The CORDET Framework
C2 Implementation
- USER REQUIREMENTS -

Alessandro Pasetti & Vaclav Cechticky

26 June 2014

Revision 0.5.0
PP-SP-COR-0002

P&P Software GmbH
High Tech Center 1
8274 Tägerwilen
Switzerland

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

Abstract

This document defines, justifies, and verifies the User Requirements for the C2 Implementation of the CORDET Framework. The CORDET Framework is a software framework for service-oriented embedded applications. The CORDET Framework defines an application in terms of the services it provides to other applications and in terms of the services it uses from other applications.

The CORDET Framework is implementation-independent. The C2 Implementation is a C-language implementation of the CORDET components. The main features of the C1 Implementation are: small memory footprint, small CPU demands, scalability, and high reliability.

The C2 Implementation is provided with a Qualification Data Package which can be used to support the certification of applications built using its components.

Contents

List of Figures

List of Tables

1 Introduction

This document defines, justifies and verifies the user requirements for the *C2 Implementation*. The C2 Implementation is a C-language implementation of the CORDET Framework. The CORDET Framework is a software framework for service-oriented distributed embedded applications. The CORDET Framework defines an application in terms of the services it provides to other applications and in terms of the services it uses from other applications.

A service is implemented by a set of commands through which an application is asked to perform certain activities and by a set of reports through which an application gives visibility over its internal state. The CORDET Framework defines the components to receive, send, distribute, and process commands and reports. The CORDET Framework is defined in [4].

1.1 Intended Use of C1 Implementation

Although the C2 Implementation can be used wherever there is a need to implement a system of distributed applications which exchange CORDET service requests, the high reliability of the implementation, the emphasis placed on formally specifying and verifying its expected behaviour, and the small demands on memory and processing resources mean that the C2 Implementation is especially well-suited for implementing mission-critical embedded applications within a service-oriented distributed architecture.

Thus, the intended use of the C2 Implementation is to support the implementation of the CORDET service concept for mission-critical embedded applications.

1.2 Requirement Definition

Requirements are defined in tables with the following format:

CR-'x'/'V'	⟨**Requirement Title**⟩
Requirement	⟨Formulation of requirement⟩
Note	⟨Explicatory notes for requirement⟩
Justification	⟨Justification of requirement⟩
Implementation	⟨Dscription of how requirement is implemented⟩
Verification	⟨Dscription of how requirement is verified⟩

Here, the suffix 'x' is a numerical identifier which uniquely identifies the requirement within this document. The suffix 'V' identifies the verification method for the requirement according to the convention presented in section 1.2.3.

The explicatory notes are appended to the definition of the requirements where there is a need to clarify the terms which are used in their formulation.

In addition to their definition, this document also provides the following infor-

mation for each requirement: a justification of the requirement; a description of how the requirement is implemented; and a description of how the requirement is verified.

1.2.1 Requirement Justification

For each requirement, a *justification* is provided which *validates* the requirement. Requirements are justified with respect to the intended use of the C2 Implementation. The intended use of the C2 Implementation is to support the implementation of the CORDET service concept for mission-critical embedded applications (see section 1.1). Hence, a requirement is justified in proportion to its ability to further the adequacy of the C2 Implementation to support the implementation of the CORDET service concept in an environment where memory and processing resources are constrained and where reliability is of paramount importance.

1.2.2 Requirement Implementation

For each requirement, the function or data structure or other code-level construct in the source code which implements it is identified.

1.2.3 Requirement Verification

Verification information is provided for each requirement to demonstrate the correct implementation of the requirement. The following verification methods are possible:

- Verification by Review ('R'): the requirement is verified by inspecting the code or its documentation.
- Verification by Analysis ('A'): the requirement is verified by analysing the code, possibly with the help of a tool.
- Verification by Test ('T'): the requirement is verified by one or more test cases in the Test Suite.

One single verification method is defined for each requirement. This is identified as part of the requirement definition (see the description of the requirement format in section 1.2).

The Test Suite which is used for the verification by test is a complete application which demonstrates all aspects of the behaviour of the CORDET components. It consists of a sequence of Test Cases which are independent of each other. Each Test Case focuses on one particular functional aspect of the C2 Implementation. The Test Suite is distributed with the C2 Implementation. It is documented as part of the Doxygen documentation for the C2 Implementation and is described in the C2 Implementation User Manual (see reference [5]).

2 Functional Requirements

This section defines the functional requirements for the C2 Implementation. The functional requirements are those which define the functional behaviour of the components which implement the CORDET Framework.

2.1 CORDET Framework Requirements

The CORDET Framework is specified through a set of formal requirements defined in reference [4]. Four types of requirements are recognized in reference [4]:

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

An implementation of the CORDET Framework should cover the first two types of requirements (the other two types of requirements are only relevant to application developers who wish to instantiate the framework to build a specific application). This section defines this coverage for the C2 Implementation.

CR-2.1.1/T	**CORDET Standard Requirements**
Requirement	The C2 Implementation shall implement the standard requirements of the CORDET Framework of [4].
Justification	The intended use of the C2 Implementation is to implement the CORDET Framework.
Implementation	Appendix A shows how each standard requirement defined in reference [4] is implemented in the C2 Implementation.
Verification	Appendix A shows how each standard requirement defined in reference [4] is verified in the C2 Implementation.

CR-2.1.2/R	CORDET Adaptation Requirements
Requirement	The C2 Implementation shall implement the adaptation requirements of the CORDET Framework of reference [4].
Justification	The intended use of the C2 Implementation is to implement the CORDET Framework.
Implementation	The adaptation requirements of reference [4] define a number of adaptation points. Appendix B shows how each adaptation point of reference [4] is implemented in the C2 Implementation.
Verification	See explanatory text at the beginning of appendix B. Note also that the Adaptation Requirements are verified by showing that a running application can be built by closing each Adaptation Point (or using the default value of an Adaptation Point). This is done in the Test Suite. The Test Suite exercises all framework functionalities (it has 100% statement coverage) and therefore needs all Adaptation Points to be closed.

2.2 C2 Adaptation Points

CR-2.2.1/R	C2 Adaptation Points
Requirement	The C2 Implementation shall support the adaptation points listed in table C.1.
Justification	These adaptation points arise as a result of the design choices made for the C2 Implementation.
Implementation	The last column in table C.1 describes how each adaptation points is implemented and what its default value (if any) is.
Verification	See Implementation.

CR-2.2.2/T	Default Values for Adaptation Points
Requirement	The C2 Implementation shall provide default values for all adaptation points (both those defined at CORDET Framework level and those defined at C2 Implementation level).
Justification	Provision of default values facilitates the definition of test cases and demonstrators.
Implementation	The default values for the column in table C.1 describes how each adaptation points is implemented and what its default value is.
Verification	The default values for the C2 Implementation are those used for the Test Suite which constitutes a complete instantiation of the CORDET Framework and defined in appendix A of the reference [5].

2.3 Component Instantiation

CR-2.3.1/T	Component Instantiation
REQUIREMENT	The C2 Implementation shall provide *Factory Functions* to instantiate the following types of components: OutStream, OutFactory, OutManager, OutLoader, OutRegistry, InStream, InFactory, InManager, InLoader, and InRegistry.
JUSTIFICATION	The CORDET Framework distinguishes between components which are subject to *early instantiation* and those which are subject to *late instantiation* (see section 3.1 of reference [4]). The component types listed in this requirements are those which are subject to early instantiation. These are the components which must be instatiated during the application start-up.
IMPLEMENTATION	The factory functions are the functions with names like: `CrFwXxxMake` where `Xxx` is the name of the component type.
VERIFICATION	For each component type, a set of test cases is defined in: `CrFwXxxTestCases.h` where `Xxx` is the name of the component type. These test cases verify the factory functions.

CR-2.3.2/R	Attribute Setting Order
REQUIREMENT	When a factory function configures a newly-created packet, it shall set its attributes in the following order: packet report/command flag (which determines whether the packet holds a report or a command), packet source (i.e. the host application), packet group, packet type, packet sub-type, packet discriminant, and then other attributes in an undefined order.

JUSTIFICATION	The framework provides one single interface for decoding and encoding packets in module `CrFwPckt`. This is obviously suitable for application developers who wish to use the same layout for all packets used by the application, irrespective of their type or of their destination or source or their other characteristics. If this is not possible, then the getter and setter functions of interface `CrFwPckt.h` must implement logic which makes their outcome dependent on the content of the packet itself. Thus, for instance, if different packet sources use different layouts, the getter functions will have to inspect the source of a packet before deciding how to decode the value of a packet's attribute. In the case of the setter functions, this approach requires that the order in which the packet attributes are set be specified so that the logic in the setter functions can rely on this ordering to decide how to set attribute values.
IMPLEMENTATION	The only place in the CORDET Framework where newly-created packets are configured is the function to create a new OutComponent `CrFwOutFactoryMakeOutCmp`.
VERIFICATION	Inspection of the implementation of function `CrFwOutFactoryMakeOutCmp` in module `CrFwOutFactory` shows that the requirement is fulfilled.

CR-2.3.3/R	**Irreversibility of Instantiation**
REQUIREMENT	It shall not be possible to destroy an instance of the component types listed in the previous requirement.
JUSTIFICATION	The CORDET Framework specifies that components subject to early instantiation must be instantiated during the application start-up but it does not say whether they should be destroyed and re-created when the application is reset. In the interest of simplicity, the C2 Implementation bars dynamic destruction of these components.
IMPLEMENTATION	The C2 Implementation does not define any `release` function through which the instances created by the Factory Functions may be destroyed.
VERIFICATION	See implementation.

2.4 Component Factories

FW-2.4.1/R	Component Pools in Factories
Requirement	The components factories of the C2 Implementation shall manage dynamic component creation through pools of pre-allocated component instances.
Note	The component factories manage the dynamic allocation of components through a `make` and a `release` operation. The intention of this requirement is that these operations be implemented by creating a pool of pre-allocated components at initialization time and by then allocating and releasing component instances from this pool.
Justification	Use of a pre-allocated pool of component enchances static predictability of behaviour and this important for the target applications of the C2 Implementation.
Implementation	There are only two factory components in the C2 Implementation: the OutFactory and the InFactory. The OutFactory defines array `outCmp` in `CrFwOutFactory.c` to hold the pre-allocated OutComponent instances. The InFactory defines arrays `inCmd` and `inRep` in `CrFwInFactory.c` to hold the pre-allocated InCommand and InReport instances.
Verification	See Implementation.

CR-2.4.2/R	Dynamic Memory Allocation
Requirement	Dynamic memory allocation through calls to `malloc` shall be done exclusively as part of component initialization.
Justification	The component instantiation model of the CORDET Framework dictates that resource allocation be done as part of a component's Initialization Procedure (see section 3.2 of [4].
Implementation	Calls to `malloc` are used in the following functions: `CrFwInManagerInitAction`, `CrFwOutManagerInitAction` and `CrFwOutRegistryInitAction`.
Verification	Functions `CrFwInManagerInitAction`, `CrFwOutManagerInitAction` and `CrFwOutRegistryInitAction` implement the initialization action of the InManager, OutManager and OutRegistry components and are therefore executed as part of these component initialization.

CR-2.4.3/R	**Dynamic Memory Release**
REQUIREMENT	If a component performs a `malloc` call as part of its initialization action, then it shall also perform a matching `free` call as part of its shutdown action.
JUSTIFICATION	The shutdown action is symmetric to the initialization action. This requirement therefore helps ensure that there are no memory leaks.
IMPLEMENTATION	Calls to `free` are used in the following functions: `CrFwInManagerShutdown`, `CrFwOutManagerShutdown` and `CrFwOutRegistryShutdown`.
VERIFICATION	Functions `CrFwInManagerShutdown`, `CrFwOutManagerShutdown` and `CrFwOutRegistryShutdown` implement the shutdown operation of the InManager, OutManager and OutRegistry components which are the components which perform calls to `malloc` as part of their initialization (see previous requirement).

3 Non-Functional Requirements

This section defines the non-functional requirements of the C2 Implementation. Non-functional requirements impose overall constraints on the use, design, or implementation of the C2 Implementation.

3.1 Coding Requirements

CR-3.1.1/R	**Implementation Language**
Requirement	The C2 Implementation shall be implemented in the C language.
Justification	The C Language is the standard language for embedded applications.
Implementation	All the modules offered by the C2 Implementation are implemented in C.
Verification	See implementation.

CR-3.1.2/T	**Compiler Warning**
Requirement	The C2 Implementation shall not generate any warnings when compiled with the GCC compiler with all warnings enabled.
Justification	Warning may indicate weaknesses in the code or potential errors.
Implementation	See verification.
Verification	The C2 Implementation Acceptance Test Procedure (see reference [5]) compiles all source files of the implementation using `gcc` with the option `-Wall`.

3.2 Adaptation Mechanisms

FW-3.2.1/R	**Adaptation Mechanism**
REQUIREMENT	The C2 Implementation shall exclusively support static adaptation mechanisms.
NOTE	An adaptation mechanism is static if it only allows the adaptation to be performed at compile time. Thus, static adaptation forces application developers to decide how to close an adaptation point at compile time.
JUSTIFICATION	Restriction to static adaptation mechanisms enhances static predictability of behaviour which is important for mission-critical applications.
IMPLEMENTATION	The adaptation mechanisms supported by the C2 Implementation are (see section 6 of reference [5]): • Define Constant: a framework component uses a `#DEFINE` constant whose value may be overridden by application developers. • Define Function: a framework component uses a function pointer and application developers must provide an implementation for the missing function (or, if available, may choose to use the default implementation provided at framework level) • Implement Interface: the framework defines an interface as a C header file and application developers must provide an implementation for it. • Define Type: a framework component uses a variables of a type defined as a `typedef` and application developers may override the default type definition.
VERIFICATION	The adaptation mechanisms listed above are compile-time adaptation mechanisms.

3.3 Resource Requirements

FW-3.3.1/T	Code Memory Footprint
Requirement	The code memory footprint of the C2 Implementation shall be independent of the number of instances of framework components required by an application.
Note	Ideally, it would be desirable to impose a requirement on the memory occupation of the C2 Implementation. This is not possible because memory occupation depends on the tool chain used to compile an application and on the target processor. This requirement aims to restrict memory occupation in a manner which is independent of the compilation tool chain and of the execution hardware.
Justification	Embedded applications are often memory-constrained.
Implementation	See requirement verification.
Verification	The C2 Implementation provides a set of factory functions and factory components to create instances of framework components. There is no code generation facility (neither explicit, nor implicit through the use of macros) which generates *ad hoc* code for each component instance. Thus, the code base of the C2 Implementation is fixed and independent of the number of deployed component instances.

3.4 Verification Requirements

CR-3.4.1/T	**Test Coverage**
REQUIREMENT	The C2 Implementation shall be provided with a Test Suite offering 100% statement, branch and condition coverage.
JUSTIFICATION	The level of coverage provided by the requirement is that typically used in mission-critical applications.
IMPLEMENTATION	The Test Suite is implemented in a set of Test Cases defined in `CrFwXxxTestCases.h` where `Xxx` is the name of a component type. The `main` program for the Test Suite is in `CrFwTestSuite.h`.
VERIFICATION	The Acceptance Test Procedure of the C2 Implementation (see [5]) uses the `gcov` tool to measure the statement and branch coverage of the Test Suite. Note that the C2 Implementation does not use any boolean expressions in the decision points of the code (e.g. in the `if` clauses). Decision are always taken on the basis of the outcome of the evaluation of a single primitive Boolean condition. Hence, branch coverage implies condition coverage.

3.5 Dependency Requirements

CR-3.5.1/R	External Modules
Requirement	The C2 Implementation shall not require any external modules other than C's `stdlib` and `string` and the procedure and state machine modules of the C1 Implementation.
Justification	Minimization of dependencies on external libraries helps minimize the memory footprint of the application using the C2 Implementation and facilitates its qualification. The `stdlib` and `string` modules are likely to be used in any C application and hence they accepted. The C1 Implementation is provided with a qualification data package and its state machine and procedure modules have no external dependencies other than `stdlib` and `string`.
Implementation	See verification.
Verification	Inspection of the C2 Implementation files shows that no external modules other than `stdlib` and `string` and the state machine modules `FwSm*.h` and the procedure modules `FwPr*.h` of the C1 Implementation are used. The compilation and linking process for the Test Suite shows that no other libraries need be linked.

A CORDET Framework Standard Requirements

The C2 Implementation implements the standard requirements of the CORDET Framework. Table A.1 lists the standard requirements of the CORDET Framework as they are defined in reference [4] and, for each requirement, it describes how the requirement is implemented in the C2 Implementation and how the implementation is verified. The requirement identifier (first column in the table) is the same as used in reference [4].

The requirements often refer to state machine or procedure diagrams. A complete list of the diagrams of the state machines and procedures which define the behaviour of the CORDET components can be found in section E.

Table A.1: Implementation of CORDET Framework Requirements

ID	Requirement Text	Requirement Implementation	Requirement Verification
BAS-1	All components provided by the CORDET Framework shall implement the behaviour of the Base State Machine of figure E.1.	The behaviour of the Base State Machine is implemented in `CrFwBaseCmp`. Each framework component is built around a state machine instance which is derived from the Base State Machine (see section 5 of reference [5]). State machine derivation is done using the extension mechanism of the C1 Implementation of the FW Profile (see [5]) which guarantees that the derived state machines have the same behaviour as the base state machine. Framework component are instantiated by Make functions (see section 6.1 of reference [5]). The creation of their state machine as an extension of the Base State Machine is done in these Make functions.	The behaviour of the Base State Machine is verified in table D.1. The behaviour of its two procedures (Initialization Procedure and Reset Procedure) is verified in table D.7 and D.8). The inheritance of this behaviour by all other framework state machines is guaranteed by the extension mechanism of the FW Profile as it is implemented in the C1 Implementation and by the fact that all framework components are created as extension of a Base State Machine.

ID	Requirement Text	Requirement Implementation	Requirement Verification
BAS-2	The CORDET Framework shall implement an API through which applications can query a CORDET Component for its current state (including, if applicable, its current sub-state).	Only two levels of state machine embedding are used in the C2 Implementation. Query of the outer state (which is a state of the Base State Machine) is provided by functions: `CrFwCmpIsInCreated`, `CrFwCmpIsInInitialized` and `CrFwCmpIsInConfigured`. Embedded states are only embedded in state CONFIGURED and query of this embedded state is provided by functions with names like `CrFw⟨Type⟩IsIn⟨State⟩(⟨Inst⟩)`.	The state query functions are guaranteed to be verified because the Test Suite has 100% statement coverage.
AST-1	The CORDET Framework shall implement the Application State Machine of figure E.3.	The Application State Machine is implemented in `CrFwAppSm`.	The behaviour of the Application State Machine is verified in table D.2.
AST-3	The CORDET Framework shall provide an API through which applications can query the Application State Machine for its current state.	This API is provided by functions `CrFwAppSmIsInStartUp`, `CrFwAppSmIsInNormal`, `CrFwAppSmIsInReset` and `CrFwAppSmIsInShutdown`.	The state query functions are guaranteed to be verified because the Test Suite has 100% statement coverage.
FAC-1	The factory components shall be provided as extensions of the Base Component.	The InFactory component is created by function `CrFwInFactoryMake` which creates it as an extension of the Base State Machine. Similarly, OutFactory component is created by function `CrFwOutFactoryMake` which creates it as an extension of the Base State Machine.	InFactory creation is verified in the test cases in `CrFwInFactoryTestCase.h` and OutFactory creation is verified in the test cases in `CrFwOutFactoryTestCase.h`.

ID	Requirement Text	Requirement Implementation	Requirement Verification
FAC-2	The factory components shall define an API offering two operations: `Make` and `Release`.	The make operations for the InFactory are implemented in functions `CrFwInFactorMakeInCmd` and `CrFwInFactorMakeInRep`. The release operations are implemented in functions `CrFwInFactorReleaseInCmd` and `CrFwInFactorReleaseInRep`. The make operation for the OutFactory is implemented in function `CrFwOutFactorMakeOutCmp`. The release operation is implemented in function `CrFwOutFactorReleaseOutCmp`.	The make operation for InCommands is verified in test case `CrFwInCmdTestCase1`. The make operation for InReports is verified in test case `CrFwInRepTestCase1`. The make operation for OutComponents is verified in test case `CrFwOutCmpTestCase1`.
FAC-3	The `Make` operation shall either fail and return nothing or succeed and return a component instance of the type specified by the `Make` arguments.	The make operations for the InFactory are implemented in functions `CrFwInFactorMakeInCmd` and `CrFwInFactorMakeInRep`. The make operation for the OutFactory is implemented in function `CrFwOutFactorMakeOutCmp`.	For the InFactory, successful creation is verified in `CrFwInFactoryTestCase1`; unsuccessful creation is verified in `CrFwInFactoryTestCase2` and `CrFwInFactoryTestCase5`. For the OutFactory, successful creation is verified in `CrFwOutFactoryTestCase1`; unsuccessful creation is verified in `CrFwOutFactoryTestCase2`.

ID	Requirement Text	Requirement Implementation	Requirement Verification
FAC-4	The `Release` operation shall take as argument the component instance to be released.	The release operations for the InFactory are implemented in functions `CrFwInFactorReleaseInCmd` and `CrFwInFactorReleaseInRep`. The release operation for the OutFactory is implemented in function `CrFwOutFactorReleaseOutCmp`.	The release functions are guaranteed to be verified because the Test Suite has 100% statement coverage.
OST-1	The CORDET Framework shall provide an OutStream component as an extension of the Base Component.	The OutStream component is created by function `CrFwOutStreamMake` which creates it as an extension of the Base State Machine.	OutStream creation is verified in the test cases in `CrFwOutStreamTestCase.h`.
OST-2	The behaviour of the OutStream component in state CONFIGURED shall be as defined by the *OutStream State Machine* of figure E.4.	The function `CrFwOutStreamMake` builds an instance of an OutStream by first extending a Base State Machine and then embedding within its CONFIGURED state an OutStream State Machine.	The behaviour of the OutStream State Machine is verified in table D.3.
OST-4	The Packet Queue in the OutStream shall be managed as a FIFO queue.	The packet queue of the OutStream uses the implementation of module `CrFwPcktQueue`. Module.	FIFO order management of packets in packet queues is verified in `CrFwPacketQueueTestCase1`.
OST-6	The OutStream shall provide visibility over the state of its Packet Queue (number of packets in the queue and number of empty slots still available).	The number of pending packet is provided by function `CrFwOutStreamGetNOfPendingPckts` and the queue size is provided by function `CrFwOutStreamGetPcktQueueSize`.	The functions to check the number of pending packets and the size of the Packet Queue are verified in `CrFwOutStreamTestCase1` and `CrFwOutStreamTestCase3`.

ID	Requirement Text	Requirement Implementation	Requirement Verification
IST-1	The CORDET Framework shall provide an InStream component as an extension of the Base Component.	The InStream component is created by function `CrFwInStreamMake` which creates it as an extension of the Base State Machine.	InStream creation is verified in the test cases in `CrFwOutStreamTestCase.h`.
IST-2	The behaviour of the InStream component in state CONFIGURED shall be as defined by the *InStream State Machine* of figure E.5 and by the *Packet Collect Procedure* of figure E.6.	The function `CrFwInStreamMake` builds an instance of an InStream by first extending a Base State Machine and then embedding within its CONFIGURED state an InStream State Machine. The Packet Collect Procedure is implemented in function `DoActionB` in `CrFwInStream.h`.	The behaviour of the InStream State Machine is verified in table D.4. The behaviour of the Packet Collect Procedure is verified in table D.9
IST-3	The Packet Queue in the InStream shall be managed as a FIFO queue.	The packet queue of the InStream uses the implementation of module `CrFwPcktQueue`. Module.	FIFO order management of packets in packet queues is verified in `CrFwPacketQueueTestCase1`.
IST-5	The InStream shall provide visibility over the state of its Packet Queue (number of packets in the queue and number of empty slots still available).	The number of pending packet is provided by function `CrFwInStreamGetNOfPendingPckts` and the queue size is provided by function `CrFwInStreamGetPcktQueueSize`.	The function to check the number of pending items is verified in `CrFwInStreamTestCase3`; the function to verify the size of the packet queue is verified in `CrFwInStreamTestCase4`.
OSR-1	The CORDET Framework shall provide an OutStreamRegistry component as an extension of the Base Component.	The OutStreamRegistry function is implemented in the OutStream itself (in function `CrFwOutStreamGet`).	The `CrFwOutStreamGet`) function is verified in test case `CrFwInStreamTestCase4`.

ID	Requirement Text	Requirement Implementation	Requirement Verification
OSR-3	The OutStreamRegistry component shall define an API offering one operation: `OutStreamGet`.	The OutStreamGet operation is implemented by function `CrFwOutStreamGet`.	The `CrFwOutStreamGet`) function is verified in test case `CrFwInStreamTestCase4`.
OSR-4	The `OutStreamGet` operation shall either fail and return nothing, or succeed and return the OutStream component associated to the command or report destination specified in its argument.	The OutStreamGet operation is implemented by function `CrFwOutStreamGet`.	Both kinds of return values for the `CrFwOutStreamGet`) function are verified in test case `CrFwInStreamTestCase4`.
OSR-5	The encoding of the command or report destination passed in a call the `OutStreamGet` operation shall be the same as the encoding of the destination attribute of commands and reports.	The argument of `CrFwOutStreamGet` is of type `CrFwDestSrc_t` and this is the same type as used for a report or command destination in function `CrFwPcktSetDest`.	The `CrFwOutStreamGet`) function is verified in test case `CrFwInStreamTestCase4`.

ID	Requirement Text	Requirement Implementation	Requirement Verification
OCM-1	The CORDET Framework shall provide an OutComponent component as an extension of the Base Component.	The OutComponent components are created by function `CrFwOutFactoryMakeOutCmp` which returns an instance taken from a pool of pre-allocated components. The pre-allocated components are created by function `OutFactoryInitAction` as extensions of a Base OutComponent which is created by function `CrFwOutCmpMakeBase` and which is itself an extension of a Base State Machine.	The function `CrFwOutFactoryMakeOutCmp` is verified in test cases `CrFwOutCmpTestCase1` to `CrFwOutCmpTestCase6`.
OCM-2	The behaviour of the OutComponent in state CONFIGURED shall be as defined by the OutComponent State Machine of figure E.7.	The function `CrFwOutCmpMakeBase` builds the Base OutComponent from which all OutComponents are derived by extending the Base State Machine and then embedding within its CONFIGURED state an OutComponent State Machine.	The behaviour of the OutComponent State Machine is verified in table D.5.
OCM-4	The OutComponent component shall provide access to the attributes of the command or report instance that the OutComponent encapsulates.	The command or report attributes can be accessed through functions with names like: `CrFwOutCmpGet*`.	The getter functions for the command and report attributes are verified in test case `CrFwOutCmpTestCase1`.
OFT-1	The OutFactory component shall encapsulate the instance creation process for OutComponent components.	Instances of OutComponents are created by function `CrFwOutFactoryMakeOutCmp`.	Function `CrFwOutFactoryMakeOutCmp` is verified in test case `CrFwOutFactoryTestCase4`.

ID	Requirement Text	Requirement Implementation	Requirement Verification
OFT-2	The `Make` operation of the OutFactory component shall take as arguments the service type, command or report sub-type and discriminant value of the command or report to be encapsulated by the OutComponent.	Instances of OutComponents are created by function `CrFwOutFactoryMakeOutCmp`.	Function `CrFwOutFactoryMakeOutCmp` is verified in test case `CrFwOutFactoryTestCase4`.
OFT-3	The OutComponents returned by the `Make` operation of the OutFactory shall have their service type, command/report sub-type, and discriminant attribute set in accordance with the value of the arguments of the `Make` operation.	Instances of OutComponents are created by function `CrFwOutFactoryMakeOutCmp`.	The correctness of the type, sub-type and discriminant of a newly created OutComponent is verified in test case `CrFwOutCmpTestCase1`.
OFT-4	The OutComponents returned by the `Make` operation of the OutFactory shall have their identifier attribute set to represent the number of components successfully created by the factory since it was initialized.	Instances of OutComponents are created by function `CrFwOutFactoryMakeOutCmp`.	The correctness of the instance identifier of a newly created OutComponent is verified in test case `CrFwOutCmpTestCase1`.

ID	Requirement Text	Requirement Implementation	Requirement Verification
OLD-1	The CORDET Framework shall provide an OutLoader component as an extension of the Base Component.	The OutLoader component is created by function `CrFwOutLoaderMake` which creates it as an extension of the Base State Machine.	OutLoader creation is verified in the test cases in `CrFwOutLoaderTestCase.h`.
OLD-3	The OutLoader component shall offer a `Load` operation to load an OutComponent instance into an OutManager.	The Load operation is implemented by function `CrFwOutLoaderLoad`.	Function `CrFwOutLoaderLoad` is verified in test case `CrFwOutLoaderTestCase1`.
OLD-4	Execution of the `Load` operation shall cause the *Load Procedure* of figure E.8 to be run.	The Load operation is implemented by function `CrFwOutLoaderLoad`.	The Load Procedure has one single branch which is tested in `CrFwOutLoaderTestCase1`.
OMG-1	The CORDET Framework shall provide an OutManager component as an extension of the Base Component.	The OutManager component is created by function `CrFwOutManagerMake` which creates it as an extension of the Base State Machine.	OutLoader creation is verified in the test cases in `CrFwOutManagerTestCase.h`.
OMG-3	The OutManager component shall offer a `Load` operation to load an OutComponent instance in the POCL.	The Load operation is implemented by function `CrFwOutManagerLoad`.	Function `CrFwOutManagerLoad` is verified in test case `CrFwOutManagerTestCase2`.

ID	Requirement Text	Requirement Implementation	Requirement Verification
OMG-4	The `Load` operation shall run the OutManager Load Procedure of figure E.9.	The OutManager Load Procedure is implemented by function `CrFwOutManagerLoad`.	The Load Procedure has two branches both of which are verified in test case `CrFwOutManagerTestCase2`. The 'POCL Full' branch is also verified in test cases `CrFwOutManagerTestCase3` and `CrFwOutManagerTestCase4`.
ORG-1	The CORDET Framework shall provide an OutRegistry component as an extension of the Base Component.	The OutRegistry component is created by function `CrFwOutRegistryMake` which creates it as an extension of the Base State Machine.	OutRegistry creation is verified in the test cases in `CrFwOutManagerTestCase.h`.
ORG-3	The OutRegistry shall offer a `StartTracking` operation to run the Registry Start Tracking Procedure of figure E.11.	The StartTracking operation is implemented in function `CrFwOutRegistryStartTracking`.	The Registry Start Tracking Procedure has two branches both of which are verified in test case `CrFwOutRegistryTestCase7`.
ORG-4	The OutRegistry shall offer an `Update` operation to run the Registry Update Procedure of figure E.11.	The Update operation is implemented in function `CrFwOutRegistryStartTracking`.	The Registry Update Procedure has two branches both of which are verified in test case `CrFwOutRegistryTestCase7`.

ID	Requirement Text	Requirement Implementation	Requirement Verification
ORG-5	The OutRegistry component shall provide an API through which the state of a command or report in the repository (PENDING, ABORTED, and TERMINATED) can be queried.	The query operation is implemented in function `CrFwOutRegistryGetState`.	The query function is verified in test cases `CrFwOutRegistryTestCase7` to `CrFwOutRegistryTestCase9`. All possible outcomes of the query function (PENDING, NOT_TRACKED, ABORTED and TERMINATED) are verified.
ORG-6	The OutRegistry component shall provide an API through which the enable state of a service type, service sub-type or discriminant value can be set and read.	The set operation is implemented in function `CrFwOutRegistrySetEnable`. The get operation is implemented in function `CrFwOutRegistryIsEnabled`.	Functions `CrFwOutRegistrySetEnable` and `CrFwOutRegistryIsEnabled` are verified in test cases `CrFwOutRegistryTestCase3` to `CrFwOutRegistryTestCase6`.
ORG-7	The OutRegistry component shall provide an API through which the enable state of a specific out-going command or report can be determined in accordance with the logic of the Enable State Determination Procedure of figure E.12.	The logic to determine the enable state of an OutComponent is implemented in function `CrFwOutRegistryIsEnabled`.	The behaviour of the Enable State Determination Procedure e is verified in table D.10.

ID	Requirement Text	Requirement Implementation	Requirement Verification
ORG-8	The OutRegistry shall use the command/report identifier attribute as the key to store and make available information about commands and reports.	The argument of function `CrFwOutRegistryGetState` is the command or report identifier.	The query function is verified in test cases `CrFwOutRegistryTestCase7` to `CrFwOutRegistryTestCase9`.
IFT-1	The InFactory component shall encapsulate the instance creation process for InCommand and InReport components.	Instances of InCommand are created by function `CrFwInFactoryMakeInCmd`. Instances of InReport are created by function `CrFwInFactoryMakeInRep`.	Function `CrFwInFactoryMakeInCmd` is verified in test case `CrFwInCmdTestCase1`. Function `CrFwInFactoryMakeInRep` is verified in test case `CrFwInRepTestCase1`.
IFT-2	The `Make` operation of the InFactory component shall take as arguments the service type, command or report sub-type and discriminant value of the command or report to be encapsulated by the InCommand or InReport.	Instances of InCommand are created by function `CrFwInFactoryMakeInCmd`. Instances of InReport are created by function `CrFwInFactoryMakeInRep`.	Function `CrFwInFactoryMakeInCmd` is verified in test case `CrFwInCmdTestCase1`. Function `CrFwInFactoryMakeInRep` is verified in test case `CrFwInRepTestCase1`.

ID	Requirement Text	Requirement Implementation	Requirement Verification
IFT-3	The InCommands or InReports returned by the `Make` operation of the OutFactory shall have their service type, command/report sub-type, and discriminant attribute set in accordance with the value of the arguments of the Make operation.	Instances of InCommand are created by function `CrFwInFactoryMakeInCmd`. Instances of InReport are created by function `CrFwInFactoryMakeInRep`.	The correctness of the type, sub-type and discriminant of a newly created InCommand is verified in test case `CrFwInCmdTestCase1`. The correctness of the type, sub-type and discriminant of a newly created InReport is verified in test case `CrFwInRepTestCase1`.
ILD-1	The CORDET Framework shall provide an InLoader component as an extension of the Base Component.	The InLoader component is created by function `CrFwInLoaderMake` which creates it as an extension of the Base State Machine.	InLoader creation is verified in the test cases in `CrFwInLoaderTestCase.h`.
ILD-3	The InLoader component shall offer a `Load` operation to load a command or report in an InManager.	The Load operation is implemented by function `CrFwInLoaderLoad`.	Function `CrFwInLoaderLoad` is verified in test cases `CrFwInLoaderTestCase3` to `CrFwInLoaderTestCase11`.
ILD-4	The `Load` operation shall run the InLoader Execution Procedure of figure E.13.	The InLoader Execution Procedure is implemented by function `InLoaderExecAction`.	The behaviour of the InLoader Execution Procedure is verified in table D.11. This procedure uses the InLoader Load Command/Report Procedure which is verified in table D.12.

ID	Requirement Text	Requirement Implementation	Requirement Verification
ICM-1	The CORDET Framework shall provide an InCommand component as an extension of the Base Component to encapsulate an incoming command in a provider application.	The InCommand components are created by function `CrFwInFactoryMakeInCmd` which returns an instance taken from a pool of pre-allocated components. The pre-allocated components are created by function `InFactoryInitAction` as extensions of a Base InCommand which is created by function `CrFwInCmdMakeBase` and which is itself an extension of a Base State Machine.	The function `CrFwInFactoryMakeInCmd` is verified in test case `CrFwInCmdTestCase1`.
ICM-2	The behaviour of the InCommand component in state CONFIGURED shall be as defined by the InCommand State Machine of figure E.15.	The function `CrFwInCmdMakeBase` builds the Base InCommand from which all InCommands are derived by extending the Base State Machine and then embedding within its CONFIGURED state an InCommand State Machine.	The behaviour of the InCommand State Machine is verified in table D.6.
ICM-4	The InCommand component shall provide visibility over the value of all the attributes of the command it encapsulates.	The InCommand attributes can be accessed through functions with names like: `CrFwInCmdGet*`.	The `CrFwInCmdGet*` functions are verified in test case `CrFwInCmdTestCase1`.

ID	Requirement Text	Requirement Implementation	Requirement Verification
IRP-1	The CORDET Framework shall provide an InReport component as an extension of the Base Component to encapsulate an incoming report in a user application.	The InReport components are created by function `CrFwInFactoryMakeInRep` which returns an instance taken from a pool of pre-allocated components. The pre-allocated components are created by function `InFactoryInitAction` as extensions of a Base InCommand which is created by function `CrFwInRepMakeBase` and which is itself an extension of a Base State Machine.	The function `CrFwInFactoryMakeInRep` is verified in test case `CrFwInRepTestCase1`.
IRP-3	The InReport component shall provide visibility over the value of all the attributes of the command it encapsulates.	The InReport attributes can be accessed through functions with names like: `CrFwInRepGet*`.	The `CrFwInRepGet*` functions are verified in test case `CrFwInRepTestCase1`.
IMG-1	The CORDET Framework shall provide an InManager component as an extension of the Base Component.	The InManager component is created by function `CrFwInManagerMake` which creates it as an extension of the Base State Machine.	InManager creation is verified in the test cases in `CrFwInManagerTestCase.h`.
IMG-3	The InManager component shall offer a `Load` operation to load an InCommand or InReport instance in the Pending Command/Report List (PCRL).	The Load operation is implemented by function `CrFwInManagerLoad`.	Function `CrFwInManagerLoad` is verified in test case `CrFwInManagerTestCase2`.

ID	Requirement Text	Requirement Implementation	Requirement Verification
IMG-4	The Load operation shall run the InManager Load Procedure of figure E.17.	The OutManager Load Procedure is implemented by function `CrFwOutManagerLoad`.	The Load Procedure has two branches both of which are verified in test case `CrFwInManagerTestCase2`. The 'PCRL Not Full' branch is also verified in test cases `CrFwOutManagerTestCase3` to `CrFwOutManagerTestCase8`.
IRG-1	The CORDET Framework shall provide an InRegistry component as an extension of the Base Component .	The InRegistry component is created by function `CrFwInRegistryMake` which creates it as an extension of the Base State Machine.	InRegistry creation is verified in the test cases in `CrFwInRegistryTestCase.h`.
IRG-3	The InRegistry shall offer an operation StartTracking to run the Registry Start Tracking Procedure of figure E.11.	The StartTracking operation is implemented in function `CrFwInRegistryStartTracking`.	The Registry Start Tracking Procedure has two branches both of which are verified in test case `CrFwInRegistryTestCase2`.
IRG-4	The InRegistry shall offer an Update operation which runs the Registry Update Procedure of figure E.11.	The Update operation is implemented in function `CrFwInRegistryUpdateState`.	The Registry Update Procedure has two branches both of which are verified in test case `CrFwInRegistryTestCase2`.

ID	Requirement Text	Requirement Implementation	Requirement Verification
IRG-5	The InRegistry component shall provide an API through which the state of a command or report in the repository (PENDING, ABORTED, and TERMINATED) can be queried.	The query operation is implemented in function `CrFwInRegistryGetState`.	The query function is verified in test case `CrFwOutRegistryTestCase2`. All possible outcomes of the query function (PENDING, NOT_TRACKED, ABORTED and TERMINATED) are verified.
IRG-6	The InRegistry shall use the command/report identifier attribute as the key to store and make available information about commands and reports.	The argument of function `CrFwInRegistryGetState` is the command or report identifier.	The query function is verified in test case `CrFwInRegistryTestCase2`.

B CORDET Framework Adaptation Points

The C2 Implementation implements the adaptation points of the CORDET Framework. Table B.1 lists the adaptation points of the CORDET Framework as they are defined in reference [4]. The requirement identifier (first column in the table) is the same as used in reference [4].

For each adaptation point, the last column in the table either describes how the adaptation point is implemented in the C2 Implementation or it explains why it is not directly implemented. The latter is the case for the following kinds of adaptation points:

- Adaptation points which are "closed at framework level" and which are therefore not present in the C2 Implementation. This is the case when component B is derived (through the extension mechanism of the FW Profile) from component A and component A has defined an adaptation point which is inherited by component B but which component B closes (i.e. the value of the adaptation point on component B is fixed and cannot be modified by users of the framework). Thus, for instance, the Execution Procedure is an adaptation point for the Base Component (BAS-6) but the framework components which are derived from the Base Component have a specific Execution Procedure which is not intended to be modified by application developers and therefore "close" the adaptation point BAS-6 defined on the Base Component.
- Adaptation points which have been mapped to other adaptation points which are specific to the C2 Implementation. In these cases, the last column in the table identifies the C2 Implementation adaptation point (the full list of C2 Implementation Points is provided in appendix C).
- Adaptation points which are closed as a result of the design choices made by the C2 Implementation. This is, for instance, the case for the adaptation points for the OutStreamRegistry. In the C2 Implementation, this component has been merged with the OutStream component and hence its adaptation points have been merged with those of the OutStream.

The definitions of the CORDET adaptation points often refer to state machine or procedure diagrams. A complete list of the diagrams of the state machines and procedures which define the behaviour of the CORDET components can be found in section E.

Table B.1: CORDET Adaptation Points

AP ID	Adaptation Point	Default Value	Implementation
BAS-1	Initialization Check in Initialization Procedure of Base Component	Always returns: 'check successful'	The Base Component is not available for direct use by application developers. This Adaptation Point is therefore not directly supported by the C2 Implementation but, where required, is supported by components which are derived from the Base Component.
BAS-2	Initialization Action in Initialization Procedure of Base Component	Do nothing and return: 'action successful'	The Base Component is not available for direct use by application developers. This Adaptation Point is therefore not directly supported by the C2 Implementation but, where required, is supported by components which are derived from the Base Component.
BAS-3	Configuration Check in Reset Procedure of Base Component	Always returns: 'check successful'	The Base Component is not available for direct use by application developers. This Adaptation Point is therefore not directly supported by the C2 Implementation but, where required, is supported by components which are derived from the Base Component.
BAS-4	Configuration Action in Reset Procedure of Base Component	Do nothing and return: 'action successful'	The Base Component is not available for direct use by application developers. This Adaptation Point is therefore not directly supported by the C2 Implementation but, where required, is supported by components which are derived from the Base Component.

AP ID	Adaptation Point	Default Value	Implementation
BAS-5	Shutdown Action of Base Component	Do nothing	The Base Component is not available for direct use by application developers. This Adaptation Point is therefore not directly supported by the C2 Implementation but, where required, is supported by components which are derived from the Base Component.
BAS-6	Execution Procedure of Base Component	Do the same dummy action (return without doing anything) whenever the procedure is executed	The Base Component is not available for direct use by application developers. This Adaptation Point is therefore not directly supported by the C2 Implementation but, where required, is supported by components which are derived from the Base Component.
AST-1	Application Start-Up Procedure	No default provided at framework level	Implementation of `CrFwAppStartUpProc.h`. Only a test stub is provided as default at framework level.
AST-2	Application Reset Procedure	No default provided at framework level	Implementation of `CrFwAppResetProc.h`. Only a test stub is provided as default at framework level.
AST-3	Application Shutdown Procedure	No default provided at framework level	Implementation of `CrFwAppShutdownProc.h`. Only a test stub is provided as default at framework level.
AST-4	State Machine Embedded in state START_UP of Application State Machine	No state machine embedded in state START_UP	`#DEFINE` constant in `CrFwAppSmUserPar.h`
AST-5	State Machine Embedded in state NORMAL of Application State Machine	No state machine embedded in state NORMAL	`#DEFINE` constant in `CrFwAppSmUserPar.h`

AP ID	Adaptation Point	Default Value	Implementation
AST-6	State Machine Embedded in state RESET of Application State Machine	No state machine embedded in state RESET	`#DEFINE` constant in `CrFwAppSmUserPar.h`
AST-7	State Machine Embedded in state SHUTDOWN of Application State Machine	No state machine embedded in state SHUTDOWN	`#DEFINE` constant in `CrFwAppSmUserPar.h`
FAC-1	`Make` Operation to dynamically instantiate a component	No default provided at framework level	The only components which can be instantiated dynamically are report and command components. Their `make` operations are implemented in full (see `CrFwInFactoryMake*` and `CrFwOutFactoryMakeOutCmp` functions). This adaptation point is therefore closed by the C2 Implementation.
FAC-2	`Release` Operation to dynamically release a component	No default provided at framework level	The only components which can be released dynamically are report and command components. Their `release` operations are implemented in full (see `CrFwInFactoryRelease*` and `CrFwOutFactoryReleaseOutCmp` functions). This adaptation point is therefore closed by the C2 Implementation.
OST-1	Packet Queue Size for OutStream	No value defined at framework level	`#DEFINE` constant (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`

AP ID	Adaptation Point	Default Value	Implementation
OST-2	Initialization Check in Initialization Procedure of OutStream	Returns 'check successful' if the size of the Packet Queue has been set to a positive integer	Function pointers (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`. Default implementation is provided in `CrFwOutStream.h`..
OST-3	Initialization Action in Initialization Procedure of OutStream	Allocate resources for Packet Queue and return 'Action Successful' iff the allocation succeeds	Function pointers (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`. Default implementation is provided in `CrFwOutStream.h`..
OST-4	Configuration Check in Initialization Procedure of OutStream	Same value as in Base Component	Function pointers (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`. Default implementation is provided in `CrFwOutStream.h`..
OST-5	Configuration Action in Reset Procedure of OutStream	Reset the Packet Queue and return 'Action Successful'	Function pointers (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`. Default implementation is provided in `CrFwOutStream.h`..
OST-6	Shutdown Action of OutStream	Reset the Packet Queue	Function pointers (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`. Default implementation is provided in `CrFwOutStream.h`..
OST-7	Execution Procedure of OutStream (closes BAS-6)	Same value as in Base Component	This Adaptation Point is closed at framework level.
OST-8	Packet Hand-Over Operation of OutStream	No value defined at framework level	Function pointers (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`. Only a test stub is provided as default at framework level.

AP ID	Adaptation Point	Default Value	Implementation
OST-9	Operation to set Sequence Counter in Outgoing Packets	No value defined at framework level	Implemented by Adaptation Point C2-PCK-1.
OST-12	Operation to Report Packet Queue Full	Generate OUTSTREAM_PQ_FULL Error Report	Implemented by Adaptation Point C2-ERR-1.
IST-1	Size of the Packet Queue in InStream	Default size is 1	`#DEFINE` constant (one for each InStream in the application) in `CrFwInStreamUserPar.h`
IST-2	Initialization Check in Initialization Procedure of InStream	Returns 'check successful' if the size of the Packet Queue has been set to a positive integer	Function pointers (one for each InStream in the application) in `CrFwInStreamUserPar.h`. Default implementation is provided in `CrFwInStream.h`.
IST-3	Initialization Action in Initialization Procedure of InStream	Allocate resources for Packet Queue and return 'Action Successful' iff the allocation succeeds	Function pointers (one for each InStream in the application) in `CrFwInStreamUserPar.h`. Default implementation is provided in `CrFwInStream.h`.
IST-4	Configuration Action in Reset Procedure of InStream	Reset the Packet Queue and return 'Action Successful'	Function pointers (one for each InStream in the application) in `CrFwInStreamUserPar.h`. Default implementation is provided in `CrFwInStream.h`.
IST-5	Shutdown Action of InStream	Reset the Packet Queue	Function pointers (one for each InStream in the application) in `CrFwInStreamUserPar.h`. Default implementation is provided in `CrFwInStream.h`.
IST-6	Execution Procedure of InStream (closes BAS-6)	Same value as in Base Component	This Adaptation Point is closed at framework level.

AP ID	Adaptation Point	Default Value	Implementation
IST-7	Operation to Get Packet Source from Incoming Packet	No value defined at framework level	Implemented by Adaptation Point C2-PCK-1.
IST-8	Operation to Get Packet Sequence Counter from Incoming Packet	No value defined at framework level	Implemented by Adaptation Point C2-PCK-1.
IST-9	Operation to Report Sequence Counter Error	Generate INSTREAM_SC_ERR Error Report with expected and actual sequence counter values	Implemented by Adaptation Point C2-ERR-1.
IST-10	Operation to Report Packet Queue Full	Generate INSTREAM_PQ_FULL Error Report	Implemented by Adaptation Point C2-ERR-1.
IST-11	Packet Collect Operation for InStream	No default defined at framework level	Function pointers (one for each InStream in the application) in `CrFwInStreamUserPar.h`. Only a test stub is provided as default at framework level.
IST-12	Packet Available Check Operation for InStream	No default defined at framework level	Function pointers (one for each InStream in the application) in `CrFwInStreamUserPar.h`. Only a test stub is provided as default at framework level.
OSR-1	Initialization Check in Initialization Procedure of OutStreamRegistry	Same value as in Base Component	In the C2 Implementation, the OutStreamRegistry is not implemented as a separate component (it is merged with the OutStream). This adaptation point is closed in the C2 Implementation.

AP ID	Adaptation Point	Default Value	Implementation
OSR-2	Initialization Action in Initialization Procedure of OutStreamRegistry	Same value as in Base Component	In the C2 Implementation, the OutStreamRegistry is not implemented as a separate component (it is merged with the OutStream). This adaptation point is closed in the C2 Implementation.
OSR-3	Configuration Check in Reset Procedure of OutStreamRegistry	Returns 'check successful' if the information to set up the link between the packet destinations and the OutStreams is available.	In the C2 Implementation, the OutStreamRegistry is not implemented as a separate component (it is merged with the OutStream). This adaptation point is closed in the C2 Implementation.
OSR-4	Configuration Action in Reset Procedure of OutStreamRegistry	Set up and configure the link between the packet destinations and the OutStreams.	In the C2 Implementation, the OutStreamRegistry is not implemented as a separate component (it is merged with the OutStream). This adaptation point is closed in the C2 Implementation.
OSR-5	Shutdown Action of OutStreamRegistry (closes BAS-5)	Same value as in Base Component	In the C2 Implementation, the OutStreamRegistry is not implemented as a separate component (it is merged with the OutStream). This adaptation point is closed in the C2 Implementation.
OSR-6	Execution Procedure of OutStreamRegistry (closes BAS-6)	Same value as in Base Component	In the C2 Implementation, the OutStreamRegistry is not implemented as a separate component (it is merged with the OutStream). This adaptation point is closed in the C2 Implementation.
OSR-7	Get OutStream Operation of OutStreamRegistry	No default provided at framework level	`#DEFINE` constants (one for each OutStream in the application) in `CrFwOutStreamUserPar.h` define the destination associated to each OutStream.

AP ID	Adaptation Point	Default Value	Implementation
OCM-1	Initialization Check in Initialization Procedure of OutComponent	Same value as in Base Component	OutComponents are provided by the OutFactory in the CONFIGURED state and cannot therefore be initialized and configured by the user. This adaptation point is closed in the C2 Implementation.
OCM-2	Initialization Action in Initialization Procedure of OutComponent	Same value as in Base Component	OutComponents are provided by the OutFactory in the CONFIGURED state and cannot therefore be initialized and configured by the user. This adaptation point is closed in the C2 Implementation.
OCM-3	Configuration Check in Reset Procedure of OutComponent	Same value as in Base Component	OutComponents are provided by the OutFactory in the CONFIGURED state and cannot therefore be initialized and configured by the user. This adaptation point is closed in the C2 Implementation.
OCM-4	Configuration Action in Reset Procedure of OutComponent	Same value as in Base Component	OutComponents are provided by the OutFactory in the CONFIGURED state and cannot therefore be initialized and configured by the user. This adaptation point is closed in the C2 Implementation.
OCM-5	Shutdown Action in Base Component of OutComponent	Same value as in Base Component	OutComponents are provided by the OutFactory in the CONFIGURED state and are not intended to be ever shut down. This adaptation point is closed in the C2 Implementation.
OCM-6	Execution Procedure of OutComponent (closes BAS-6)	Same value as in Base Component	The OutComponents are not intended to be ever executed. This adaptation point is closed in the C2 Implementation.

AP ID	Adaptation Point	Default Value	Implementation
OCM-7	Service Type Attribute of OutComponent	No default provided at framework level	This Adaptation Point is implemented by adaptation point OFA-2.
OCM-8	Command/Report Sub-Type Attribute of OutComponent	No default provided at framework level	This Adaptation Point is implemented by adaptation point OFA-2.
OCM-9	Destination Attribute of OutComponent	No default provided at framework level	This Adaptation Point is implemented by adaptation point OFA-2.
OCM-10	Acknowledge Level Attribute of OutComponent	Default value is: 'no acknowledge required' (only relevant for OutCommands)	This Adaptation Point is implemented by adaptation point C2-PCK-1.
OCM-11	Discriminant Attribute of OutComponent	Default value is: 'no discriminant'	This Adaptation Point is implemented by adaptation point OFA-2.
OCM-12	Parameter Attribute of OutComponent	Default value is: 'no parameters'	This Adaptation Point is implemented indirectly: applications must extend OutComponents and must define the range of parameters for each OutComponent and the operations to set their values.
OCM-13	Enable Check Operation of OutComponent	Query the OutRegistry for the enable status of the command or report encapsulated in the OutComponent and set value of isEnable accordingly	`#DEFINE` constants (one for each kind of OutComponent in the application) in `CrFwOutFactoryUserPar.h` define the pointer to the function implementing the operation. A default is provided a framework level.

AP ID	Adaptation Point	Default Value	Implementation
OCM-14	Ready Check Operation of OutComponent	Set value of isReady flag to true	`#DEFINE` constants (one for each kind of OutComponent in the application) in `CrFwOutFactoryUserPar.h` define the pointer to the function implementing the operation. A default is provided a framework level.
OCM-15	Repeat Check Operation of OutComponent	Return "No Repeat"	`#DEFINE` constants (one for each kind of OutComponent in the application) in `CrFwOutFactoryUserPar.h` define the pointer to the function implementing the operation. A default is provided a framework level.
OCM-16	Update Action of OutComponent	Set Time Stamp of OutComponent to current time	`#DEFINE` constants (one for each kind of OutComponent in the application) in `CrFwOutFactoryUserPar.h` define the pointer to the function implementing the operation. A default is provided a framework level.
OCM-17	Serialize Operation of OutComponent	No default defined at framework level	`#DEFINE` constants (one for each kind of OutComponent in the application) in `CrFwOutFactoryUserPar.h` define the pointer to the function implementing the operation. A default is provided a framework level.
OCM-18	Operation to Report Invalid Destination of an OutComponent	Generate SNDPCKT_INV_DEST Error Report with invalid destination as a parameter	Implemented by Adaptation Point C2-ERR-1.

AP ID	Adaptation Point	Default Value	Implementation
OLD-1	Initialization Check in Initialization Procedure of OutLoader	Returns 'check successful' if the size of the LOM (List of OutManagers) has been set to a positive integer value.	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which always returns 'check successful' is provided in `CrFwOutLoader.h`.
OLD-2	Initialization Action in Initialization Procedure of OutLoader	Allocate resources for LOM and return 'Action Successful' iff the allocation succeeds	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which takes no action is provided in `CrFwOutLoader.h`.
OLD-3	Configuration Check in Reset Procedure of OutLoader	Returns 'check successful' iff all the information is available to update (or initialize) the value of the LOM.	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which always returns 'check successful' is provided in `CrFwOutLoader.h`.
OLD-4	Configuration Action in Reset Procedure of OutLoader	Update (or initialize) the LOM and return 'Action Successful'	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which takes no action is provided in `CrFwOutLoader.h`.
OLD-5	Shutdown Action of OutLoader	Same as in Base Component.	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which takes no action is provided in `CrFwOutLoader.h`.
OLD-6	Execution Procedure of OutLoader (closes BAS-6)	Same as in Base Component.	This Adaptation Point is closed at framework level.
OLD-7	OutManager Selection Operation	Select the first OutManager in the LOM	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which always returns the first OutManager in the LOM is provided in `CrFwOutLoader.h`.

AP ID	Adaptation Point	Default Value	Implementation
OLD-8	OutManager Activation Operation	Do nothing	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which takes no action is provided in `CrFwOutLoader.h`.
OLD-9	Operation to set Set Time-Stamp in Outgoing Packets	No value defined at framework level	Implemented by Adaptation Point C2-PCK-1.
OMG-1	Size of POCL of OutManager	Default size is 1.	`#DEFINE` constants (one for each OutManager) in `CrFwOutManagerUserPar.h`
OMG-2	Initialization Check in Initialization Procedure of OutManager (closes BAS-1)	Returns 'check successful' if the size of the POCL has been set to a positive integer value.	This Adaptation Point is closed at framework level.
OMG-3	Initialization Action in Initialization Procedure of OutManager (closes BAS-2)	Allocate resources for POCL and return 'Action Successful' iff the allocation succeeds	This Adaptation Point is closed at framework level.
OMG-4	Configuration Check in Reset Procedure of OutManager (closes BAS-3)	Same as in Base Component	This Adaptation Point is closed at framework level.
OMG-5	Configuration Action in Reset Procedure (closes BAS-4)	Release all OutComponents in the POCL; reset the POCL; reset the counter of successfully loaded OutComponents; and return 'Action Successful'	This Adaptation Point is closed at framework level.

AP ID	Adaptation Point	Default Value	Implementation
OMG-6	Shutdown Action in Base Component of OutManager (closes BAS-5)	Release all OutComponents in the POCL; reset the POCL	This Adaptation Point is closed at framework level.
OMG-7	Execution Procedure in Base Component of OutManager (closes BAS-6)	Implemented as procedure of Manager Execution Procedure	This Adaptation Point is closed at framework level.
OMG-8	Operation to Report POCL of OutManager Full	Generate OUTMANAGER_POCL_FULL Error Report	Implemented by Adaptation Point C2-ERR-1.
ORG-1	Maximum Number of Trackable Commands/Reports for OutRegistry	Default value is 1.	`#DEFINE` constant in `CrFwOutRegistryUserPar.h` defines types, sub-types and range of discriminant values supported by application.
ORG-2	Initialization Check in Initialization Procedure of OutRegistry (closes BAS-1)	Returns 'check successful' if the maximum number of trackable commands/reports has been set to a positive integer value.	This Adaptation Point is closed at framework level.
ORG-3	Initialization Action in Initialization Procedure of OutRegistry (closes BAS-2)	Allocate the resources for tracking the commands and reports and returns: 'action successful' if the allocation succeeds or 'action failed' if the allocation fails.	This Adaptation Point is closed at framework level.

AP ID	Adaptation Point	Default Value	Implementation
ORG-4	Configuration Check in Reset Procedure of OutRegistry (closes BAS-3)	Same value as in Base Component	This Adaptation Point is closed at framework level.
ORG-5	Configuration Action in Reset Procedure of OutRegistry (closes BAS-4)	Set the enable state for all kinds of commands and reports to: 'enabled'; clear all information about tracked commands and reports; and return: 'action successful'.	This Adaptation Point is closed at framework level.
ORG-6	Shutdown Action of OutRegistry (closes BAS-5)	Set the enable state for all kinds of commands and reports to: 'enabled'; clear all information about tracked commands and reports.	This Adaptation Point is closed at framework level.
ORG-7	Execution Procedure of OutRegistry (closes BAS-6)	Same value as in Base Component	This Adaptation Point is closed at framework level.
ILD-1	Initialization Check in Initialization Procedure of InLoader (closes BAS-1)	Return "check successful' iff the sizes of the LIM is a positive integer	This Adaptation Point is closed at framework level.
ILD-2	Initialization Action in Initialization Procedure of InLoader (closes BAS-2)	Allocate resources for the LIM and return "Action Successful' iff the allocation succeeds	This Adaptation Point is closed at framework level.

AP ID	Adaptation Point	Default Value	Implementation
ILD-3	Configuration Check in Reset Procedure of InLoader (closes BAS-3)	Returns "check successful' if: (a) the information to update (or initialize) the content of the LIM is valid; and (b) the information to re0route packets is valid.	This Adaptation Point is closed at framework level.
ILD-4	Configuration Action in Reset Procedure of InLoader (closes BAS-4)	(a) update (or initialize) content of LIM; and (b) update (or initialize) packet re0routing information.	This Adaptation Point is closed at framework level.
ILD-5	Shutdown Action of InLoader (closes BAS-5)	Same as in Base Component.	This Adaptation Point is closed at framework level.
ILD-6	Execution Procedure of InLoader (closes BAS-6)	Implemented as InLoader Execution Procedure.	This Adaptation Point is closed at framework level.
ILD-7	Size of List of InManagers in InLoader	Default size is 2.	The InLoader of the C2 Implementation does not explicitly define a List of InManager. This Adaptation Point is subsumed in the Adaptation Point for the selection of InManager (C2-ILD-TBD)
ILD-8	Content of List of InManagers in InLoader	No default provided at framework level.	The InLoader of the C2 Implementation does not explicitly define a List of InManager. It only defines the function to return the InManager where the InReport or InCommand must be loaded (see C2-ILD-2).

AP ID	Adaptation Point	Default Value	Implementation
ILD-9	Operation to Determine Re-Routing Destination of Packets	Re0routing destination is set to the destination of the incoming packet.	Function pointer in `CrFwInLoaderUserPar.h`. Default implementation is provided `CrFwInLoader.h`.
ILD-10	Operation to Get Packet Destination	No default provided at framework level.	Implemented by Adaptation Point C2-PCK-1.
ILD-11	Operation to Check Packet Destination Validity	Always returns "destination is valid'.	The check of the destination validity is performed by the function which returns the re-routing destination.
ILD-12	Operation to Report Packet Destination Invalid	Generate error report INLOADER_INV_DEST with the destination identifier as a parameter	Implemented by Adaptation Point C2-ERR-1.
ILD-13	Operation to Get Packet Type	No default provided at framework level	Implemented by Adaptation Point C2-PCK-1.

AP ID	Adaptation Point	Default Value	Implementation
ILD-14	Operation to Report Acceptance Failure	For InCommands: generate command acknowledge report CMD_ACK_ACC_FAIL with command's identifier and with identifier of reason of failure as parameters.. For InReports: generate error report INLOADER_ACC_FAIL with report's identifier and with identifier of reason of acceptance failure as parameters.	Implemented by Adaptation Point C2-ACK-1.
ILD-15	Operation to Report Acceptance Success	Generate command acknowledge report CMD_ACK_ACC_SUCC with command's identifier as parameter.	Implemented by Adaptation Point C2-ACK-1.
ILD-16	Operation to Deserialize Packet	No default provided at framework level.	Packets are not deserialized in the C2 Implementation. Instead, the packet itself is attached to the component encapsulating the incoming report or command. This adaptation point is closed in the C2 Implementation.
ILD-17	Operation to Select InManager where Incoming Report or Command is Loaded	For InCommands, select first InManager in LIM; for InReport, select second InManager in LIM.	Function pointer in `CrFwInLoaderUserPar.h`. Default implementation is provided `CrFwInLoader.h`.

AP ID	Adaptation Point	Default Value	Implementation
ILD-18	Operation to Check Packet Type Validity	No default provided at framework level	This check is implemented in function *CrFwInFactoryMakeInCmd* for incoming command and in function *CrFwInFactoryMakeInRep* for incoming report. These functions check that the type is supported by the application.
ICM-1	Initialization Check in Initialization Procedure of InCommand	Returns "check successful' if information for initializing InCommand using data in incoming packet is valid	This Adaptation Point is closed in the C2 Implementation because InCommands are provided by the InFactory in the CONFIGURED state (but a validity check is provided in C2-ICM-1 to implement the acceptance check).
ICM-2	Initialization Action in Initialization Procedure of InCommand	Use information in incoming packet to initialize InCommand and return "action successful'	This Adaptation Point is closed in the C2 Implementation because InCommands are provided by the InFactory in the CONFIGURED state (but a validity check is provided in C2-ICM-1 to implement the acceptance check).
ICM-3	Configuration Check in Reset Procedure of InCommand	Returns "check successful' if information for configuring InCommand using data in incoming packet is valid	This Adaptation Point is closed in the C2 Implementation because InCommands are provided by the InFactory in the CONFIGURED state (but a validity check is provided in C2-ICM-1 to implement the acceptance check).

AP ID	Adaptation Point	Default Value	Implementation
ICM-4	Configuration Action in Reset Procedure of InCommand	Use information in incoming packet to configure InCommand and return "action successful'	This Adaptation Point is closed in the C2 Implementation because InCommands are provided by the InFactory in the CONFIGURED state (but a validity check is provided in C2-ICM-1 to implement the acceptance check).
ICM-5	Shutdown Action of InCommand (closes BAS-5)	Same value as in Base Component	This Adaptation Point is closed at framework level.
ICM-6	Execution Procedure of InCommand (closes BAS-6)	Same value as in Base Component	This Adaptation Point is closed at framework level.
ICM-7	Ready Check of InCommand	Return "command is ready'	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided by function `CrFwSmCheckAlwaysTrue`.
ICM-8	Start Action of InCommand	Set action outcome to "success'	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided by function `CrFwSmEmptyAction`.
ICM-9	Progress Action of InCommand	Set action outcome to "completed'	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided by function `CrFwSmEmptyAction`.
ICM-10	Termination Action of InCommand	Set action outcome to "success'	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided by function `CrFwSmEmptyAction`.

AP ID	Adaptation Point	Default Value	Implementation
ICM-11	Abort Action of InCommand	Do nothing	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided by function `CrFwSmEmptyAction`.
ICM-12	Operation to Report Start Failed for InCommand	Generate command acknowledge report CMD_ACK_STR_FAIL with command's identifier and with identifier of reason of failure as parameters.	Implemented by Adaptation Point C2-ACK-1.
ICM-13	Operation to Report Start Successful for InCommand	Generate command acknowledge report CMD_ACK_STR_SUCC with command's identifier as parameter.	Implemented by Adaptation Point C2-ACK-1.
ICM-14	Operation to Report Progress Failed for InCommand	Generate command acknowledge report CMD_ACK_PRG_FAIL with command's identifier, progress step and with identifier of reason of failure as parameters.	Implemented by Adaptation Point C2-ACK-1.
ICM-15	Operation to Report Progress Successful for InCommand	Generate command acknowledge report CMD_ACK_PRG_SUCC with command's identifier and progress step as parameters.	Implemented by Adaptation Point C2-ACK-1.

AP ID	Adaptation Point	Default Value	Implementation
ICM-16	Operation to Report Termination Failed for InCommand	Generate command acknowledge report CMD_ACK_TRM_FAIL with command's identifier and with identifier of reason of failure as parameters.	Implemented by Adaptation Point C2-ACK-1.
ICM-17	Operation to Report Report Termination Successful for InCommand	Generate command acknowledge report CMD_ACK_TRM_FAIL with command's identifier as parameter.	Implemented by Adaptation Point C2-ACK-1.
ICM-18	Service Type Attribute of InCommand	No default provided at framework level	Implemented by Adaptation Point C2-IFA-4.
ICM-19	Command Sub-Type Attribute of InCommand	No default provided at framework level	Implemented by Adaptation Point C2-IFA-4.
ICM-20	Discriminant Attribute of InCommand	Default value is: "no discriminant'	Implemented by Adaptation Point C2-IFA-4.
ICM-21	Parameter Attributes of InCommand	Default value is: "no parameters'	This Adaptation Point is implemented indirectly: applications must extend InCommands and must define the range of parameters for each kind of InCommand and the operations to get their values.

AP ID	Adaptation Point	Default Value	Implementation
IRP-1	Initialization Check in Initialization Procedure of InReport	Returns "check successful' if information for initializing InReport using data in incoming packet is valid	This Adaptation Point is closed by the C2 Implementation because InReports are provided by the InFactory in the CONFIGURED state (but a validity check is provided in C2-ICM-1 to implement the acceptance check).
IRP-2	Initialization Action in Initialization Procedure of InReport	Use information in incoming packet to initialize InReport and return "action successful'	This Adaptation Point is closed by the C2 Implementation because InReports are provided by the InFactory in the CONFIGURED state (but a validity check is provided in C2-ICM-1 to implement the acceptance check).
IRP-3	Configuration Check in Reset Procedure of InReport	Returns "check successful' if information for configuring InReport using data in incoming packet is valid	This Adaptation Point is closed by the C2 Implementation because InReports are provided by the InFactory in the CONFIGURED state (but a validity check is provided in C2-ICM-1 to implement the acceptance check).
IRP-4	Configuration Action in Reset Procedure of InReport	Use information in incoming packet to configure InReport and return "action successful'	This Adaptation Point is closed by the C2 Implementation because InReports are provided by the InFactory in the CONFIGURED state (but a validity check is provided in C2-ICM-1 to implement the acceptance check).
IRP-5	Shutdown Action of InReport (closes BAS-5)	Same value as in Base Component	This Adaptation Point is closed at framework level.

AP ID	Adaptation Point	Default Value	Implementation
IRP-6	Execution Procedure of InReport (closes BAS-6)	Same value as in Base Component	This Adaptation Point is closed at framework level.
IRP-7	Update Action of InReport	Do nothing	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided at framework level.
IRP-8	Service Type Attribute of InReport	No default provided at framework level	Implemented by Adaptation Point C2-IFA-3.
IRP-9	Sub-Type Attribute of InReport	No default provided at framework level	Implemented by Adaptation Point C2-IFA-3.
IRP-10	Discriminant Attribute of InReport	Default value is: "no discriminant'	Implemented by Adaptation Point C2-IFA-3.
IRP-11	Parameter Attribute of InReport	Default value is: "no parameters'	This Adaptation Point is implemented indirectly: applications must extend InCommands and must define the range of parameters for each kind of InCommand and the operations to get their values.
IMG-1	Size of PCRL of InManager	Default size is 1.	`#DEFINE` constants (one for each InManager) in `CrFwInManagerUserPar.h`
IMG-2	Initialization Check in Initialization Procedure of InManager (closes BAS-1)	Returns "check successful' if the size of the PCRL has been set to a positive integer value.	This Adaptation Point is closed at framework level.
IMG-3	Initialization Action in Initialization Procedure of InManager (closes BAS-2)	Allocate resources for PCRL and return "Action Successful' iff the allocation succeeds	This Adaptation Point is closed at framework level.

AP ID	Adaptation Point	Default Value	Implementation
IMG-4	Configuration Check in Reset Procedure of InManager (closes BAS-3)	Same as in Base Component	This Adaptation Point is closed at framework level.
IMG-5	Configuration Action in Reset Procedure of InManager (closes BAS-4)	Release all InCommands and InReports in the PCRL; reset the counter of successfully loaded InCommands and InReports; reset the PCRL; and return "Action Successful'	This Adaptation Point is closed at framework level.
IMG-6	Shutdown Action of InManager (closes BAS-5)	Release all InCommands and InReports in the PCRL; reset the PCRL;	This Adaptation Point is closed at framework level.
IMG-7	Execution Procedure of InManager (closes BAS-6)	Implemented as InManager Execution Procedure.	This Adaptation Point is closed at framework level.
IMG-8	Operation to Report PCRL of InManager Full	Generate INMANAGER_PCRL_FULL Error Report	Implemented by Adaptation Point C2-ERR-1.
IRG-1	Maximum Number of Trackable InCommands/InReports in InRegistry	Default value is 1.	`#DEFINE` constant in `CrFwInRegistryUserPar.h`

AP ID	Adaptation Point	Default Value	Implementation
IRG-2	Initialization Check in Initialization Procedure of InRegistry (closes BAS-1)	Returns "check successful' if the maximum number of trackable InCommands/InReports has been set to a positive integer value.	This Adaptation Point is closed at framework level.
IRG-3	Initialization Action in Initialization Procedure of InRegistry (closes BAS-2)	Allocate the resources for tracking the commands and reports and returns: "action successful' if the allocation succeeds or "action failed' if the allocation fails.	This Adaptation Point is closed at framework level.
IRG-4	Configuration Check in Reset Procedure of InRegistry (closes BAS-3)	Same value as in Base Component	This Adaptation Point is closed at framework level.
IRG-5	Configuration Action in Reset Procedure (closes BAS-4)	Clear all information about tracked InCommands and InReports; return: "action successful'.	This Adaptation Point is closed at framework level.
IRG-6	Shutdown Action of InRegistry (closes BAS-5)	Clear all information about tracked InCommands and InReports.	This Adaptation Point is closed at framework level.
IRG-7	Execution Procedure of InRegistry (closes BAS-6)	Same value as in Base Component	This Adaptation Point is closed at framework level.

C C2 Adaptation Points

The C2 Implementation implements the adaptation points of the CORDET Framework. Table B.1 lists the adaptation points of the CORDET Framework as they are defined in reference [4] and, for each adaptation point, it describes how the adaptation point is implemented in the C2 Implementation.

Besides the adaptation points defined by the CORDET Framework, the C2 Implementation also provides a number of additional adaptation points which arise as a result of the design choices made for the C2 Implementation. These adaptaton points are called *C2 Adaptation Points* and are listed in table C.1. The last column in the table shows how each adaptation point is implemented and which default value (if any) is offered for it by the C2 Implementation.

The definitions of the C2 adaptation points often refer to state machine or procedure diagrams. A complete list of the diagrams of the state machines and procedures which define the behaviour of the CORDET components can be found in section E.

Table C.1: C2 Adaptation Points

AP ID	Adaptation Point	Implementation
C2-CST-1	Identifier of Host Application	`#DEFINE` constant in `CrFwUserConstants.h`
C2-CST-2	Range of Service Type, Sub-Type and Discriminants for In-Commands and InReports	`#DEFINE` constants in `CrFwUserConstants.h`
C2-AST-1	Application Start-Up Procedure	Implementation of `CrFwAppStartUpProc.h`. Only a test stub is provided as default at framework level.
C2-AST-2	Application Reset Procedure	Implementation of `CrFwAppResetProc.h`. Only a test stub is provided as default at framework level.
C2-AST-3	Application Shutdown Procedure	Implementation of `CrFwAppShutdownProc.h`. Only a test stub is provided as default at framework level.
C2-AST-4	State Machine Embedded in state START_UP of Application State Machine	`#DEFINE` constant in `CrFwAppSmUserPar.h`
C2-AST-5	State Machine Embedded in state NORMAL of Application State Machine	`#DEFINE` constant in `CrFwAppSmUserPar.h`
C2-AST-6	State Machine Embedded in state RESET of Application State Machine	`#DEFINE` constant in `CrFwAppSmUserPar.h`

AP ID	Adaptation Point	Implementation
C2-AST-7	State Machine Embedded in state SHUTDOWN of Application State Machine	`#DEFINE` constant in `CrFwAppSmUserPar.h`
C2-PCK-1	Operations to Set and Get the Values of Command and Report Attributes in a Packet	Implementation of `CrFwPckt.h`. Only a test stub is provided as default at framework level.
C2-ERR-1	Operations to Report Errors Values of Command and Report Attributes in a Packet	Implementation of `CrFwRepErr.h`. Only a test stub is provided as default at framework level.
C2-OFA-1	OutFactory Capacity	`#DEFINE` constant in `CrFwOutFactoryUserPar.h` defines maximum number of OutComponents which can be allocated by the factory.
C2-OFA-2	OutComponent Kinds	`#DEFINE` constants in `CrFwOutFactoryUserPar.h` define the kinds of OutComponents supported by the application. An OutComponent kind is defined through its service type, command or report sub-type, and discriminant value. For each supported OutComponent kind, function pointers are defined implementing the OutComponent checks and actions.
C2-IFA-1	InFactory Capacity for InReports	`#DEFINE` constant in `CrFwInFactoryUserPar.h` defines maximum number of InReports which can be allocated by the factory.
C2-IFA-2	InFactory Capacity for InCommands	`#DEFINE` constant in `CrFwInFactoryUserPar.h` defines maximum number of InCommands which can be allocated by the factory.

AP ID	Adaptation Point	Implementation
C2-IFA-3	InReport Kinds	`#DEFINE` constants in `CrFwInFactoryUserPar.h` define the kinds of InReports supported by the application. An InReport kind is defined through its service type, command or report sub-type, and discriminant value. For each supported InReport kind, function pointers are defined implementing the InReport checks and actions.
C2-IFA-4	InCommand Kinds	`#DEFINE` constants in `CrFwInFactoryUserPar.h` define the kinds of InCommands supported by the application. An InCommand kind is defined through its service type, command or report sub-type, and discriminant value. For each supported InCommand kind, function pointers are defined implementing the InCommand checks and actions.
C2-OST-1	Number of OutStreams in the Application	`#DEFINE` constant in `CrFwOutStreamUserPar.h`
C2-OST-2	Packet Queue Size for OutStream	`#DEFINE` constant (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`
C2-OST-3	Destination associated to OutStream	`#DEFINE` constant (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`
C2-OST-4	Initialization Check in Initialization Procedure of OutStream	Function pointers (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`. Default implementation is provided in `CrFwOutStream.h`..
C2-OST-5	Initialization Action in Initialization Procedure of OutStream	Function pointers (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`. Default implementation is provided in `CrFwOutStream.h`..

AP ID	Adaptation Point	Implementation
C2-OST-4	Configuration Check in Initialization Procedure of OutStream	Function pointers (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`. Default implementation is provided in `CrFwOutStream.h`..
C2-OST-6	Configuration Action in Reset Procedure of OutStream	Function pointers (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`. Default implementation is provided in `CrFwOutStream.h`..
C2-OST-7	Shutdown Action of OutStream	Function pointers (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`. Default implementation is provided in `CrFwOutStream.h`..
C2-OST-8	Packet Hand-Over Operation of OutStream	Function pointers (one for each OutStream in the application) in `CrFwOutStreamUserPar.h`. Only a test stub is provided as default at framework level.
C2-IST-1	Number of InStreams in the Application	`#DEFINE` constant in `CrFwInStreamUserPar.h`
C2-IST-2	Size of the Packet Queue in InStream	`#DEFINE` constant (one for each InStream in the application) in `CrFwInStreamUserPar.h`
C2-IST-3	Source associated to InStream	`#DEFINE` constant (one for each InStream in the application) in `CrFwInStreamUserPar.h`
C2-IST-4	Initialization Check in Initialization Procedure of InStream	Function pointers (one for each InStream in the application) in `CrFwInStreamUserPar.h`. Default implementation is provided in `CrFwInStream.h`.
C2-IST-5	Initialization Action in Initialization Procedure of InStream	Function pointers (one for each InStream in the application) in `CrFwInStreamUserPar.h`. Default implementation is provided in `CrFwInStream.h`.

AP ID	Adaptation Point	Implementation
C2-IST-6	Configuration Action in Reset Procedure of InStream	Function pointers (one for each InStream in the application) in `CrFwInStreamUserPar.h`. Default implementation is provided in `CrFwInStream.h`.
C2-IST-7	Shutdown Action of InStream	Function pointers (one for each InStream in the application) in `CrFwInStreamUserPar.h`. Default implementation is provided in `CrFwInStream.h`.
C2-IST-8	Packet Collect Operation for InStream	Function pointers (one for each InStream in the application) in `CrFwInStreamUserPar.h`. Only a test stub is provided as default at framework level.
C2-IST-9	Packet Available Check Operation for InStream	Function pointers (one for each InStream in the application) in `CrFwInStreamUserPar.h`. Only a test stub is provided as default at framework level.
C2-OST-10	Get OutStream Operation of OutStreamRegistry	`#DEFINE` constants (one for each OutStream in the application) in `CrFwOutStreamUserPar.h` define the destination associated to each OutStream.
C2-OCM-1	Enable Check Operation of OutComponent	`#DEFINE` constants (one for each kind of OutComponent in the application) in `CrFwOutFactoryUserPar.h` define the pointer to the function implementing the operation. A default is provided a framework level.
C2-OCM-2	Ready Check Operation of OutComponent	`#DEFINE` constants (one for each kind of OutComponent in the application) in `CrFwOutFactoryUserPar.h` define the pointer to the function implementing the operation. A default is provided a framework level.
C2-OCM-3	Repeat Check Operation of OutComponent	`#DEFINE` constants (one for each kind of OutComponent in the application) in `CrFwOutFactoryUserPar.h` define the pointer to the function implementing the operation. A default is provided a framework level.
C2-OCM-4	Update Action of OutComponent	`#DEFINE` constants (one for each kind of OutComponent in the application) in `CrFwOutFactoryUserPar.h` define the pointer to the function implementing the operation. A default is provided a framework level.

AP ID	Adaptation Point	Implementation
C2-OCM-5	Serialize Operation of OutComponent	`#DEFINE` constants (one for each kind of OutComponent in the application) in `CrFwOutFactoryUserPar.h` define the pointer to the function implementing the operation. A default is provided a framework level.
C2-OLD-1	Initialization Check in Initialization Procedure of OutLoader	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which always returns 'check successful' is provided in `CrFwOutLoader.h`.
C2-OLD-2	Initialization Action in Initialization Procedure of OutLoader	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which takes no action is provided in `CrFwOutLoader.h`.
C2-OLD-3	Configuration Check in Reset Procedure of OutLoader	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which always returns 'check successful' is provided in `CrFwOutLoader.h`.
C2-OLD-4	Configuration Action in Reset Procedure of OutLoader	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which takes no action is provided in `CrFwOutLoader.h`.
C2-OLD-5	Shutdown Action of OutLoader	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which takes no action is provided in `CrFwOutLoader.h`.
C2-OLD-6	OutManager Selection Operation	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which always returns the first OutManager in the LOM is provided in `CrFwOutLoader.h`.
C2-OLD-7	OutManager Activation Operation	Function pointer in `CrFwOutLoaderUserPar.h`. Default implementation which takes no action is provided in `CrFwOutLoader.h`.
C2-OMG-1	Number of OutManagers in Application	`#DEFINE` constants in `CrFwOutManagerUserPar.h`
C2-OMG-2	Size of POCL of OutManager	`#DEFINE` constants (one for each OutManager) in `CrFwOutManagerUserPar.h`

AP ID	Adaptation Point	Implementation
C2-ORG-1	Maximum Number of Trackable Commands/Reports for OutRegistry	`#DEFINE` constant in `CrFwOutRegistryUserPar.h` defines types, sub-types and range of discriminant values supported by application.
C2-ORG-2	Number of Service Types/Sub-Types supported by Application	`#DEFINE` constant in `CrFwOutRegistryUserPar.h` defines types, sub-types and range of discriminant values supported by application.
C2-ORG-3	Range of Services supported by Application	`#DEFINE` constant in `CrFwOutRegistryUserPar.h` defines types, sub-types and range of discriminant values supported by application.
C2-PCK-1	Operations to Report the Outcome of the Processing and Execution of an Incoming Command	Implementation of `CrFwRepInCmdOutcome.h`. Only a test stub is provided as default at framework level.
C2-ILD-1	Operation to Determine Re-Routing Destination of Packets	Function pointer in `CrFwInLoaderUserPar.h`. Default implementation is provided `CrFwInLoader.h`.
C2-ILD-1	Operation to Select InManager where Incoming Report or Command is Loaded	Function pointer in `CrFwInLoaderUserPar.h`. Default implementation is provided `CrFwInLoader.h`.
C2-ICM-1	Validity Check for InCommand	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided by function `CrFwPrCheckAlwaysTrue`.
C2-ICM-2	Ready Check of InCommand	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided by function `CrFwSmCheckAlwaysTrue`.
C2-ICM-3	Start Action of InCommand	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided by function `CrFwSmEmptyAction`.

AP ID	Adaptation Point	Implementation
C2-ICM-4	Progress Action of InCommand	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided by function `CrFwSmEmptyAction`.
C2-ICM-5	Termination Action of InCommand	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided by function `CrFwSmEmptyAction`.
C2-ICM-6	Abort Action of InCommand	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided by function `CrFwSmEmptyAction`.
C2-IRP-1	Validity Check for InReport	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided at framework level.
C2-IRP-2	Update Action of InReport	Function pointer in `CrFwInFactoryUserPar.h`. Default implementation is provided at framework level.
C2-IMG-1	Number of InManagers in Application	`#DEFINE` constants in `CrFwInManagerUserPar.h`
C2-IMG-2	Size of PCRL of InManager	`#DEFINE` constants (one for each InManager) in `CrFwInManagerUserPar.h`
C2-IRG-1	Maximum Number of Trackable InCommands/InReports in InRegistry	`#DEFINE` constant in `CrFwInRegistryUserPar.h`
C2-TIM-1	Operations to Get the Current Time	Implementation of `CrFwTime.h`. Only a test stub is provided as default at framework level.
C2-TYP-1	Definition of Primitive Types	Definition of `typedef`.values in `CrFwUserConstants.h`. Default values are pre-defined in this header file.
C2-CST-1	Identifier of Host Application	`#DEFINE` constant in `CrFwUserConstants.h`.

D CORDET Framework Behaviour

The C2 Implementation implements the behaviour of the CORDET Framework. The behaviour of the CORDET Framework is defined through a set of state machines and procedures (see E for a full list of their diagrams). Thus, the C2 Implementation implements the behaviour of the CORDET Framework state machines and procedures. This section provides the verification evidence which demonstrates correct implementation of the CORDET Framework state machines and procedures.

D.1 Verification of State Machine Behaviour

Correct implementation of the state machine behaviour is verified at the level of the C1 Implementation in reference [3]. At the level of the C2 Implementation it is therefore only necessary to verify that the state machines are correctly configured. This is done by performing tests which:

1. For every transition in the state machine, execute the state transition
2. For every transition originating in a proper state which has a guard, attempt to execute the state transition with the guard evaluating to false
3. For every do action in the state machine, execute the state machine

Note that the first two bullets also verify execution of all entry, exit and transition actions in the state machine. Tables D.1 to D.6 provide this verification evidence for each state machine defined in reference [4]. For each element in the previous list to be verified, the tables give the name of the test case in the Test Suite where that element is verified. Note that no attempt is made to list all test cases which verify a given element; rather the ojective is to identify one test case for each element to be verified.

Table D.1: Verification of Base State Machine

Element	Test Case
Transition from Initial Pseudo-State to CREATED	`CrFwBaseCmpTestCase1`
Transition from CREATED to CREATED	`CrFwInStreamTestCase5`
Transition from CREATED to INITIALIZED	`CrFwBaseCmpTestCase1`, `CrFwInStreamTestCase5`
Transition from INITIALIZED to INITIALIZED	`CrFwInStreamTestCase5`
Transition from INITIALIZED to CONFIGURED	`CrFwBaseCmpTestCase1`, `CrFwInStreamTestCase5`
Transition from CONFIGURED to CONFIGURED	`CrFwInStreamTestCase4`

Element	Test Case
Transition from CONFIGURED to Final Pseudo-State	`CrFwInStreamTestCase6`
Do Action in CONFIGURED	`CrFwInLoaderTestCase2` to `CrFwInLoaderTestCase11` verify that execution of InLoader component in state CONFIGURED triggers execution of its Execution Procedure

Table D.2: Verification of Application State Machine

Element	Test Case
Transition from Initial Pseudo-State to START-UP	`CrFwAppSmTestCase1`
Transition from START-UP to NORMAL with transition guard true	`CrFwAppSmTestCase1`
Transition from START-UP to NORMAL with transition guard false	`CrFwAppSmTestCase1`
Transition from NORMAL to RESET	`CrFwAppSmTestCase1`
Transition from RESET to NORMAL with transition guard true	`CrFwAppSmTestCase1`
Transition from RESET to NORMAL with transition guard false	`CrFwAppSmTestCase1`
Transition from NORMAL to SHUTDOWN	`CrFwAppSmTestCase1`
Transition from SHUTDOWN to Final-Pseudo State with transition guard true	`CrFwAppSmTestCase1`
Transition from SHUTDOWN to Final-Pseudo State with transition guard false	`CrFwAppSmTestCase1`

Table D.3: Verification of OutStream State Machine

Element	Test Case
Transition from Initial Pseudo-State to READY	`CrFwOutStreamTestCase1`
Transition from READY to Choice Pseudo-State	`CrFwOutStreamTestCase1`

Element	Test Case
Transition from Choice Pseudo-State to BUFFERING	`CrFwOutStreamTestCase1`
Transition from BUFFERING to BUFFERING	`CrFwOutStreamTestCase1`
Transition from BUFFERING to Choice Pseudo-State	`CrFwOutStreamTestCase3`
Transition from Choice Pseudo-State to READY	`CrFwOutStreamTestCase3`
In Enqueue Action, Branch with PQ not full	`CrFwOutStreamTestCase1`
In Enqueue Action, Branch with PQ full	`CrFwOutStreamTestCase1`
In Send or Enqueue Action, Branch with Packet Not Originating in Application	`CrFwOutStreamTestCase6`
In Send or Enqueue Action, Branch with Middleware Accepting Packet	`CrFwOutStreamTestCase3`
In Send or Enqueue Action, Branch with Middleware Rejecting Packet	`CrFwOutStreamTestCase3`
In Send or Enqueue Action, Branch with Legal Group	`CrFwOutStreamTestCase3`
In Send or Enqueue Action, Branch with Illegal Group	`CrFwOutStreamTestCase7`
In Flush Packet Queue Action, Branch with Packet Originating in Application	`CrFwOutStreamTestCase3`
In Flush Packet Queue Action, Branch with Legal Packet Group	`CrFwOutStreamTestCase3`
In Flush Packet Queue Action, Branch with Packet Not Originating in Application	`CrFwOutStreamTestCase6`
In Flush Packet Queue Action, Branch with Middleware Accepting Packet	`CrFwOutStreamTestCase3`
In Flush Packet Queue Action, Branch with Middleware Rejecting Packet	`CrFwOutStreamTestCase3`
In Flush Packet Queue Action, Branch with Legal Group	`CrFwOutStreamTestCase3`
In Flush Packet Queue Action, Branch with Illegal Group	`CrFwOutStreamTestCase7`

Table D.4: Verification of InStream State Machine

Element	Test Case
Transition from Initial Pseudo-State to WAITING	`CrFwInStreamTestCase1`, `CrFwInStreamTestCase4`
Transition from Initial Pseudo-State to PCKT_AVAIL	`CrFwInStreamTestCase6`
Transition from Initial PCKT_AVAIL to PCKT_AVAIL through Choice Pseudo-State	`CrFwInStreamTestCase2`
Transition from Initial PCKT_AVAIL to PCKT_AVAIL through Self-Transition	`CrFwInStreamTestCase3`, `CrFwInStreamTestCase4`
Transition from Initial PCKT_AVAIL to WAITING	`CrFwInStreamTestCase2`
Transition from Initial WAITING to PCKT_AVAIL	`CrFwInStreamTestCase2`, `CrFwInStreamTestCase3`, `CrFwInStreamTestCase4`

Table D.5: Verification of OutComponent State Machine

Element	Test Case
Transition from Initial Pseudo-State to LOADED	`CrFwOutCmpTestCase2` to `CrFwOutCmpTestCase6`
Transition from LOADED to ABORTED	`CrFwOutCmpTestCase2`, `CrFwOutCmpTestCase6`
Transition from LOADED to PENDING	`CrFwOutCmpTestCase3` to `CrFwOutCmpTestCase9`
Transition from PENDING to TERMINATED with guard true and a valid OutStream (i.e. transition is triggered by Repeat Check returning 'no repeat')	`CrFwOutCmpTestCase7`
Transition from PENDING to TERMINATED with guard true due to an invalid OutStream (i.e. transition is triggered by Send Packet Procedure having set `isRepeat` to 'no repeat')	`CrFwOutCmpTestCase8`
Transition from PENDING to TERMINATED with guard false	`CrFwOutCmpTestCase7`

Element	Test Case
Transition from PENDING to ABORTED with guard true	`CrFwOutCmpTestCase9`
Transition from PENDING to ABORTED with guard false	`CrFwOutCmpTestCase7`, `CrFwOutCmpTestCase8`

Table D.6: Verification of InCommand State Machine

Element	Test Case
Transition from Initial Pseudo-State to ACCEPTED	`CrFwOutCmpTestCase1` to `CrFwOutCmpTestCase3` and `CrFwOutCmpTestCase5` to `CrFwOutCmpTestCase11`
Transition from ACCEPTED to Choice Psedo-State with guard false	`CrFwOutCmpTestCase2`
Transition from ACCEPTED to ABORTED	`CrFwOutCmpTestCase3`
Transition from ACCEPTED to PROGRESS	`CrFwOutCmpTestCase2`
Transition from ACCEPTED to PROGRESS	`CrFwOutCmpTestCase2`
Transition from PROGRESS to ABORTED with guard false	`CrFwOutCmpTestCase5`
Transition from PROGRESS to Choice Pseudo-State with guard false	`CrFwOutCmpTestCase5`, `CrFwOutCmpTestCase6`
Transition from PROGRESS to TERMINATED	`CrFwOutCmpTestCase6`
Direct Transition from PROGRESS to ABORTED	`CrFwOutCmpTestCase7`
Transition from PROGRESS to ABORTED via Choice Pseudo-State	`CrFwOutCmpTestCase8`

D.2 Verification of Procedure Behaviour

Correct implementation of the procedure behaviour is verified at the level of the C1 Implementation in reference [3]. At the level of the C2 Implementation it is therefore only necessary to verify that the procedures are correctly configured. This is done by performing tests which execute every control flow in the procedure and, for every control flow with a guard, execute the control flow both when the guard is true and when it is false.

Tables D.1 to D.12 provide this verification evidence for each state machine defined in reference [4]. For each element in the previous list to be verified, the tables give the name of the test case in the Test Suite where that element is verified. Note that no attempt is made to list all test cases which verify a given element; rather the ojective is to identify one test case for each element to be verified. For convenience, the diagram representing a procedure is shown next to the table which verifies it.

Table D.7: Verification of Initialization Procedure

Element	Test Case
Execution of procedure with `Outcome` equal to Success	`CrFwBaseCmpTestCase1`
Execution of procedure with `Outcome` equal to Failure	`CrFwInStreamTestCase5`

Table D.8: Verification of Reset Procedure

Element	Test Case
Execution of procedure with `Outcome` equal to Success	`CrFwBaseCmpTestCase1`
Execution of procedure with `Outcome` equal to Failure	`CrFwInStreamTestCase5`

Table D.9: Verification of Packet Collect Procedure

Element	Test Case
Execution of procedure with `Flag_1` equal to True; Packet Queue not full; and MW in State PCKT_AVAIL	`CrFwInStreamTestCase2`, `CrFwInStreamTestCase3`
Execution of procedure with `Flag_1` equal to True; Packet Queue not full; and MW not in State PCKT_AVAIL	`CrFwInStreamTestCase2`, `CrFwInStreamTestCase3`
Execution of procedure branch with `Flag_1` equal to False	`CrFwInStreamTestCase4`

Element	Test Case
Execution of procedure branch with Packet Queue Full	`CrFwInStreamTestCase4`
Execution of procedure branch with Illegal Group	`CrFwInStreamTestCase7`

Table D.10: Verification of Enable State Determination Procedure

Element	Test Case
Execution of procedure with Service Type Disabled	`CrFwOutRegistryTestCase3`, `CrFwOutRegistryTestCase4`
Execution of procedure with Service Type Enabled	`CrFwOutRegistryTestCase3`, `CrFwOutRegistryTestCase4`, `CrFwOutRegistryTestCase5`
Execution of procedure with Service Sub-Type Disabled	`CrFwOutRegistryTestCase3`, `CrFwOutRegistryTestCase4`
Execution of procedure with Service Sub-Type Enabled	`CrFwOutRegistryTestCase3`, `CrFwOutRegistryTestCase4`, `CrFwOutRegistryTestCase5`
Execution of procedure with Out-Going Command or Report with Discriminant	`CrFwOutRegistryTestCase4`, `CrFwOutRegistryTestCase5`
Execution of procedure with Out-Going Command or Report with no Discriminant	`CrFwOutRegistryTestCase3`
Execution of procedure with Discriminant Enabled	`CrFwOutRegistryTestCase4`, `CrFwOutRegistryTestCase5`

Table D.11: Verification of InLoader Execution Procedure

Element	Test Case
Execution of procedure with No Packet Returned by InStream	`CrFwInLoaderTestCase2`
Execution of procedure with Packet Returned by InStream	`CrFwInLoaderTestCase3` to `CrFwInLoaderTestCase11`
Execution of procedure with Packet Destination Invalid	`CrFwInLoaderTestCase3`
Execution of procedure with Packet Destination Valid and Packet Destination not the Host Application	`CrFwInLoaderTestCase4`

Element	Test Case
Execution of procedure with Packet Destination Valid and Packet Destination is the Host Application	`CrFwInLoaderTestCase5`
Execution of procedure with Packet Destination Valid and Packet Destination is the Host Application	`CrFwInLoaderTestCase5`
Execution of procedure with Packet Destination Valid and Packet Destination is the Host Application	`CrFwInLoaderTestCase5`

Table D.12: Verification of InLoader Load Command/Report Procedure

Element	Test Case
Execution of procedure with Packet Type Invalid	`CrFwInLoaderTestCase5`
Execution of procedure with Packet Type Valid	`CrFwInLoaderTestCase6`
Execution of procedure when Make Operation Fails	`CrFwInLoaderTestCase6`
Execution of procedure when Make Operation Succeeds	`CrFwInLoaderTestCase7`
Execution of procedure when InCommand or InReport is in State CONFIGURED	`CrFwInLoaderTestCase8`
Execution of procedure when InCommand or InReport is not in State CONFIGURED	`CrFwInLoaderTestCase7`
Execution of procedure when Load Operation Fails	`CrFwInLoaderTestCase8`
Execution of procedure when Load Operation Succeeds	`CrFwInLoaderTestCase9`
Execution of procedure when Component Being Loaded is an InCommand and No Acknowledgement of Acceptance is Required	`CrFwInLoaderTestCase10`
Execution of procedure when Component Being Loaded is an InCommand and Acknowledgement of Acceptance is Required	`CrFwInLoaderTestCase11`

E State Machine and Procedure Diagrams

For convenience, this appendix shows all the state machine and procedure diagrams referred to in the test. The description of the state machine diagrams can be found in references [4] and [5].

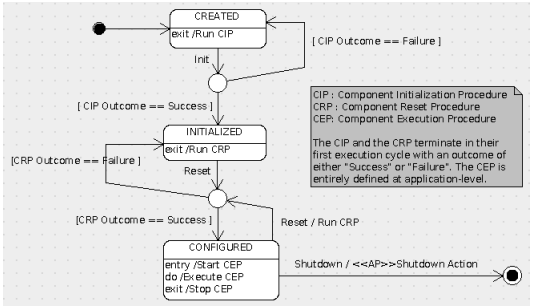

Fig. E.1: Base State Machine

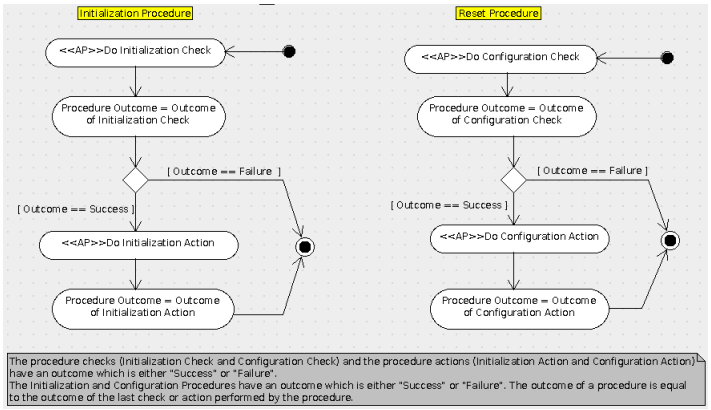

Fig. E.2: Initialization and Reset Procedures

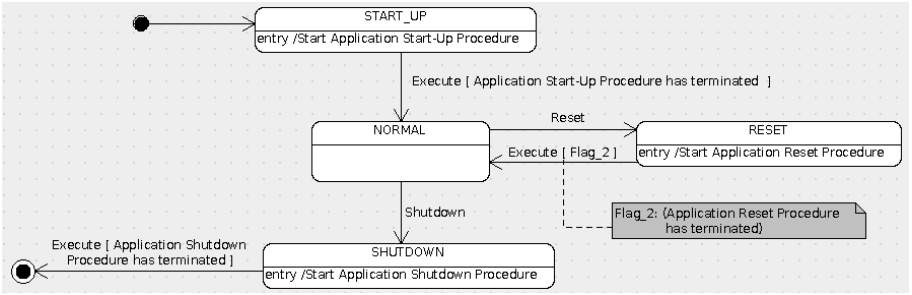

Fig. E.3: Application State Machine

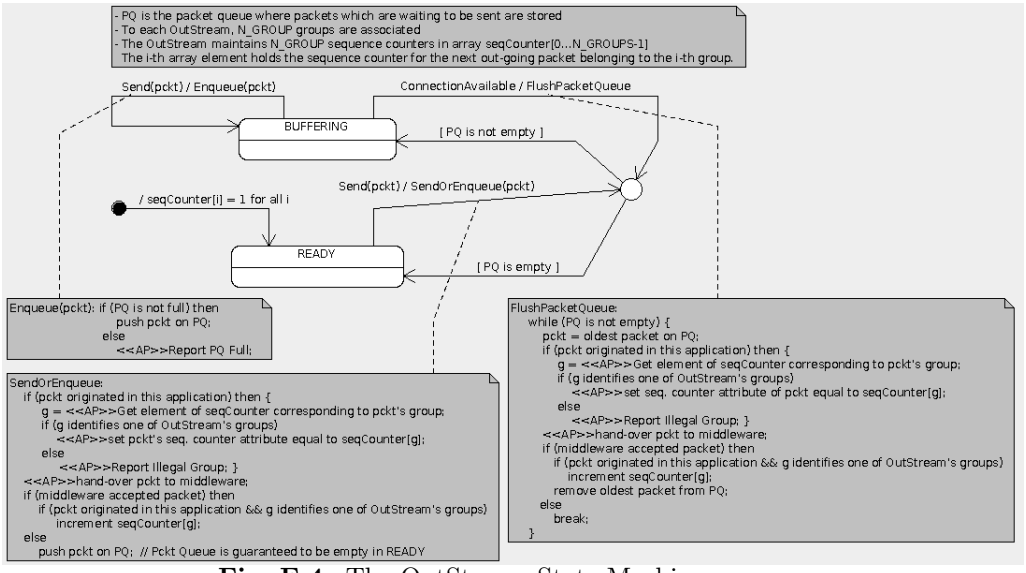

Fig. E.4: The OutStream State Machine

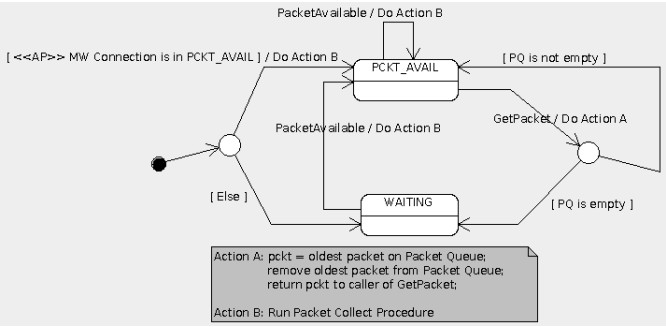

Fig. E.5: The InStream State Machine

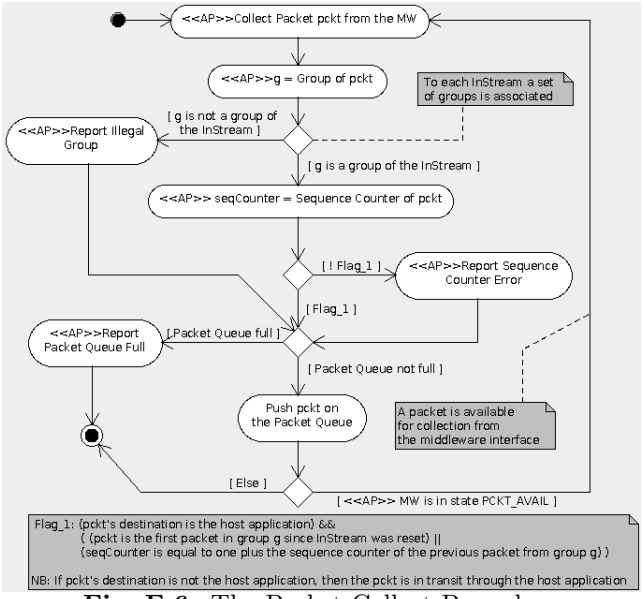

Fig. E.6: The Packet Collect Procedure

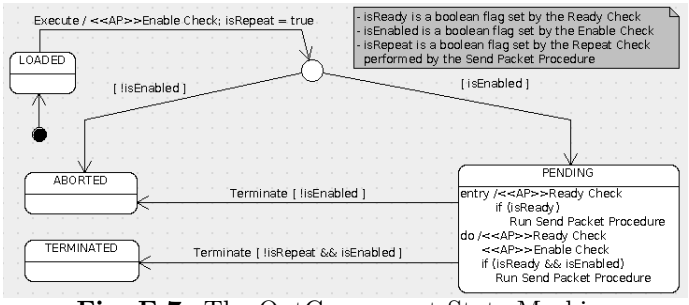

Fig. E.7: The OutComponent State Machine

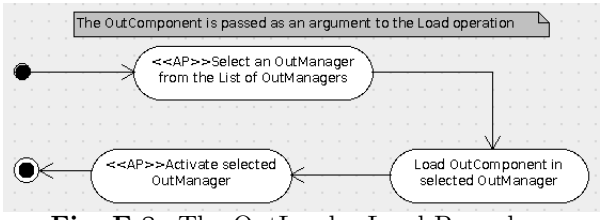

Fig. E.8: The OutLoader Load Procedure

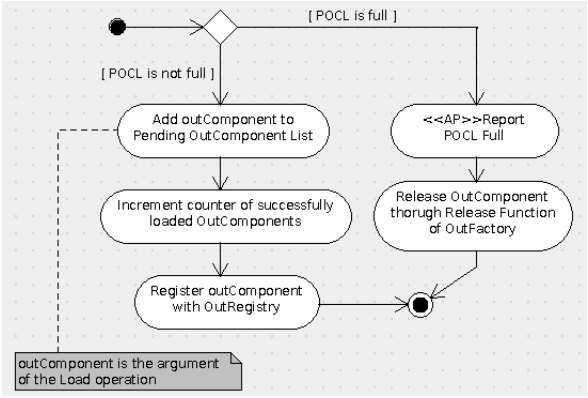

Fig. E.9: The OutManager Load Procedure

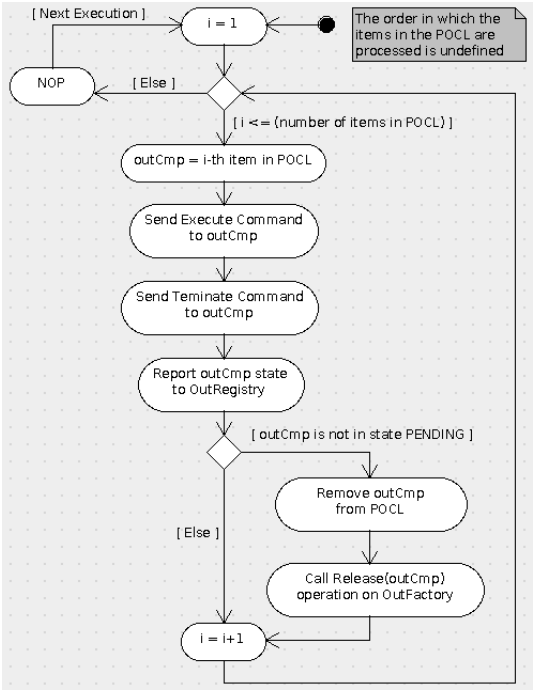

Fig. E.10: The OutManager Execution Procedure

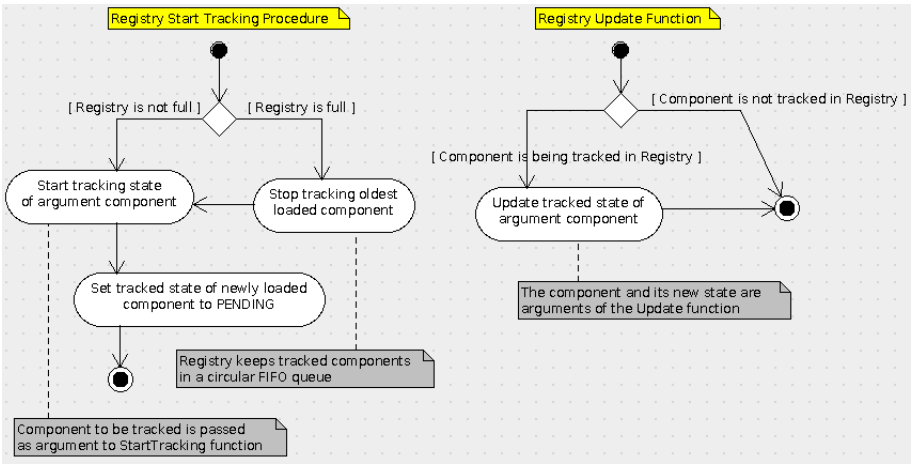

Fig. E.11: The Registry Start Tracking and Registry Update Procedures

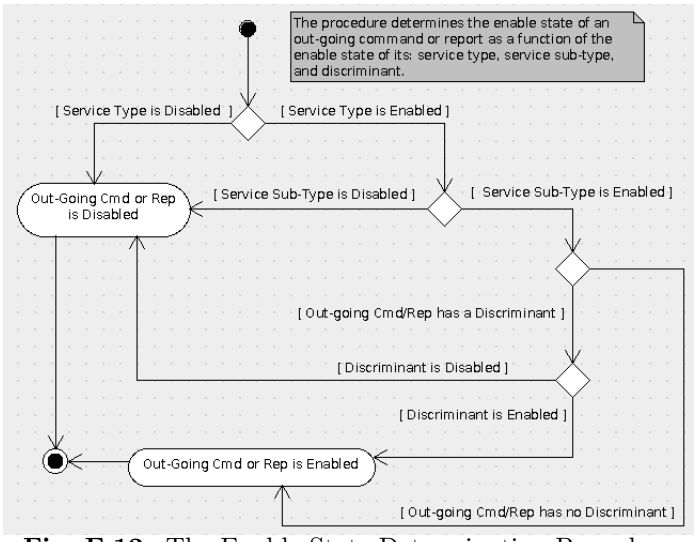

Fig. E.12: The Enable State Determination Procedure

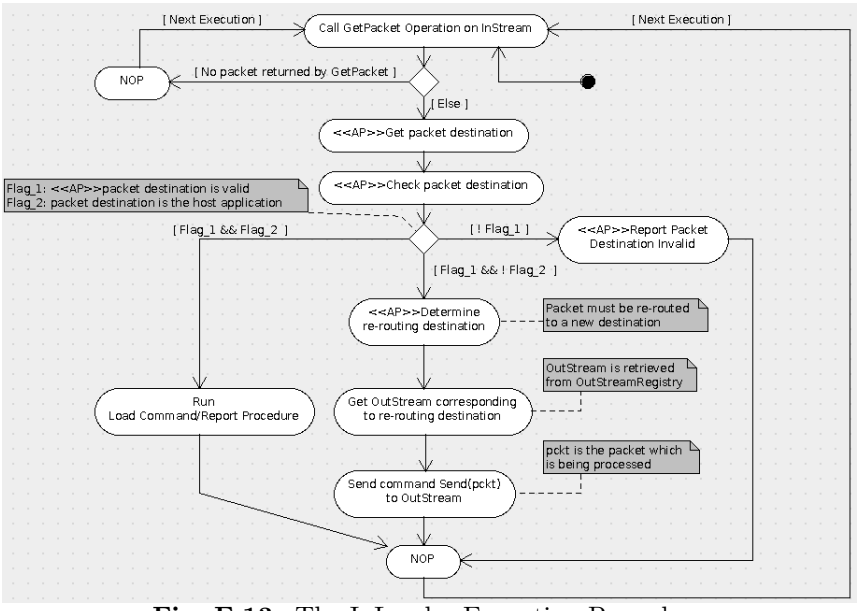

Fig. E.13: The InLoader Execution Procedure

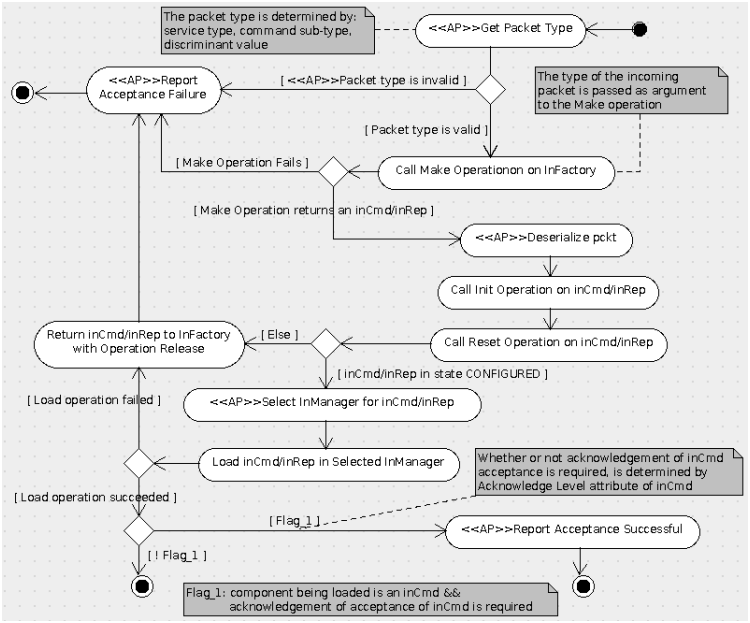

Fig. E.14: The InLoader Load Command/Report Procedure

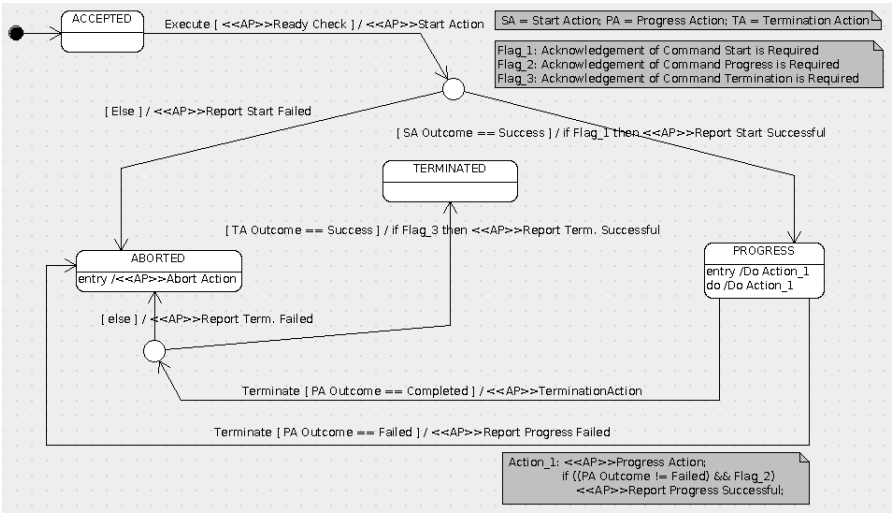

Fig. E.15: The InCommand State Machine

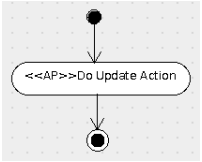

Fig. E.16: The InReport Execution Procedure

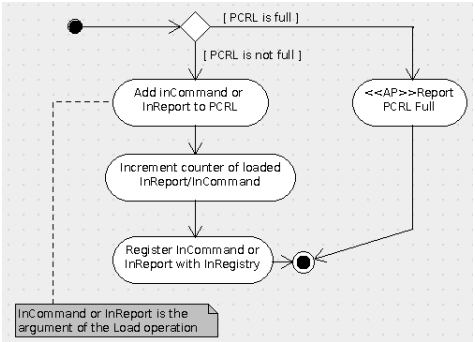

Fig. E.17: The InManager Load Procedure

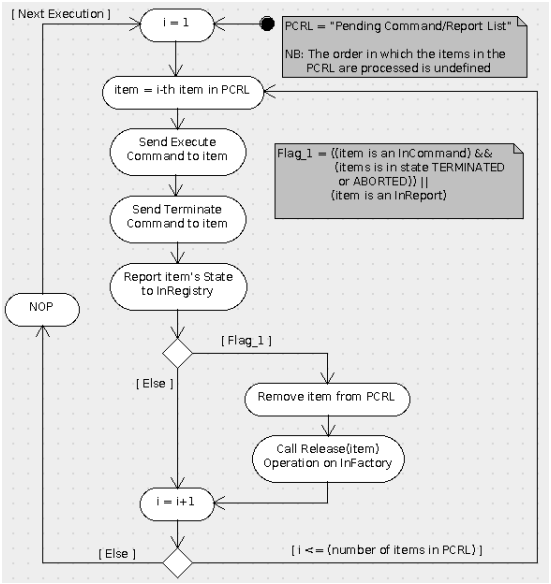

Fig. E.18: The InManager Execution Procedure

References

[1] Alessandro Pasetti, Vaclav Cechticky: *The FW Profile*. PP-DF-COR-00001, Revision 1.3, P&P Software GmbH, Switzerland, 2013

[2] Alessandro Pasetti, Vaclav Cechticky: *The Framework Profile - C1 Implementation User Manual*. PP-UM-COR-00001, Revision 1.2.0, P&P Software GmbH, Switzerland, 2013 Available from: `www.pnp-software.com/fwprofile`

[3] Alessandro Pasetti, Vaclav Cechticky: *The Framework Profile - C1 Implementation User Requirements*. PP-SP-COR-00001, Revision 1.2.0, P&P Software GmbH, Switzerland, 2013 Available from: `www.pnp-software.com/fwprofile`

[4] Alessandro Pasetti, Vaclav Cechticky: *The CORDET Framework*. PP-DF-COR-00002, Revision 1.3.0, P&P Software GmbH, Switzerland, 2014

[5] Alessandro Pasetti, Vaclav Cechticky: *The CORDET Framework - User Manual*. PP-UM-COR-00002, Revision 0.4.0, P&P Software GmbH, Switzerland, 2014

