CORDET Framework - PUS Extension
Software User Manual

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

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

Written By:	Alessandro Pasetti
Checked By:	n.a.
Document Ref.:	PP-UM-PUX-0001
Issue:	0.2
Created On:	20/05/2019, at: 22:40

Contents

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-VR]	The PUS Extension of the CORDET Framework – Verification Report	PP-RP-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 user manual for the PUS Extension of the CORDET Framework. The PUS Extension of the CORDET Framework is aimed at on-board satellite applications. It provides an implementation of a subset of the PUS services defined in reference [PS-SP]. The currently supported services are listed in table TBD.

The PUS Extension of the CORDET Framework is built as an instantiation of the CORDET Framework of reference [CR-SP]. Readers of this user manual are expected to be familiar with the documentation of the CORDET Framework.

The PUS Extension of the CORDET Framework supports the development of PUS-compliant on-board applications. Figure 2.1 illustrates the structure of such an application. The core of the application is the CORDET Framework which implements the management of in- and out-going commands and reports. The CORDET Framework has two main sets of interfaces:

- An interface towards the underlying middleware which carries raw commands and reports as byte packets. This interface is encapsulated in framework components `InStream` and `OutStream`. Each such component manages one command/report destination or source.
- An interface towards the application to implement concrete commands and reports. Incoming commands are implemented in instances of framework component `InCommand` and out-going reports are implemented in instances of framwork component `OutComponent`.

The CORDET Framework is built on an abstract model of commands and reports and therefore it is independent of the specific actions which a command executes or of the specific data which a report carries. The PUS Extension extends the CORDET Framework by providing concrete implementations for the commands and reports of the PUS services listed in table 2.1.

Reference [PX-SP] describes and specifies the PUS Extension. Its software has undergone an extensive qualification programme whose outcome is documented in reference [PX-VR]. A TM/TC ICD is available in reference [PX-IC] documenting the commanding and reporting interface used to test the PUS Extension.

This commanding and reporting was defined through the CORDET Editor. The CORDET Editor is a proprietary tool of P&P Software GmbH to model PUS-based applications. The CORDET Editor includes a code-generating front-end which generates C-code to implement the data pool of a PUS application and the functions to access the parameters of its commands and reports. Part of the code in the Delivery File of the PUS Extension of the CORDET Framework was generated by the CORDET Editor. Users will normally have to replace this code with code implementing their own commanding and reporting interface.

The CORDET Framework is flight-proven having been used for the payload software of the CHEOPS satellite and of the SMILE satellite. Its PUS Extension is currently a beta version.

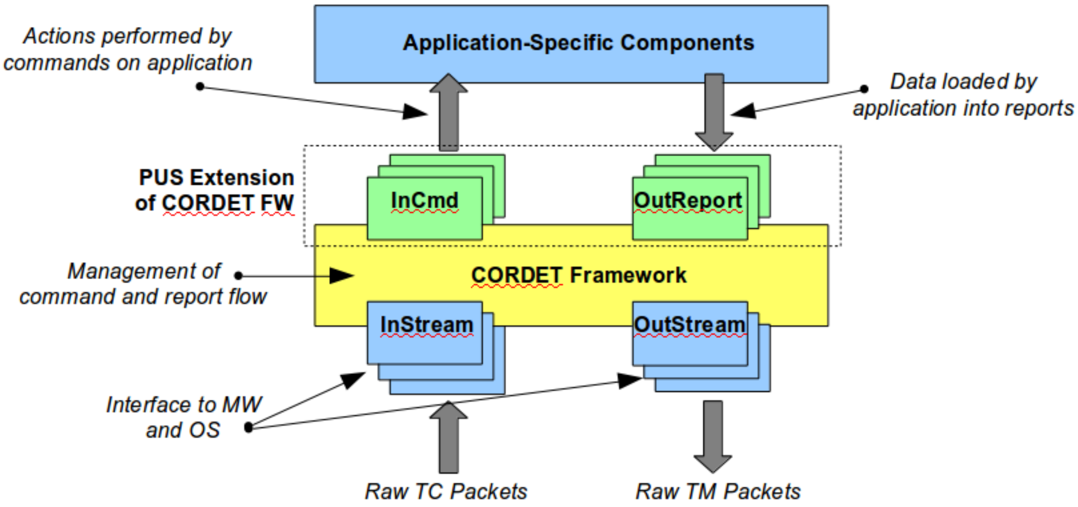
Actions performed by commands on application
Application-Specific Components
Data loaded by application into reports
PUS Extension of CORDET FW
InCmd
OutReport
Management of command and report flow
CORDET Framework
InStream
OutStream
Interface to MW and OS
Raw TC Packets
Raw TM Packets

Fig. 2.1: Overview of PUS Extension Application

Table 2.1: Service Development Status

N	Service Name	Status
1	Request Verification Service	Supported in full
3	Housekeeping Service	Supported as per table 6.6
5	Event Reporting Service	Supported in full
11	Time-Based Scheduling	Under development
12	On-Board Monitoring Service	Under development
13	Large Packet Transfer Service	Under development
17	Test Service	Supported in full

Table 2.2: List of Supported Commands/Reports for Service 1

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

Type	CORDET Name	PUS Name
TC(3,31)	ModPerHkCmd	Modify Collection Interval of Housekeeping Report Structure
TC(3,32)	ModPerDiagCmd	Modify Collection Interval of Diagnostic Report Structure
TM(5,1)	Rep1	Informative Event Report (Level 1)
TM(5,2)	Rep2	Low Severity Event Report (Level 2)
TM(5,3)	Rep3	Medium Severity Event Report (Level 3)
TM(5,4)	Rep4	High Severity Event Report (Level 4)
TC(5,5)	EnbCmd	Enable Generation of Event Identifiers
TC(5,6)	DisCmd	Disable Generation of Event Identifiers
TC(5,7)	RepDisCmd	Report the List of Disabled Event Identifiers
TM(5,8)	DisRep	Disabled Event Identifier Report
TC(11,1)	EnbTbsCmd	Enable Time-Based Schedule Execution Function
TC(11,2)	DisTbsCmd	Disable Time-Based Schedule Execution Function
TC(11,3)	ResTbsCmd	Reset Time-Based Schedule
TC(11,4)	InsTbaCmd	Insert Activities into Time-Based Schedule
TC(11,5)	DelTbaCmd	Delete Activities from Time-Based Schedule
TC(11,20)	EnbSubSchedCmd	Enable Time-Based Sub-Schedules
TC(11,21)	DisSubSchedCmd	Disable Time-Based Sub-Schedules
TC(11,22)	CreGrpCmd	Create Time-Based Scheduling Groups
TC(11,23)	DelGrpCmd	Delete Time-Based Scheduling Groups
TC(11,24)	EnbGrpCmd	Enable Time-Based Scheduling Groups
TC(11,25)	DisGrpCmd	Disable Time-Based Scheduling Groups
TC(11,26)	RepGrpCmd	Report Status of Time-Based Scheduling Groups
TM(11,27)	GrpRep	Time-Based Scheduling Group Status Report
TC(12,1)	EnbParMonDefCmd	Enable Parameter Monitoring Definitions
TC(12,2)	DisParMonDefCmd	Disable Parameter Monitoring Definitions
TC(12,3)	ChgTransDelCmd	Change Maximum Transition Reporting Delay
TC(12,4)	DelAllParMonCmd	Delete All Parameter Monitoring Definitions
TC(12,5)	AddParMonDefCmd	Add Parameter Monitoring Definitions
TC(12,6)	DelParMonDefCmd	Delete Parameter Monitoring Definitions
TC(12,7)	ModParMonDefCmd	Modify Parameter Monitoring Definitions
TC(12,8)	RepParMonDefCmd	Report Parameter Monitoring Definitions
TM(12,9)	RepParMonDefRep	Parameter Monitoring Definition Report
TC(12,10)	RepOutOfLimitsCmd	Report Out Of Limit Monitors
TM(12,11)	RepOutOfLimitsRep	Out Of Limit Monitors Report
TM(12,12)	CheckTransRep	Check Transition Report
TC(12,13)	RepParMonStatCmd	Report Status of Parameter Monitors
TM(12,14)	RepParMonStatRep	Parameter Monitor Status Report

Type	CORDET Name	PUS Name
TC(12,15)	EnbParMonFuncCmd	Enable Parameter Monitoring Function
TC(12,16)	DisParMonFuncCmd	Disable Parameter Monitoring Function
TC(12,17)	EnbFuncMonCmd	Enable Functional Monitoring Function
TC(12,18)	DisFuncMonCmd	Disable Functional Monitoring Function
TC(12,19)	EnbFuncMonDefCmd	Enable Functional Monitoring Definitions
TC(12,20)	DisFuncMonDefCmd	Disable Functional Monitoring Definitions
TC(12,21)	ProtFuncMonDefCmd	Protect Functional Monitoring Definitions
TC(12,22)	UnprotFuncMonDefCmd	Unprotect Functional Monitoring Definitions
TC(12,23)	AddFuncMonDefCmd	Add Functional Monitoring Definitions
TC(12,24)	DelFuncMonDefCmd	Delete Functional Monitoring Definitions
TC(12,25)	RepFuncMonDefCmd	Report Functional Monitoring Definitions
TM(12,26)	RepFuncMonDefRep	Report Functional Monitoring Definitions
TC(12,27)	RepFuncMonStatCmd	Report Status of Functional Monitors
TM(12,28)	RepFuncMonStatRep	Status of Functional Monitors Report
TM(13,1)	DownFirstRep	First Downlink Part Report
TM(13,2)	DownInterRep	Intermediate Downlink Report
TM(13,3)	DownLastRep	Last Downlink Part Report
TC(13,9)	UpFirstCmd	First Uplink Part
TC(13,10)	UpInterCmd	Intermediate Uplink Part
TC(13,11)	UpLastCmd	Last Uplink Part
TM(13,16)	UpAbortRep	Large Packet Uplink Abortion Report
TC(13,129)	StartDownCmd	Trigger Large Packet Down-Transfer
TC(13,130)	AbortDownCmd	Abort Large Packet Down-Transfer
TC(17,1)	AreYouAliveCmd	Perform Are-You-Alive Connection Test
TM(17,2)	AreYouAliveRep	Are-You-Alive Connection Report
TC(17,3)	ConnectCmd	Perform On-Board Connection Test
TM(17,4)	ConnectRep	On-Board Connection Test Report
TC(255,1)	Sample1	Sample 1 Command

3 Installation & Content Overview

The PUS Extension of the CORDET Framework can be accessed in two forms:

- As one single zip file (the *delivery file*) available from the project web site at `https://www.pnp-software.com/cordetfw/`
- As a public GitHub project at: `https://github.com/pnp-software/cordetfw-pus`

The delivery file is entirely self-contained and includes a complete instantiation of the PUS Extension. The GitHub project, instead, does not contain the code generated from the CORDET Editor model of the PUS Extension. Users should therefore normally start from the Delivery File and this User Manual accordingly focuses on the Delivery File.

The Delivery File should be expanded in a dedicated directory, which becomes the *host directory* for the PUS Extension of the CORDET Framework. Table 3.1 gives an overview of the structure of the host directory. More details are found in subsequent subsections.

The PUS Extension software is delivered as source code and therefore no further installation operations are needed. A Test Suite is provided together with Unix script files to compile and link it.

Table 3.1: Structure of Host Directory

Sub-Dir.	**Sub-Directory Description**
`/doc`	Support documentation. See section 3.3.
`/lib`	External libraries used by the PUS Extension. See section 3.1.
`/log`	Test reports and log files. See section 3.6.
`/src`	Source code for the PUS Extension. See section 3.2.
`/tests`	Source code for the Test Suite. See section 3.2.

3.1 Dependency on External Libraries

The PUS Extension is built as an instantiation of the CORDET Framework. This is explained in sections 2 and 3 of reference [PX-SP].

The CORDET Framework (both its source code and its documentation) is included in the delivery file of the PUS Extension (in directory `lib`).

The behaviour of some of the services supported by the PUS Extension is specified by means of state machines and procedures (activity diagrams). The implementation of the PUS Extension therefore requires an implementation of state machines and procedures. The PUS Extension does not include an own implementation of state machines and procedures. Instead, it uses the state machine and procedure modules of the FW Profile (see reference [FW-SP]).

The FW Profile (both its source code and its documentation) is included in the delivery file of the PUS Extension (in directory `lib/cordetfw/lib`).

Note that the FW Profile, the CORDET Framework and its PUS Extension are published under the same MPL licence.

3.2 Source Code

The source code in the Delivery File covers one instantiation of the PUS Extension for the *Test Suite* (see section 3.5).

At source code level, an instantiation of the PUS Extension of the CORDET Framework can be split into four parts:

- **Invariant Code** consisting of: (a) the implementation of the FW Profile, (b) the implementation of the CORDET Framework, and (c) the implementation of the behaviour of the commands and reports supported by the PUS Extension. This code is common to all instantiations of the PUS Extension.
- **Configurable Code** consisting of the part of the code which must be modified to be adapted to the needs of each end-application (see section 4). This part is customized for each instantiations of the PUS Extension.
- **Application-Specific Code** implementing the application-specific (i.e. non-framework) part of the target application.

The Delivery File contains the invariant code and the configuration and application-specific code for the Test Suite Application. This is an application instantiated from the PUS Extension of the CORDET Framework for testing purposes (see section 3.5).

The source code in the PUS file is split into several sub-directories as presented in table 3.2. Users who wish to build a new application by instantiating the PUS Extension should take the invariant directories without changes and should customize the software in the remaing directories to match their needs. The instantiation process is described in section 4.

Table 3.2: Sub-Directory

Sub-Directory	Sub-Directory Description
`/src/DataPool`	**Configurable Code**: Implementation of the data pool for the Test Suite Application. This code is generated by the CORDET Editor.
`/src/Dum`	**Application-Specific Code**: Implementation of the Dummy Service (a private service used by the Test Suite application).
`/src/Evt`	**Invariant Code**: Implementation of the Event Reporting Service.
`/src/Hk`	**Invariant Code**: Implementation of the Housekeeping Service.
`/src/Lpt`	**Invariant Code**: Implementation of the Large Packet Transfer Service. This code is untested and should not be used.
`/src/Mon`	**Invariant Code**: Implementation of the Monitoring Service. This code is untested and should not be used.
`/src/Pckt`	**Configurable Code**: Implementation of the functions to access parameters in the PUS packets. This code is generated by the CORDET Editor.
`/src/Scd`	**Invariant Code**: Implementation of the Scheduling Service. This code is untested and should not be used.
`/src/Tst`	**Invariant Code**: Implementation of the Test Service.
`/src/Ver`	**Invariant Code**: Implementation of the Command Verification Service.

Sub-Directory	Sub-Directory Description
/lib	**Invariant Code**: Source code for the CORDET Framework and FW Profile. This code is used unchanged in all applications instantiated from the PUS Extension.
/tests	**Application-Specific Code**: Implementation of Test Suite application.
/tests/PusConfig	**Configurable Code**: Configuration code for the CORDET Framework. The header files `CrFwOutRegistryUserPar.h`, `CrFwOutFactoryUserPar.h` and `CrFwInFactoryUserPar.h` are generated by the CORDET Editor.

3.3 Support Documentation

The PUS Extension is delivered with the following support documents:

- The **PUS Extension Definition Document** which specifies the PUS Extenstion
- A **User Manual** (this document) which describes how the C2 Implementation is used
- A **Verification Report** which provides validation and verification evidence for the PUS Extension of the CORDET Framework
- A **TM/TC ICD** which defines the command and reporting interface used for the Test Suite application. All tables in this document are generated from the CORDET Editor.

These documents, together with the Test Suite and the detailed software documentation in the Doxygen web site, constitute the **Qualification Data Package** (QDP) for the PUS Extension. The QDP is provided for users who need to certify their application or, more generally, who need to provide evidence of its correctness. The QDP contains the typical information which is required for software certification purposes. It can therefore be included in the certification data package of end-applications and it relieves the user of the need to produce such information for the PUS Extension part of their applications.

3.4 Doxygen Documentation

All the source code in the PUS Extension (including the test suite and the CORDET Framework code) is documented in accordance with doxygen rules. The entry point to the Doxygen documentation is the `index.html` file in the `/doc/doxygen` directory.

3.5 Test Suite

The Test Suite is a complete application which demonstrates all aspects of the behaviour of the PUS Extension of the CORDET Components.

The main program of the Test Suite application is in file `CrPsTestSuite.c`. This program consists of a set of test cases. For each supported service, one or more test cases are defined. Each test case exercises a specific aspect of the behaviour of a CORDET Component. The Test Suite aims to offer 100% code, branch, and condition coverage of the CORDET Components. This target will be confirmed in release 1.0.0 of the PUS Extension.

On a Unix platform, the Test Suite application can be built by running one of the support scripts delivered with the PUS Extension (see section 3.7).

3.6 Generation of Delivery File

The Delivery File contains all the inputs required to generate both the software and the documentation for the PUS Extension of the CORDET Framework. Table 3.3 describes the step-by-step procedure to generate a new version of the PUS Extension. Its starting point is the availability of a complete delivery.

Table 3.3: Generation of a New PUS Extension Delivery

N	Step
1	Export the models from the FW Profile Editor through the "Download C Code Dynamic" action in the editor. This action returns one single zipped file holding the models in `json` format, the code generated from them, and the image files with the state machine and procedure diagrams.
2	Run the following code generators in CORDET Editor: (a) ICD Generator, (b) Data Pool Generator, (c) Specification Generator, (d) Packet Access Function Generator, and (e) CordetFw Generator
3	Run the `ImportGenProducts.sh` script to import the auto-generated items into the delivery directory. The instruction for running the script are in the comment at the beginning of the script. The items to be imported are: (a) code outputs generated by the CORDET Editor, (b) data pool csv description file generated by the CORDET Editor, (c) constants csv description file generated by the CORDET Editor, and (d) code generated by the FW Profile Editor.
4	Update the framework documentation as needed. The library documentation is in latex format in the following directories: `doc/int/um` (user manual), `doc/int/pus` (specification), `doc/int/vr` (verification report), `doc/int/icd` (TM/TC ICD).
5	Update the document issue numbers in file `doc/int/common/RefDoc.csv` and in the `MakeDeliveryFile.sh` script
6	Run Doxygen on the entire source code of the delivery by running command: `doxywizard DoxygenConfig.txt` from the top-level directory of the framework. Verify that neither errors nor warnings are reported by doxygen.
7	Delete all items generate in previous tests of the framework by running command: `make clean`.
8	Build the executable of the Test Suite with "all warnings" enabled by running command: `make test`. This step should be done twice with `gcc` and with `clang` (the compiler is selected by editing one line at the top of the `makefile`). Verify that neither errors nor warnings are reported by the compiler or linker.
9	Run `scan-build` by again running command `make clean` and then running command `scan-build -o log make test`. Verify that all issues reported by `scan-build` are false positives.
10	Run the Test Suite with command: `make run-test` and verify that all test cases are executed successfully.
11	Run again the Test Suite with Valgrind through command: `make run-test-valgrind`. Verify that the Test Suite runs to completion and that no memory leaks are reported by Valgrind.
12	Run `lcov` on all library source files through command: `make gen-lcov`. Verify coverage levels in the `lcov` output in directory `lcov`. See also section 6 of reference [PX-VR].

N	Step
13	Take a snapshot of the summary table of `lcov` and store it in file `doc/int/images/LcovCodeCovReport.png` from where it will be used for building the Verification Report.
14	Compute the software metrics by running command `lizard -C 10 src`. Use the output of this command to fill in the table in section 11.1 of reference [TiVr].
15	Generate the PDF version of the library documents by compiling the latex documents. Compilation must be run twice to ensure that all cross-references are correctly linked. Verify that compilation is successful.
16	When all above steps have been successfully completed, generate the delivery data package by running script `MakeDeliveryFile.sh`.

With reference to point 5, it is noted that, depending on the test timing, Valgrind may report 3 possible memory leaks originating in function `pthread_create`. This is due to the fact that the test cases in module `CrFwSocketTestCase` create threads but do not join them before terminating. This potential leak does not affect the framework code and is therefore accepted.

3.7 Support Scripts

To build the Test Suite a Makefile is provided in the root directory. This Makefile can be used with the generally available `make` tool to generate different targets. The following targets are supported:

- `make test` Generates the test suite
- `make run-test` Runs the test suite
- `make run-test-valgrind` Runs the test suite
- `make gen-lcov` Generates the gcov files which contain the coverage information
- `make gen-lcov` Generates the `lcov` files which contain the coverage information in html format

The test suite is created in the `/bin` sub-directory.

Additionally, the following scripts are provided:

- `ImportGenProducts.sh` copies the files generated by the FW Profile and the CORDET Editor to the `cordetfw-pus` directory
- `MakeDeliveryFile.sh` generates the delivery file

3.8 Verification Environment

The PUS Extension of the CORDET Framework has been developed and verified in the following enviroment (the first two items were obtained by entering `g++ -v` and `uname -a` at the terminal):

- Ubuntu 5.4.0-6ubuntu1 16.04.10 on an x86_64 target
- Linux ap 4.4.0-141-generic #167-Ubuntu SMP Wed Dec 5 10:40:15 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux

- Version 5.4.0 of `gcc`
- Version 3.8.0-2ubuntu4 (tags/RELEASE_380/final) of `clang` for x86_64-pc-linux-gnu with posix thread model
- Version 1.3.1 of the FW Profile
- Version 1.14.10 of `lizard` (computation of software metrics)
- Version 1.8.11 of `Doxygen`

The tool `scan-build` was used to perform a static check on the framework code. It does not seem to be possible to obtain the version of the installed tool.

4 Mode of Use

The PUS Extension of the CORDET Framework extends the CORDET Framework in the sense that: (a) it closes some of its adaptation points and (b) it adds some new adaptation points. Table 4.1 lists the CORDET Framework adaptation points which are closed by the PUS Extension and the adaptation points which are added by the PUS Extension.

The instantiation process for the PUS Framework extends the instantiation process of the CORDET Framework described in section 24 of the CORDET Framework User Manual in reference [CR-SP]. For convenience, the instantiation steps for the CORDET Framework are listed in table 4.2. Steps 5.1 to 5.4 have been added to cover the instantantion activities which are specific to the PUS Extension. These instantiation activities would normally be performed by modifying the PUS Extension model in the CORDET Editor and by then re-generating its code using the editor's code generators.

4.1 Memory Management

The PUS Extension of the CORDET Framework allocates memory both on the stack and on the heap. Stack usage has not been measured but it is noted that no recursive function calls are made by the PUS Extension code[1].

Memory allocation from the heap is done exclusively during the initialization phase. There is therefore no risk of memory leaks. This has been verified both with the help of the `Valgrind` tool and through static code analysis (see section 9 in reference [PX-VR].

4.2 Real-Time Aspects

The PUS Extension, like the CORDET Framework, is purely passive and does not create any own threads. The same considerations as for the real-time aspects of the CORDET Framework (see section 22 of the CORDET Framework User Manual in reference [CR-SP]) also apply to the PUS Extension.

4.3 Operational Constraints

The operational constraints are listed in the "Related Pages->Constraints" page of the doxygen documentation. Compliance with these constraints is mandatory and failure to comply with them will result in unpredictable (and probably undesirable) behaviour.

4.4 Limitations

Limitations (i.e. functionalities which are specified but not implemented) are listed in the "Related Pages->Limitations" page of the doxygen documentation.

4.5 Known Bugs

Bugs are tracked through the "Issues" facility of the `cordetfw-pus` project.

[1] Recursion 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 (see the FW Profile User Manual in reference [FwProf]). Since no nested state machines are used in the PUS Extension, it can be assumed that no recursion is used.

Table 4.1: Adaptation Points Closed or Created by PUS Extension of CORDET Framework

Req. ID	Origin	Description	Default Valut	Implementation
DP-1	New AP	Data Pool Load Procedure	Procedure does nothing and terminates when it is executed for the first time	Module `CrPsDataPoolFunc`
DP-7	New AP	Definition of Data Items in the Data Pool Component	The default data pool implements the observables and the parameters for the services supported by the PUS Extension	For each service `srv`, the associated observables and parameters are implemented in module `DataPool/CrPsDpSrv` 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	Getter and setter methods are provided for all observables and the parameters for the services supported by the PUS Extension	See previous adaptation point
DP-9	New AP	Data Pool Refresh Operation	Loads the values of the debug variables (loads the values of the memory locations pointed at by the elements of data pool parameters `debugVarAddr` into the elements of data pool variable `debugVar`, see section 8.3)	Module `CrPsDataPoolFunc`
OCM-1	Closes OCM-1	Initialization Check in Initialization Procedure of OutComponent	Always returns 'check successful'	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 `CrFwOutFactory.c`

Req. ID	Origin	Description	Default Valut	Implementation
OCM-2	Closes OCM-2	Initialization Action in Initialization Procedure of OutComponent	Do nothing and return 'action successful'	See close-out of OCM-1
OCM-3	Closes OCM-3	Configuration Check in Reset Procedure of OutComponent	Always returns 'check successful'	See close-out of OCM-1
OCM-4	Closes OCM-4	Configuration Action in Reset Procedure of OutComponent	Do nothing and return 'action successful'	See close-out of OCM-1
OCM-5	Closes OCM-5	Shutdown Action in Base Component of OutComponent	Do nothing	See close-out of OCM-1
OCM-6	Closes OCM-6	Execution Procedure of OutComponent	Do nothing	See close-out of OCM-1
OCM-7	Closes OCM-7	Service Type Attribute of OutComponent	Set equal to PUS service type	Service identifiers for supported services are defined in `CrPsServTypesId.h`
OCM-8	Closes OCM-8	Command/Report Sub-Type Attribute of OutComponent	Set equal to PUS service sub-type	Sub-service identifiers for supported services are defined in `CrPsServTypesId.h`
OCM-9	Closes OCM-9	Destination Attribute of OutComponent	See definition of individual reports	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	Not relevant to out-going report	n.a.
OCM-11	Closes OCM-11	Discriminant Attribute of OutComponent	See definition of individual reports	Destination is set at the point of instantiation of a report according to the rules defined in the report definition tables

Req. ID	Origin	Description	Default Valut	Implementation
OCM-12	Closes OCM-12	Parameter Attribute of OutComponent	See definition of individual reports	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	See definition of individual reports	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 definition of individual reports	See close-out of OCM-13
OCM-15	Closes OCM-15	Repeat Check Operation of OutComponent	See definition of individual reports	See close-out of OCM-13
OCM-16	Closes OCM-16	Update Action of OutComponent	See definition of individual reports	See close-out of OCM-13
OCM-17	Closes OCM-17	Serialize Operation of OutComponent	Build a packet with the layout specified by the PUS	See close-out of OCM-13
OCM-18	Closes OCM-18	Operation to Report Invalid Destination of an OutComponent	Generate SNDPCKT_INV_DEST Error Report	Error reports are generated throug 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.

Req. ID	Origin	Description	Default Valut	Implementation
ICM-1	Closes ICM-1	Initialization Check in Initialization Procedure of InCommand	Always returns ’check successful’	Incoming commands are provided by the `CrFwInfactory` in the CONFIGURED state which configures them to have an initilization check which is always successful.
ICM-2	Closes ICM-2	Initialization Action in Initialization Procedure of InCommand	Do nothing	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	Returns ’check successful’ if packet length and checksum are correct	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	Use information in incoming packet to configure InCommand and return “action successful’	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	Release all resources allocated to the InCommand	Incoming commands are provided by the `CrFwInfactory` in the CONFIGURED state which configures them to have a shutdown action which does nothing

Req. ID	Origin	Description	Default Valut	Implementation
ICM-6	Closes ICM-6	Execution Procedure of InCommand	Do nothing	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	See definition of individual commands	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 definition of individual commands	See close-out of ICM-7
ICM-9	Closes ICM-9	Progress Action of InCommand	See definition of individual commands	See close-out of ICM-7
ICM-10	Closes ICM-10	Termination Action of InCommand	See definition of individual commands	See close-out of ICM-7
ICM-11	Closes ICM-11	Abort Action of InCommand	See definition of individual commands	See close-out of ICM-7
ICM-12	Closes ICM-12	Operation to Report Start Failed for InCommand	See definition of service 1	Operation to report outcome of acceptance, start, execution and termination checks are implemented by function `CrFwRepInCmdOutcome` which generates service 1 reports
ICM-13	Closes ICM-13	Operation to Report Start Successful for InCommand	See definition of service 1	See close-out of ICM-12
ICM-14	Closes ICM-14	Operation to Report Progress Failed for InCommand	See definition of service 1	See close-out of ICM-12

Req. ID	Origin	Description	Default Valut	Implementation
ICM-15	Closes ICM-15	Operation to Report Progress Successful for InCommand	See definition of service 1	See close-out of ICM-12
ICM-16	Closes ICM-16	Operation to Report Termination Failed for InCommand	See definition of service 1	See close-out of ICM-12
ICM-17	Closes ICM-17	Operation to Report Report Termination Successful for InCommand	See definition of service 1	See close-out of ICM-12
ICM-18	Closes ICM-18	Service Type Attribute of InCommand	Set equal to PUS service type	Service identifiers for supported services are defined in `CrPsServTypesId.h`
ICM-19	Closes ICM-19	Command Sub-Type Attribute of InCommand	Set equal to PUS service sub-type	Sub-service identifiers for supported services are defined in `CrPsServTypesId.h`
ICM-20	Closes ICM-20	Discriminant Attribute of InCommand	See definition of individual commands	Discriminants are set at the point of instantiation of a command
ICM-21	Closes ICM-21	Parameter Attributes of InCommand	See definition of individual commands	Parameters for commands in service Xyz are read using the getter functions generated by the CORDET Editor in `CrPsPcktXyz.h`
S1-1	Closes ILD-12	Operation to Report Packet Destination Invalid by InLoader	Run the Packet Re-Routing Failure Procedure of figure 7.1	The procedure is implemented iin module `CrPsCmdPrgrFail` (generated by FW Profile Editor) and it is run from function `CrFwRepInCmdOutcome`
S1-2	Closes ILD-14	Operation to Report Acceptance Failure by InLoader	Run the Command Verification Failure Procedure of figure 7.3	The procedure is implemented iin module `CrPsCmdReroutingFail` (generated by FW Profile Editor) and it is run from function `CrFwRepErrInstanceIdAndDest`

Req. ID	Origin	Description	Default Valut	Implementation
S1-3	Closes ILD-13	Operation to Report Acceptance Success by InLoader	Run the Command Verification Success Procedure of figure 7.2	The procedure is implemented iin 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	Run the Command Verification Failure Procedure of figure 7.3	The procedure is implemented iin 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	Run the Command Verification Success Procedure of figure 7.4	The procedure is implemented iin 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	Run the Command Progress Failure Procedure of figure 7.5	The procedure is implemented iin 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	Run the Command Progress Success Procedure of figure 7.4	The procedure is implemented iin 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	Run the Command Verification Failure Procedure of figure 7.3	The procedure is implemented iin 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	Run the Command Verification Success Procedure of figure 7.2	The procedure is implemented iin module `CrPsCmdVerSucc` (generated by FW Profile Editor) and it is run from function `CrFwRepInCmdOutcome`

Req. ID	Origin	Description	Default Valut	Implementation
S3-1	New AP	Definition of DAta Pool Parameters and Variables	Some data pool items are defined at framework level and are listed in the 'Observables and Parameters' sections of the supported services	Applications must create modules like the `CrPsDpXyz` to implement the new data items
S5-1	New AP	Definition of Event Reports	Some event reports are pre-defined at PUS Extension level and are listed in section A of [PX-SP]	Applications must expand the `CrPsDpEvt` module and the `CrPsPcktEvt` to cover the new event reports
S12-1	New AP	Definition of Parameter Monitoring Procedures	Three parameter monitoring procedures are defined (limit check, expected value and delta check)	This service is not yet implemented
S12-2	New AP	Definition of Parameter Monitoring Definition List (PMDL)	By default, all parameter monitors are empty	This service is not yet implemented
S12-3	New AP	Definition of Functional Monitoring Definition List (FMDL)	By default, all parameter monitors are empty	This service is not yet implemented
S13-1	New AP	Operation to access the i-th LPT Buffer	No default defined at framework level	This service is not yet implemented

Table 4.2: Framework Instantiation Specification and Implementation Steps

N	Step Name	Specification Sub-Step	Implementation Sub-Step
1	Identify Target Application	Identify the application for which the framework is being instantiated.	The Application Identifier is specified in `CrFwUserConstants.h`.
2	Identify Service Users	Identify the users of the services provided by the target application. Each service user is identified through its Application Identifier.	The service user identifiers are used to define the sources of incoming commands (InCommands) for the application in `CrFwInStreamUserPar.h` and the destination of out-going reports (OutCompnents) in `CrFwOutStreamUserPar.h`.
3	Identify Service Providers	Identify the providers of the services used by the target application. Each service provider is identified through its Application Identifier.	The service provider identifiers are used to define the sources of incoming reports (InReports) for the application in `CrFwInStreamUserPar.h` and the destination of out-going commands (OutCompnents) in `CrFwOutStreamUserPar.h`.
4	Define Used Services	Define the services which are used by the target application. Each service is defined through: its identifier (the "service type"); a description of the purpose of the service; the external entity which provides the service; the commands and reports which implement the service.	The range of services used by the application is defined in `CrFwInFactoryUserPar.h` and `CrFwOutFactoryUserPar.h`. Also, a list of services supported by the application is defined in `CrFwOutRegistryUserPar.h`.
5	Define Provided Services	Define the services which are provided by the target application. Each service is defined through: its identifier (the "service type"); a description of the purpose of the service; the external entity which uses the service; the commands and reports which implement the service.	The range of services provided by the application is defined in `CrFwInFactoryUserPar.h` and `CrFwOutFactoryUserPar.h`. Also, a list of services supported by the application is defined in `CrFwOutRegistryUserPar.h`.

N	Step Name	Specification Sub-Step	Implementation Sub-Step
5.1	Select Pre-Defined Provided Services	From the services provided by the PUS Extension, select those which are to be included in the target application. Note that it is not possible to select only a sub-set of the commands and reports of a supported service: once a pre-defined service is selected for inclusion in the target application, all its commands and reports are imported.	Services supported by the PUS Extension are identifier by a 2- or 3-letter acronym. The code implementing service Xyz is spread over three locations: (a) the code implementing the commands and reports for supported service Xyz is in directory `src/Xyz`; (b) the data pool items belonging to that service are in module: `src/DataPool/CrPsDpXyz`; (c) the packet accessor functions are in module `src/Pckt/CrPsPcktXyz`.
5.2	Customize the Selected Pre-Defined Services	For the selected services pre-defined by the PUS Extension, define: (a) the value of their constatns; (b) the size of the types of their data pool items and packet parameters; and (c) the endianity of its packet parameters.	The PUS Extension constants are defined in file `CrPsConstants.h`. The size of the data types are defined in `CrPsTypes.h`. A change of endianity requires update of all packet accessor functions in: `CrFwPckt.h`, in all the `CrPsPcktXyz.h`
5.3	Define Command Rejection Codes	If the target application supports the Command Verification Service pre-defined by the PUS Extension, define the command rejection codes supported by the application in terms of their identifiers and of their verification data item.	The range of command rejection codes is defined in enumerated type `CrPsFailCode_t`.
5.4	Define Event Reports	If the target application supports the Event Reporting Service pre-defined by the PUS Extension, define the event reports supported by the application in terms of their event identifiers, of their severity level, and of the parameters they carry.	The range of event identifiers is defined in enumerated type `CrPsEvtId_t`. A change in the set of event identifiers requires the module `CrPsPcktEvt.h` to be re-generated.
6	Identify Re-Routing Capabilities	Define the applications to which incoming packets received must be re-routed.	The re-routing information is defined in the re-routing function which is provided to the framework as a function pointer in `CrFwInLoaderUserPar.h` and for which two defaults are provided by the InLoader component. Also, re-routing contributes to the definition of InStreams and OutStreams (InStreams are required to receive re-routed packets and OutStreams are required to forward them).

N	Step Name	Specification Sub-Step	Implementation Sub-Step
7	Define Incoming Commands	For each provided service not pre-defined by the PUS Extension, define the commands which implement it (i.e the commands which the application must be able to receive and process) in terms of: their attributes, their acceptance and ready checks, their start action, progress action, termination action, and abort action.	The detailed definition of the incoming commands is done in `CrFwInFactoryUserPar.h`. Also, for each command, a C-module must be provided which implements the functions encapsulating the command actions and checks. See the command modules pre-defined by the PUS Extension for an example.
8	Define Incoming Reports	For each used service, define the reports which implement it (i.e. the reports which the application must be able to receive and process) in terms of: their attributes, their acceptance check, and their update action.	The detailed definition of the incoming reports is done in `CrFwInFactoryUserPar.h`. Also, for each report, a C-module must be provided which implements the functions encapsulating the report actions and checks. See module `CrFwInRepSample1` for an example.
9	Define Outgoing Commands and Reports	For each provided service not pre-defined by the PUS Extension, define the reports which implement it and for each used service, define the commands which implement it in terms of: their attributes, their enable check, and their ready, and repeat check and their update action.	The detailed definition of the out-going commands and reports is done in `CrFwOutFactoryUserPar.h`. Also, for outgoing reports or commands which do not use the default implementations of the OutComponent adaptation points, a C-module must be provided which implements the functions encapsulating the report or command actions and checks. See module `CrFwOutCmpSample1` for an example.
10	Assign Commands and Reports to Groups	Define command and report groups and define rules for assigning commands and reports to groups.	The definition of the assignment rules is done in the implementation of the getter and setter functions for the group attribute in module `CrFwPckt`.
11	Define Command and Report Layout	For each command and report which can be either generated or received by the target application, define the layout of the packet which carries it.	The packet layout is implicitly implemented in the setter and getter functions of the `CrFwPckt.h` interface. The application developer must provide a complete implementation for this interface. A stub implementation is provided in the configuration directory `/cr/src/crConfigTestSuite`.

N	Step Name	Specification Sub-Step	Implementation Sub-Step
12	Define Packet Allocation Policy	Define the allocation policy for the packets which the application creates when it receives a command or report.	The packet allocation policy is implemented in the `make` function of the `CrFwPckt.h` interface. The application developer must provide a complete implementation for this interface. A stub implementation is provided in the configuration directory `/cr/src/crConfigTestSuite`.
13	Define Command and Report Capacity	Define: the maximum number of incoming commands which the target application can hold at any given time; the maximum number of incoming reports which the target application can hold at any given time; and the maximum number of outgoing commands or reports which the application can hold at any given time.	The capacities for incoming commands and reports are defined as `#DEFINE` constants in `CrFwInFactoryUserPar.h`. The capacity for out-going commands and reports is defined as a `#DEFINE` constant in `CrFwOutFactoryUserPar.h`.
14	Define Application Modes	Define the sub-states in the states of the Application State Machine.	For each set of sub-states, a state machine implementing them is defined which is then embedded in one of the states of the Application State Machine. The embedded state machines are defined in `CrFwAppSmUserApp.h`.
15	Define Incoming Middleware Interface	Define the interface to the middleware which is responsible for receiving the commands and reports for the target application.	For each source of commands or reports, one InStream is defined. The size of the InStream packet queues and the pointers to the functions which implement the InStream operations are defined in `CrFwInStreamUserPar.h`. Also, for each InStream a C module must be defined which implements the InStream functions. A test stub is provided in `CrFwInStreamStub`.
16	Define Out-Going Middleware Interface	Define the interface to the middleware which is responsible for sending the commands and reports originating in the target application.	For each command or report destination, one OutStream is defined. The size of the OutStream packet queues and the pointers to the functions which implement the OutStream operations are defined in `CrFwOutStreamUserPar.h`. Also, for each OutStream a C module must be defined which implements the OutStream functions. A test stub is provided in `CrFwOutStreamStub`.

N	Step Name	Specification Sub-Step	Implementation Sub-Step
17	Define InManagers	Define the number of InManagers and the size of their Pending Command/Report Lists (PCRLs).	These items are defined as `#DEFINE` constants in `CrFwInManagerUserPar.h`
18	Define InManager Selection Function	Define the logic to select the InManager where an incoming command or report is loaded.	A pointer to this function is defined in `CrFwInLoaderUserPar.h`. A default implementation is provided by the InLoader (see `CrFwInLoader.h`).
19	Define InRegistry	Define the maximum number of commands and reports which can be tracked by the InRegistry.	This item is defined as a `#DEFINE` constant in `CrFwInRegsitryUserPar.h`.
20	Define OutManagers	Define the number of OutManagers and the size of their Pending OutComponent Lists (POCLs).	These items are defined as `#DEFINE` constants in `CrFwOutManagerUserPar.h`.
21	Define OutManager Selection Function	Define the logic to select the OutManager where an out-going command or report is loaded.	A pointer to this function is defined in `CrFwOutLoaderUserPar.h`. A default implementation is provided by the OutLoader (see `CrFwOutLoader.h`).
22	Define OutRegistry	Define the maximum number of commands and reports which can be tracked by the OutRegistry.	This item is defined as a `#DEFINE` constant in `CrFwOutRegistryUserPar.h`.
23	Define Start-Up Procedure	Define the start-up procedure for the application. This in particular includes the sequence in which framework components are instantiated, initialized and configured.	Implement the Application Start-Up Procedure by providing an implementation for `CrFwAppStartUpProc.h`. A test stub is provided in `CrFwAppStartUpProc.c`.
24	Define Reset Procedure	Define the reset procedure for the application. This in particular includes the sequence in which framework components are reset.	Implement the Application Reset Procedure by providing an implementation for `CrFwAppResetProc.h`. A test stub is provided in `CrFwAppResetProc.c`.
25	Define Shutdown Procedure	Define the shutdown procedure for the application. This in particular includes the sequence in which framework components are shutdown.	Implement the Application Shutdown Procedure by providing an implementation for `CrFwAppShutdownProc.h`. A test stub is provided in `CrFwAppShutdownProc.c`.

N	Step Name	Specification Sub-Step	Implementation Sub-Step
26	Define Time Interface	Define the means through which the current time is acquired. This is needed for time-stamping out-going commands and reports in the OutStream.	The time acquisition interface is defined in `CrFwTime.h`. The application developer must provide a complete implementation for this interface. A stub implementation is provided in the configuration directory `/cr/src/crConfigTestSuite`.
27	Define Error Reporting Interface	Define the response to the generation of error reports.	The respone to error reports is defined in `CrFwRepErr.h`. The application developer must provide a complete implementation for this interface. A test implementation is provided in the configuration directory `/cr/src/crConfigTestSuite`.
28	Define InCommand Outcome Reporting	Define the means through which the outcome of the processing of incoming commands is reported.	The respone to the reports of InCommand outcomes is defined in `CrFwRepInCmdOutcome.h`. The application developer must provide a complete implementation for this interface. A test implementation is provided in the configuration directory `/cr/src/crConfigTestSuite`.
29	Define Primitive Types	Define the range of the primitive types used by the framework components. The driver for this definition is the need to optimize the memory footprint of the application.	The primitive types are defined through `typedef`'s in `CrFwUserConstants.h`. Application developers can override the default definitions in this file (but note that, in most cases, the default definitions should be adequate).