

CORDET FRAMEWORK - PUS EXTENSION

Software Verification Report

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:	n.a.
Document Ref.:	PP-RP-PUX-0001
Issue:	0.2
Created On:	20/05/2019, at: 22:40

Contents

1	References	3
2	Introduction	4
2.1	Framework Constituents	4
2.2	Open Issues	4
3	Requirement Traceability	6
4	Requirement Verification	18
4.1	Verification of Standard Requirements	18
4.2	Verification of Commands and Reports	18
4.3	Verification of Data Pool	19
4.4	Verification of Command Rejection Codes	20
4.5	Verification of FW Profile Procedures	20
4.6	Verification of Design-Level Functions	20
5	Static Code Analysis	33
6	Code Coverage	34
7	Unit-Level Tests	36
8	Comment Coverage	37
9	Memory Management	38
10	Schedulability	39
11	Metrics	40
11.1	Software Metrics	40
11.2	Process Metrics	40
12	Automatically Generated Code	41
12.1	Generation by the FW Profile Editor	41
12.2	Generation by the CORDET Editor	42
13	Qualification Status of External Libraries	43
A	FW Profile Editor and Code Generator	44
A.1	Overview of FW Profile Editor	44
A.2	Generated Code	46
A.3	Use of Generated Code	46
A.4	Summary of Mode of Use of FW Profile Editor	46

1 References

The documents referenced in this document are listed in table 1.1.

The following documents are for reference and/or guideline only.

Table 1.1: Referenced documents

Ref	Description	Doc. Number	Iss.
[CR-SP]	The CORDET Framework, www.pnp-software.com/cordetfw	Release	1
[FW-SP]	The Framework Profile, www.pnp-software.com/fwprofile	Release	1.3.1
[PS-SP]	Ground Systems and Operations, Telemetry and Telecommand Packet Utilization Standard	ECSS-E-70-41C	C
[PX-SP]	The PUS Extension of the CORDET Framework – Specification	PP-SP-PUX-001	0.2
[PX-UM]	The PUS Extension of the CORDET Framework – User Manual	PP-UM-PUX-001	0.2
[PX-IC]	The PUS Extension of the CORDET Framework – TM/TC Interface Control Document	PP-IC-PUX-001	0.2

2 Introduction

This document is the software verification report for the PUS Extension of the CORDET Framework. It presents and evaluates the verification evidence collected for the framework at design and implementation level. Table 2.1 lists all the verification issues which are addressed in this document. The rightmost column points to the section within the document where each issue is discussed in detail.

The PUS Extension of the CORDET Framework is specified in reference [CR-SP] and its user manual is in reference [CR-UM].

Table 2.1: Verification Issues Addressed in This Document

Name	Short Description	Sect.
Req. traceability	Verify that all requirements are implemented	3
Req. verification	Verify that all requirements are verified	4
Static code analysis	Verify that the code is free of statically identifiable bugs	5
Code coverage	Verify level of code coverage (statement, branch and condition) achieved through unit-level tests	6
Unit tests	Verify that all unit-level tests are successfully executed	7
Comment coverage	Verify that all parts of the code are commented	8
Mem. management	Verify that there are no memory leaks	9
Schedulability	Verify that the framework supports application- and system-level schedulability analyses	10
Software metrics	Verify that the level of complexity of the framework code is acceptable	11.1
Automatically generated code	Verify that the quality of the automatically generated code used in the framework is adequate	12
External libraries	Verify that the qualification levels of the external libraries used by the framework is adequate	13

2.1 Framework Constituents

For the purposes of assessing the qualification status of the PUS Extension of the CORDET Framework, it is useful to divide its code into the constituent parts listed in table 2.2. The second column in the table identifies the generation method of each component and is one of: (a) 'Reused', if the component is imported into the PUS Extension of the CORDET Framework without changes; (b) 'Generated', if the component is auto-coded; or (c) 'Manual', if the component is developed manually. The third column in the table gives the qualification status of the component which is either 'Qualified', if sufficient verification evidence is already available for the component, or 'To Be Qualified', if the component must be verified within the PUS Extension of the CORDET Framework Project itself.

2.2 Open Issues

An *Open Issue* is an issue whose resolution may have an impact on the design or implementation of the framework. The open issues are tracked in the framework's GitHub project.

Table 2.2: PUS Extension of the CORDET Framework Components

ID	Description	Gen. Status	Qual. Status	Remarks
1	C Libraries	Reused	Qualified	C libraries used by the framework are listed in section 3.1 of reference [CR-UM]; they are assumed to be qualified through their extensive use in open software projects
2	FW Profile Library	Reused	Qualified	This library is provided with a qualification data package which is evaluated in section 13
3	CORDET Framework Library	Reused	Qualified	This library is provided with a qualification data package which is evaluated in section 13
4	Configuration Code	Generated	To Be Qualified	Configuration code for the state machines and procedures used in the framework. Its qualification status is evaluated in section 12.
5	All Other Code	Manual	To Be Qualified	Manually developed code of the framework

3 Requirement Traceability

The PUS Extension of the CORDET Framework is specified in reference [CR-SP]. The framework is specified in terms of three kinds of requirements:

- *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.

Table 3.1 shows how each Standard Requirement is implemented in the framework code.

The Adaptation Points are of two kinds: adaptation points which are inherited from the CORDET Framework and new adaptation points which are defined by the PUS Extension. The new adaptation points are those with an identifier of the form Sx. Table 3.2 list both kinds of adaptation points and shows how each is mapped to code. Note that, in the case of adaptation points which are inherited from the CORDET Framework and are closed by the PUS Extension, the mapping to code takes the form of a pointer to a code-level module where the close-out is done; in the case of new adaptation points, instead, the mapping to code takes the form of a statement explaining how the adaptation point can be closed at code level by an application.

The Use Constraint Requirements are not mapped to code.

Table 3.1: Traceability of Standard Requirements to Implementation

Req. ID	Requirement Text	Implementation
DP-1	The PUS Extension of the CORDET Framework shall provide a Data Pool component implementing the behaviour of the Data Pool State Machine of figure 4.1 in [PX-SP].	Component is generated by FW Profile Editor in module CrPsDataPool
DP-3	In state READY, the Data Pool component shall provide operations to let other components access the current value of its data items	Accessor functions are generated by CORDET Editor in modules in directory src/DataPool
CRA-1	Components encapsulating a command or a report shall implement all attributes defined for them by the PUS	For each service srv which it supports, the PUS Extension provides a header file /Pckt/CrPsPcktSrv.h which defines the layout of the commands and reports in that service and the functions to access their parameters.
CRA-2	Components encapsulating a command or a report shall provide operations to access in read and write mode all their PUS-defined attributes	Parameter are accessed by first extracting the packet encapsulating a command or report and by then using the functions described at the previous requirement
CRA-3	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	See previous requirement
FAC-1	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	The implementation provided by the CORDET Framework in module CrFwInFactory is used unchanged. The packet allocation policy must be provided by applications by implementing interface CrFwPckt.h
FAC-2	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	The implementation provided by the CORDET Framework in module CrFwOutFactory is used unchanged. The packet allocation policy must be provided by applications by implementing interface CrFwPckt.h
FAC-3	The two factory components shall maintain and make accessible through the data pool the observables listed in table 5.1 in [PX-SP]	Observables are implemented in generated module CrPsDpFct

Req. ID	Requirement Text	Implementation
S1-1	The PUS Extension of the CORDET Framework shall provide OutComponents implementing the service 1 reports listed in tables 7.2 to 7.10 in [PX-SP]	Report Xyz is implemented in module CrPsOutCmpXyz. The header file of this module and a skeleton body file is generated by the CORDET Editor.
S1-3	The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 7.11 in [PX-SP]	The service 1 observables are implemented in module CrPsDpVer. This module is generated by the CORDET Editor.
S3-1	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 8.1 in [PX-SP]	See the data structures defined in CrPsDpHk
S3-2	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	This requirement is not implemented
S3-4	The PUS Extension of the CORDET Framework shall provide OutComponents and InCommands implementing the reports and commands defined in tables 4.1 to 8.19 in [PX-SP]	Report Xyz is implemented in module CrPsOutCmpXyz. Command Xyz is implemented in module CrPsInCmdXyz. The header files of these modules and skeleton body files are generated by the CORDET Editor.
S3-5	The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables and parameters listed in table 8.21 in [PX-SP]	See the data structures defined in module CrPsDpHk
S5-1	The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 9.10 in [PX-SP]	See the data structures defined in module CrPsDpEvt
S5-4	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	See functions in CrPsEvtGenPreDefEvt module

Req. ID	Requirement Text	Implementation
S5-5	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	See functions in CrPsEvtGenPreDefEvt module
S5-6	If the OutComponent retrieval from the OutFactory fails, the Generate Pre-Defined Event Function shall generate an error report of type OUTFACTORY_FAIL	See functions in CrPsEvtGenPreDefEvt module
S5-7	The PUS Extension of the CORDET Framework shall support the commands and reports specified in tables 9.1 to 9.8 in [PX-SP]	Report Xyz is implemented in module CrPsOutCmpXyz. Command Xyz is implemented in module CrPsInCmdXyz. The header files of these modules and skeleton body files are generated by the CORDET Editor.
S5-8	The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables and parameters listed in table 9.10 in [PX-SP]	See the data structures defined in CrPsDpEvt
S11-1	The PUS Extension of the CORDET Framework shall implement a Time-Based Schedule (TBS) consisting of SCD_N_TBA Time-Based Activities (TBAs) with the attributes defined in table 10.1 in [PX-SP]	This service is not yet implemented
S11-2	The PUS Extension of the CORDET Framework shall provide a Time-Based Execution Procedure implementing the behaviour shown in figure 10.7 in [PX-SP]	This service is not yet implemented
S11-6	The PUS Extension of the CORDET Framework shall support the commands and reports specified in tables 10.2 to 10.14	This service is not yet implemented
S12-1	The PUS Extension of the CORDET Framework shall provide a Monitoring Function Procedure implementing the behaviour shown in figure 11.1	This service is not yet implemented

Req. ID	Requirement Text	Implementation
S12-2	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	This service is not yet implemented
S12-3	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	This service is not yet implemented
S12-4	The PUS Extension of the CORDET Framework shall provide the Limit Check Monitoring Procedure implementing the behaviour shown in figure 11.2	This service is not yet implemented
S12-5	The PUS Extension of the CORDET Framework shall provide the Expected Value Monitoring Procedure implementing the behaviour shown in figure 11.3	This service is not yet implemented
S12-6	The PUS Extension of the CORDET Framework shall provide the Delta Value Monitoring Procedure implementing the behaviour shown in figure 11.4	This service is not yet implemented
S12-7	The PUS Extension of the CORDET Framework shall provide the Process CTL Procedure implementing the behaviour shown in figure 11.5	This service is not yet implemented
S12-8	The PUS Extension of the CORDET Framework shall provide the Functional Monitor Notification Procedure implementing the behaviour shown in figure 11.6	This service is not yet implemented
S12-14	The PUS Extension of the CORDET Framework shall support the commands and reports specified in tables 11.6 to 11.33	This service is not yet implemented
S13-1	The PUS Extension of the CORDET Framework shall support the commands and reports specified in tables 12.1 to 12.9	This service is not yet implemented

Req. ID	Requirement Text	Implementation
S13-2	The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables and parameters listed in table 12.11 in [PX-SP]	This service is not yet implemented
S17-1	The PUS Extension of the CORDET Framework shall support the commands and reports specified in tables 13.1 to 13.4 in [PX-SP]	Report Xyz is implemented in module CrPsOutCmpXyz. Command Xyz is implemented in module CrPsInCmdXyz. The header files of these modules and skeleton body files are generated by the CORDET Editor.
S17-3	The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables and parameters listed in table 13.6 in [PX-SP]	See the data structures defined in CrPsDpTst

Table 3.2: Traceability of Adaptation Points to Implementation

Req. ID	Origin	Description	Implementation
DP-1	New AP	Data Pool Load Procedure	Module <code>CrPsDataPoolFunc</code>
DP-7	New AP	Definition of Data Items in the Data Pool Component	For each service <code>srv</code> , the associated observables and parameters are implemented in module <code>DataPool/CrPsDpSrv</code> together with the setter and getter functions to access their value
DP-8	New AP	Operation to access the Current Value of a Data Item	See previous adaptation point
DP-9	New AP	Data Pool Refresh Operation	Module <code>CrPsDataPoolFunc</code>
OCM-1	Closes OCM-1	Initialization Check in Initialization Procedure of OutComponent	OutComponents are provided by the OutFactory in the CONFIGURED state and their initialization, configuration, and execution actions and checks are set by OutFactory. This AP is therefore closed in the implementation of <code>CrFwOutFactory.c</code>
OCM-2	Closes OCM-2	Initialization Action in Initialization Procedure of OutComponent	See close-out of OCM-1
OCM-3	Closes OCM-3	Configuration Check in Reset Procedure of OutComponent	See close-out of OCM-1
OCM-4	Closes OCM-4	Configuration Action in Reset Procedure of OutComponent	See close-out of OCM-1
OCM-5	Closes OCM-5	Shutdown Action in Base Component of OutComponent	See close-out of OCM-1
OCM-6	Closes OCM-6	Execution Procedure of OutComponent	See close-out of OCM-1
OCM-7	Closes OCM-7	Service Type Attribute of OutComponent	Service identifiers for supported services are defined in <code>CrPsServTypesId.h</code>
OCM-8	Closes OCM-8	Command/Report Sub-Type Attribute of OutComponent	Sub-service identifiers for supported services are defined in <code>CrPsServTypesId.h</code>

Req. ID	Origin	Description	Implementation
OCM-9	Closes OCM-9	Destination Attribute of OutComponent	Destination is set at the point of instantiation of a report according to the rules defined in the report definition tables
OCM-10	Closes OCM-10	Acknowledge Level Attribute of OutComponent	n.a.
OCM-11	Closes OCM-11	Discriminant Attribute of OutComponent	Destination is set at the point of instantiation of a report according to the rules defined in the report definition tables
OCM-12	Closes OCM-12	Parameter Attribute of OutComponent	Parameters for reports in service Xyz are set using the setter functions generated by the CORDET Editor in CrPsPcktXyz.h
OCM-13	Closes OCM-13	Enable Check Operation of OutComponent	The report-specific checks and actions are defined through CR_FW_OUTCMP_INIT_KIND_DESC in CrFwOutFactoryUserPar.h. This header file is generated by the CORDET Framework to match the definition of the behaviour of each type of report.
OCM-14	Closes OCM-14	Ready Check Operation of OutComponent	See close-out of OCM-13
OCM-15	Closes OCM-15	Repeat Check Operation of OutComponent	See close-out of OCM-13
OCM-16	Closes OCM-16	Update Action of OutComponent	See close-out of OCM-13
OCM-17	Closes OCM-17	Serialize Operation of OutComponent	See close-out of OCM-13
OCM-18	Closes OCM-18	Operation to Report Invalid Destination of an OutComponent	Error reports are generated through calls to functions defined by interface CrFwRepErr.h. The error codes are defined by enumerated type CrFwRepErrCode_t. The interface must be implemented by applications and the enumerated type may be extended by applications.
ICM-1	Closes ICM-1	Initialization Check in Initialization Procedure of InCommand	Incoming commands are provided by the CrFwInfactory in the CONFIGURED state which configures them to have an initialization check which is always successful.

Req. ID	Origin	Description	Implementation
ICM-2	Closes ICM-2	Initialization Action in Initialization Procedure of InCommand	Incoming commands are provided by the CrFwInfactory in the CONFIGURED state which configures them to have an initialization action which does nothing
ICM-3	Closes ICM-3	Configuration Check in Reset Procedure of InCommand	Incoming commands are provided by the CrFwInfactory in the CONFIGURED state which configures them to have a configuration check which verifies the CRC through the CRC-related functions of module CrFwPckt . The check on the CRC implicitly verifies the length of the packet holding the command because the CRC is located at the end of the packet
ICM-4	Closes ICM-4	Configuration Action in Reset Procedure of InCommand	Incoming commands are provided by the CrFwInfactory in the CONFIGURED state which configures them to have a configuration action which does nothing.
ICM-5	Closes ICM-5	Shutdown Action of InCommand	Incoming commands are provided by the CrFwInfactory in the CONFIGURED state which configures them to have a shutdown action which does nothing
ICM-6	Closes ICM-6	Execution Procedure of InCommand	Incoming commands are provided by the CrFwInfactory in the CONFIGURED state which configures them to have an execution action which does nothing
ICM-7	Closes ICM-7	Ready Check of InCommand	The command-specific checks and actions are defined through CR_FW_INCMD_INIT_KIND_DESC in CrFwInFactoryUserPar.h . This header file is generated by the CORDET Framework to match the definition of the behaviour of each type of command.
ICM-8	Closes ICM-8	Start Action of InCommand	See close-out of ICM-7
ICM-9	Closes ICM-9	Progress Action of InCommand	See close-out of ICM-7
ICM-10	Closes ICM-10	Termination Action of InCommand	See close-out of ICM-7
ICM-11	Closes ICM-11	Abort Action of InCommand	See close-out of ICM-7

Req. ID	Origin	Description	Implementation
ICM-12	Closes ICM-12	Operation to Report Start Failed for InCommand	Operation to report outcome of acceptance, start, execution and termination checks are implemented by function <code>CrFwRepInCmdOutcome</code> which generates service 1 reports
ICM-13	Closes ICM-13	Operation to Report Start Successful for InCommand	See close-out of ICM-12
ICM-14	Closes ICM-14	Operation to Report Progress Failed for InCommand	See close-out of ICM-12
ICM-15	Closes ICM-15	Operation to Report Progress Successful for InCommand	See close-out of ICM-12
ICM-16	Closes ICM-16	Operation to Report Termination Failed for InCommand	See close-out of ICM-12
ICM-17	Closes ICM-17	Operation to Report Report Termination Successful for InCommand	See close-out of ICM-12
ICM-18	Closes ICM-18	Service Type Attribute of InCommand	Service identifiers for supported services are defined in <code>CrPsServTypesId.h</code>
ICM-19	Closes ICM-19	Command Sub-Type Attribute of InCommand	Sub-service identifiers for supported services are defined in <code>CrPsServTypesId.h</code>
ICM-20	Closes ICM-20	Discriminant Attribute of InCommand	Discriminants are set at the point of instantiation of a command
ICM-21	Closes ICM-21	Parameter Attributes of InCommand	Parameters for commands in service Xyz are read using the getter functions generated by the CORDET Editor in <code>CrPsPcktXyz.h</code>
S1-1	Closes ILD-12	Operation to Report Packet Destination Invalid by InLoader	The procedure is implemented in module <code>CrPsCmdPrgrFail</code> (generated by FW Profile Editor) and it is run from function <code>CrFwRepInCmdOutcome</code>
S1-2	Closes ILD-14	Operation to Report Acceptance Failure by InLoader	The procedure is implemented in module <code>CrPsCmdReroutingFail</code> (generated by FW Profile Editor) and it is run from function <code>CrFwRepErrInstanceIdAndDest</code>

Req. ID	Origin	Description	Implementation
S1-3	Closes ILD-13	Operation to Report Acceptance Success by InLoader	The procedure is implemented in module CrPsCmdVerSucc (generated by FW Profile Editor) and it is run from function CrFwRepInCmdOutcome
S1-4	Closes ICM-12	Operation to Report Start Failed for InCommand	The procedure is implemented in module CrPsCmdVerFail (generated by FW Profile Editor) and it is run from function CrFwRepInCmdOutcome
S1-5	Closes ICM-13	Operation to Report Start Successful for InCommand	The procedure is implemented in module CrPsCmdVerSucc (generated by FW Profile Editor) and it is run from function CrFwRepInCmdOutcome
S1-6	Closes ICM-14	Operation to Report Progress Failed for InCommand	The procedure is implemented in module CrPsCmdPrgrFail (generated by FW Profile Editor) and it is run from function CrFwRepInCmdOutcome
S1-7	Closes ICM-15	Operation to Report Progress Successful for InCommand	The procedure is implemented in module CrPsCmdPrgrSucc (generated by FW Profile Editor) and it is run from function CrFwRepInCmdOutcome
S1-8	Closes ICM-16	Operation to Report Termination Failed for InCommand	The procedure is implemented in module CrPsCmdVerFail (generated by FW Profile Editor) and it is run from function CrFwRepInCmdOutcome
S1-9	Closes ICM-17	Operation to Report Report Termination Successful for InCommand	The procedure is implemented in module CrPsCmdVerSucc (generated by FW Profile Editor) and it is run from function CrFwRepInCmdOutcome
S3-1	New AP	Definition of Data Pool Parameters and Variables	Applications must create modules like the CrPsDpXyz to implement the new data items
S5-1	New AP	Definition of Event Reports	Applications must expand the CrPsDpEvt module and the CrPsPcktEvt to cover the new event reports
S12-1	New AP	Definition of Parameter Monitoring Procedures	This service is not yet implemented
S12-2	New AP	Definition of Parameter Monitoring Definition List (PMDL)	This service is not yet implemented

Req. ID	Origin	Description	Implementation
S12-3	New AP	Definition of Functional Monitoring Definition List (FMDL)	This service is not yet implemented
S13-1	New AP	Operation to access the i-th LPT Buffer	This service is not yet implemented

4 Requirement Verification

The PUS Extension of the CORDET Framework is specified in reference [PX-SP] through a set of requirements of the following kinds:

- *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.

The Use Constraint Requirements and the Adaptation Requirements are not directly verified. The standard requirements are often formulated in terms of expected behaviour for the commands and reports supported by the PUS Extension. Hence, the following verification evidence is provided for them:

- Section 4.1 traces each standard requirement to its verification evidence
- Section 4.2 traces each command or report supported by the PUS Extension to the test cases which verify it
- Section 4.3 traces the data pool items to the test cases which verify them
- Section ?? traces the command rejection codes to the test cases which verify them
- Section 4.5 traces each FW Profile procedure used by the PUS Extension to the test cases which verify it

4.1 Verification of Standard Requirements

Table 4.2 lists the standard requirements of the PUS Extension and provides for each requirement its verification evidence.

4.2 Verification of Commands and Reports

The PUS Extension defines the behaviour of the commands and the reports supported by the PUS Extension. The PUS Extension is built on the CORDET Framework. In the CORDET world, 'command' and 'reports' are defined through: (a) their *parameters*; (b) the *actions* they perform during their execution; and (c) the *conditional checks* which determine whether and when these actions are executed.

The CORDET Framework defines the life-cycle of a generic command or report in terms of these parameters, actions and checks. The PUS Extension defines concrete commands and reports by specifying the name and type of their parameters and the content of their actions and checks. This is done through tables like the one shown in table 4.1.

Table 4.1: Specification of HkCreHkCmd Component

Name	HkCreHkCmd(3,1)
Description	Create a housekeeping report structure
Parameters	SID, collection interval and identifiers of parameters of the report to be created
Discriminant	The structure identifier (SID) of the packet to be created
Ready Check	Default implementation (command is always ready)
Start Action	Run the procedure Start Action of HkCreate Command of figure 8.1
Progress Action	<p>The following actions are performed:</p> <ul style="list-style-type: none"> (a) The definition of the new report is added to the RDL (b) The destination of the new report is set equal to the source of the present command (c) The enabled status of the new report is set to 'disabled' (d) The cycle counter of the new report is set to zero (e) The newly created housekeeping report is loaded in the OutLoader (f) The action outcome is set to 'success' (g) The completion outcome is set to 'completed'
Termination Action	Default implementation (set action outcome to 'success')
Abort Action	Default implementation (set action outcome to 'success')

For all actions and checks, a default implementation is defined at CORDET Framework level. Wherever possible, the definition of a concrete command or report by the PUS Extension makes use of these default implementations. The default implementations are already verified at CORDET Framework level. Hence, Verification at PUS Extension level only needs to cover the non-default actions and checks and the commands or report parameters.

Tables 4.3 and 4.4 list the commands and reports implemented by the PUS Extension and map them to the test case which verify them. In most cases, more than one test case is used to cover each command or report because different test cases cover different aspects of the of the command's or report's behaviour. Note that the guarantee of coverage of all non-default actions and checks is provided by the code coverage analysis in section 6.

The implementation of the functions which access the command and report parameters is generated automatically from the description of the command and report layout (see reference [PX-IC]). Verification of correctness is done implicitly through the test cases listed in tables 4.3 and 4.4 which must access the command and report parameters in order to verify their behaviour.

4.3 Verification of Data Pool

Table 4.5 lists the data pool items for which dedicated verification tests are done in the test suite and associates to each data pool item the test case the item is verified. Coverage is not complete but this is acceptable because the functions to access the data pool items are generated automatically from the data pool description.

4.4 Verification of Command Rejection Codes

Table 4.6 lists the command rejections codes for which dedicated verification tests are done in the test suite and associates to each command rejection code the test case the code is verified.

4.5 Verification of FW Profile Procedures

Part of the behaviour of the PUS Extension is specified through FW Profile procedures. Table 4.8 lists the procedures and the test cases where they are verified. All branches of the procedures are covered by the test cases.

4.6 Verification of Design-Level Functions

The PUS Extension requirements define a small number of design-level functions. Table 4.7 lists them and associates them to the test cases which verify them. If a function operates under different conditions, all conditions are verified.

Table 4.2: Traceability of Standard Requirements to Verification Evidence

Req. ID	Requirement Text	Verification
DP-1	The PUS Extension of the CORDET Framework shall provide a Data Pool component implementing the behaviour of the Data Pool State Machine of figure 4.1 in [PX-SP].	The configuraton code for the Data Pool component is generated by the FW Profile editor (see section A for a discussion of its verification). No behaviour is associated by default to the actions and guards in the Data Pool component.
DP-3	In state READY, the Data Pool component shall provide operations to let other components access the current value of its data items	The data pool accessor functions are generated by the CORDET Editor (see section 12 for a discussion of its verification). Table 4.5 lists the data pool items which are directly verified by the test cases.
CRA-1	Components encapsulating a command or a report shall implement all attributes defined for them by the PUS	The layout of the commands and reports is defined in the CORDET Editor and is described in [PX-IC]. Verification of consistency to the PUS is done by review of this document.
CRA-2	Components encapsulating a command or a report shall provide operations to access in read and write mode all their PUS-defined attributes	The parameter accessor functions are generated by the CORDET Editor (see section 12 for a discussion of its verification). They are implicitly verified as part of the test cases which verify the PUS commands and reports.
CRA-3	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	See previous requirement
FAC-1	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	The test cases create at least one instance of each command supported by the PUS Extension (see section 4.2). The commands are created by the InFactory.
FAC-2	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	The test cases create at least one instance of each report supported by the PUS Extension (see section 4.2). The commands are created by the OutFactory.

Req. ID	Requirement Text	Verification
FAC-3	The two factory components shall maintain and make accessible through the data pool the observables listed in table 5.1 in [PX-SP]	Extent of verification of observables is shown in section 4.3
S1-1	The PUS Extension of the CORDET Framework shall provide OutComponents implementing the service 1 reports listed in tables 7.2 to 7.10 in [PX-SP]	All service 1 reports are verified by test (see section 4.2)
S1-3	The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 7.11 in [PX-SP]	Extent of verification of observables is shown in section 4.3
S3-1	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 8.1 in [PX-SP]	The service 3 commands access the RDL. The RDL is therefore verified in the test cases which exercise the service 3 commands (see section 4.2). Dedicated tests are done in test case CrPsHkTestCase1.
S3-2	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	This requirement is not implemented
S3-4	The PUS Extension of the CORDET Framework shall provide OutComponents and InCommands implementing the reports and commands defined in tables 4.1 to 8.19 in [PX-SP]	All service 3 commands and reports are verified by test (see section 4.2)
S3-5	The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables and parameters listed in table 8.21 in [PX-SP]	Extent of verification of observables is shown in section 4.3
S5-1	The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables listed in table 9.10 in [PX-SP]	Extent of verification of observables is shown in section 4.3

Req. ID	Requirement Text	Verification
S5-4	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	See test cases CrPsEvtTestCase3 and CrPsEvtTestCase9
S5-5	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	See test cases CrPsEvtTestCase3 and CrPsEvtTestCase9
S5-6	If the OutComponent retrieval from the OutFactory fails, the Generate Pre-Defined Event Function shall generate an error report of type OUTFACTORY_FAIL	See test cases CrPsEvtTestCase9
S5-7	The PUS Extension of the CORDET Framework shall support the commands and reports specified in tables 9.1 to 9.8 in [PX-SP]	All service 5 commands and reports are verified by test (see section 4.2)
S5-8	The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables and parameters listed in table 9.10 in [PX-SP]	Extent of verification of observables is shown in section 4.3
S11-1	The PUS Extension of the CORDET Framework shall implement a Time-Based Schedule (TBS) consisting of SCD_N_TBA Time-Based Activities (TBAs) with the attributes defined in table 10.1 in [PX-SP]	
S11-2	The PUS Extension of the CORDET Framework shall provide a Time-Based Execution Procedure implementing the behaviour shown in figure 10.7 in [PX-SP]	
S11-6	The PUS Extension of the CORDET Framework shall support the commands and reports specified in tables 10.2 to 10.14	

Req. ID	Requirement Text	Verification
S12-1	The PUS Extension of the CORDET Framework shall provide a Monitoring Function Procedure implementing the behaviour shown in figure 11.1	
S12-2	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	
S12-3	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	
S12-4	The PUS Extension of the CORDET Framework shall provide the Limit Check Monitoring Procedure implementing the behaviour shown in figure 11.2	
S12-5	The PUS Extension of the CORDET Framework shall provide the Expected Value Monitoring Procedure implementing the behaviour shown in figure 11.3	
S12-6	The PUS Extension of the CORDET Framework shall provide the Delta Value Monitoring Procedure implementing the behaviour shown in figure 11.4	
S12-7	The PUS Extension of the CORDET Framework shall provide the Process CTL Procedure implementing the behaviour shown in figure 11.5	
S12-8	The PUS Extension of the CORDET Framework shall provide the Functional Monitor Notification Procedure implementing the behaviour shown in figure 11.6	
S12-14	The PUS Extension of the CORDET Framework shall support the commands and reports specified in tables 11.6 to 11.33	

Req. ID	Requirement Text	Verification
S13-1	The PUS Extension of the CORDET Framework shall support the commands and reports specified in tables 12.1 to 12.9	
S13-2	The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables and parameters listed in table 12.11 in [PX-SP]	
S17-1	The PUS Extension of the CORDET Framework shall support the commands and reports specified in tables 13.1 to 13.4 in [PX-SP]	All service 17 commands and reports are verified by test (see section 4.2)
S17-3	The PUS Extension of the CORDET Framework shall maintain and make accessible through the data pool the observables and parameters listed in table 13.6 in [PX-SP]	Extent of verification of observables is shown in section 4.3

Table 4.3: Verification of Commands

Command	Verified Item	Test Case
Are-You-Alive	Start Action Nominal Branch	CrPsTstTestCase2
	Start Action Failure Branch	CrPsTstTestCase2
	Progress Action	CrPsTstTestCase2
Connection Test	Start Action Illegal Destination Branch	CrPsTstTestCase3
	Start Action Nominal Branches	CrPsTstTestCase3
	Progress Action Left Branch	CrPsTstTestCase3
	Progress Action Middle Branch	CrPsTstTestCase3
	Start Action Retrieval of Second OutComponent Fails	CrPsTstTestCase4
	Start Action Retrieval of First OutComponent Fails	CrPsTstTestCase4
	Progress Action right Branch (failure)	CrPsTstTestCase4
Create Diagnostic Report	Successful execution	CrPsHkTestCase10
Create HK	Start Failure with error code VER_FULL_RDL	CrPsHkTestCase2

Command	Verified Item	Test Case
	Start Failure with error code VER_ILL_SID	CrPsHkTest Case2
	Start Failure with error code VER_ILL_DP_ID	CrPsHkTest Case2
	Start Failure with error code VER_ILL_NID	CrPsHkTest Case2
	Start Failure with error code VER_SID_IN_USE	CrPsHkTest Case2
	Successful execution	CrPsHkTest Case3
Delete Diagnostic Report Structure	SID is loaded and disabled	CrPsHkTest Case6
Delete HK Report Structure	SID is not loaded in the RDL	CrPsHkTest Case6
	SID is illegal	CrPsHkTest Case6
	SID is enabled	CrPsHkTest Case6
	SID is loaded and disabled	CrPsHkTest Case6
Disable Generation of Event Identifiers	Multiple legal event identifiers	CrPsEvtTest Case6
	Mixed legal and illegal event identifiers	CrPsEvtTest Case7
Disable Generation of a Diagnostic Report Structure	One loaded SID	CrPsHkTest Case5
Disable Generation of a HK Report Structure	All SIDs are not loaded in the RDL	CrPsHkTest Case5
	Some SIDs are loaded in the RDL and others are illegal	CrPsHkTest Case5
	All SIDs are loaded in the RDL	CrPsHkTest Case5
Enable Generation of Event Identifiers	Multiple legal event identifiers	CrPsEvtTest Case6
	Mixed legal and illegal event identifiers	CrPsEvtTest Case7
Enable Generation of a Diagnostic Report Structure	One loaded SID	CrPsHkTest Case4
Enable Generation of a HK Report Structure	All SIDs are not loaded in the RDL	CrPsHkTest Case4
	Some SIDs are loaded in the RDL and others are illegal	CrPsHkTest Case4
	All SIDs are loaded in the RDL	CrPsHkTest Case4
Generate One-Shot Report for HK Parameters	SID is loaded in the RDL	CrPsHkTest Case8
	SID is not loaded in the RDL	CrPsHkTest Case8
	SID is out-of-range	CrPsHkTest Case8
Hk Config	Operation to set the define report in the RDL	CrPsHkTest Case1
	Operation to clear a SID from the RDL	CrPsHkTest Case1

Command	Verified Item	Test Case
Housekeeping Parameter Report Structure Report		CrPsHkTestCase9
Modify Diagnostic Report Collection Interval	SID is loaded in the RDL	CrPsHkTestCase7
Modify HK Report Collection Interval	SID is loaded in the RDL	CrPsHkTestCase7
	SID is not loaded in the RDL	CrPsHkTestCase7
	SID is out-of-range	CrPsHkTestCase7
Report Diagnostic Parameter Report Structure	SID is loaded in the RDL	CrPsHkTestCase12
Report Disabled Event Identifiers	Nominal case	CrPsEvtTestCase8

Table 4.4: Verification of Reports

Report	Verified Item	Test Case
		CrPsTstTestCase2
Diagnostic	Execution while enabled with collection period equal to 1	CrPsHkTest Case10
Diagnostic Parameter Report Structure		CrPsHkTest Case12
Disabled Event Identifiers	Case of single response report	CrPsEvtTest Case8
	Case of multiple response reports	CrPsEvtTest Case8
	Case of full OutFactory	CrPsEvtTest Case9
Event Severity 1	Nominal case	CrPsEvtTest Case2
Event Severity 2	Nominal case	CrPsEvtTest Case3
Event Severity 3	Nominal case	CrPsEvtTest Case4
Event Severity 4	Nominal case	CrPsEvtTest Case5
Failed Acceptance Verification		CrPsVerTest Case4
Failed Progress of Execution Verification		CrPsVerTest Case4
Failed Routing Verification		CrPsVerTest Case2
Failed Start of Execution Verification		CrPsVerTest Case4
Failed Termination of Execution Verification		CrPsVerTest Case4
Hk	Execution while disabled	CrPsHkTest Case3
	Execution while enabled with collection period equal to 1	CrPsHkTest Case3
	Execution while enabled with collection period greater than 1	CrPsHkTest Case3
	Execution while enabled with collection period equal to zero	CrPsHkTest Case3
Housekeeping Parameter Report Structure	SID is loaded in the RDL	CrPsHkTest Case9
	SID is illegal	CrPsHkTest Case9
Start Failure	Failure Code VER_REP_CR_FD	CrPsTstTest Case3
	Failure Code VER_REP_CR_FD	CrPsTstTest Case4
Successful Acceptance Verification Report		CrPsVerTest Case3

Report	Verified Item	Test Case
Successful Progress of Execution Verification Report		CrPsVerTestCase3
Successful Start of Execution Verification Report		CrPsVerTestCase3
Successful Termination of Execution Verification Report		CrPsVerTestCase3
Termination Failure	Failure Code VER_TST_TO	CrPsTstTestCase4

Table 4.5: Verification of Data Pool Items

Data Pool Item	Test Case
EvtLastEvtEid	CrPsEvtTestCase2
EvtLastEvtEid_2	CrPsEvtTestCase3
EvtLastEvtEid_3	CrPsEvtTestCase4
EvtLastEvtEid_4	CrPsEvtTestCase5
EvtLastEvtTime	CrPsEvtTestCase2
EvtLastEvtTime_2	CrPsEvtTestCase3
EvtLastEvtTime_3	CrPsEvtTestCase4
EvtLastEvtTime_4	CrPsEvtTestCase5
EvtNOOfDetectedEvts	CrPsEvtTestCase2
EvtNOOfDetectedEvts_2	CrPsEvtTestCase3
EvtNOOfDetectedEvts_3	CrPsEvtTestCase4
EvtNOOfDetectedEvts_4	CrPsEvtTestCase5
HkCycleCnt	CrPsHkTestCase8
HkIsEnabled	CrPsHkTestCase3
HkPeriod	CrPsHkTestCase7
areYouAliveSrc	CrPsTstTestCase1
areYouAliveTimeOut	CrPsTstTestCase1
	CrPsTstTestCase4
failCode	CrPsVerTestCase4
failCodePrgrFailed	CrPsTstTestCase3
	CrPsTstTestCase4
failData	CrPsEvtTestCase7
hkDest	CrPsHkTestCase4
invDestRerouting	CrPsVerTestCase2
nOfDisabledEid	CrPsEvtTestCase6
nOfReroutingFailed	CrPsVerTestCase2
onBoardConnectDestLst	CrPsTstTestCase1
pcktIdReroutingFailed	CrPsVerTestCase2

Table 4.6: Verification of Command Rejection Codes

Command Rejection Code	Test Case
VER_ENB_SID	CrPsHkTestCase6
VER_FULL_RDL	CrPsHkTestCase2
VER_ILL_DP_ID	CrPsHkTestCase2
VER_ILL_EID	CrPsEvtTestCase7
VER_ILL_NID	CrPsHkTestCase2
VER_ILL_SID	CrPsHkTestCase2
VER_MI_S3_FD	CrPsHkTestCase4
	CrPsHkTestCase6
	CrPsHkTestCase8
VER_REP_CR_FD	CrPsTstTestCase3

Command Rejection Code	Test Case
VER_SID_IN_USE	CrPsHkTestCase2

Table 4.7: Verification of Design-Level Functions

Function	Verified Item	Test Case
Evt Config	Operation to retrieve position and severity level of event identifier	CrPsEvtTestCase1
	Operation to set the enable status of an event identifier	CrPsEvtTestCase1
	Operation to read the enable status of an event identifier	CrPsEvtTestCase1
	Operation to read the severity level of an event identifier	CrPsEvtTestCase1
Generate Pre-Defined Event	Nominal case	CrPsEvtTestCase3
	Nominal case	CrPsEvtTestCase5
	Parameterless event OUTFACTORY_FAIL error case	CrPsEvtTestCase9

Table 4.8: Verification of Design-Level Procedures

Function	Verified Item	Test Case
Command Progress Failed	Nominal branch	CrPsVerTestCase4
Command Progress Failure	OUTFACTORY_FAIL branch	CrPsVerTestCase5
Command Progress Success	Nominal branches	CrPsVerTestCase3
	OUTFACTORY_FAIL branch	CrPsVerTestCase5
Command Verification Failed	Nominal branch	CrPsVerTestCase4
	OUTFACTORY_FAIL branch	CrPsVerTestCase5
Command Verification Success	Nominal branches	CrPsVerTestCase3
	OUTFACTORY_FAIL branch	CrPsVerTestCase5
Packet Rerouting Failure	Report handling branch	CrPsVerTestCase2
	Nominal command handling branch	CrPsVerTestCase2
	Non-Nominal command handling branch	CrPsVerTestCase2

5 Static Code Analysis

Two forms of static code analysis were done:

- Compilation of the PUS Extension of the CORDET Framework with the `clang` compiler with all warnings enabled
- Compilation with the `clang` static analyzer (`scan-build`)

The `clang` compiler does not report any errors.

The `scan-build` reports no errors in the PUS Extension code.

Based on the above, it is concluded that no statically detectable issues are present in the code of the PUS Extension of the CORDET Framework.

6 Code Coverage

Code coverage is measured using the `gcov` tool. The main page of the Doxygen documentation for the PUS Extension of the CORDET Framework gives access to the coverage web page generated by the `lcov` tool. For convenience, figure 6.1 shows the summary report from the `lcov` tool. The coverage information was derived by running the Test Suite (see section 3.5 in reference [PX-UM]).

In order to facilitate the assessment of branch coverage information, the following was done:

- The library source code is compiled with no optimization (compiler option `-O0`)
- The conditions in `if-clauses` consist of primitive boolean conditions. This means that condition coverage becomes equivalent to decision coverage.

Note that the code in the `DataPool` and `Pckt` subdirectory is generated automatically by the CORDET Editor and consists mostly of getter and setter functions.

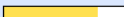
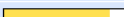
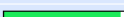

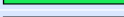
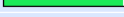
LCOV - code coverage report

Current view: **top level**

Test: **filtered_coverage.info**

Date: **2019-05-02 23:20:51**

	Hit	Total	Coverage
Lines:	1736	1865	93.1 %
Functions:	263	272	96.7 %
Branches:	234	293	79.9 %

Directory	Line Coverage ↕	Functions ↕	Branches ↕
DataPool	 77.7 % 153 / 197	89.8 % 53 / 59	54.2 % 13 / 24
Evt	 87.7 % 264 / 301	89.5 % 17 / 19	77.5 % 100 / 129
Hk	 97.4 % 262 / 269	100.0 % 20 / 20	95.1 % 78 / 82
Pckt	 100.0 % 473 / 473	100.0 % 93 / 93	- 0 / 0
Tst	 98.6 % 212 / 215	100.0 % 32 / 32	85.0 % 17 / 20
Ver	 90.7 % 372 / 410	98.0 % 48 / 49	68.4 % 26 / 38

Generated by: [LCOV version 1.12](#)

Fig. 6.1: Summary Coverage Report of lcov Tool

7 Unit-Level Tests

The unit level tests are implemented in the Test Suite described in section 3.5 of reference [PX-UM]. The outcome of running the entire test suite is in the log file `/log/TestSuiteRun.log`. Inspection of this log file shows that all test cases in the test suite have been successfully passed.

8 Comment Coverage

The code of the PUS Extension of the CORDET Framework is documented using doxygen comments. The verification of the coverage of these comments is done as follows:

- Doxygen is configured to issue a warning in case any coding item is not documented
- It is verified that warnings issue by Doxygen are justified

The outcome of this verification activity can be checked in file `/logs/Doxygen.out` in the delivery file which holds the output of the execution of Doxygen. The following warnings are issued by Doxygen:

- Warning "...too many nodes. Consider increasing DOT_GRAPH_MAX_NODES"
- Warning for undocumented members for functions `CrPsTestOnBoardConnectionPrgrN1` and `CrPsHkConfigGetParamId`
- Warning "return type of member `__attribute__` is not documented"

The first and third warnings are accepted. The reason for the second warning is instead not understood.

9 Memory Management

Verification of absence of memory leaks is done at two levels:

- Statically, the only statements resulting in an allocation of memory from the heap are those where FW Profile state machine or procedures are created but these objects are only created during the application initialization phase (in the `config` modules) and are never destroyed afterwards
- Dynamically, `Valgrind` reports no memory leaks during the execution of the Test Suite for the PUS Extension of the CORDET Framework. The outcome of running the entire test suite is in the log file `/log/TestSuiteRunWithValgrind.log`.

Based on these considerations, it is concluded that there are no memory leaks in the PUS Extension of the CORDET Framework.

No measurement has been made of stack usage but it is stressed that the PUS Extension of the CORDET Framework does not make any recursive function calls¹.

¹Recursion is used in the FW Profile Library but, in that case, the depth of recursion is the same as the depth of nesting of state machines.

10 Schedulability

The PUS Extension of the CORDET Framework is not a self-standing executable. Hence, it does not make sense to ask whether the framework is schedulable *per se*. The more relevant question is: does the library support the analysis of the schedulability of the application within which it is embedded? This question can be split into the following sub-questions:

- Is the threading model implemented by the framework compatible with application-level schedulability analysis?
- Is the execution time of framework code well characterized?

The answer to the first question is provided in section 21 of the User Manual of the CORDET Framework.

The answer to the second question would require timing measurements to be done on the framework code. No such measurements have yet been done. In any case, these measurements would have to be done on the final target where the application instantiated from the framework runs.

11 Metrics

The first sub-section presents the software metrics. The second sub-section presents the process metrics.

11.1 Software Metrics

Software metrics were computed with the aim of measuring the complexity of the code. The tool used to compute the software metrics was `lizard` version 1.14.10. Its output is in the log file `Metrics.log` in the `log` directory of the delivery file.

Table 11.1 lists the software metrics and their values. The following considerations apply:

- Metrics SM1 to SM5 refer to the PUS Extension code excluding the CORDET Framework and FW Profile code
- Metrics SM2.1 to SM2.4 refer to the the PUS Extension code excluding the CORDET Framework and FW Profile code and excluding the code generated by the CORDET Editor
- The only function with CCN greater than 12 is `CrPsEvtConfigIter` which has a CCN of 18.

Table 11.1: Software Metrics

ID	Metrics Name	Value
SM1	Number of Lines of Code	8779
SM2	Number of Functions	1152
SM3	Average LOC per Function	6.2
SM4	Average Cyclomatic Complexity Number (CCN)	1.1
SM5	Number of Functions with CCN greater than 12	1
SM2.1	Number of Lines of Non-Editor Code	3420
SM2.2	Number of Non-Editor Functions	222
SM2.3	Average LOC per Non-Editor Function	10.2
SM2.4	Average Cyclomatic Complexity Number (CCN) for Non-Editor code	1.6

11.2 Process Metrics

The process metrics tracked for this project are listed in table 11.2.

Table 11.2: Process Metrics

ID	Metrics Name	Value
PM1	Number of Open Issues (see section 2.2)	n.a.
PM2	Number of Non-Conformances since Release 1.0 (see section ??)	n.a.
PM3	Number of Requirement Changes since Release 1.0 (see section ??)	n.a.

The process metrics will only be collected after release 1.0 of the framework

12 Automatically Generated Code

Two forms of code generations are used in the PUS Extension of the CORDET Framework:

- Part of the specification of the PUS Extension of the CORDET Framework is expressed through state machines and procedures compliant with the FW Profile. Their implementation is generated by the FW Profile Editor.
- The layout of the commands and reports supported by the PUS Extension is defined in the CORDET Editor which generates the functions to access the command and report parameters.

These two forms of code generations are discussed in the next two sub-sections.

12.1 Generation by the FW Profile Editor

The state machines and procedures used to specify the PUS Extension in reference [PX-SP] are built within the *FW Profile Editor*. The FW Profile Editor is a web-based tool which supports the definition of state machines and procedures compliant with the FW Profile. The editor includes an auto-coding back-end which generates the C-code which creates and configures the state machines and procedures. The auto-generated modules in the PUS Extension of the CORDET Framework are those whose author in the doxygen comment at the top of the module is: "FW Profile code generator". Appendix A gives an overview of the FW Profile Editor.

Use of an auto-coding approach has the following advantages:

- Guarantee of consistency between specification and implementation
- Faster turn-around time
- Reduced verification effort

The first advantage follows from the fact that the implementation is directly and automatically derived from the specification. The second advantage follows from the fact that code is generated automatically which is faster than manual development. The gains in verification effort follow from the fact that generated code can be tested more quickly than manually crafted code (because errors are systematic and hence only coding patterns need to be verified).

The software implementing the state machines and procedures consist of two parts:

1. The software in the FW Profile Library which implements the generic state machine and procedure behaviour; and
2. The software generated by the FW Profile Editor implementing the instantiation and configuration code for a state machine or procedure.

The verification approach for the first item is discussed in section 13. For the second item the following considerations apply:

1. The generated code consists of a linear (no branching) sequence of functions calls; hence one single test provides full coverage of the entire generated code.
2. The functions called by the generated code are configuration functions provided by the FW Profile Library and the order in which they are called is either immaterial or, if it is subject to constraints, their violation leads to errors detected by a built-in

consistency check (see next bullet).

3. The FW Profile Library provides functions `FwSmCheckRec` and `FwPrCheck` which perform extensive consistency checks on the configuration of, respectively, a state machine or a procedure; these functions detect errors such as omission of a configuration action or execution of a configuration action with invalid parameters. These functions are called as part of the start-up of the PUS Extension of the CORDET Framework (TBC).
4. Implementation errors in the code generation process are likely to result in systematic coding errors triggering major errors in the test suite for the PUS Extension of the CORDET Framework.
5. The test suite for the PUS Extension of the CORDET Framework (see section 7) covers every state and every transition of every state machine and every node and every branch of every procedure (see section 4). Thus, any configuration errors in a state machine or procedure would be caught by the test suite.

Based on the above, it is concluded that the automatically generated code implementing the configuration of the state machines and procedures of the PUS Extension of the CORDET Framework can be considered to be adequately verified.

12.2 Generation by the CORDET Editor

The layout of the commands and reports supported by the PUS Extension of the CORDET Framework is defined in the CORDET Editor. The CORDET Editor is a proprietary web-based tool which supports the definition of PUS-compliant applications. The editor includes an auto-coding back-end which generates the following implementation-level artifacts:

- The modules implementing the data pool (modules in directory `src/DataPool`)
- The modules implementing the functions to access the parameters in the supported commands and reports (modules in directory `src/Pckt`)

Use of the auto-coding approach has the same advantages as in the case of the FW Profile Editor. The verification approach for this auto-generated code is as follows:

- The generated code consists of linear (no branching) sequences of statements; hence, for a given function, one single test provides full coverage of the entire generated code.
- The Test Suite of the PUS Extension has a very high coverage of the generated code (see section 6)
- Implementation errors in the code generation process are likely to result in systematic coding errors triggering major errors in the test suite for the PUS Extension of the CORDET Framework.

Based on the above, it is concluded that the code generated by the CORDET Editor can be considered to be adequately verified.

13 Qualification Status of External Libraries

The PUS Extension of the CORDET Framework uses two external libraries:

- The CORDET Framework library from reference [CR-SP]
- The FW Profile Library from reference [FW-SP]

Both libraries are publicly available under the Mozilla Public Licence v2. This licence allows their code to be embedded within the PUS Extension of the CORDET Framework code without imposing any requirements on the latter. In fact, the full code of the two libraries is included in the delivery file of the PUS Extension of the CORDET Framework.

Both libraries are provided with an own qualification data package (downloadable from references [CR-SP] and [FW-SP]) which covers the following items:

- A set of **Software Requirements** which formally specify the implementation
- A set of **Behavioural Models** which describe the behaviour of the library components
- An **Implementation Traceability Matrix** which shows how each requirement is implemented
- A **Verification Traceability Matrix** which shows how the implementation of each requirement is verified
- A **Validation Traceability Matrix** which justifies each requirement with respect to the intended use of the library
- A **Test Suite** with 100% statement, function, branch, and condition coverage for the entire library implementation (excluding error branches for system calls)
- **Doxygen Documentation** covering the entire library source code
- A **User Manual** which explains how the library should be used

It is also noted that the FW Profile Library has been successfully used in several industrial project and the CORDET Framework library has been successfully used in at least one on-board software project.

Based on the above evidence, it is concluded that the FW Profile Library and the CORDET Framework library have an adequate quality for use within the PUS Extension of the CORDET Framework.

A FW Profile Editor and Code Generator

This appendix describes the FW Profile Editor which is used to define the state machines and procedures which specify the behaviour of the PUS Extension of the CORDET Framework and to generate the code of the functions implementing this behaviour.

A.1 Overview of FW Profile Editor

The FW Profile Editor allows a user to draw a state machine or a procedure (activity diagram) in a web-based interface accessible from [FwProfEd]. The semantics of the state machines and procedures is that of the FW Profile of reference [FwProf]. Figure A.1 shows a screenshot of the FW Profile Editor.

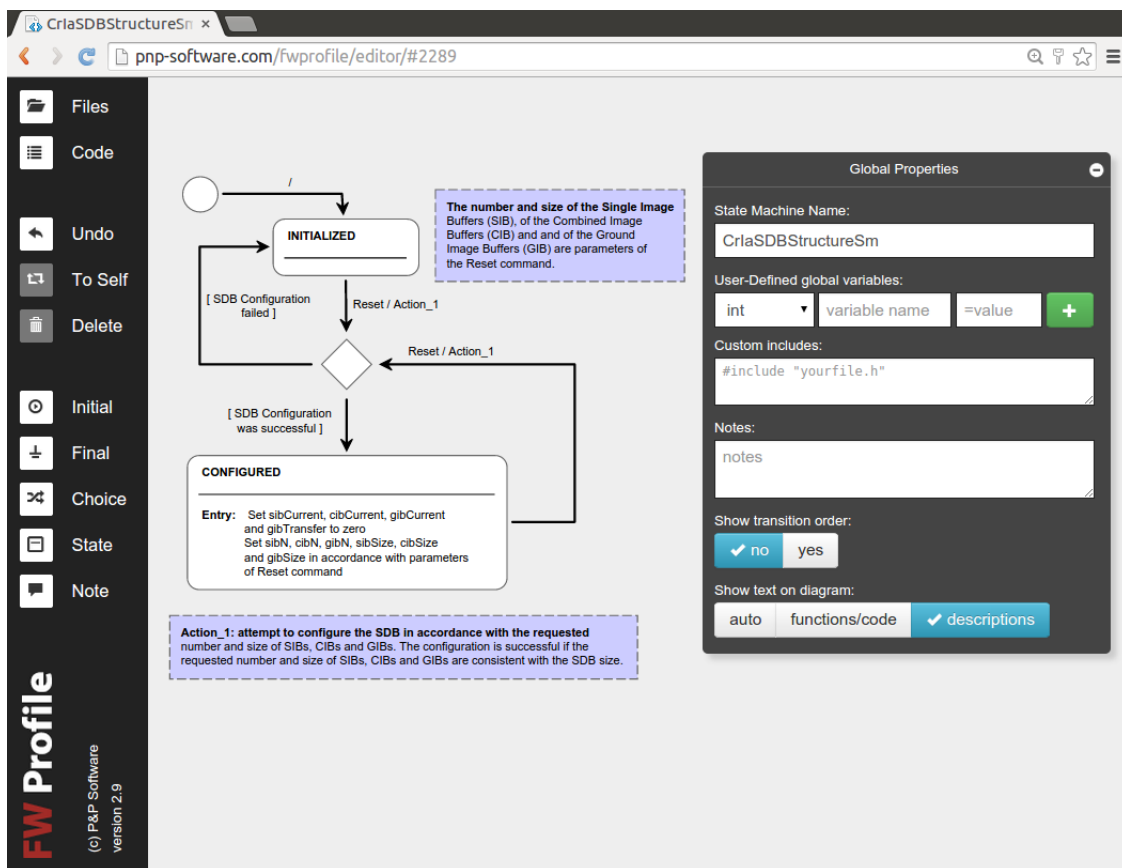


Fig. A.1: Screenshot of the Framework Profile Editor

For each state machine or procedures, the FW Profile Editor offers two "views": the "description view" and the "code view". Users can change between the code view and the description view using a switch in the web-based interface of the editor (see box at the bottom of the "Global Properties" palette in figure A.1).

The two views are illustrated in figures A.2 and A.3 through the example of the Out Command State Machine. The description view is used as a specification in the USP of the PUS Extension of the CORDET Framework (see reference [TsUsp]). The actions and guards of the state machine are described through a specification of what they should do. The code

view associates code to the actions and guards in the state machine. The code may either take the form of a function name or it may consist of valid C code which directly implements the function. In the example of figure A.3, the former option. Thus, for instance, `OutCmdPendingEntry` is the name of a function which implements the entry action of state `PENDING` in the `OutCommand` State Machine.

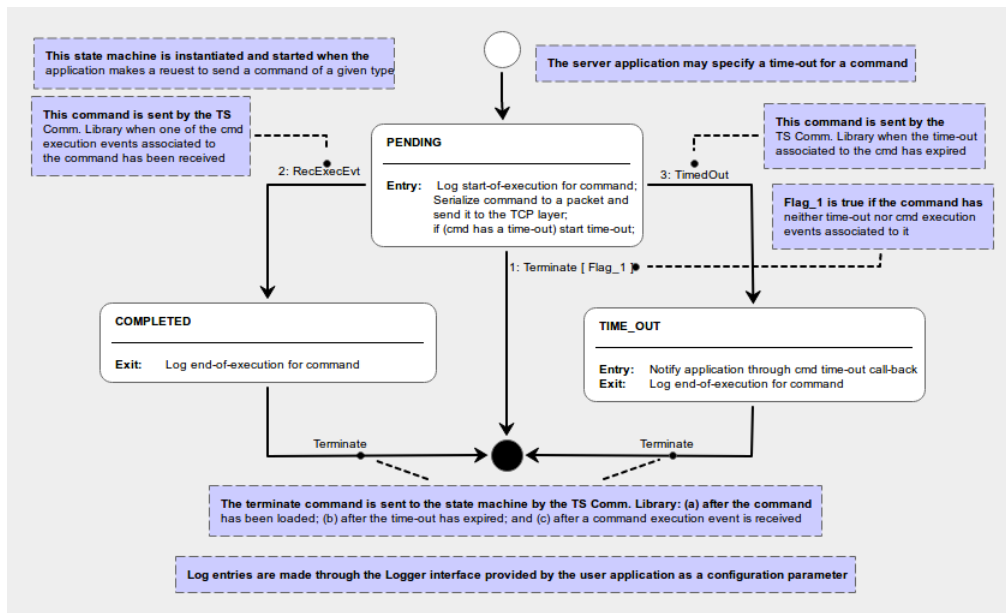


Fig. A.2: Description View OutCommand State Machine

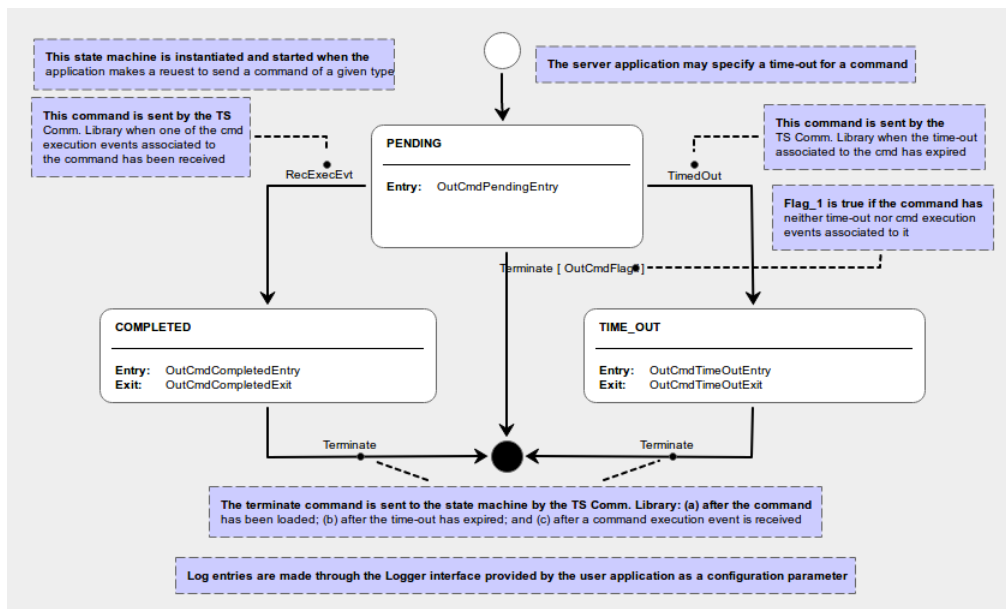


Fig. A.3: Code View of OutCommand State Machine

A.2 Generated Code

The code generated by the FW Profile Tool for each state machine or procedure consists of three files:

- A header file which declares the following two items:
 1. The function to instantiate and configure the state machine or procedure, and
 2. Any additional functions implementing actions or guards of the state machine or procedure.
- A body file which implements the function which instantiates and configures the state machine or procedure.
- A dummy `main` program which instantiates and configures the state machine or procedure and sends a few commands to it. This main program is only intended for demonstration/testing purposes and is not integrated with the application code.

As noted in the previous section, the FW Profile Editor also gives users the option to define the implementation of an action or guard by directly entering the implementing code through the tool interface. In that case, the tool also generates a private (`static`) function with the code provided by the user.

The code generated by the FW Profile Editor must be linked with the FW Profile Library (the necessary `#include` statements are automatically generated). The library consists of a set of C modules which implement the behaviour of a generic state machine or procedure. Use of this library means that the code generated by the editor only needs to implement the configuration of a state machine or procedure (as opposed to implementing the state machine transition logic or the procedure execution logic).

For state machines, the configuration code consists of a linear sequence of function calls which "build" the state machine by adding states and transitions and attaching them their attributes (the state and transition actions and the transition guards). Similarly, the configuration of a procedure consists of a sequence of function calls which build the procedure by adding nodes and control flows and to it and attaching them their attributes (the node actions and the control flow guards).

A.3 Use of Generated Code

With the PUS Extension of the CORDET Framework, the auto-generated code is used to create and configure the state machines and procedures. The code must be linked with the manually written code of the PUS Extension of the CORDET Framework but does not need to manually modified in any way.

The PUS Extension of the CORDET Framework code calls the `Create` function when it needs to instantiate a state machine or procedure. The `Create` function returns one instance of the state machine or procedure which is fully configured and ready to be used. No further interaction with the generated code is necessary.

A.4 Summary of Mode of Use of FW Profile Editor

The FW Profile Editor and its code generator are used as follows in the PUS Extension of the CORDET Framework:

1. Specification of state machines and procedures using the "description view" of the FW

Profile Editor.

2. Definition of "code view" for the state machines and procedures within the FW Profile Editor. For each action or guard with non-default implementation, a function is defined and its name is entered in the tool.
3. Generation of code using the FW Profile Editor code generator. For each state machine or procedure, two files are generated (one header file and one body file).
4. For each state machine or procedure, creation of a body file with the implementation of the functions implementing the non-default actions or guards of the state machine or procedure.