 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-

²It 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 time³ 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

³The 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 return 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	<i>The PUS Extension of the CORDET Framework shall provide a Data Pool component as an extension of the Base Component</i>
P-DP-2/A	<i>The Data Pool Component shall support the adaptation points specified in table 5.1</i>
P-DP-3/S	<i>When it is configured, the Data Pool Component shall provide operations to let other components access the current value of its data items</i>
P-DP-4/S	<i>When it is configured, the Data Pool Component shall provide operations to let other components update the current value of its data items</i>
P-DP-5/S	<i>Deleted</i>
P-DP-5/C	<i>An application shall instantiate the Data Pool Component only once</i>
P-DP-6/C	<i>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</i>

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: Data Items for Factory Components

Name	Description
nOfAllocatedInRep	Number of InReports which are currently allocated (i.e. which have been successfully created by the InFactory and not yet released)
nOfAllocatedInCmd	Number of InCommands which are currently allocated (i.e. which have been successfully created by the InFactory and not yet released)
nOfAllocatedOutCmp	Number of OutComponents which are currently allocated (i.e. which have been successfully created by the OutFactory and not yet released)
nOfFailedInRep	Number of InReports whose creation by the InFactory failed
nOfFailedInCmd	Number of InCommands whose creation by the InFactory failed
nOfFailedOutCmp	Number of OutComponents whose creation by the OutFactory failed
nOfTotAllocatedInRep	Number of InReports successfully created by the InFactory since application start
nOfTotAllocatedInCmd	Number of InCommands successfully created by the InFactory since application start
nOfTotAllocatedOutCmp	Number of OutComponents successfully created by the InFactory 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 is 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	<i>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</i>
P-FAC-2/S	<i>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</i>
P-FAC-3/S	<i>The two factory components shall maintain and make accessible through the data pool the observables listed in table 6.1</i>

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.

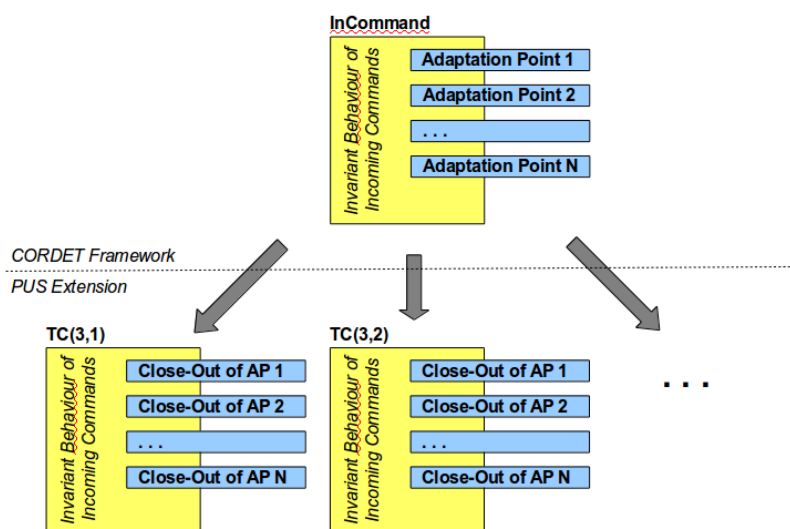


Fig. 7.1: Extension of InCommand Component

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 lists the command and report components provided by the PUS Extension of the CORDET Framework to support the PUS services. The first column in the table gives the name of the CORDET component which implements the command or report; the second column gives its PUS names as it is given in section 8 of [PS-SP]; and the third column gives the [type,sub-type] pair which identifies the command or report. Note that, in some cases, the same CORDET components implements two PUS commands or reports. This is the case where the two commands or reports share the same behaviour and only differ for the value of some of their attributes (e.g. the (3,25) and (3,26) reports).

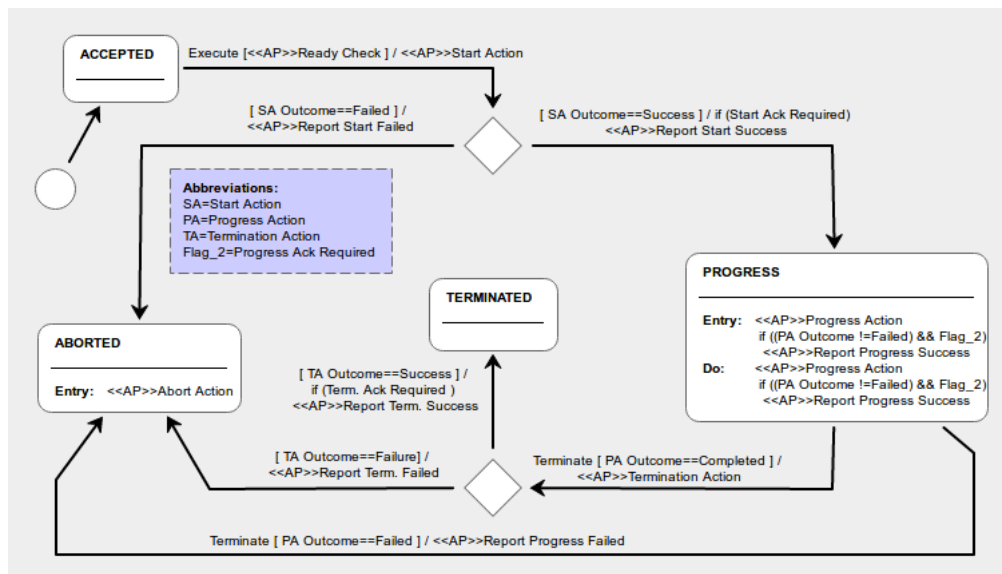


Fig. 7.2: Model of InCommand Component

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'

AP ID	Adaptation Point	Close-Out Value
P-OCM-4	Configuration Action in Reset Procedure of OutComponent (Closes OCM-4)	Do nothing and return 'action successful'
P-OCM-5	Shutdown Action in Base Component of OutComponent (Closes OCM-5)	Do nothing
P-OCM-6	Execution Procedure of Out-Component (Closes OCM-6)	Do nothing
P-OCM-7	Service Type Attribute of Out-Component (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 Out-Component (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 Out-Component (Closes OCM-11)	See definition of individual reports
P-OCM-12	Parameter Attribute of Out-Component (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 Out-Component (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 In-Command (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 In-Command (Closes ICM-20)	See definition of individual commands
P-ICM-21	Parameter Attributes of In-Command (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 13 and TBD generate event reports and therefore need service 5
- TBD

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	TBD
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	<i>The InCommand components provided by the PUS Extension of the CORDET Framework shall close the InCommand adaptation points as stated in table 7.1</i>
P-PCR-2/A	<i>The OutComponent components provided by the PUS Extension of the CORDET Framework shall close the OutComponent adaptation points as stated in table 7.2</i>
P-PCR-3/C	<i>Applications shall comply with the service dependencies listed in table 7.3</i>

Table 7.5: Supported PUS Commands and Reports

CORDET Name	PUS Name	Type
	Request Verification Service	(1,*)
ReqVerRep	Successful Acceptance Verification Report	(1,1)
ReqVerRep	Failed Acceptance Verification Report	(1,2)
ReqVerRep	Successful Start of Execution Verification Report	(1,3)
ReqVerRep	Failed Start of Execution Verification Report	(1,4)
ReqVerRep	Successful Progress of Execution Verification Report	(1,5)
ReqVerRep	Failed Progress of Execution Verification Report	(1,6)
ReqVerRep	Successful Completion of Execution Verification Report	(1,7)
ReqVerRep	Failed Completion of Execution Verification Report	(1,8)
ReqVerRep	Failed Routing Verification Report	(1,10)
	Housekeeping Service	(3,*)
HkCreateCmd	Create a Housekeeping Parameter Report Structure	(3,1)
HkCreateCmd	Create a Diagnostic Parameter Report Structure	(3,2)
HkDeleteCmd	Delete a Housekeeping Parameter Report Structure	(3,3)
HkDeleteCmd	Delete a Diagnostic Parameter Report Structure	(3,4)
HkEnableCmd	Enable Periodic Generation of a Housekeeping Parameter Report Structure	(3,5)
HkDisableCmd	Disable Periodic Generation of a Housekeeping Parameter Report Structure	(3,6)
HkEnableCmd	Enable Periodic Generation of a Diagnostic Parameter Report Structure	(3,7)
HkDisableCmd	Disable Periodic Generation of a Diagnostic Parameter Report Structure	(3,8)
HkRepStructCmd	Report Housekeeping Parameter Report Structure	(3,9)
HkRepStructRep	Housekeeping Parameter Report Structure Report	(3,10)
HkRepStructCmd	Report Diagnostic Parameter Report Structure	(3,11)
HkRepStructRep	Diagnostic Parameter Report Structure Report	(3,12)
HkRep	Housekeeping Parameter Report	(3,25)
HkRep	Diagnostic Parameter Report	(3,26)
HkOneShotCmd	Generate One-Shot Report for Housekeeping Parameters	(3,27)
HkOneShotCmd	Generate One-Shot Report for Diagnostic Parameters	(3,28)
	Event Reporting Service	(5,*)
EvtRep	Informative Event Report (Level 1)	(5,1)
EvtRep	Low Severity Event Report (Level 2)	(5,2)
EvtRep	Medium Severity Event Report (Level 3)	(5,3)
EvtRep	High Severity Event Report (Level 4)	(5,4)
EvtEnableCmd	Enable Generation of Event Identifiers	(5,5)
EvtDisableCmd	Disable Generation of Event Identifiers	(5,6)
EvtRepDisabledCmd	Report the List of Disabled Event Identifiers	(5,7)
EvtRepDisabledRep	List of Disabled Event Identifiers	(5,8)
	On-Board Monitoring Service	(12,*)
MonEnbParMonCmd	Enable Parameter Monitoring Definitions	(12,1)

CORDET Name	PUS Name	Type
MonDisParMonCmd	Disable Parameter Monitoring Definitions	(12,2)
MonChgTransDelCmd	Change Maximum Transition Reporting Delay	(12,3)
MonDelAllParMonCmd	Delete All Parameter Monitoring Definitions	(12,4)
MonAddParMonCmd	Add Parameter Monitoring Definitions	(12,5)
MonDelParMonCmd	Delete Parameter Monitoring Definitions	(12,6)
MonModParMonCmd	Modify Parameter Monitoring Definitions	(12,7)
		(,)
MonChkTransRep	Check Transition Report	(12,12)
		(,)
		(,)
MonEnbMonFncCmd	Enable Parameter Monitoring Function	(12,15)
MonDisMonFncCmd	Disable Parameter Monitoring Function	(12,16)
		(,)
		(,)
		(,)
	Large Packet Transfer Service	(13,*)
LptDownFirstRep	First Downlink Part Report	(13,1)
LptDownInterRep	Intermediate Downlink Report	(13,2)
LptDownLastRep	Last Downlink Part Report	(13,3)
LptUpFirstCmd	First Uplink Part Report	(13,9)
LptUpInterCmd	Intermediate Uplink Part Report	(13,10)
LptUpLastCmd	Last Uplink Part Report	(13,11)
LptUpAbortRep	Large Packet Uplink Abortion Report	(13,16)
LptStartDownCmd	Trigger Large Packet Down-Transfer	(13,129)
LptAbortDownCmd	Abort Large Packet Down-Transfer	(13,130)
	Test Service	(17,*)
AreYouAliveCmd	Perform Are-You-Alive Connection Test	(17,1)
AreYouAliveRep	Are-You-Alive Connection Report	(17,2)
OnBoardConnectCmd	Perform On-Board Connection Test	(17,3)
OnBoardConnectRep	On-Board Connection Report	(17,4)

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 reports (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 <i>Report Packet Destination Invalid Operation</i> 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 <i>Report Acceptance Failure Operation</i> which is called by the InLoader Load Command/Report Procedure when an incoming command has failed its acceptance check.	ILD-14
Acceptance Success Notification	This notification is issued by the <i>Report Acceptance Success Operation</i> 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 <i>Report Start Successful for InCommand Operation</i> 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 <i>Report Start Failed for InCommand Operation</i> 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 <i>Report Progress Successful for InCommand Operation</i> 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 <i>Report Progress Failed for InCommand Operation</i> 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
Execution Termination Success Notification	This notification is issued by the <i>Report Termination Successful for InCommand Operation</i> 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 <i>Report Termination Failed for InCommand Operation</i> 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 8.4. 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 reports (1,5) and (1,6) report, respectively, the success and failure of a progress step. The CORDET Framework has the concept of 'Progress Step' which is a counter which counts the number of times an InCommand has been executed since it was 'in progress' (i.e. since it entered state PROGRESS). It is recognized that this mechanism may result in a step granularity which is too fine for some applications. The default logic for the generation of the (1,5) and (1,6) reports is then as follows:

- A return value of 'failed' for the Progress Action of the InCommand is interpreted as a progress step failure which triggers a (1,6) report.
- A return value of 'continue' for the Progress Action of the InCommand may be interpreted as a progress step success which triggers a (1,5) report according to an application-specific logic to be inserted in adaptation point ICM-15 (Operation to Report Progress Success for InCommand).

Note that the second bullet implies that the adaptation point ICM-15 cannot be closed at framework level but must instead remain open so that applications may decide the conditions under which a progress action has completed a step.

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 one single OutComponent, ReqVerRep, to encapsulate any of the nine service reports. Use of a single component is legitimate because all service 1 reports share the same behaviour (i.e. they only differ in the data they carry but have the same checks and definitions).

The ReqVerRep component is implemented as an extension of the OutComponent component. It is therefore defined by the way it closes the adaptation points of the OutComponent. Table 8.4 lists the OutComponent adaptation points and shows 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 Data is stored in data pool item **verFailData**. Its purpose 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.

Table 8.2 formally specifies the ReqVerRep component 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 verification failure code or who wish to disable a certain service 1 sub-type can do so by setting its status to 'disabled' in the OutRegistry. Note that this is a run-time operation which would typically be done as part of an application's initialization.

Table 8.2: Specification of ReqVerRep Component

Name	ReqVerRep
Description	Service 1 verification reports
Parameters	Parameter values are as defined by the service 1 procedures called in adaptation points S1-1 to S1-9
Discriminant	No discriminant attribute is defined for service 1 success reports. For failure reports, the failure code acts as discriminant.
Destination	The destination of service 1 reports is set equal to the source of the command being verified
Enable Check	Service 1 reports retrieve their enable status from the OurRegistry as a function of their type, sub-type and discriminant
Ready Check	Service 1 reports are always ready
Repeat Check	Service 1 reports are never repeated
Update Action	No action

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 8.3 lists the data pool data items which are maintained by service 1.

Table 8.3: Observables for Service 1 (Request Verification)

Name	Description
nOfAccFailed	Number of commands which have failed their acceptance check since the application was last reset
failCodeAccFailed	Failure code of last command which failed its Acceptance Check
pcktIdAccFailed	Packet identifier of last command which failed its Acceptance Check
nOfStartFailed	Number of commands which have failed their Start Check since the application was last reset
failCodeStartFailed	Failure code of last command which failed its Start Check
pcktIdStartFailed	Packet identifier of last command which failed its Start Check
nOfPrgrFailed	Number of commands which have failed their Progress Check since the application was last reset
failCodePrgrFailed	Failure code of last command which failed its Progress Check
pcktIdPrgrFailed	Packet identifier of last command which failed its Progress Check
stepPrgrFailed	Step identifier of last command which failed its Progress Check
nOfTermFailed	Number of commands which have failed their Termination Check since the application was last reset
failCodeTermFailed	Failure code of last command which failed its Termination Check

Name	Description
pcktIdTermFailed	Packet identifier of last command which failed its Termination Check
nOfReroutingFailed	Number of commands for which re-routing failed
pcktIdReroutingFailed	Packet identifier of last command for which re-routing failed
invDestRerouting	Destination of last command for which re-routing failed
verFailData	Verification Failure Data (data item of fixed size but variable type with command-specific information about the last verification failure)

8.3 Service 1 Adaptation Points

Table 8.4 lists the CORDET Framework adaptation points which are closed or overridden by the request verification service.

Table 8.4: Adaptation Points for Service 1 (Request Verification)

AP ID	Adaptation Point	Close-Out Value
P-S1-1	Operation to Report Packet Destination Invalid by In-Loader (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 (Overrides ICM-15)	Determine if a progress step has been completed and, if so, run the Command Progress Success Procedure
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 8.5: Requirements for Service 1 (Request Verification)

Req. ID	Requirement Text
P-S1-1/S	<i>The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, a ReqVerRep component to encapsulate a service 1 report</i>
P-S1-2/A	<i>The ReqVerRep component shall close the OutComponent adaptation points as indicated in table 8.2</i>
P-S1-3/A	<i>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 8.4</i>
P-S1-4/S	<i>The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 8.3</i>
P-S1-5/S	<i>The PUS Extension of the CORDET Framework shall support the service 1 failure codes listed in table C.1</i>
P-S1-6/C	<i>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</i>
P-S1-7/C	<i>Applications shall be responsible for configuring the OutRegistry component to selectively disable failure verification reports which they do not need</i>

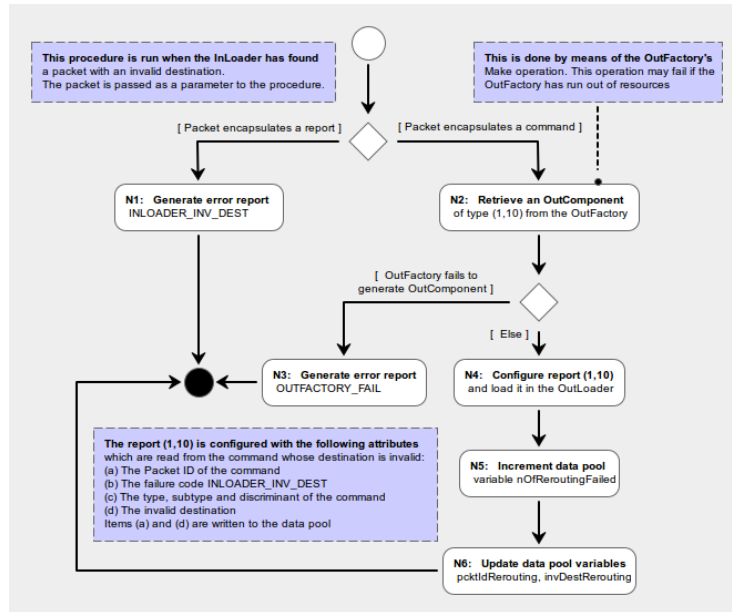


Fig. 8.1: Packet Rerouting Failure Procedure

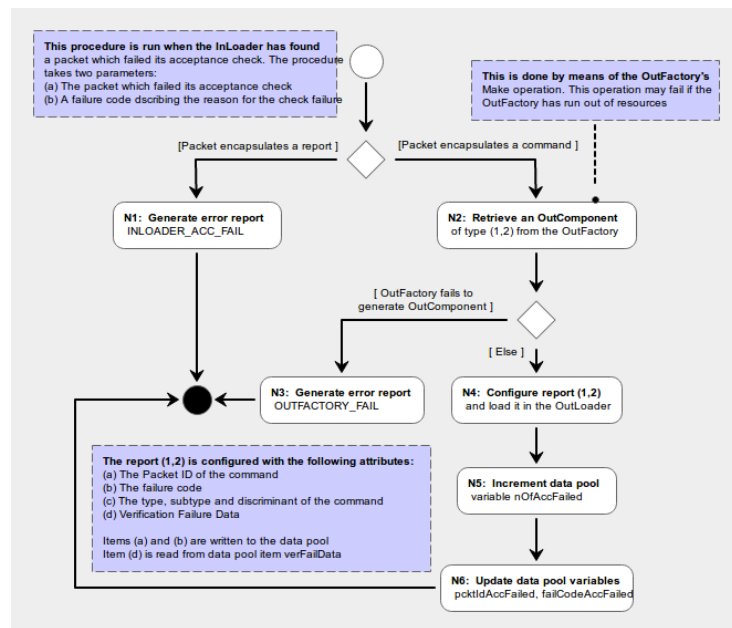


Fig. 8.2: Packet Acceptance Failure Procedure

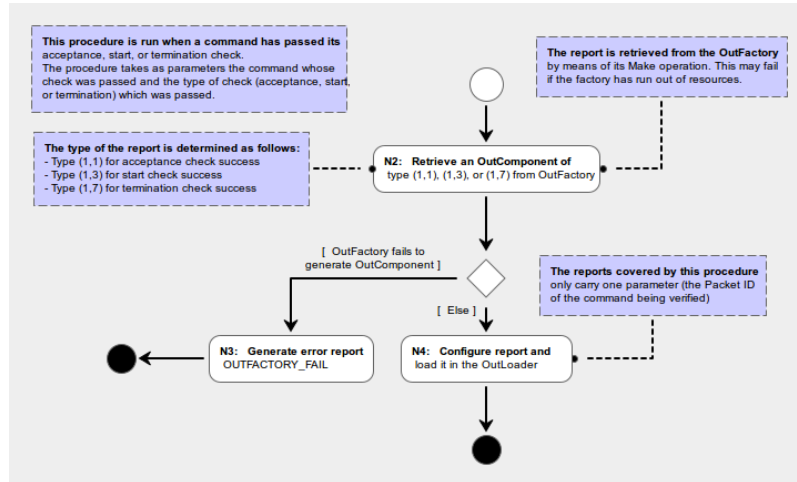


Fig. 8.3: Command Verification Success Procedure

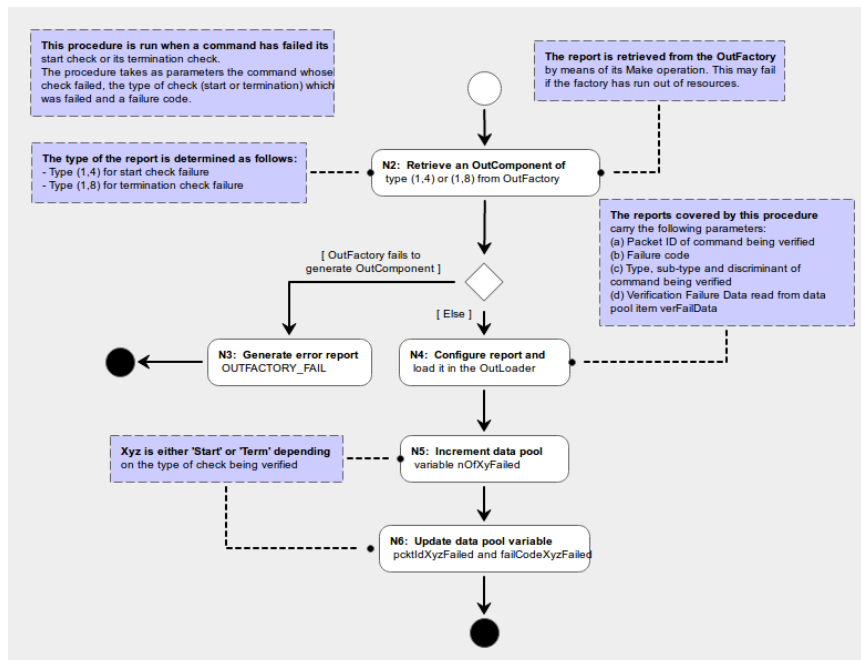


Fig. 8.4: Command Verification Failure Procedure

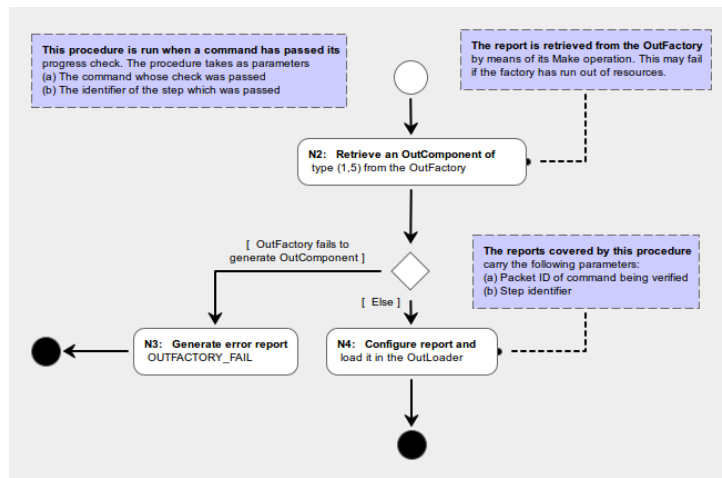


Fig. 8.5: Command Progress Success Procedure

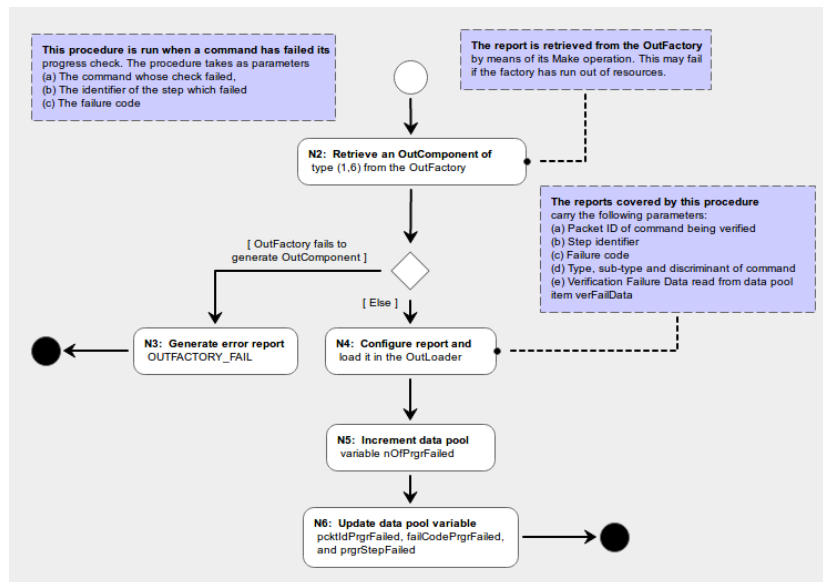


Fig. 8.6: Command Progress Failure Procedure

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 items⁴. 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.

⁴The 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: $(nSimple + nRep[1] + \dots + nRep[nGroup])$, 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: $(nSimple + nRep[1] * rep[1] + \dots + nRep[nGroup] * rep[nGroup])$

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

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 can be set using service TBD. 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 uses service TBD to load 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

Tables 9.2 to 9.9 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 'one-shot' mode in response to a (3,27) or (3,28) command.

Table 9.2: Specification of HkCreateCmd Component

Name	HkCreateCmd
Description	Command (3,1) or (3,2) to Create a Housekeeping or Diagnostic Report Structure
Parameters	SID, collection interval and identifiers of parameters of the housekeeping/diagnostic report to be created with a layout as in clauses 8.3.2.1 and 8.3.2.2 of [PS-SP]
Discriminant	None
Ready Check	Return "command is 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	Set action outcome to 'success'
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

Table 9.3: Specification of HkDeleteCmd Component

Name	HkDeleteCmd
Description	Command (3,3) or (3,4) to Delete a Housekeeping or Diagnostic Report Structure
Parameters	List of SIDs whose definition is to be deleted with layout as in clauses 8.3.2.3 and 8.3.2.4 of [PS-SP]
Discriminant	None
Ready Check	Return "command is ready"
Start Action	Run the procedure Start Action of HkDelete Command of figure 9.2
Progress Action	Delete the entries in the RDL which have been marked in the Start Action corresponding to the SIDs which have been identified as valid by the Start Action and then set the action outcome to 'completed'
Termination Action	Set action outcome to 'success'
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

Table 9.4: Specification of HkEnableCmd Component

Name	HkEnableCmd
Description	Command (3,5) or (3,7) to Enable Periodic Generation of a Housekeeping or Diagnostic Report Structure
Parameters	List of SIDs to be enabled with a layout as in clauses 8.3.2.5 and 8.3.2.7 of [PS-SP]
Discriminant	None
Ready Check	Return “command is 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	Set action outcome to ‘success’
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

Table 9.5: Specification of HkDisableCmd Component

Name	HkDisableCmd
Description	Command (3,6) or (3,8) to Disable Periodic Generation of a Housekeeping or Diagnostic Report Structure
Parameters	List of SIDs to be disabled with a layout as in clauses 8.3.2.6 and 8.3.2.8 of [PS-SP]
Discriminant	None
Ready Check	Return “command is 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	Set action outcome to ‘success’
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

Table 9.6: Specification of HkRepStructCmd Component

Name	HkRepStructCmd
Description	Command (3,9) or (3,11) to Report Structure of a Housekeeping or Diagnostic Report
Parameters	List of SIDs whose structure is to be reported with a layout as in clauses 8.3.2.9 and 8.3.2.11 of [PS-SP]
Discriminant	None
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 Report Housekeeping 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	Do nothing
Operation to Report Progress Successful	Do nothing (no progress success reports are generated by this command)

Table 9.7: Specification of HkRepStructRep Component

Name	HkRepStructRep
Description	Housekeeping or Diagnostic Structure Report (3,10) or (3,12)
Parameters	The definition of a SID in the RDL with the layout defined in clauses 8.3.2.10 and 8.3.2.12 of [PS-SP]
Discriminant	The Structure Identifier (SID) of the report
Destination	The destination is set equal to the source of the (3,9) or (3,11) command which triggered the report
Enable Check	The enable status is read from the <code>isEnabled</code> field of the Report Definition corresponding to the report’s SID
Ready Check	Report is always ready
Repeat Check	Report is never repeated
Update Action	Load the SID definition from the RDL

Table 9.8: Specification of HkRep Component

Name	HkRep
Description	Periodic Housekeeping or Diagnostic Report (3,25) or (3,26)
Parameters	The values of the data items associated to the report's SID in the RDL with the layout defined in clauses 8.3.2.25 and 8.3.2.26 of [PS-SP]
Discriminant	The Structure Identifier (SID) of the report
Destination	The destination is read from the dest field of the Report Definition corresponding to the report's SID
Enable Check	The report is enabled if its SID is defined in the RDL
Ready Check	Run the procedure Ready Check of HkRep Report of figure 9.5
Repeat Check	Report is always 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 9.9: Specification of HkOneShotCmd Component

Name	HkOneShotCmd
Description	Command (3,27) or (3,28) to Generate One-Shot Housekeeping Report,
Parameters	List of SIDs for which the one-shot report is to be generated with a layout as in clauses 8.3.2.27 and 8.3.2.28 of [PS-SP]
Discriminant	None
Ready Check	Return "command is 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, set the cycle counter equal to the period. Set the action outcome to 'completed'
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
Operation to Report Progress Successful	Do nothing (no progress success reports are generated by this command)

9.5 Service 3 Constants

The service 3 constants are listed in table 9.10.

Table 9.10: Constants for Service 3 (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/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

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. They are listed in table 9.11.

Table 9.11: Observables for Service 3 (Housekeeping Service)

Name	Description
cycleCnt	Array of HK_N_REP_DEF elements. The i-th element is the cycle counter for the i-th Report Definitions in the RDL
sampleBufId	Array of HK_N_REP_DEF elements. The i-th element is the identifier of the Sampling Buffer for the i-th Report Definition in the RDL
debugVar	Array of HK_N_DEBUG_VAR elements. The i-th element is the value of the i-th debug variable

9.7 Service 3 Parameters

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. They are listed in table 9.12.

Table 9.12: Parameters for Service 3 (Housekeeping Service)

Name	Description
sid	Array of HK_N_REP_DEF elements. The i-th element is the SID of the i-th Report Definition in the RDL
period	Array of HK_N_REP_DEF elements. The i-th element is the period of the i-th Report Definition in the RDL
isEnabled	Array of HK_N_REP_DEF elements. The i-th element is the enable status of the i-th Report Definition in the RDL
dest	Array of HK_N_REP_DEF elements. The i-th element is the destination of the i-th Report Definition in the RDL
debugVarAddr	Array of HK_N_DEBUG_VAR elements. The i-th element is the address of the i-th debug variable

9.8 Service 3 Requirements

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

Table 9.13: Requirements for Service 3 (Housekeeping Service)

Req. ID	Requirement Text
P-S3-1/S	<i>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</i>
P-S3-2/S	<i>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</i>
P-S3-3/S	<i>The PUS Extension of the CORDET Framework shall provide a Sampling Buffer Configuration Function to configure a sampling buffer for a given report</i>
P-S3-4/S	<i>The PUS Extension of the CORDET Framework shall provide a Sampling Buffer Setter Function to load a data item value in a sampling buffer</i>
P-S3-5/S	<i>The PUS Extension of the CORDET Framework shall provide a Sampling Buffer Getter Function to retrieve a data item value from a sampling buffer</i>
P-S3-6/C	<i>Application shall be responsible for loading a sampling buffer with the values of super-commutated data items</i>
P-S3-7/S	<i>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</i>
P-S3-8/S	<i>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</i>

Req. ID	Requirement Text
P-S3-9/S	<i>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</i>
P-S3-10/S	<i>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</i>
P-S3-11/S	<i>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</i>
P-S3-12/S	<i>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</i>
P-S3-13/S	<i>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</i>
P-S3-14/S	<i>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</i>
P-S3-15/S	<i>The hkCreateCmd component shall close the InCommand adaptation points as indicated in table 9.2</i>
P-S3-16/S	<i>The hkDeleteCmd component shall close the InCommand adaptation points as indicated in table 9.3</i>
P-S3-17/S	<i>The hkEnableCmd component shall close the InCommand adaptation points as indicated in table 9.4</i>
P-S3-18/S	<i>The hkDisableCmd component shall close the InCommand adaptation points as indicated in table 9.5</i>
P-S3-19/S	<i>The hkRepStructCmd component shall close the InCommand adaptation points as indicated in table 9.6</i>
P-S3-20/S	<i>The hkRepStructRep component shall close the OutComponent adaptation points as indicated in table 9.7</i>
P-S3-21/S	<i>The hkRep component shall close the OutComponent adaptation points as indicated in table 9.8</i>
P-S3-22/S	<i>The hkRepOneShot component shall close the OutComponent adaptation points as indicated in table ??</i>
P-S3-23/S	<i>The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 9.11</i>
P-S3-24/S	<i>The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the parameters listed in table 9.12</i>

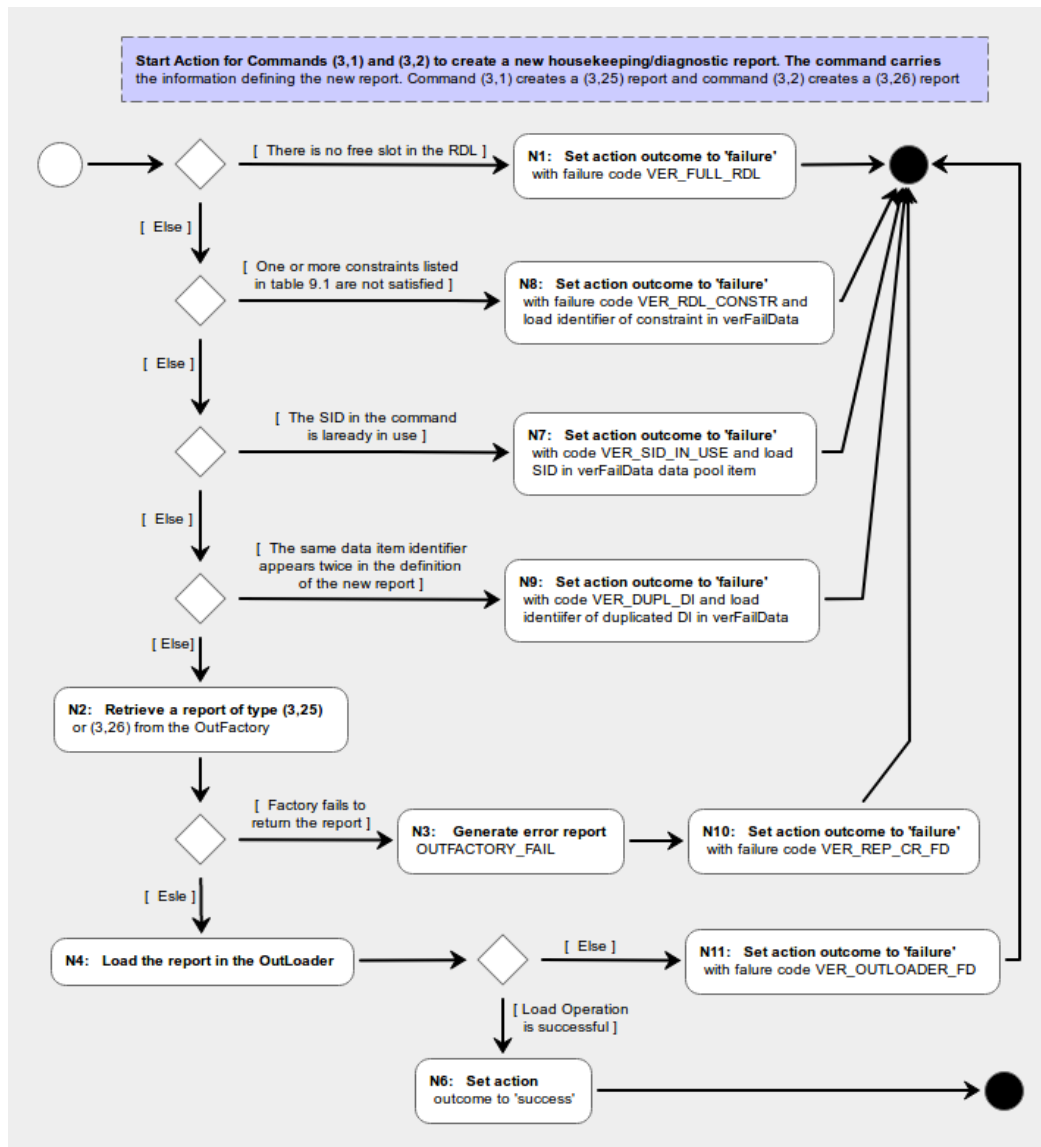


Fig. 9.1: Start Action of Command HkCreateCmd

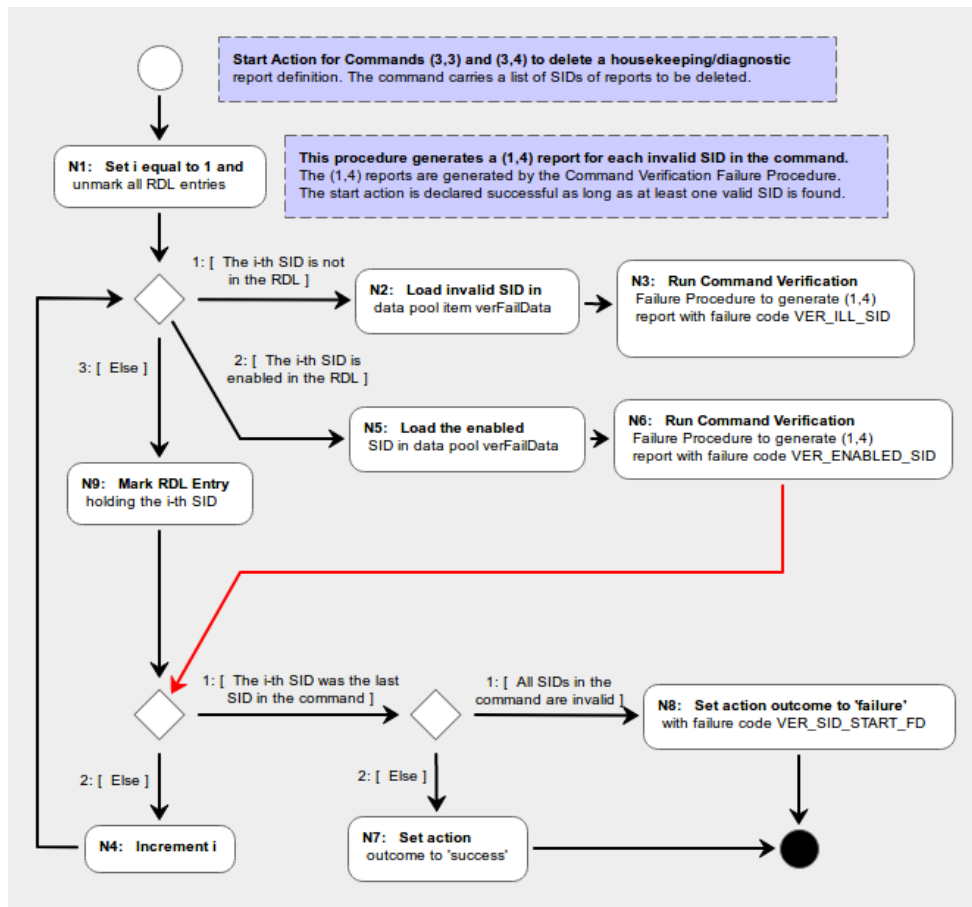


Fig. 9.2: Start Action of Command HkDeleteCmd

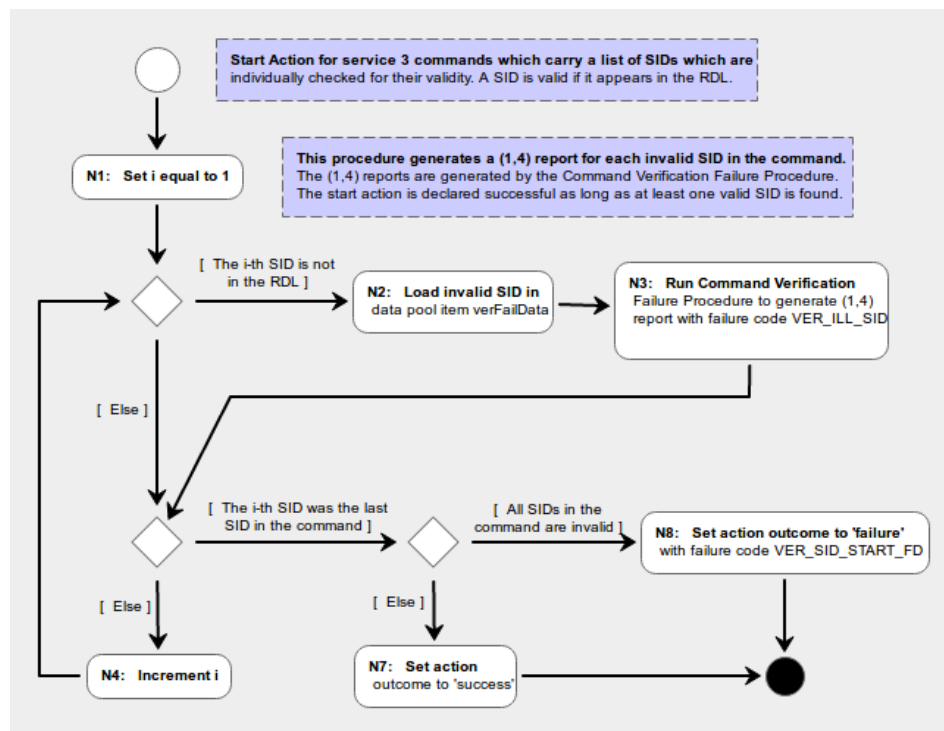


Fig. 9.3: Start Action of Multi-SID Commands

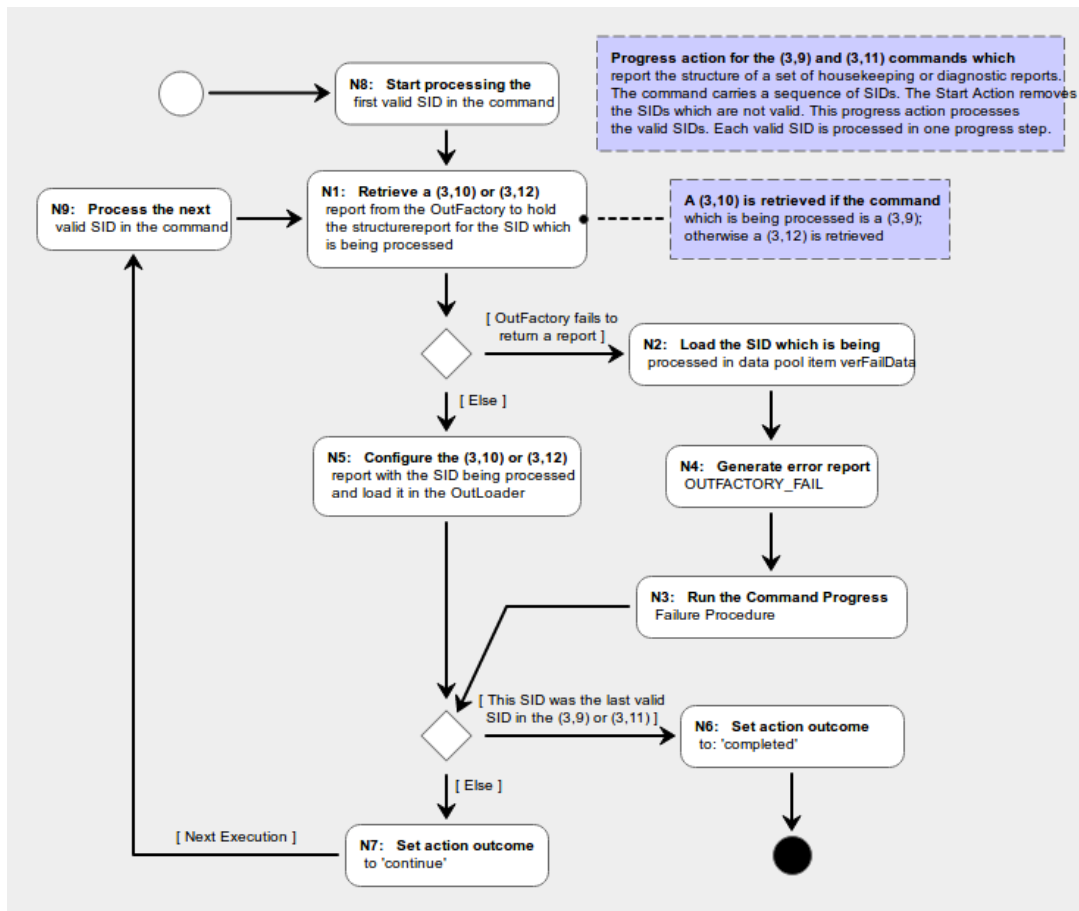


Fig. 9.4: Progress Action of Command HkRepStructCmd

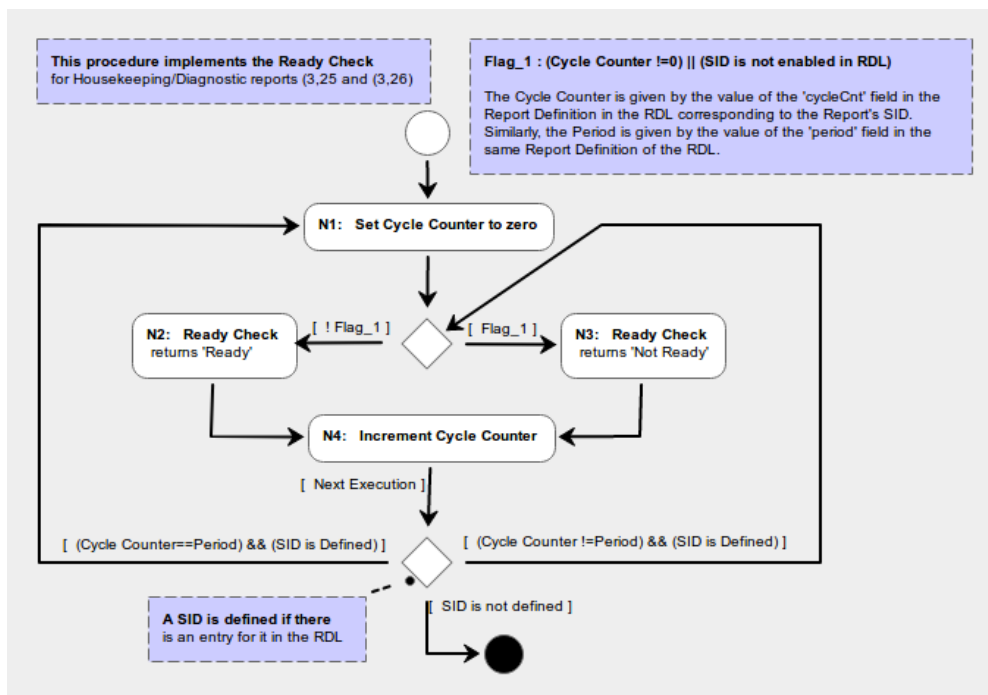


Fig. 9.5: Ready Check of Report HkRep

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 consequently defines one single component `EvtHk` which may encapsulate an event report of any 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. No event identifiers are pre-defined at framework level.

Applications use event reports as follows:

- When an application encounters a situation which should trigger the generation of an event report, it retrieves an `EvtHk` component from the `OutFactory`. This component will encapsulate the event report. The event level and the event identifier are specified when the `EvtHk` component is retrieved from the factory because the 'make' function of the `OutFactory` takes as an argument the type, sub-type and the discriminant of the event report.
- The application configures the component with the destination and with any data which
- The application loads the `EvtHk` 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.

10.1 Pre-Defined Event Reports

The event reports listed in section A are pre-defined in the sense that they are generated in response to situations defined at framework level. For each of these events, the framework provides a Generate Pre-Defined Event Function which takes as parameters the type, sub-type and discriminant value of the event report and all its parameters and then performs the following actions:

- Retrieve an instance of an OutComponent for the event report from the OutFactory
- If the retrieval is successful, it configures the event report with the event parameter data and loads it in the OutLoader
- If the retrieval is not successful, it generates an error report OUTFACTORY_FAIL

10.2 Service 5 Report and Command Definition

Tables 10.1 to 10.2 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 same component EvtRep implements all four event reports (5,1) to (5,4). This is legitimate because all event reports - irrespective of their severity level - have the same behaviour.
- 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 EvtRep Component

Name	EvtRep
Description	Event Report of Type (5,1) to (5,4)
Parameters	Parameter values are as application-specific
Discriminant	The Event Identifier (EID)
Destination	The destination is application-specific
Enable Check	Update service 5 observable nOfDetectedEvt_x ('x' is the event severity level) and then retrieve the enable status from the Our-Registry as a function of the report type, sub-type and discriminant
Ready Check	Report is always ready
Repeat Check	Report is never repeated
Update Action	Update service 5 observables: nOfGenEvtRep_x, lastEvtEid_i, lastEvtTime_x ('x' is the event severity level). Note that the parameter values are set by the application which creates the event report at the time it creates the event report.

Table 10.2: Specification of EvtEnableCmd Component

Name	EvtEnableCmd
Description	Command (5,5) to Enable Generation of a List of Event Identifiers
Parameters	List of EIDs to be enabled with a layout as in clause 8.5.2.5 of [PS-SP]
Discriminant	None
Ready Check	Return “command is ready”
Start Action	Run the procedure Start Action of Multi-EID Command of figure 10.1
Progress Action	For each valid EID found by the Start Action of the command: set the corresponding element of the array isEidEnabled to true and then decrement nDisabledEid_x (‘x’ is the severity level of the EID). Set the action outcome to ‘completed’.
Termination Action	Set action outcome to ‘success’
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

Table 10.3: Specification of EvtDisableCmd Component

Name	EvtDisableCmd
Description	Command (5,5) to Disable Generation of a List of Event Identifiers
Parameters	List of EIDs to be disabled with a layout as in clause 8.5.2.6 of [PS-SP]
Discriminant	None
Ready Check	Return “command is ready”
Start Action	Run the procedure Start Action of Multi-EID Command of figure 10.1
Progress Action	For each valid EID found by the Start Action of the command: set the corresponding element of the array isEidEnabled to false and then increment nDisabledEid_x (‘x’ is the severity level of the EID). Set the action outcome to ‘completed’.
Termination Action	Set action outcome to ‘success’
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

Table 10.4: Specification of EvtRepDisabledCmd Component

Name	EvtRepDisabledCmd
Description	Command (5,7) to Generate Report (5,8) with a List of Disabled Event Identifiers
Parameters	None
Discriminant	None
Ready Check	Return “command is ready”
Start Action	Retrieve (5,8) report from OutFactory and set action outcome to “success” if retrieval succeeds. If the retrieval fails, generate error report OUTFACTORY_FAILED and set outcome of Start Action to ‘failed’
Progress Action	Configure the (5,8) report with a destination equal to the source of the (5,7) command, load it in the OutLoader, and set the action outcome to ‘completed’
Termination Action	Set action outcome to ‘success’
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

Table 10.5: Specification of EvtRepDisableRep Component

Name	EvtRepDisabledRep
Description	List of Disabled Event Identifiers Report (5,8)
Parameters	The list of disabled event identifiers with the layout defined in clause 8.5.2.8 of [PS-SP]
Discriminant	None
Destination	The destination is set equal to the source of the (3,7) command which triggered the report
Enable Check	Report is always enabled
Ready Check	Report is always ready
Repeat Check	Report is never repeated
Update Action	Load the list of disabled event identifiers from arrays isEidEnabled1 to isEidEnabled4

10.3 Service 5 Constants

The service 5 constants are listed in table 10.6. Note that the event identifiers must be defined in ascending order. This constraint is introduced to allow implementation to optimize the search for event identifiers.

Table 10.6: Constants for Service 5 (Event Reporting Service)

Name	Description
EVT_N_EID	Number of event identifiers supported by the application
EVT_EID	Array of EVT_N_EID elements holding the event identifiers in ascending order
EVT_EID_LEVEL	Array of EVT_N_EID elements holding the severity level of the event identifiers

10.4 Service 5 Observables

The service 5 observables consist of counters and flags which are updated by the service commands. They are listed in table 10.7.

Table 10.7: Observables for Service 5 (Event Reporting Service)

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

10.5 Service 5 Requirements

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

Table 10.8: Requirements for Service 5 (Event Reporting Service)

Req. ID	Requirement Text
P-S5-25/S	<i>The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, an EvtRep component to encapsulate an event report of type (5,1) to (5,4)</i>
P-S5-26/S	<i>The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, an EvtEnableCmd component to encapsulate a (5,5) command</i>
P-S5-27/S	<i>The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, an EvtDisableCmd component to encapsulate a (5,6) command</i>
P-S5-28/S	<i>The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, an EvtRepDisabledCmd component to encapsulate a (5,7) command</i>
P-S5-29/S	<i>The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, an EvtRepDisabledRep component to encapsulate a (5,8) report</i>
P-S5-30/A	<i>The EvtRep component shall close the OutComponent adaptation points as indicated in table 10.1</i>
P-S5-31/A	<i>The EvtEnableCmd component shall close the InCommand adaptation points as indicated in table 10.2</i>
P-S5-32/A	<i>The EvtDisableCmd component shall close the InCommand adaptation points as indicated in table 10.3</i>
P-S5-33/A	<i>The EvtRepDisabledCmd component shall close the InCommand adaptation points as indicated in table 10.4</i>
P-S5-34/A	<i>The EvtRepDisabledRep component shall close the OutComponent adaptation points as indicated in table 10.5</i>
P-S5-35/S	<i>The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 12.11</i>
P-S5-36/C	<i>Applications shall be responsible for configuring the EvtRep component with the event parameters at the point where the EvtRep component is created</i>
P-S5-37/C	<i>Applications shall be responsible for configuring the OutRegistry component to selectively disable event reports whose default enable status is: 'disabled'</i>
P-S5-38/S	<i>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</i>
P-S5-39/S	<i>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</i>
P-S5-40/S	<i>If the OutComponent retrieval from the OutFactory fails, the Generate Pre-Defined Event Function shall generate an error report of type OUTFACTORY_FAIL</i>

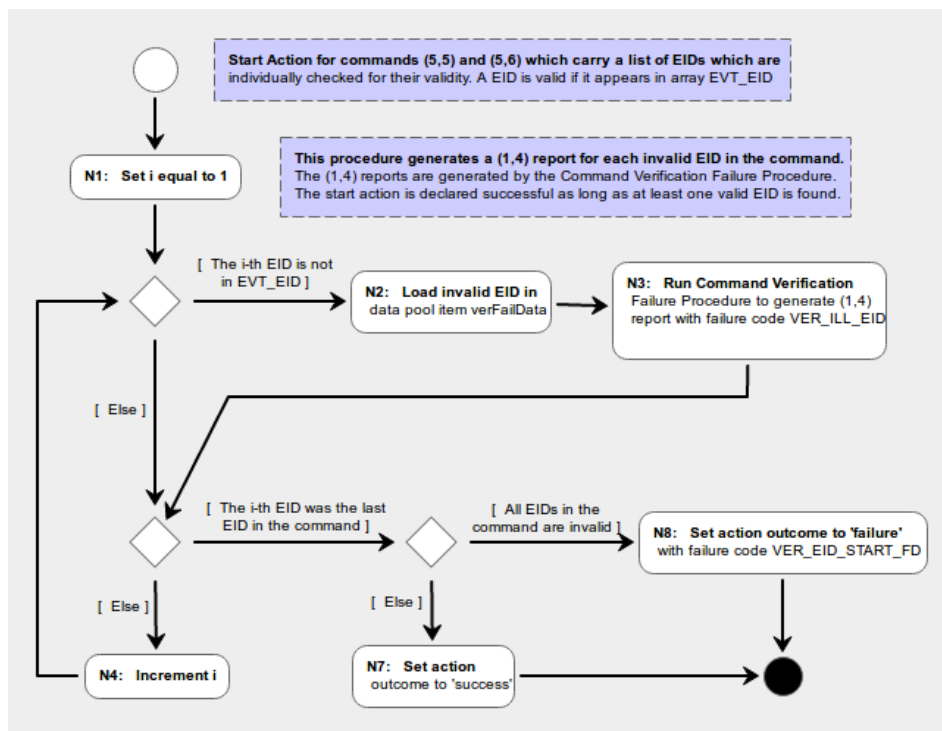


Fig. 10.1: Start Action of Multi-EID Commands

11 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.

11.1 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 pool⁵. 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 unconditionally 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 11.1. 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 11.1. The first item is the identifier of the data pool item whose value the parameter monitor checks.

⁵It 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 11.2. When the checking status is updated, the following actions are executed:

- An entry is made in the Check Transition List (see section 11.1.2)
- The procedure to process the Check Transition List is run (see section 11.1.2)
- If an event is associated to the parameter monitor (i.e. if field **evtId** in table 11.1 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 11.1), 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 $[0..(\text{per}-1)]$ 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 individual 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 11.1: 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 11.1.1
monPrType	Identifier of the Monitor Procedure type which checks the parameter value	See section 11.1.1
monPrRetVal	Most recent return value of the Monitor Procedure	See table 11.2
monPrPrevRetVal	Previous return value of the Monitor Procedure (or INVALID after the monitoring procedure or the monitoring function has been enabled)	See table 11.2
checkStatus	Checking status of monitored parameter	See table 11.2
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

11.1.1 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 11.2.
- Expected Value Monitor Procedure: verifies whether the monitored parameter has pre-defined value. See figure 11.3.
- 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 11.4.

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 11.2 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 dynam-

ically using command (12,7), their monitor procedures are re-started by first stopping them and then starting them.

Table 11.2: 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 successive values) is above its upper limit
MON_DEL_BELOW	Parameter delta-value (difference between successive values) is below its lower limit

11.1.2 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 11.3.

Entries are added to the CTL by the Parameter Monitoring Procedure (see figure 11.1) 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 11.5. 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 11.6).

Table 11.3: Attributes of a Check Transition

Name	Description	Constraint
<code>dataItemId</code>	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 11.1.1
expValChkMask	In the case of an Expected Value Monitor violation, the expected value check mask	See section 11.1.1
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 11.2
prevCheckStatus	Checking status in the cycle before the violation was detected	A CUC time value

11.2 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 11.6.

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 11.5. 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 11.4: Attributes of a Functional Monitor

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

Table 11.5: Checking Statuses of a Functional Monitor

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

11.3 Service 12 Report and Command Definition

Tables 11.6 to TBD.

Table 11.6: Specification of MonEnbMonDefCmd Component

Name	MonEnbMonDefCmd
Description	Command (12,1) to enable one or more monitoring definitions
Parameters	The identifiers of the monitoring definitions to be enabled
Discriminant	None
Ready Check	Default implementation The command is always ready
Start Action	Run the procedure Start Action of Multi-Parameter Monitor Commands of figure 11.7
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
Termination Action	Default implementation Do nothing
Abort Action	Default implementation Do nothing

Table 11.7: Specification of MonDisParMonDefCmd Component

Name	MonDisParMonDefCmd
Description	Command (12,2) to disable one or more monitoring definitions
Parameters	The identifiers of the monitoring definitions to be disabled
Discriminant	None
Ready Check	Default implementation The command is always ready
Start Action	Run the procedure Start Action of Multi-Parameter Monitor Commands of figure 11.7
Progress Action	For every valid Parameter Monitor Identifier in the command: stop its Monitor Procedure and set its checking status (attribute checkStatus) to UNCHECKED
Termination Action	Default implementation Do nothing
Abort Action	Default implementation Do nothing

Table 11.8: Specification of MonChgTransDelCmd Component

Name	MonChgTransDelCmd
Description	Command (12,3) 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 The 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

Name	MonChgTransDelCmd
Termination Action	Default implementation Do nothing
Abort Action	Default implementation Do nothing

Table 11.9: Specification of MonDelAllParMonCmd Component

Name	MonDelAllParMonCmd
Description	TC(12,4) to delete all parameter monitoring definitions
Parameters	None
Discriminant	None
Ready Check	Default implementation The 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)
Termination Action	Default implementation Do nothing
Abort Action	Default implementation Do nothing

Table 11.10: Specification of MonAddParMonCmd Component

Name	MonAddParMonCmd
Description	Command (12,5) 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 The command is always ready
Start Action	Run the procedure Start Action of (12,5) Command of figure 11.8
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
Termination Action	Default implementation Do nothing
Abort Action	Default implementation Do nothing

Table 11.11: Specification of MonDelParMonCmd Component

Name	MonDelParMonCmd
Description	TC(12,6) to delete one or more parameter monitoring definitions
Parameters	The identifiers of the parameter monitors to be deleted
Discriminant	None
Ready Check	Default implementation The command is always ready
Start Action	Run the procedure Start Action of (12,6) Command of figure ??
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)

Name	MonDelParMonCmd
Termination Action	Default implementation Do nothing
Abort Action	Default implementation Do nothing

Table 11.12: Specification of MonModParMonCmd Component

Name	MonModParMonCmd
Description	Command (12,7) 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 The command is always ready
Start Action	Run the procedure Start Action for (12,7) Command of figure 11.10
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 Do nothing
Abort Action	Default implementation Do nothing

Table 11.13: Specification of MonRepParMonCmd Component

Name	MonRepParMonCmd
Description	Command (12,7) 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 The command is always ready
Start Action	Run the Start Action of (12,8) Command Procedure of figure 11.11
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 Do nothing

Table 11.14: Specification of MonRepParMonRep Component

Name	MonRepParMonRep
Description	TM(12,9) to report some or all parameter monitoring definitions in the Parameter Monitoring Definition List (PMDL)

Name	MonRepParMonRep
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
Enable Check	Default implementation The report is always enabled
Ready Check	Default implementation The report is always ready
Repeat Check	Default implementation The report is not repeated
Update Action	Default implementation Do nothing

Table 11.15: Specification of MonCheckTransRep Component

Name	MonCheckTransRep
Description	Report the content of the Check Transition List (CTL).
Parameters	The entries in the Check Transition List.
Discriminant	None
Enable Check	Default implementation The report is always enabled
Ready Check	Default implementation The report is always ready
Repeat Check	Default implementation The report is not repeated
Update Action	Default implementation Do nothing

Table 11.16: Specification of MonRepParMonStatCmd Component

Name	MonRepParMonStatCmd
Description	Command (12,13) to report the status of all parameter monitors
Parameters	None
Discriminant	None
Ready Check	Default implementation The 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 Do nothing

Table 11.17: Specification of MonRepParMonStatRep Component

Name	MonRepParMonStatRep
Description	TM(12,14) to report the status of all parameter monitors currently defined in the Parameter Monitoring Definition List (PMDL)
Parameters	The checking status of all parameter monitors currently defined in the PDML
Discriminant	None
Enable Check	Default implementation Report is enabled
Ready Check	Default implementation Report is ready
Repeat Check	Default implementation Do not repeat
Update Action	Default implementation Do nothing

Table 11.18: Specification of MonEnbMonFuncCmd Component

Name	MonEnbMonFuncCmd
Description	TC(12,15) 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 and set service 12 parameter servUser equal to the source of this command.
Termination Action	Default implementation Set action outcome to 'success'
Abort Action	Default implementation Do nothing

Table 11.19: Specification of MonDisMonFuncCmd Component

Name	MonDisMonFuncCmd
Description	Command (12,16) 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
Termination Action	Default implementation Set action outcome to 'success'
Abort Action	Default implementation Do nothing

Table 11.20: Specification of MonAddFuncMonDefCmd Component

Name	MonAddFuncMonDefCmd
Description	TC(12,23) to add one or more functional monitoring definitions to the FMDL
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
Start Action	Run the Start Action of (12,23) Command Procedure of figure 11.12.
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'. Set the action outcome to 'success'.

Name	MonAddFuncMonDefCmd
Termination Action	Default implementation
Abort Action	Default implementation

11.4 Service 12 Constants

The service 12 constants are listed in table 11.20.

Table 11.21: Constants for Service 12 (On-Board 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
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_N_FPMON	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

11.5 Service 12 Observables

The service 12 observables are listed in table 11.21. 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 done 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.

Table 11.22: Observables for Service 12 (On-Board Monitoring)

Name	Description
CtlTimeFirstEntry	Time when first entry has been added to the CTL
MaxRepDelay	Maximum reporting delay
NmbAvailFuncMon	Number of available functional monitors in the FMDL
NmbAvailParMon	Number of available parameter monitors in the PMDL
NmbEnbFuncMon	Number of enabled functional monitors in the FMDL

Name	Description
NmbEnbParMon	Number of enabled parameter monitors in the PMDL
FuncMonEnable	Functional monitoring enable status
ParMonEnbStatus	Enable state of parameter monitoring function

11.6 Service 12 Parameters

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 11.23: Parameters for Service 12 (On-Board Monitoring Service)

Name	Description
servUser	Identifier of the application acting as user of the on-board monitoring service
ctlRepDelay	Maximum reporting delay for the Check Transition List expressed in multiples of the monitoring period MON_PER

11.7 Service 12 Adaptation Points

The table in this section lists the adaptation points for the On-Board Monitoring Service.

Table 11.24: 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 List (New AP)	By default, all parameter monitors are empty

11.8 Service 12 Requirements

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

Table 11.25: Requirements for Service 12 (On-Board Monitoring Service)

Req. ID	Requirement Text
P-S12-1/S	<i>The PUS Extension of the CORDET Framework shall provide a Monitoring Function Procedure implementing the behaviour shown in figure 11.1</i>
P-S12-1.1/S	<i>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 11.1</i>
P-S12-1.2/S	<i>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 11.3</i>
P-S12-3/S	<i>The PUS Extension of the CORDET Framework shall provide the Limit Check Monitoring Procedure implementing the behaviour shown in figure 11.2</i>
P-S12-4/S	<i>The PUS Extension of the CORDET Framework shall provide the Expected Value Monitoring Procedure implementing the behaviour shown in figure 11.3</i>
P-S12-5/S	<i>The PUS Extension of the CORDET Framework shall provide the Delta Value Monitoring Procedure implementing the behaviour shown in figure 11.4</i>

Req. ID	Requirement Text
P-S12-6/C	<i>Applications shall be responsible for initializing the data pool items <code>nmbAvailPMon</code> and <code>nmbEnbPMon</code> giving the number of available and enabled parameter monitors in the parameter monitoring list</i>
P-S12-7/C	<i>As part of their initialization, applications shall be responsible for configuring and starting the monitor procedures associated to the pre-defined parameter monitors</i>
P-S12-8/C	<i>During their initialization, applications shall start the Monitoring Function Procedure and the Check Transition List Procedure</i>
P-S12-8/C	<i>During normal operation, applications shall cyclically execute the Monitoring Function Procedure with a period of <code>MON_PER</code></i>

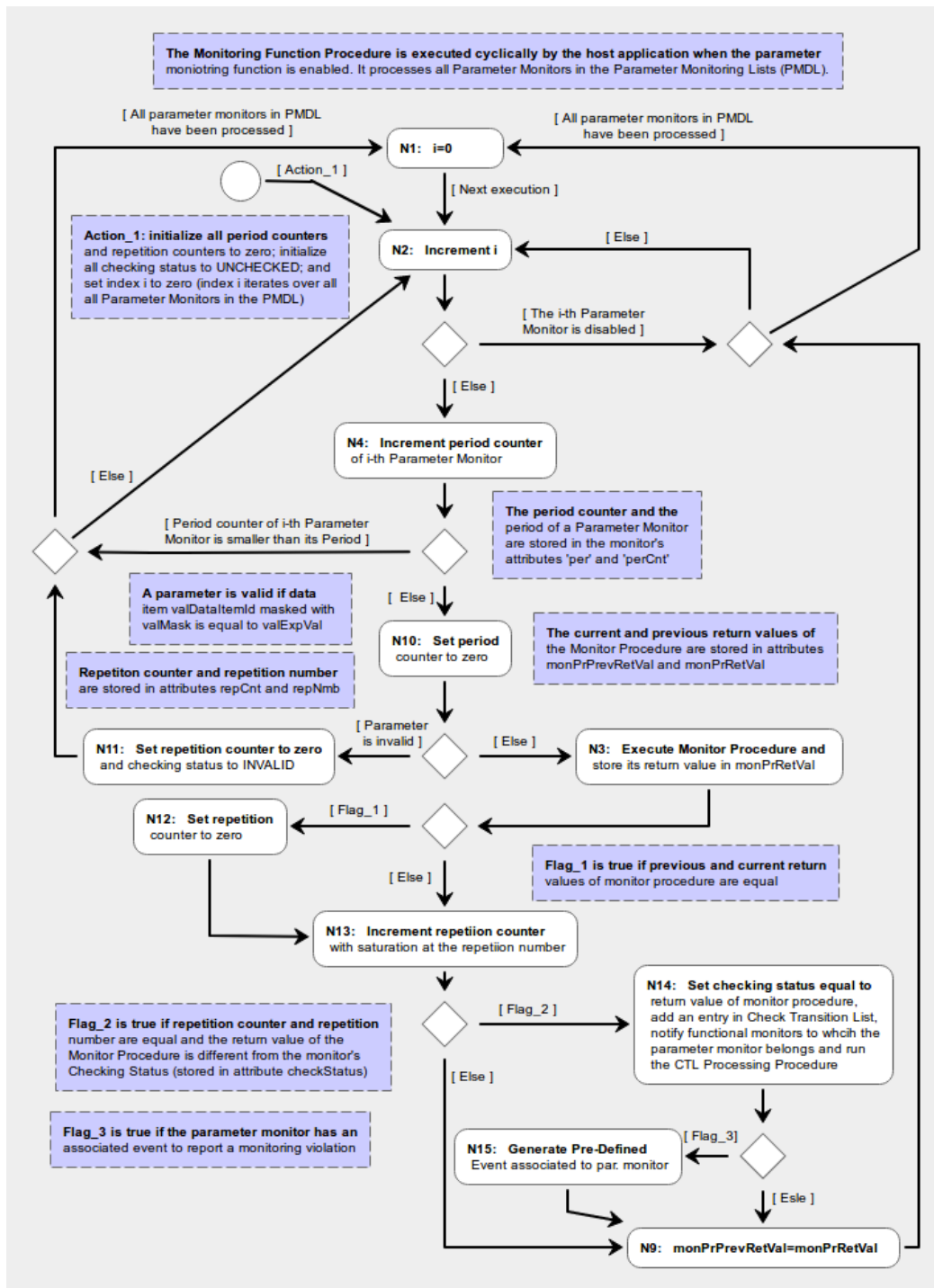


Fig. 11.1: Parameter Monitoring Procedure

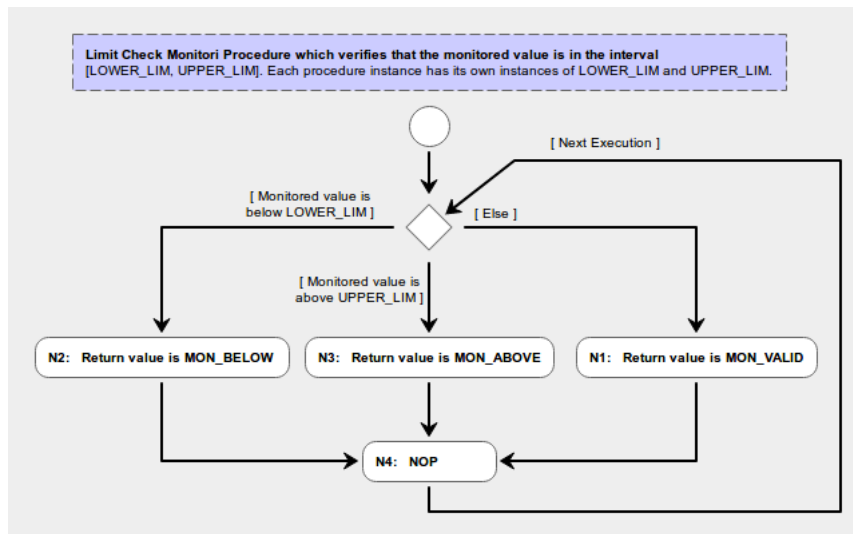


Fig. 11.2: Limit Check Monitor Procedure

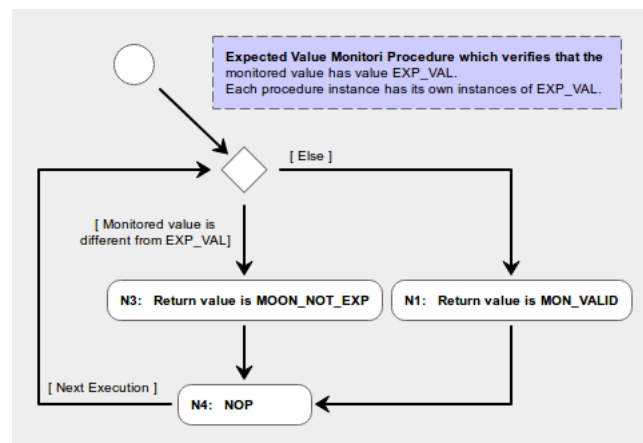


Fig. 11.3: Expected Value Monitor Procedure

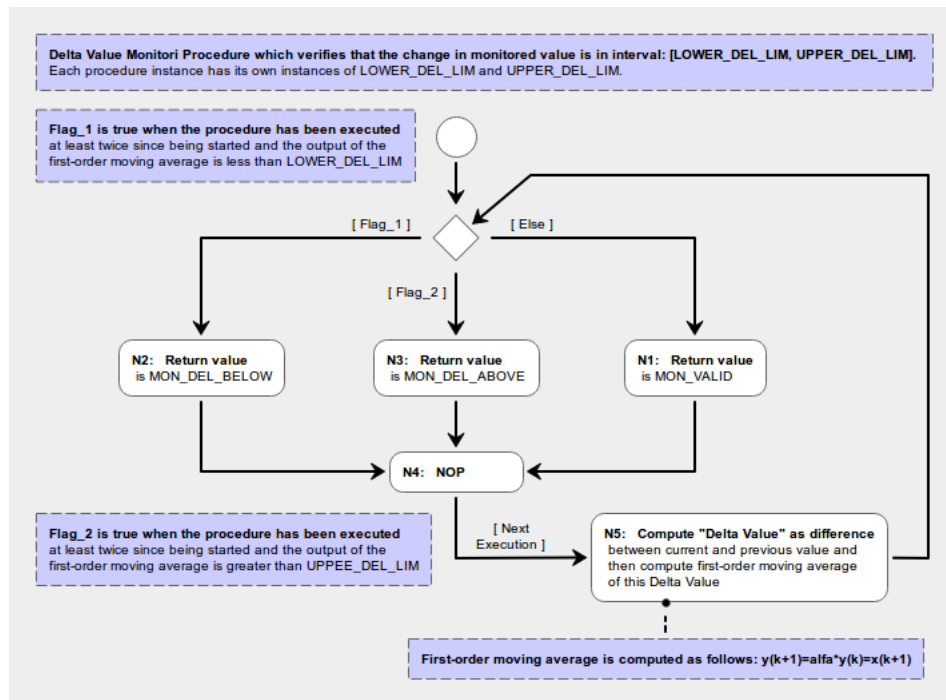


Fig. 11.4: Delta Value Monitor Procedure

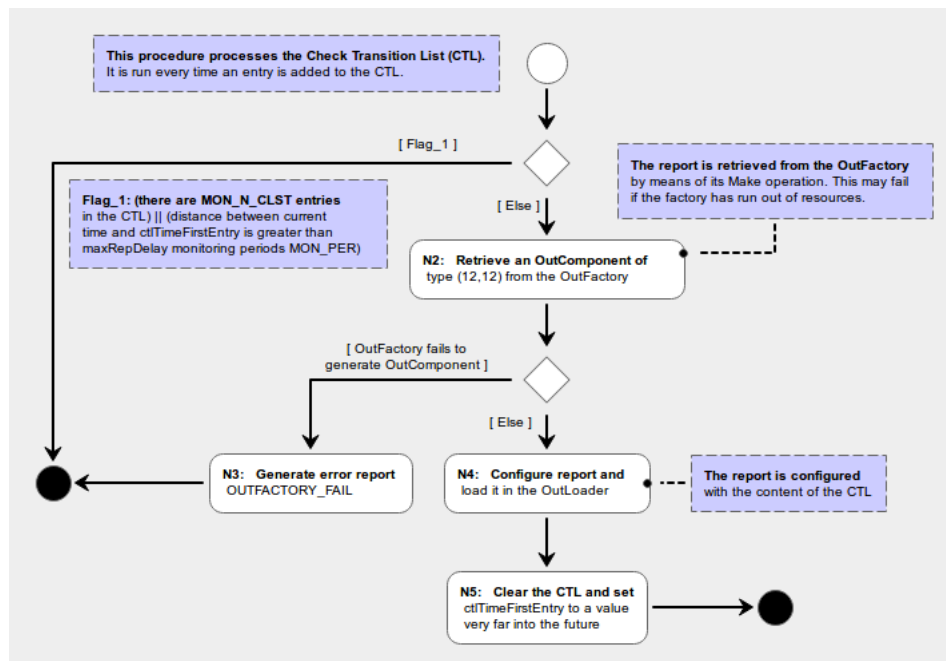


Fig. 11.5: CTL Processing Procedure

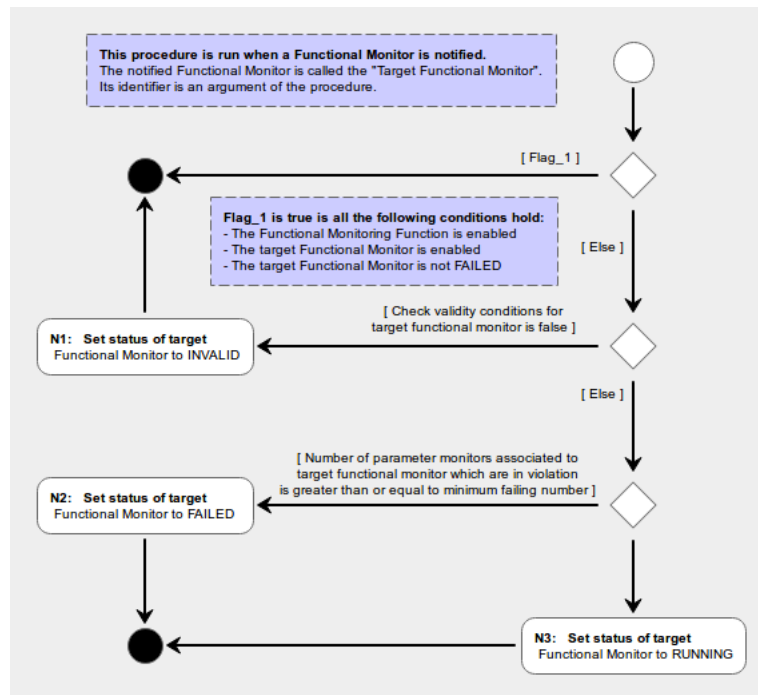


Fig. 11.6: Functional Monitor Notification Procedure

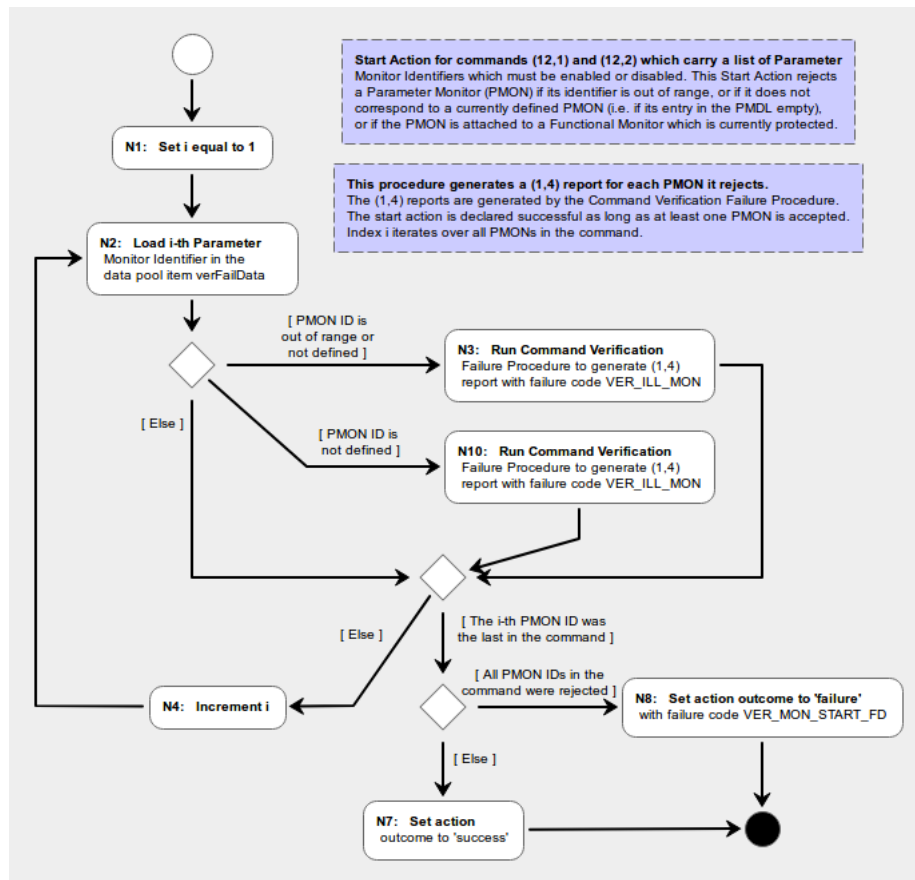


Fig. 11.7: Start Action of (12,1) and (12,2) Command Procedure

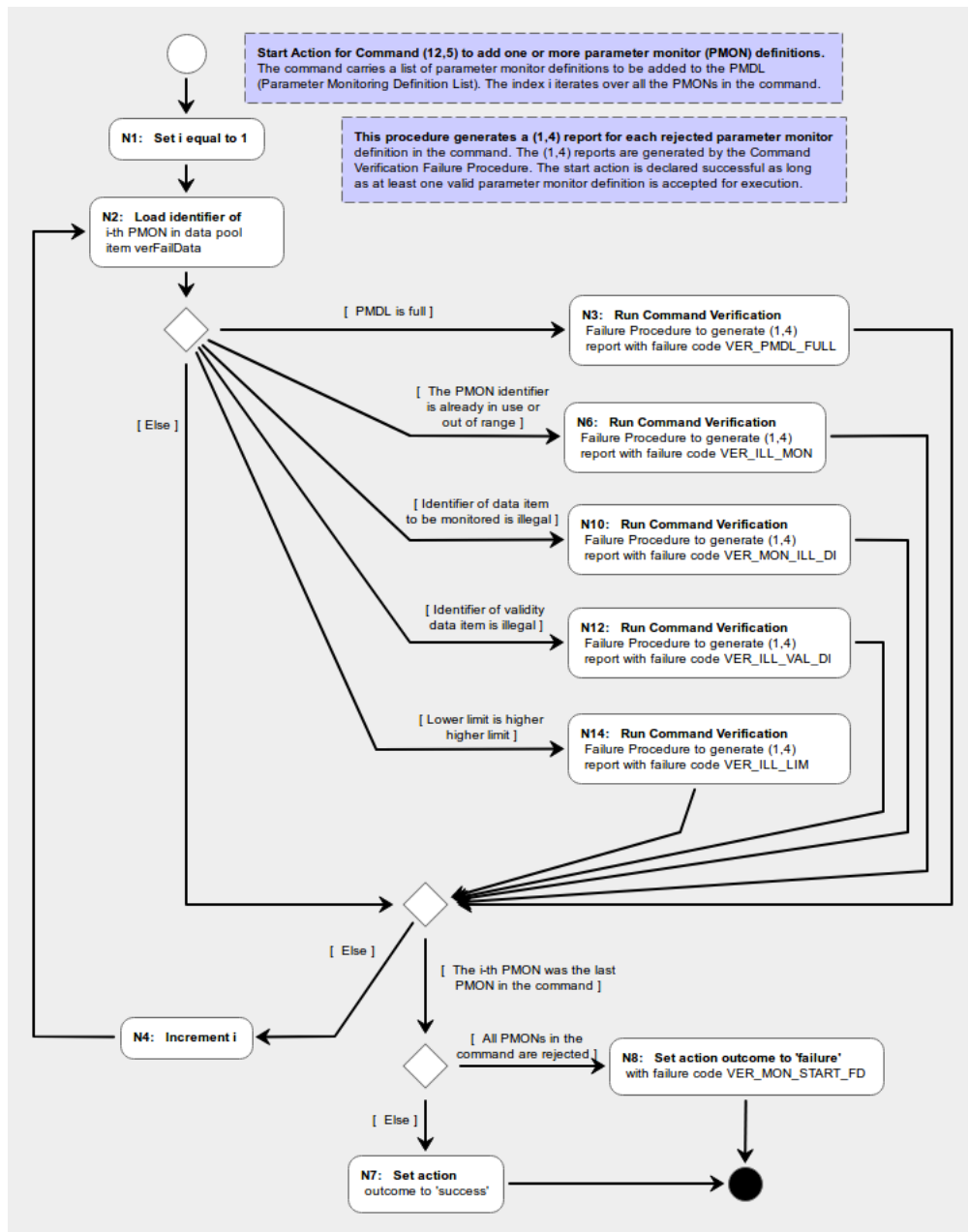


Fig. 11.8: Start Action of (12,5) Command Procedure

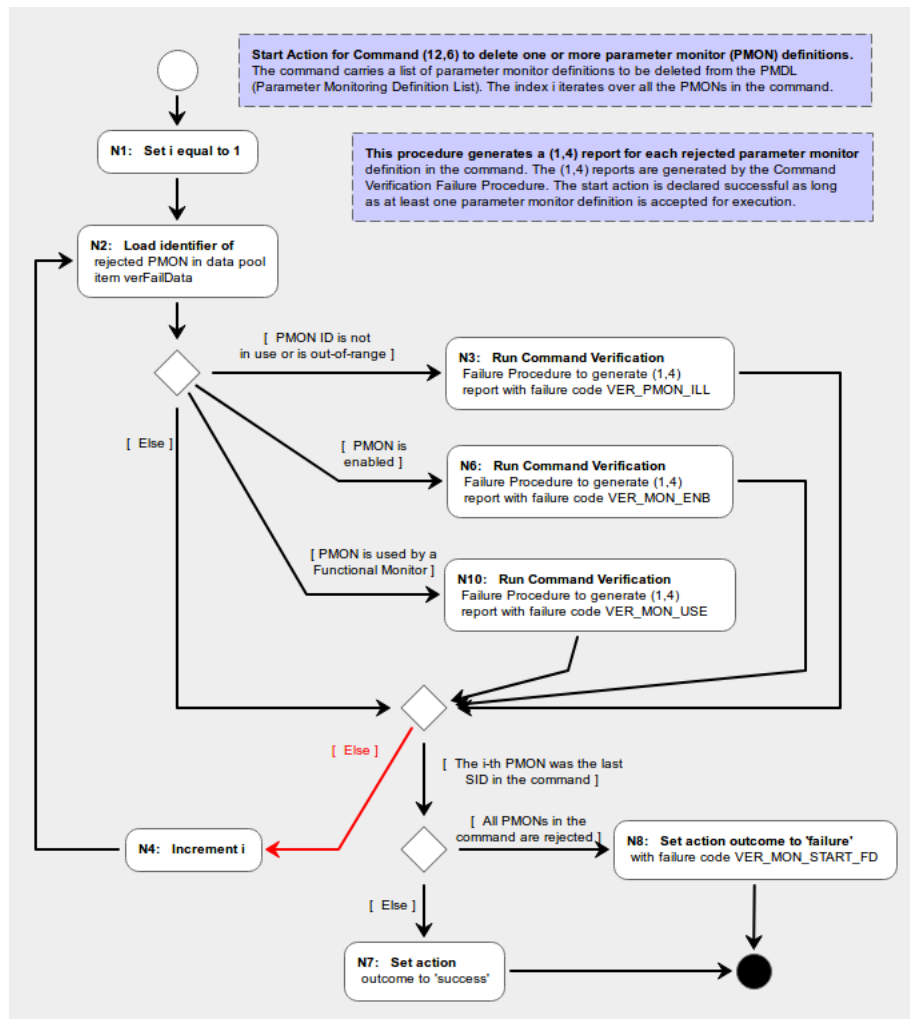


Fig. 11.9: Start Action of (12,6) Command Procedure

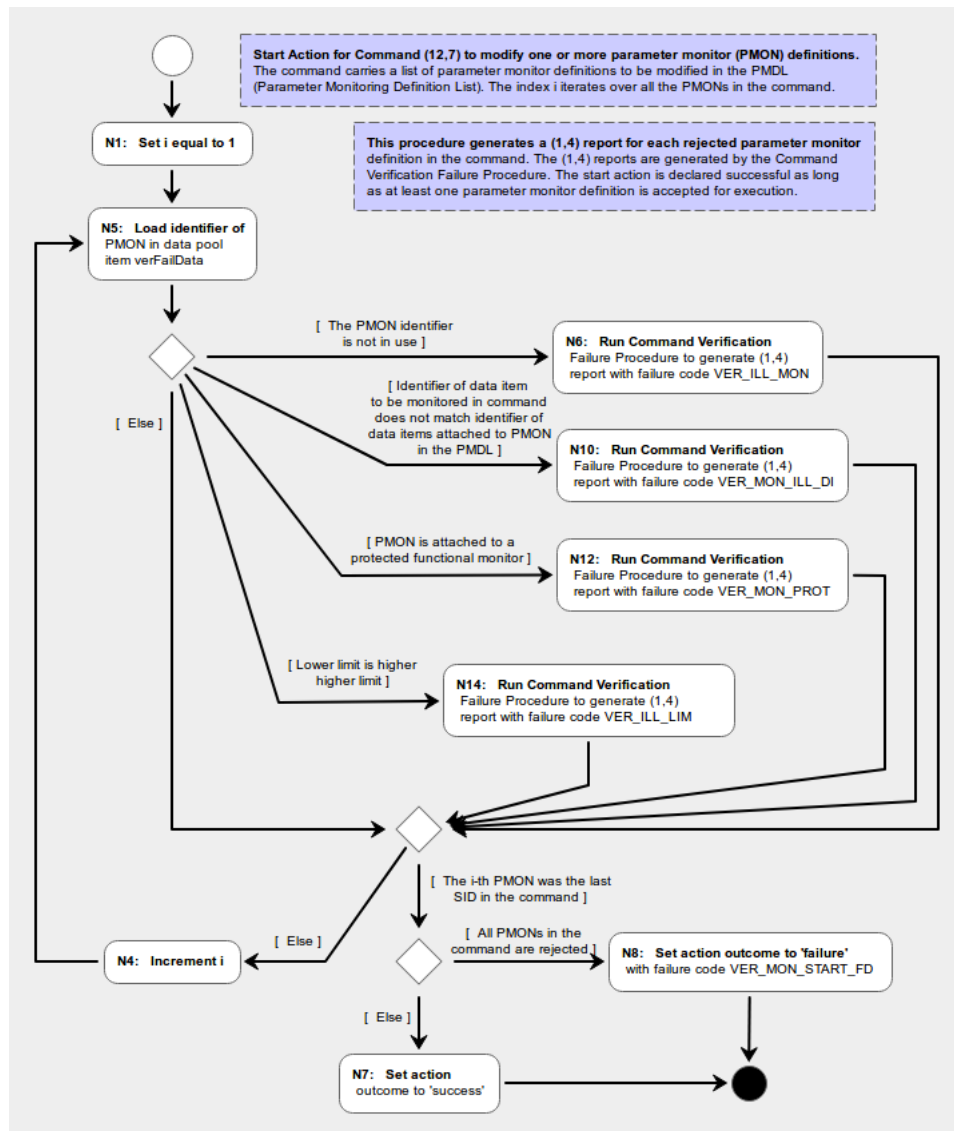


Fig. 11.10: Start Action of (12,7) Command Procedure

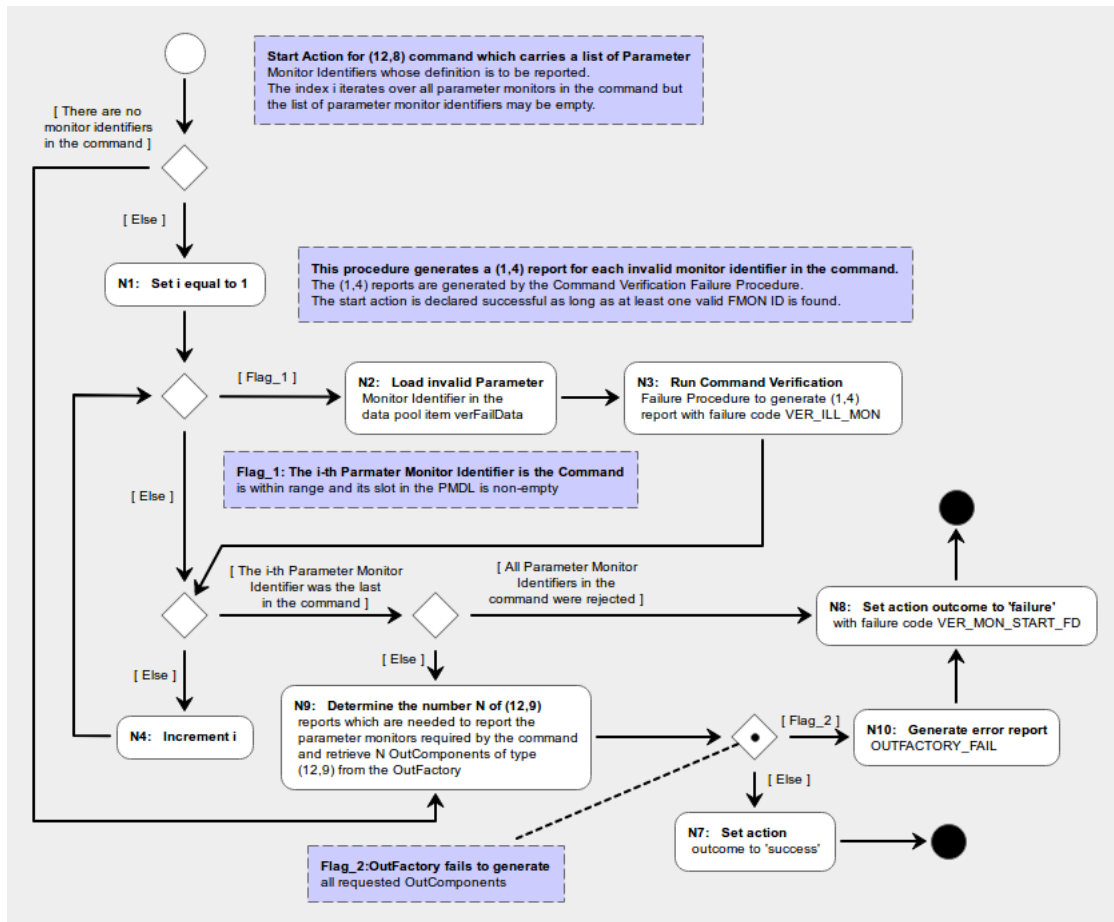


Fig. 11.11: Start Action of (12,8) Command Procedure

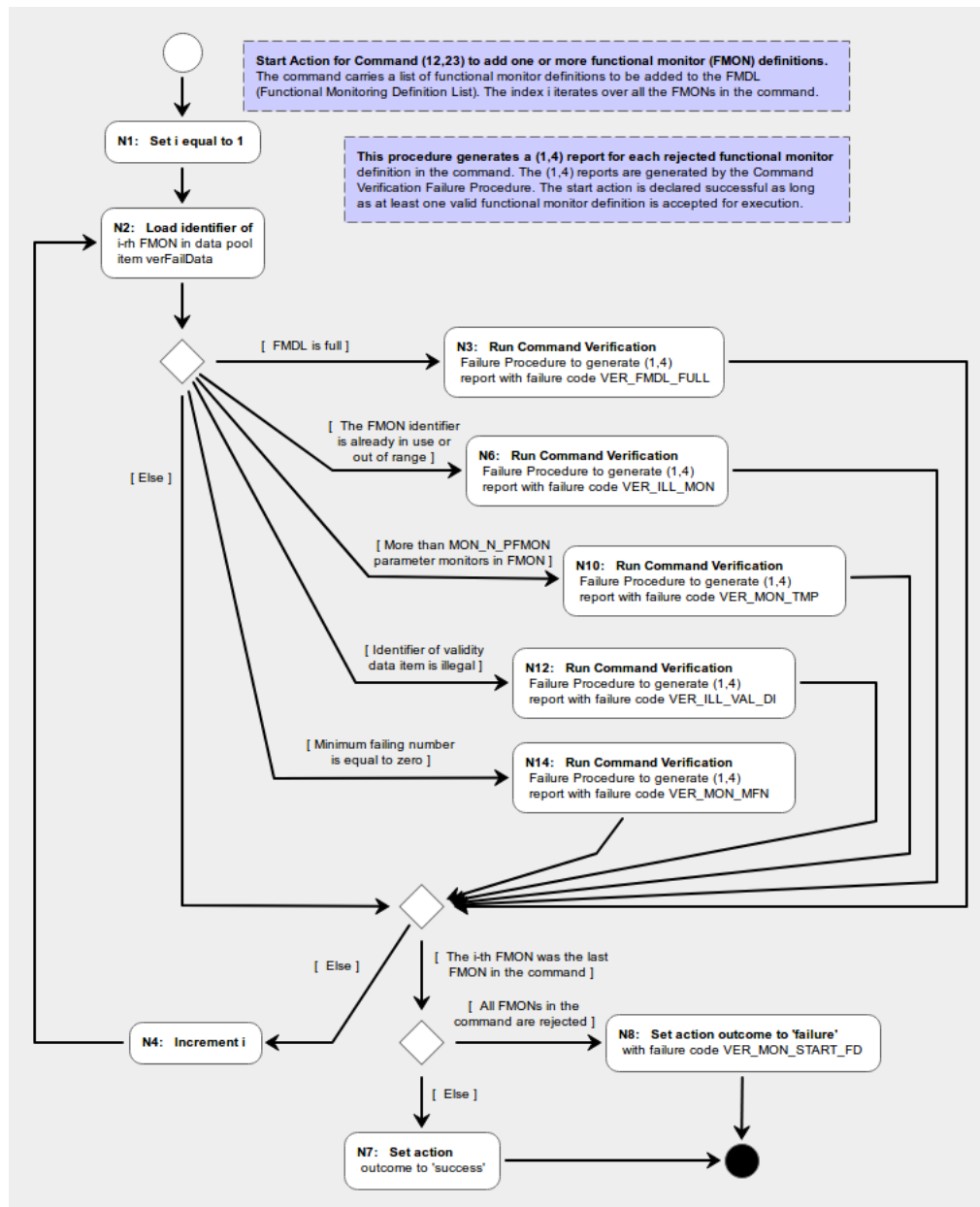


Fig. 11.12: Start Action of (12,23) Command Procedure

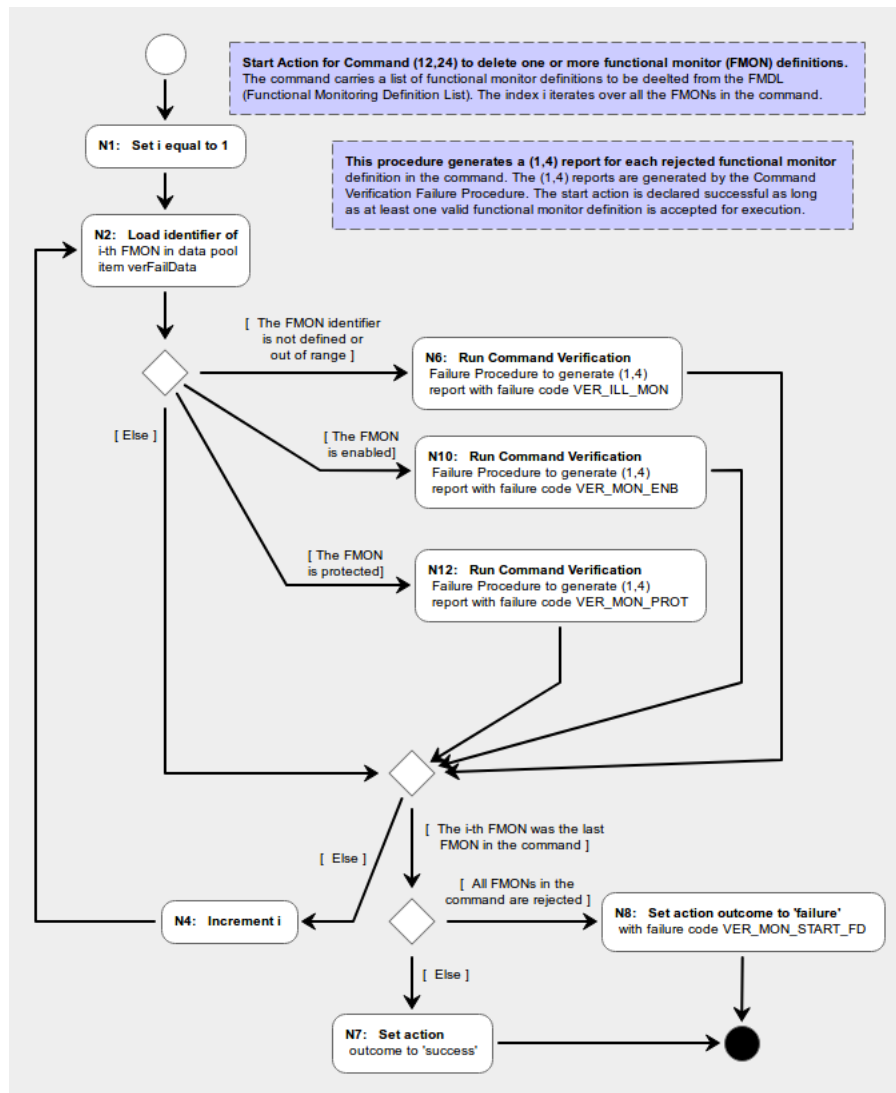


Fig. 11.13: Start Action of (12,24) Command Procedure

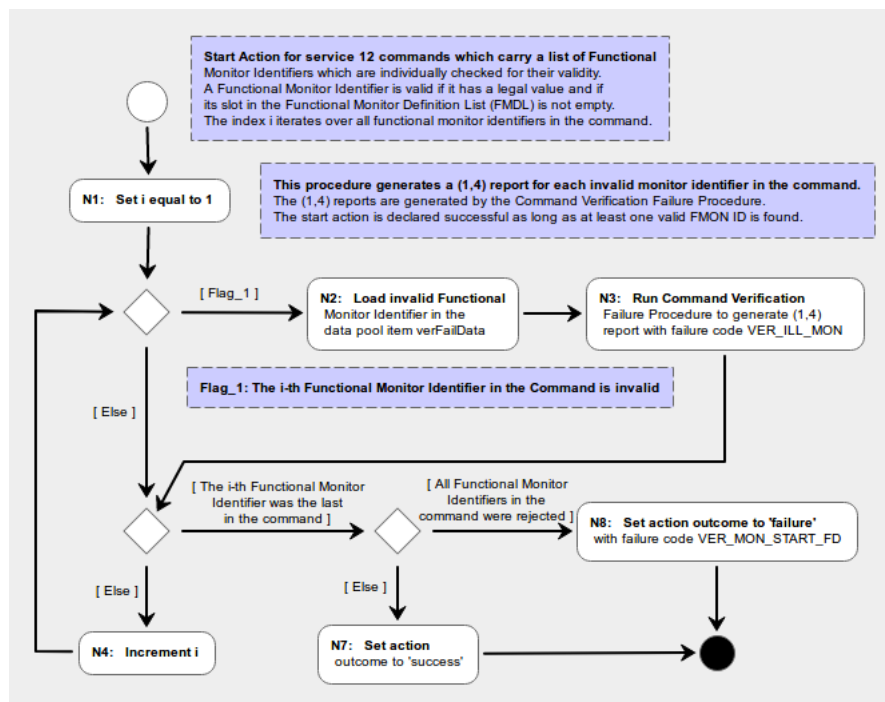


Fig. 11.14: Start Action of Multi-Functional Monitor Command Procedure

12 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.

12.1 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 equal 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 12.7). 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: $(\text{largeMsgTransId} \text{ MOD } \text{LPT_N_BUF})$.

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

12.2 The LPT State Machine

The LPT State Machine controls a down- or up-transfer. Its diagram is shown in figure 12.1.

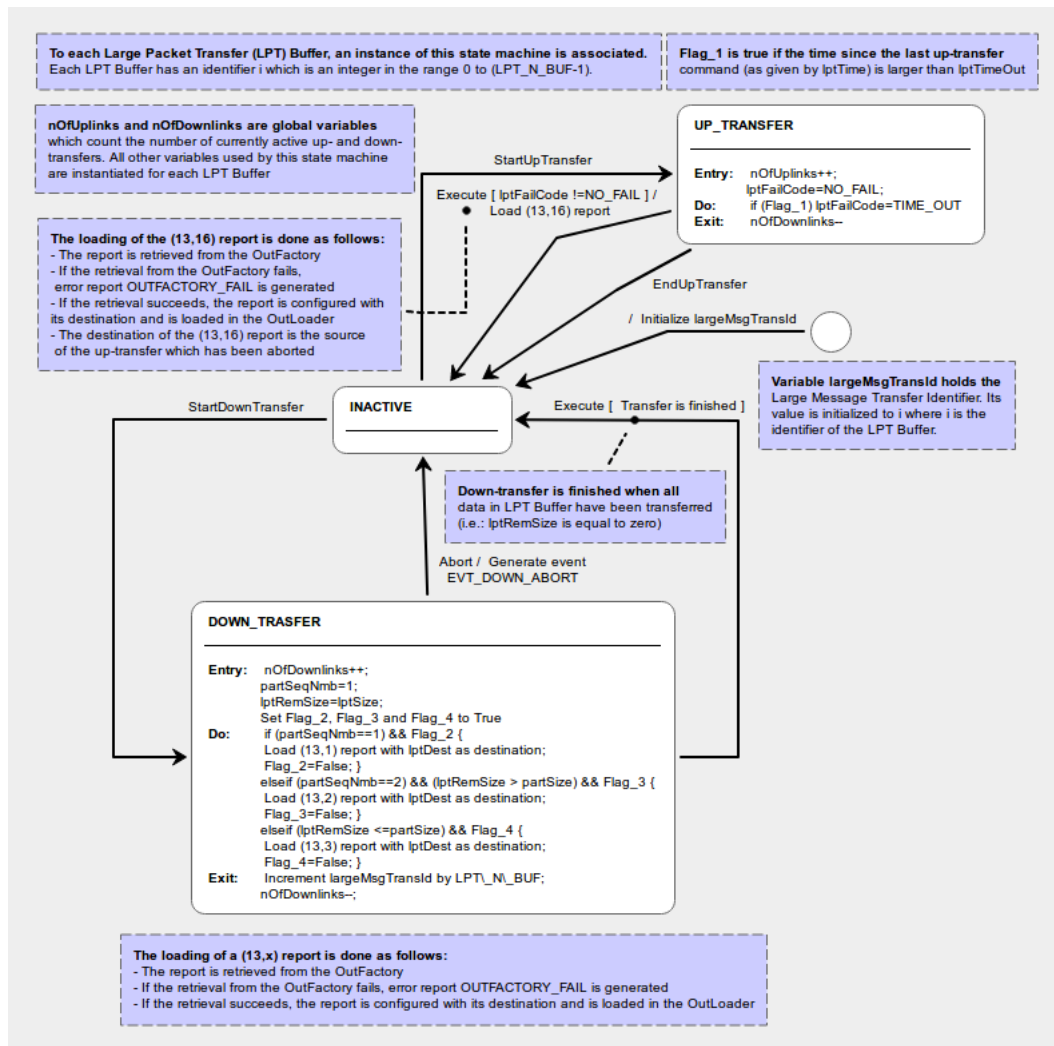


Fig. 12.1: Large Packet Transfer (LPT) State Machine

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**.

12.2.1 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 Out-

- Loader 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.

12.2.2 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. Avoiding 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.

12.3 Service 13 Report and Command Definition

Tables 12.1 to 12.9 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 12.1: Specification of `LptDownFirstRep` Component

Name	<code>LptDownFirstRep</code>
Description	First Downlink Part Report
Parameters	Large message transaction identifier, part sequence number and transfer data with the layout defined in clause 8.13.2.1 of [PS-SP]
Discriminant	None
Destination	The destination is loaded from parameter <code>lptDest</code> of the LPT Buffer holding the Large Packet to be transferred
Enable Check	Report is enabled if the LPT State Machine is in state <code>DOWN_TRANSFER</code>
Ready Check	Report is always ready
Repeat Check	Report is never repeated
Update Action	Load the first part of the large packet from the LPT Buffer; set the transaction identifier equal to <code>largeMsgTransId</code> ; set the part number equal to <code>partSeqNmb</code> ; increment <code>partSeqNmb</code> ; and decrement <code>lptRemSize</code> by <code>partSize</code>

Table 12.2: Specification of LptDownInterRep Component

Name	LptDownInterRep
Description	Intermediate Downlink Report
Parameters	Large message transaction identifier, part sequence number and transfer data with the layout defined in clause 8.13.2.2 of [PS-SP]
Discriminant	None
Destination	The destination is loaded from parameter lptDest of the LPT Buffer holding the Large Packet to be transferred
Enable Check	Report is enabled if the LPT State Machine is in state DOWN_TRANSFER
Ready Check	Report is always ready
Repeat Check	Report is repeated as long as lptRemSize is greater than partSize
Update Action	Load the next part of the large packet from the LPT Buffer; set the transaction identifier equal to largeMsgTransId ; set the part number equal to partSeqNmb ; increment partSeqNmb ; and decrement lptRemSize by partSize

Table 12.3: Specification of LptDownLastRep Component

Name	LptDownLastRep
Description	Last Downlink Part Report
Parameters	Large message transaction identifier, part sequence number and transfer data with the layout defined in clause 8.13.2.3 of [PS-SP]
Discriminant	None
Destination	The destination is loaded from parameter lptDest of the LPT Buffer holding the Large Packet to be transferred
Enable Check	Report is enabled if the LPT State Machine is in state DOWN_TRANSFER
Ready Check	Report is always ready
Repeat Check	Report is never repeated
Update Action	Load the last part of the large packet from the LPT Buffer, set the transaction identifier equal to largeMsgTransId ; set the part number equal to partSeqNmb ; set lptRemSize equal to zero

Table 12.4: Specification of LptUpFirstCmd Component

Name	LptUpFirstCmd
Description	Command (13,9) to carry the first part of an up-transfer
Parameters	Large message transaction identifier, part sequence number and part data for up-transfer with a layout as in clause 8.13.2.4 of [PS-SP]
Discriminant	None
Ready Check	Return “command is ready”
Start Action	Determine the identifier of the LPT Buffer for the up-transfer by computing: $(x \text{ MOD } \text{LPT_N_BUF})$ where ‘x’ is the Large Message Transaction Identifier. 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	Send command StartUpTransfer to LPT State Machine; copy the up-transfer data to LPT Buffer and set lptSize to be equal to the amount of copied data; set lptTime to the current time; set patSeqNbm to 1; set lptSrc to the source of the command
Termination Action	Set action outcome to ‘success’
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

Table 12.5: Specification of LptUpInterCmd Component

Name	LptUpInterCmd
Description	Command (13,10) to carry an intermediate part of an up-transfer
Parameters	Large message transaction identifier, part sequence number and part data for up-transfer with a layout as in clause 8.13.2.5 of [PS-SP]
Discriminant	None
Ready Check	Return “command is ready”
Start Action	Run the Procedure Up-Transfer Start Action of figure ??
Progress Action	Copy the up-transfer data to LPT Buffer and increment lptSize by the amount of copied data; set lptTime to the current time; set patSeqNbm to the part sequence number carried by the command
Termination Action	Set action outcome to ‘success’
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

Table 12.6: Specification of LptUpLastCmd Component

Name	LptUpLastCmd
Description	Command (13,11) to carry an intermediate part of an up-transfer
Parameters	Large message transaction identifier, part sequence number and part data for up-transfer with a layout as in clause 8.13.2.6 of [PS-SP]
Discriminant	None
Ready Check	Return “command is ready”
Start Action	Run the Procedure Up-Transfer Start Action of figure ??
Progress Action	Copy the up-transfer data to LPT Buffer and increment <code>lptSize</code> by the amount of copied data; set <code>lptTime</code> to the current time; set <code>patSeqNbm</code> to the part sequence number carried by the command; send <code>EndUpTransfer</code> command to LPT State Machine
Termination Action	Set action outcome to ‘success’
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

Table 12.7: Specification of LptUpAbortRep Component

Name	LptUpAbortRep
Description	Large Packet Uplink Abortion Report
Parameters	Large message transaction identifier, failure reason with the layout defined in clause 8.13.2.7 of [PS-SP]
Discriminant	None
Destination	The destination is the same as the source of the up-transfer being interrupted
Enable Check	Report is always enabled
Ready Check	Report is always ready
Repeat Check	Report is never repeated
Update Action	The large message transaction identifier is taken from parameter <code>largeMsgTransId</code> and the failure reason is read from variable <code>lptFailCode</code>

Table 12.8: Specification of LptStartDownCmd Component

Name	LptStartDownCmd
Description	Command (13,129) to start a down-transfer
Parameters	Large message transaction identifier
Discriminant	None
Ready Check	Return “command is ready”
Start Action	Determine the identifier of the LPT Buffer for the up-transfer by computing: $(x \text{ MOD LPT_N_BUF})$ where ‘x’ is the Large Message Transaction Identifier. 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	Set action outcome to ‘success’
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

Table 12.9: Specification of LptAbortDownCmd Component

Name	LptAbortDownCmd
Description	Command (13,130) to abort a down-transfer
Parameters	Large message transaction identifier
Discriminant	None
Ready Check	Return “command is ready”
Start Action	Determine the identifier of the LPT Buffer for the up-transfer by computing: $(x \text{ MOD LPT_N_BUF})$ where ‘x’ is the Large Message Transaction Identifier. 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	Set action outcome to ‘success’
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

12.4 Service 13 Constants

The service 13 constants are listed in table 12.10.

Table 12.10: Constants for Service 13 (Large Packet Transfer Service)

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

12.5 Service 13 Observables

The service 13 observables are listed in table 12.11.

Table 12.11: Observables for Service 13 (Large Packet Transfer Service)

Name	Description
nOfDownlinks	Number of on-going down-link transfers
nOfUplinks	Number of on-going up-link transfers
largeMsgTransId	Array of LPT_N_BUF elements. The I-th element holds the large message transaction identifier associated to the packet in the I-th LPT Buffer
lptSize	Array of LPT_N_BUF elements. The I-th element holds the size of the large packet in the I-th LPT Buffer
lptRemSize	Array of LPT_N_BUF elements. The I-th element holds the remaining size of the large packet in the I-th LPT Buffer (the part of the large packet not yet down-transferred)
partSeqNmb	Array of LPT_N_BUF elements. The I-th element holds the part sequence number for the currently on-going large packet transfer to/from the LPT Buffer
lptSrc	Array of LPT_N_BUF elements. The I-th element holds the source for the currently on-going large packet up-transfer to the LPT Buffer
lptTime	Array of LPT_N_BUF elements. The I-th element holds the time when the last up-transfer command to the LPT Buffer was received
lptFailCode	Array of LPT_N_BUF elements. The I-th element holds the failure code for the up-transfer to the LPT Buffer

12.6 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 which returns the start address and size of the buffer itself.

Table 12.12: 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

12.7 Service 13 Parameters

The service 13 parameters are listed in table 12.13.

Table 12.13: Parameters for Service 3 (Large Packet Transfer Service)

Name	Description
lptDest	Array of LPT_N_BUF elements. The i-th element is the destination for large packets originating from the i-th LPT Buffer
partSize	Array of LPT_N_BUF elements. The i-th element is the part size for large packets originating from the i-th LPT Buffer
lptTimeOut	Array of LPT_N_BUF elements. The i-th element is the time-out for the up-transfer to the i-th LPT Buffer

12.8 Service 13 Requirements

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

Table 12.14: Requirements for Service 13 (Large Packet Transfer Service)

Req. ID	Requirement Text
P-S13-1/S	<i>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)</i>
P-S13-2/S	<i>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)</i>
P-S13-3/S	<i>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)</i>
P-S13-4/S	<i>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)</i>
P-S13-5/S	<i>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)</i>
P-S13-6/S	<i>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)</i>
P-S13-7/S	<i>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)</i>
P-S13-8/S	<i>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)</i>
P-S13-9/S	<i>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)</i>
P-S13-10/A	<i>The LptDownFirstRep component shall close the OutComponent adaptation points as indicated in table 12.1</i>
P-S13-11/A	<i>The LptDownInterRep component shall close the OutComponent adaptation points as indicated in table 12.2</i>
P-S13-12/A	<i>The LptDownLastRep component shall close the OutComponent adaptation points as indicated in table 12.3</i>
P-S13-13/A	<i>The LptUpFirstCmd component shall close the InCommand adaptation points as indicated in table 12.4</i>
P-S13-14/A	<i>The LptUpInterCmd component shall close the InCommand adaptation points as indicated in table 12.5</i>
P-S13-15/A	<i>The LptUpLastCmd component shall close the InCommand adaptation points as indicated in table 12.6</i>
P-S13-16/A	<i>The LptUpAbortRep component shall close the OutComponent adaptation points as indicated in table 12.7</i>

Req. ID	Requirement Text
P-S13-17/A	<i>The LptStartDownCmd component shall close the InCommand adaptation points as indicated in table 12.8</i>
P-S13-18/A	<i>The LptAbortDownCmd component shall close the InCommand adaptation points as indicated in table 12.9</i>
P-S13-19/S	<i>The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 12.11</i>
P-S13-20/S	<i>The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the parameters listed in table 12.13</i>
P-S13-21/C	<i>The part size or destination of a large packet transfer shall not be updated while a transfer is under way</i>
P-S13-22/C	<i>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</i>
P-S13-23/C	<i>The user shall not start an up-transfer to an LPT Buffer which is being processed by the application</i>

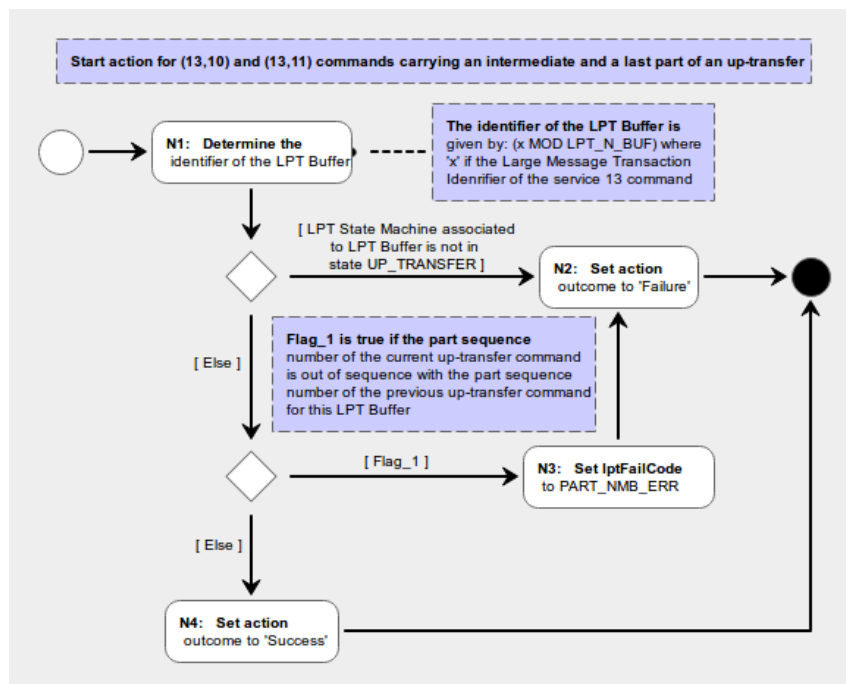


Fig. 12.2: Up-Transfer Start Action

13 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.

13.1 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 13.1 to 13.4 but the main points are as follows.

The `AreYouAliveCmd` command 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` command 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

13.1 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` with service TBD before starting an On-Board-Connection Test.

Tables 13.1 to 13.4 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 13.1: Specification of AreYouAliveCmd Component

Name	AreYouAliveCmd
Description	Command (17,1) to Perform Are-You-Alive Connection Test
Parameters	None
Discriminant	None
Ready Check	Return “command is ready”
Start Action	Retrieve (17,2) report from OutFactory and set action outcome to “success” if retrieval succeeds. If the retrieval fails, generate error report OUTFACTORY_FAILED and set outcome of Start Action to ‘failed’
Progress Action	Configure the (17,2) report with a destination equal to the source of the (17,1) command, load it in the OutLoader, and set action outcome to ‘completed’
Termination Action	Set action outcome to ‘success’
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

Table 13.2: Specification of AreYouAliveRep Component

Name	AreYouAliveRep
Description	Are-You-Alive Connection Report (17,2)
Parameters	None
Discriminant	None
Destination	The destination is set equal to the source of the (17,1) command which triggers the (17,2)
Enable Check	Report is always enabled
Ready Check	Report is always ready
Repeat Check	Report is never repeated
Update Action	No action

Table 13.3: Specification of OnBoardConnectCmd Component

Name	OnBoardConnectCmd
Description	Command (17,1) to Perform On-Board Connection Test
Parameters	Destination to which the (17,1) command must be sent
Discriminant	None
Ready Check	Return "command is ready"
Start Action	Run the procedure Start Action of OnBoardConnectCmd Command of figure 13.1
Progress Action	Run the procedure Progress Action of OnBoardConnectCmd Command of figure 13.2
Termination Action	Set action outcome to 'success' if the (17,4) report was issued and to 'failure' otherwise
Abort Action	Do nothing
Operation to Report Progress Successful	Do nothing (no progress reports are generated by this command)

Table 13.4: Specification of OnBoardConnectRep Component

Name	OnBoardConnectRep
Description	On-Board Connection Report (17,4)
Parameters	None
Discriminant	None
Destination	The destination is set equal to the source of the (17,3) command which triggers the (17,4)
Enable Check	Report is always enabled
Ready Check	Report is always ready
Repeat Check	Report is never repeated
Update Action	No action

13.2 Service 17 Observables

Service 17 maintains and makes available in the data pool one single observable listed in table 13.5.

Table 13.5: Observables for Service 17 (Test Service)

Name	Description
areYouAliveSrc	Source of the latest (17,2) report received in response to a (17,1) command triggered by a (17,3) command

13.3 Service 17 Parameters

Service 17 maintains and makes available in the data pool the parameters listed in table 13.6.

Table 13.6: Parameters for Service 17 (Test Service)

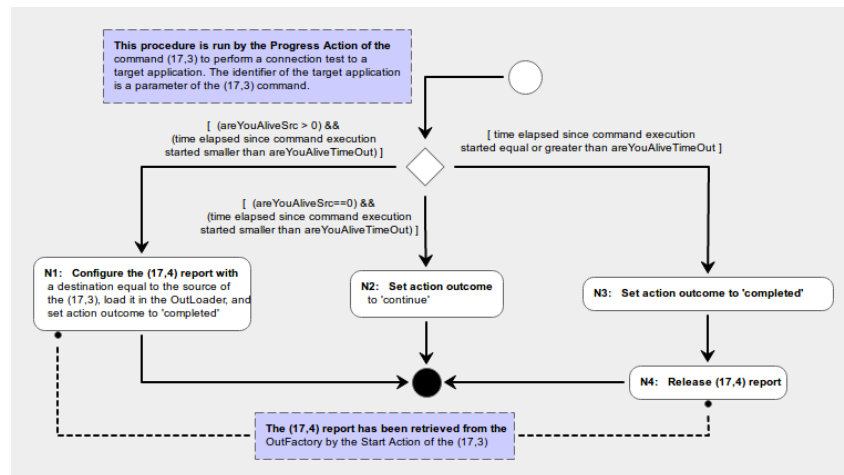
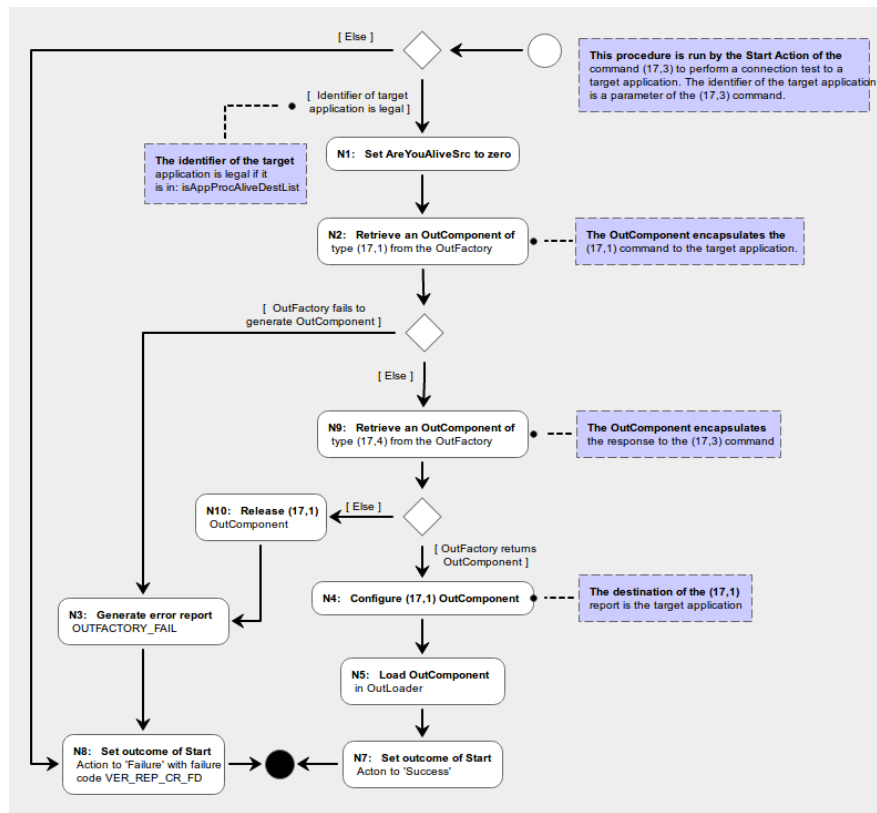
Name	Description
areYouAliveTimeOut	Time-out for the Are-You-Alive Test initiated in response to an Is-Application-Process-Alive Test
onBoardConnectDestLst	List of identifiers of target applications for an On-Board-Connection Test

13.4 Service 17 Requirements

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

Table 13.7: Requirements for Service 17 (Test Service)

Req. ID	Requirement Text
P-S17-1/S	<i>The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, an AreYouAliveCmd component to encapsulate a (17,1) command</i>
P-S17-2/S	<i>The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, an AreYouAliveRep component to encapsulate a (17,2) report</i>
P-S17-3/S	<i>The PUS Extension of the CORDET Framework shall provide, as an extension of the InCommand, an OnBoardConnectCmd component to encapsulate a (17,3) command</i>
P-S17-4/S	<i>The PUS Extension of the CORDET Framework shall provide, as an extension of the OutComponent, an OnBoardConnectCmd component to encapsulate a (17,4) report</i>
P-S17-5/A	<i>The AreYouAliveCmd component shall close the InCommand adaptation points as indicated in table 13.1</i>
P-S17-6/A	<i>The AreYouAliveRep component shall close the OutComponent adaptation points as indicated in table 13.2</i>
P-S17-7/A	<i>The OnBoardConnectCmd component shall close the InCommand adaptation points as indicated in table 13.3</i>
P-S17-8/A	<i>The OnBoardConnectRep component shall close the OutComponent adaptation points as indicated in table 13.4</i>
P-S17-9/C	<i>An application shall not be sent a (17,3) command before execution of the previous (17,3) command has completed</i>
P-S17-10/S	<i>The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 13.5</i>
P-S17-11/S	<i>The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the parameters listed in table 13.5</i>



14 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 severity level of the event
- The description of the event report
- The parameters carried by the event report

Table A.1: Error Reports

Name	Sev.	Description	Parameters
EVT_DOWN_ABORT	2	Generated by an LPT State Machine when a down-transfer is aborted	LPT State Machine Identifier
EVT_UP_ABORT	2	Generated by an LPT State Machine when an up-transfer is aborted	LPT State Machine Identifier
EVT_MON_LIM_R	3	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	3	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	3	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	3	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	3	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	3	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	2	Generated when the Monitoring Function Procedure tries to add an entry to the Check Transition List but the list is full	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 severity of the error using the same severity levels defined for service 5 reports
- The description of the error report
- The parameters carried by the error report

Table B.1: Error Reports

Name	Sev.	Description	Parameters
INLOADER_ACC_FAIL	3	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	3	Generated by InLoader when it receives a report with an invalid destination	Packet identifier of report and invalid destination
OUTFACTORY_FAIL	3	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	3	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 the invalid destination

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	Failure code for start action of commands (3,1) and (3,2) when attempt to create a new report with a SID which is already in use	SID
VER_FULL_RDL	Failure code for start action of commands (3,1) and (3,2) when they attempt to create a new report at a time when the RDL is already full	None
VER_RDL_CONSTr	Failure code for start action of commands (3,1) and (3,2) when their report configuration data violate an RDL constraint of table 9.1	The identifier of the violated constraint
VER_DUPL_DI	Failure code for start action of commands (3,1) and (3,2) when they carry the same data item identifier twice	The duplicated data item identifier
VER_ILL_SID	Failure code for start action of a service 3 command when an invalid SID is encountered	The invalid SID
VER_ENBABLED_SID	Failure code for start action of commands (3,3) and (3,4) when a SID which is enabled is encountered	The enabled SID
VER_SID_START_FD	Failure code for start action of multi-instruction service 3 commands when all the instructions in the command have been rejected	None
VER_FACT_PRGR_FD	Failure code for progress action of multi-instruction service 3 command when the attempt to retrieve a report from the OutFactory fails	The SID for which the retrieval from the OutFactory was attempted

Name	Description	Ver. Failure Data
VER_ILL_EID	Failure code for start action of a service 5 command when an illegal EID is encountered	The illegal EID
VER_EID_START_FD	Failure code for start action of multi-instruction service 5 commands when all the instructions in the command have been rejected	None
VER_ILL_MON	Failure code for start action of a service 12 command when a Parameter or Functional Monitor Identifier is out-of-range or not defined	The rejected Parameter or Functional Monitor Identifier
VER_MON_START_FD	Failure code for start action of multi-instruction service 12 commands when all the instructions in the command have been rejected	None
VER_PMDL_FULL	Failure code for start action of service 12 command which finds the Parameter Monitor Definition List (PMDL) full	The parameter monitor identifier for which the violation was found
VER_MON_ILL_DI	Failure code for start action of service 12 command which finds the data item identifier of the parameter to be monitored illegal	The parameter monitor identifier for which the violation was found
VER_MON_PROT	Failure code for start action of service 12 command which finds a parameter monitor which belongs to a protected functional monitor	The parameter monitor identifier for which the violation was found
VER_MON_ENB	Failure code for start action of service 12 command which finds a parameter or functional monitor which is enabled	The parameter or functional monitor identifier for which the violation was found
VER_MON_USE	Failure code for start action of service 12 command which finds a parameter monitor which is used by a functional monitor	The parameter monitor identifier for which the violation was found
VER_FMDL_FULL	Failure code for start action of service 12 command which finds the Functional Monitor Definition List (FMDL) full	The functional monitor identifier for which the violation was found
VER_MON_TMP	Failure code for start action of service 12 command which finds too many parameter monitors in a functional monitor	The functional monitor identifier for which the violation was found
VER_MON_MFN	Failure code for start action of service 12 command which finds a value of minimum failing number equal to zero	The functional monitor identifier for which the violation was found

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.
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
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
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
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.
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
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
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
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
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.
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
b		Each event report type shall be involved in exactly one event reporting transaction type.	C1	See justification of first requirement in this clause

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).
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 must 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)

N	Title	Requirement	C	Justification
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		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
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
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.
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

N	Title	Requirement	C	Justification
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
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
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
5.4.10a	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.
5.4.11.2.1a	Request	Each request shall be generated by exactly one subservice user.	C1	The CORDET Framework allows a command to have one single source.
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 identifiers of both their source and destination and a source sequence counter (see section 4.1 of the CORDET Framework Definition Document)

N	Title	Requirement	C	Justification
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
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)
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
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
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.
6.1.3.3a	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.
		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

N	Title	Requirement	C	Justification
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
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-*).
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.
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.
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-*).
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-*).
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
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-*).
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-*).
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-*).
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
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

N	Title	Requirement	C	Justification
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
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
b		Each housekeeping parameter report shall contain exactly one housekeeping parameter notification.	C1	See definition of hkRep component
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

N	Title	Requirement	C	Justification
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
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
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
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

N	Title	Requirement	C	Justification
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
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
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
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
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
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

N	Title	Requirement	C	Justification
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
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
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
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

N	Title	Requirement	C	Justification
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
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
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

N	Title	Requirement	C	Justification
6.3.5	Parameter functional reporting configuration subservice		n.a.	This subservice is not supported by the PUS Extension of the CORDET Framework
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 10.1)
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 10.1

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 10.1
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
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
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

N	Title	Requirement	C	Justification
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
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
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
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
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
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	n.a.	This service is not yet supported by the PUS Extension of the CORDET Framework
6.12	On-Board Monitoring	Definition of service 12		

N	Title	Requirement	C	Justification
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
6.12.2.3.1a	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
6.12.3.2.2a	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

N	Title	Requirement	C	Justification
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
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)
f		Monitoring intervals shall be expressed in 'on-board parameter minimum sampling interval' units.	C1	See statement of compliance to previous two clauses
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
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.

N	Title	Requirement	C	Justification
		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
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
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
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

N	Title	Requirement	C	Justification
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
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
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
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.
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.
		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
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

N	Title	Requirement	C	Justification
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
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
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
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

N	Title	Requirement	C	Justification
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
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
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
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
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
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
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

N	Title	Requirement	C	Justification
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
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
b		Each request to report the out-of-limits shall contain exactly one instruction to report the out-of-limits.	C1	See definition of MonRepOOLCmd command and MonRepOOLRep report
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]
6.12.4.1.2a	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.

N	Title	Requirement	C	Justification
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.
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.
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
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
6.12.4.5.2a	Enable functional monitoring definitions	The functional monitoring subservice shall provide the capability to enable functional monitoring definitions.		
b		Each request to enable functional monitoring definitions shall contain one or more instructions to enable a functional monitoring definition.		

N	Title	Requirement	C	Justification
c		Each instruction to enable a functional monitoring definition shall contain: 1. the identifier of the functional monitoring definition.		
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.		
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.		
6.12.4.5.3a	Disable functional monitoring definitions	The functional monitoring subservice shall provide the capability to disable functional monitoring definitions.		
b		Each request to disable functional monitoring definitions shall contain one or more instructions to disable a functional monitoring definition.		
c		Each instruction to disable a functional monitoring definition shall contain: 1. the identifier of the functional monitoring definition.		
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.		

N	Title	Requirement	C	Justification
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.		
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.		
b		Each request to protect functional monitoring definitions shall contain one or more instructions to protect a functional monitoring definition.		
c		Each instruction to protect a functional monitoring definition shall contain: 1. the identifier of the functional monitoring definition.		
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.		
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.		

N	Title	Requirement	C	Justification
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.		
b		Each request to unprotect functional monitoring definitions shall contain one or more instructions to unprotect a functional monitoring definition.		
c		Each instruction to unprotect a functional monitoring definition shall contain: 1. the identifier of the functional monitoring definition.		
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.		
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.		
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.		

N	Title	Requirement	C	Justification
b		Each request to add functional monitoring definitions shall contain one or more instructions to add a functional monitoring definition.		
c		Each instruction to add a functional monitoring definition shall contain: 1. the contents of the functional monitoring definition.		
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.		
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.		
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.		

N	Title	Requirement	C	Justification
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.		
b		Each request to delete functional monitoring definitions shall contain one or more instructions to delete a functional monitoring definition.		
c		Each instruction to delete a functional monitoring definition shall contain: 1. the identifier of the functional monitoring definition.		
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"."		
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.		

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.		
c		Each instruction to report a functional monitoring definition shall contain: 1. the identifier of the functional monitoring definition.		
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.		
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.		
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.		
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.		
6.13	Large Packet Transfer	Definition of service 13		

N	Title	Requirement	C	Justification
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
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
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])
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.

N	Title	Requirement	C	Justification
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.
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.

N	Title	Requirement	C	Justification
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
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.
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
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

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.
		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

N	Title	Requirement	C	Justification
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
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
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
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
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.

N	Title	Requirement	C	Justification
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
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

N	Title	Requirement	C	Justification
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
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
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
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
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
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
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
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
7.3.13a	Packet	Each packet field used to carry packets shall be of PTC 12.	C2	See justification of first requirement in clause 7.3.1a

N	Title	Requirement	C	Justification
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
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
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
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.

N	Title	Requirement	C	Justification
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 Out Component of the CORDET Framework (see section 6.1.1 of the CORDET Framework Definition Document).
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.
		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.
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
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.