Indian Radio Software Architecture (IRSA) Specification

Document: IRSA-001
Version: 1.0

May-2025

This page is intentionally left blank

Revision History

Version	Issue Date	Reason for Changes
1.0-D1	28-Feb-2023	Initial release
1.0-D2	06-Mar-2023	Internal review comments incorporated
1.0-D3	12-Aug-2024	Review comments / inputs received from stakeholders incorporated.
1.0-D4	31 Jan 2025	Review comments / inputs received from stakeholders incorporated.
1.0 (HLAC)	18 Feb 2025	Review comments / inputs received from stakeholders incorporated. Submission to HLAC for release.
1.0	23 May 2025	Review comments of HLAC incorporated.

Disclaimer

The Indian Radio Software Architecture (IRSA) Specification document applies to Software-Defined Radios (SDR) developed or intended for use in India. This document integrates and references multiple existing sources. Where applicable, prior documents have been adopted as-is, modified, or omitted based on their relevance and applicability.

Proper attribution has been provided for referenced materials, ensuring due credit to the original authors. This document does not claim ownership of externally sourced content but serves as a unified architectural guide for SDR. All external documents remain the intellectual property of their respective owners.

Terms, Conditions & Notices

This specification is prepared as per Standardization of Indigenous Software Defined Radio initiative of Directorate of Standardization (DoS) under Department of Defence Production (DDP) in Ministry of Defence (MOD) India. It is subject to change depending upon the feedback after reference implementation or input of the stakeholders. Once approved it will be binding on all future SDR development in India.

This IRSA specification is an effort to create Indian ecosystem for standard based SDR development. It is aimed to bring Indian SDR development at par with global development. It is built upon existing specification released by Joint Tactical Networking Centre, Wireless Innovation Forum and alliance for ESSOR. All documents referenced of above organizations are publicly available and their respective terms and conditions are applicable.

Contents

List of Figures

List of Tables

High Level Advisory Committee

This specification is created for Standardization of Indigenous Software Defined Radio as per initiative of Directorate of Standardization (DoS) under Department of Defence Production (DDP) in Ministry of Defence (MOD), India. High Level Advisory Committee (HLAC) has been constituted to monitor the progress of this specification. It consists of following members

1.	Director, Indian Institute of Technology (IIT), Gandhinagar (Director of an Indian Institute of Technology (IIT))	Chairman
2.	Rep, Head Quarter Integrated Defence Staff (HQ IDS)	Member
3.	Rep, Indian Army (IA) HQ	Member
4.	Rep, Indian Navy (IN) HQ	Member
5.	Rep, Indian Airforce (IAF) HQ	Member
6.	Rep, Defence Research & Development Organization (DRDO)	Member
7.	Rep, IESA / Society of Indian Defence Manufacturers (SIDM)	Member
8.	Rep, DPSUs/BEL/HAL/ECIL	Member
9.	Nominated Domain Specialists	
10.	Rep, Directorate of Standardization (DoS), Delhi	Member Secretary

Contributors

This specification is created by Defence Research and Development Organization (DRDO) with the help of Defence Public Sector Undertakings (DPSUs), academia, Indian industry, other Govt. agencies and Directorate of Standardization (DoS). The document will be approved by High Level Advisory Committee (HLAC). Following members are credited as contributors for development of this specification.

1. Adrish Banerjee, Indian Institute of Technology (IIT), Kanpur
2. Amarnadha Reddy, Lekha Wireless Solutions Pvt. Ltd.
3. Amit Tikaria, Bhabha Atomic Research Centre (BARC)
4. Anand Sharma, DRDO
5. Anil Kumar Chand, ECIL
6. Appala Naidu, Alten Global Technologies Pvt. Ltd.
7. B M Kotnala, DRDO
8. Himanshu Karnatak, DRDO
9. Krishna Kumar, Centre for Development of Advanced Computing (C-DAC)
10. Makarand Kulkarni, Tejas Networks Ltd.
11. Mallapur Veerayya, DRDO
12. Neelesh Tamrakar, DRDO
13. Pankaj Azad, DRDO
14. Prabir Datta, Hughes Systique Pvt. Ltd.
15. Rajat Sharma, Hughes Systique Pvt. Ltd.
16. Ravi Shankar, Hindustan Aeronautics Ltd. (HAL)
17. Sanjay, DRDO
18. Satish Kumar Penmetsa, Central Research Laboratory (CRL), BEL
19. Saurabh Sharma, DRDO
20. Shamsher Singh, DRDO
21. Srikanth GS, Alten Global Technologies Pvt. Ltd.
22. Surender Kumar Rohilla, Directorate of Standardization (DoS)
23. Sushil Bahuguna, Bhabha Atomic Research Centre (BARC)
24. Urbi Chatterjee, Indian Institute of Technology (IIT), Kanpur
25. Vikas Bhatia, DRDO

1 Introduction

Software Defined Radio (SDR) originated with the aim of providing standardized communication infrastructure for military radios. One of the core objectives was portable application/waveform development which could enable communication among wide range of SDRs.

Joint Tactical Networking Centre (JTNC), Wireless Innovation Forum (WINNF) and alliance for ESSOR provided different specifications over a period of time which incorporated technology updates, implementation feedbacks and catered for different types of SDR Form Factors as depicted in following figure.

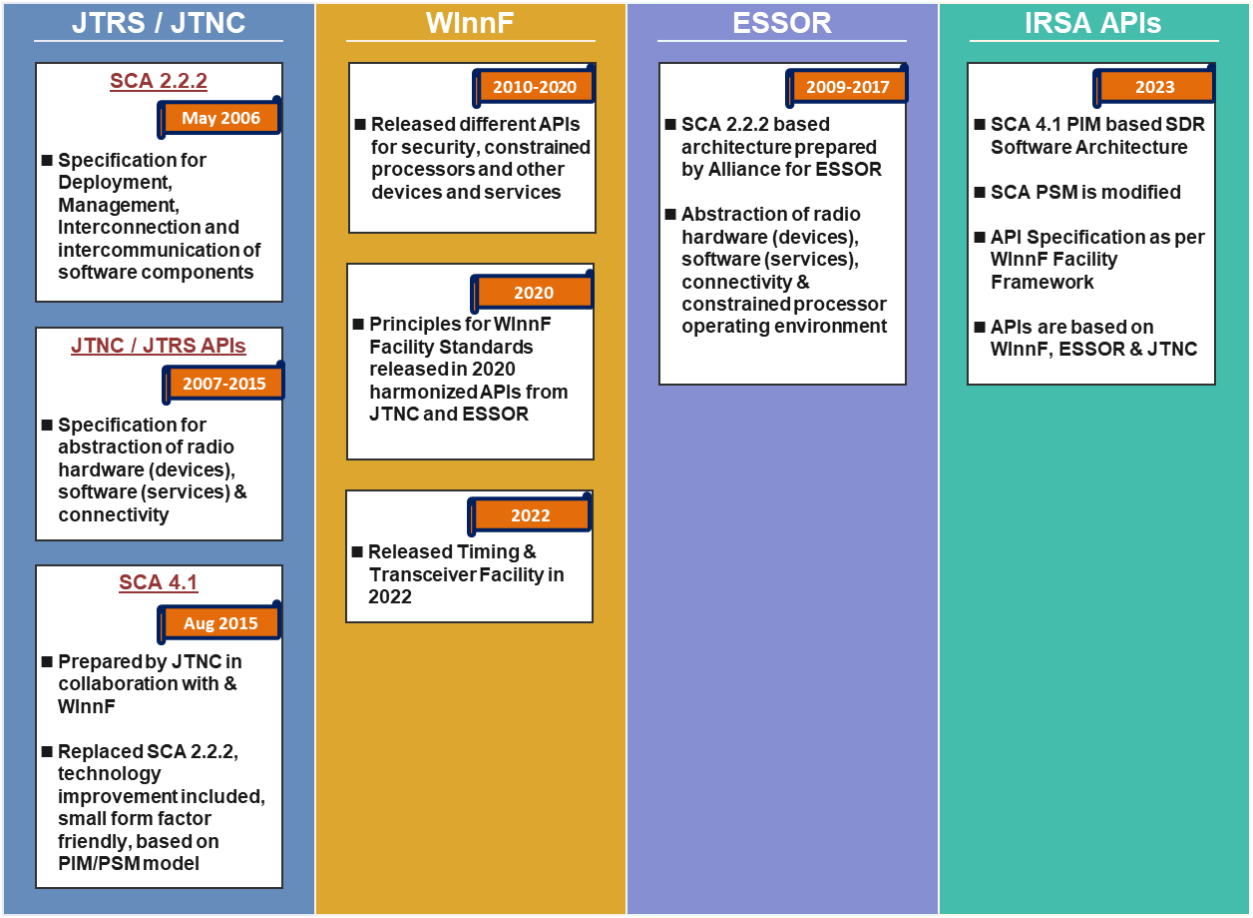

Figure 1.1 : Specification History

Software Communication Architecture (SCA) 2.2.2 [JTNC-Ref 01] specification released in 2006 had set the basis of all the subsequent developments by different agencies worldwide. Immediately after SCA specification release JTNC (previously JTRS) started releasing different Application Programming Interfaces (API) to standardize access of common radio hardware & software elements present in the radio. ESSOR architecture [ESSOR-Ref 01] was based on SCA specification but with certain modifications to JTNC

APIs. It also provided new API set for additional hardware and software elements which can be used by waveform. Another important contribution of ESSOR was extension of standardization to more processing elements like DSP and FPGA.

WINNF played a pivotal role in evolution of SCA and API specifications. It released different documents, [WInnF-Ref 01] to [WInnF-Ref 19], for wider acceptability of SCA and APIs including security APIs, certification related documents, profiles, AEPs etc. It was instrumental in creation of SCA 4.1 specification [JTNC-Ref 23] which was released by JTNC in 2015. Harmonization of JTNC and ESSOR APIs is a crucial ongoing activity in the charter of WInnF. Facility standard [WInnF-Ref 09] has been released as part of this harmonization effort.

In the presence of different standards and documents it is imperative to mandate unambiguous specifications for different requirements of SDR. Indian Radio Software Architecture (IRSA) is conceived with the aim of providing single definition for different constituents of SDR to achieve the objective of waveform portability. The approach followed by IRSA is to adopt the most suitable specifications and carry out modifications as per the requirement of Indian SDR eco system. IRSA will also cater for security specific requirements of Indian SDR eco system.

1.1 Overview

IRSA is amalgamation of specifications for different software components of SDR. IRSA components explained in this document are shown in Figure 1.2. Details of all the components are provided in this document. Chapters are created for first level components of IRSA which encompasses details of different sub-components. Security framework document is an appendix of this document and not released along with it.

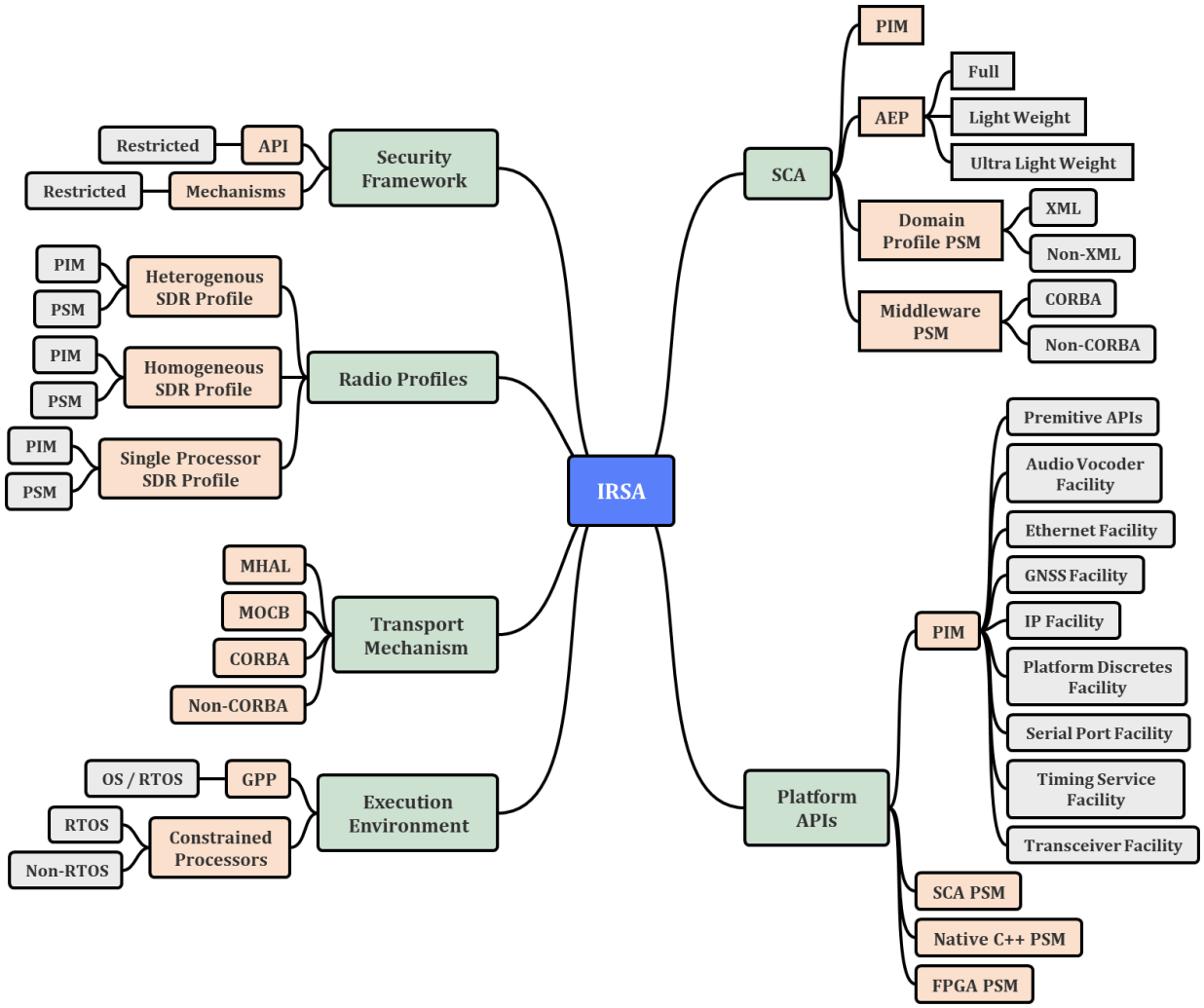

Figure 1.2 : IRSA Components

1.2 Document Outline

This document describes Indian Radio Software Architecture. As per OMG PIM/PSM model it provides PIM & PSM specification of different components. It is organized as following chapters

Chapter 1: Introduction
Chapter 2: Software Communication Architecture
Chapter 3: Platform API Specification
Chapter 4: Execution Environment
Chapter 5: Transfer Mechanisms
Chapter 6: IRSA Radio Profiles
Chapter 7: Waveform Portabilty

1.3 Purpose

IRSA aims to bring SDR development in India at par with global standards. Since no single universally accepted definition for standardization of different SDR components is available, IRSA enables that to bring maturity in Indian SDR ecosystem. The emphasis of IRSA is not on writing a specification document for each component afresh but to adopt global best practices being followed and modify only where it is required. IRSA specification has been defined with following objectives.

1. **Waveform Portability:** The communication requirements of the armed forces are met by several specific waveforms. These waveforms are required to be available on variety of SDR platforms manufactured by different OEMs. It is a prime concern for Indian SDR ecosystem where SCA core framework is widely adopted but other components are mostly customized and hampering waveform portability.

2. **Technology Neutrality:** Specification shall provide scope for use of different technologies for implementation and not be restricted. OMG PIM/PSM guidelines have been followed by WinnF and JTNC in creation of the facility framework and SCA 4.1 specification respectively. IRSA is an effort in the same direction where middleware/transfer mechanism alternatives may be realized which will be released after reference implementation and performance evaluation.

3. **Scalability:** The operational requirements of user dictate the form factors of SDR and IRSA has extended profiles to fulfil varying set of requirements of different form factors.

1.4 IRSA Scope

IRSA will enable seamless interaction among different software components of SDR and facilitate their management and control. Software components are created during SDR platform development depending on form factor requirements. Another set of software components are created during waveform development which may be hosted on different SDRs. IRSA framework encompasses of several facets for easy portability of waveform over SDR. Details of following domains are part of IRSA specification.

I. **SCA:** SCA specification is most commonly used in SDR development worldwide. IRSA mandates different PIM and PSM components of SCA 4.1 specification. IRSA is fully compliant to SCA 4.1 PIM specification and few modifications are suggested in PSM.

II. **Platform API:** It provides a set of high-level interfaces to enable an application/waveform to utilize the various devices and services of the underlying platform in any SDR. Uniformity of these interfaces assist in portability of waveforms. IRSA has adopted service-oriented approach of WINNF framework

[WInnF-Ref 09] which treats each hardware resource or software service as a facility provided by SDR platform. Functional support capability of each resource is divided in different services. Each service is a combination of different attributes and software interfaces (APIs) which are used for interaction between radio application and platform resource. These facilities encompass both PIM and PSM specifications to implement them in SCA, native C++ and FPGA environment.

III. **Execution Environment:** SDR platform is realized with help of different type of reconfigurable processing elements like GPP, DSP and FPGA depending on the form factor requirements. Execution environment of each processing element may be different and depends on corresponding form factor. IRSA assumes the presence of at least one SCA OE capable processing element in each security domain. IRSA has catered for execution environment required in resource constrained processing elements.

IV. **Transfer Mechanisms:** Software components running in SDR have to communicate which is enabled by transfer mechanism. In the presence of different processing elements and execution environments, a set of transfer mechanisms may be deployed in any SDR. The various transport mechanisms supported by IRSA are CORBA, alternative middleware (to be specified as separate PSM document like CORBA PSM), and raw middleware viz. MHAL and MOCB. Out of these transport mechanisms, IRSA has not considered CORBA for communication with components over constrained operating environment, rest shall be supported.

V. **Security API:** Complete Security framework is not in the scope of IRSA as it is being handled by different organizations as per existing policies and guidelines. IRSA will complement them by defining the security APIs to provide uniformity in the Indian SDR ecosystem. It will standardize the interface between SDR platform and waveforms thereby enabling seamless communication between waveforms and security subsystem. These APIs are restricted and not part of main document.

VI. **Radio Profiles:** SDRs are being realized in different form factors as per operational requirements. Single specification cannot be applied on all and hence the concept of three profiles has been proposed in IRSA. This is based on similar ideas applied in SCA, middleware and WINNF facility framework

Different constituents of IRSA are depicted in IRSA context diagram in Figure 1.3.

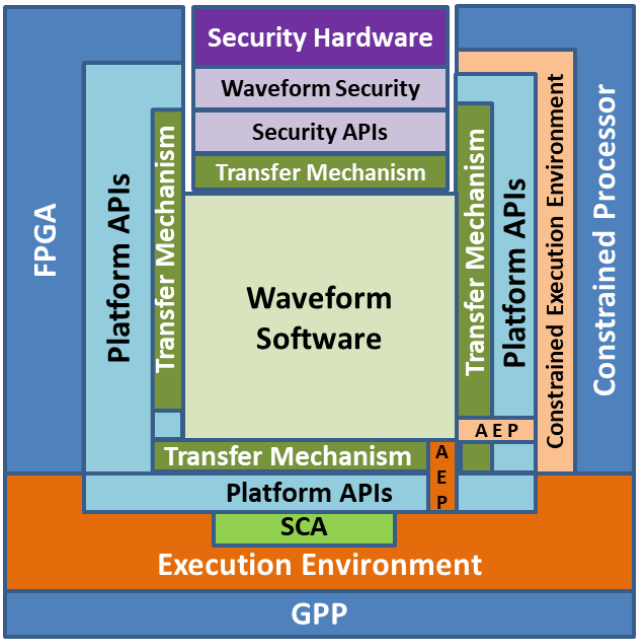

Figure 1.3 : IRSA Context Diagram

1.5 Intended Audience

This document is written to provide guidance for different stakeholders of the Indian SDR ecosystem. Following stakeholders may refer to this document to meet their objectives.

1.5.1 SDR Developers

This document will assist SDR developers in preparing their platform as per IRSA requirement. IRSA is having different domains and for each domain conformity can be achieved in different ways. This flexibility could lead to a different approach for IRSA conformance of the same type of SDR. Form factor is a governing consideration in SDR development and IRSA radio profiles are written to assist in easy conformity for different form factors. These radio profiles are specifying mandatory requirements for each domain of IRSA specific to a particular category of SDR. This will align different developers to approach IRSA conformance in same manner for one type of form factors.

1.5.2 Waveform Developers

This document will provide information to waveform developers which will assist in design and development of IRSA compliant waveform. IRSA has included platform API as per facility framework which extends API availability in DSP and FPGA other than GPP. Waveform developers will benefit from this facility framework as the platform will be abstracted across computing elements.

1.5.3 Indian Military Services

IRSA will enable the user in meeting their objective of waveform portability and in turn interoperability among different SDRs. This guiding document is encompasses all aspects of IRSA which will enable the user in planning and procurement of different types of SDR

and waveform.

1.5.4 Testing and Certification Agency

IRSA conformance testing both for waveform and SDR platform followed by certification will enable any system become part of IRSA conformant Indian SDR eco system. Different test procedures, test tools will be required to test conformance for different constituents of IRSA. This document along with its references will enable testing and certification agency in checking conformance for following domains of IRSA

i. **SCA:** SCA domain of IRSA is based on JTNC SCA 4.1 specification which has evolved and one important improvement from SCA 2.2.2 specification was an improvement in testability. SCA conformance testing will be easier as this document provides required reference for SCA conformance of both SDR platform and waveform components [WInnF-Ref 01].

ii. **Platform API:** Use of facility framework in platform API will help in API testing over different computational elements as separate PSM for each will be available. This document covers all APIs required to be tested in one place as PIM specification. PSM specifications for some of the facilities are also referenced in this document.

iii. **Execution Environment:** IRSA defines execution environments for each type of computational element. This document encompasses different profiles for execution environment.

iv. **Transfer Mechanism:** This is an important domain in SDR software components development either it is waveform or platform. This document has covered different types of communication requirement in separate sections which can assist in development of test procedures for them.

v. **IRSA SDR Profiles:** These profiles will be guiding instructions for testing and certification of same types of SDRs.

1.5.5 Standardization Agency

SDR is software intensive communication system and standardization agency has to all cater to all domains involved in it. IRSA is a single umbrella encompassing different components of SDR software which are built on well-defined global specifications. This document will facilitate standardization agency in referring single source for IRSA.

1.6 Future Directions

This guiding document along with its references has provided specifications for different domains of IRSA as per OMG PIM/PSM specification format. In the domain of platform APIs Current PSM specifications are available only for timing and transceiver services. PSM specifications for other components will be released in future.

In the domain of SCA and transfer mechanism IRSA has proposed non-CORBA PSM. Details of this technology will be released in future after reference implementation.

1.7 Reference Documents

IRSA specification referred JTNC, ESSOR and WINNF specifications for different sections. A brief of the same is shown in following Figure 1.4.

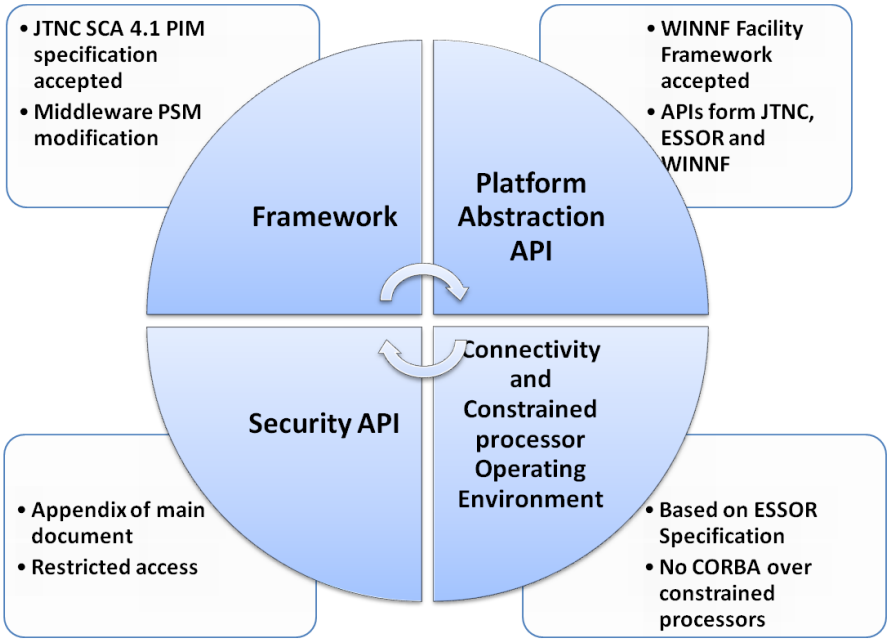

Figure 1.4 : IRSA Constituents

1.7.1 JTNC/JTRS Documents

[JTNC-Ref 01] SCA Specification, FINAL, 15 May 2006, Version 2.2.2, its errata, extensions and appendices
[JTNC-Ref 02] JTRS Standard Device IO Control API, Version 1.1.1, 29 Mar 2007
[JTNC-Ref 03] JTRS Standard Device IO Signals API, Version 1.1.1, 29 Mar 2007
[JTNC-Ref 04] JTRS Standard Device Packet API, Version 1.1.1, 29 Mar 2007
[JTNC-Ref 05] JTRS Standard Device Simple Packet API, Version 1.1.1, 29 Mar 2007
[JTNC-Ref 06] JTRS Standard Ethernet Device API, Version 1.2.2, 31 March 2008
[JTNC-Ref 07] JTRS Standard Device Message Control API, Version 1.1.3, 31 Mar 2008
[JTNC-Ref 08] JTRS Standard Packet API, Version 2.0.2, 02 April 2008
[JTNC-Ref 09] JTRS Standard Device Packet Signals API, Version 1.2.2, 02 Apr 2008

[JTNC-Ref 10] JTRS Standard JTRS CORBA Types, Version 1.0.2, 02 April 2008
[JTNC-Ref 11] JTRS Standard Device Simple Packet Signals API, Version 1.1.2, 02 Apr 2008
[JTNC-Ref 12] JTRS Standard Frequency Reference Device API, Version 1.0.3, 22 October 2008
[JTNC-Ref 13] JTRS Standard Device IO API, Version 1.0.2, 05 May 2010
[JTNC-Ref 14] JTRS Standard Audio Port Device API, Version 1.3.4, 29 Jul 2010
[JTNC-Ref 15] JTRS Platform Adapter Interface Standard, Version 1.3.3, 26 Jun 2013
[JTNC-Ref 16] JTRS Standard MHAL on Chip Bus API, Version 1.1.5, 26 June 2013
[JTNC-Ref 17] JTRS Standard Modem Hardware Abstraction Layer API, Version 3.0, 02 Oct 2013
[JTNC-Ref 18] JTRS Standard Serial Port Device API, Version 2.1.4, 26 June 2013
[JTNC-Ref 19] JTRS Standard Timing Service API, Version 1.4.4, 26 June 2013
[JTNC-Ref 20] JTRS Standard Global Positioning System API, Version 2.1.4, 24 July 2014
[JTNC-Ref 21] JTRS Standard Vocoder Service API, Version 1.4, 26 February 2015
[JTNC-Ref 22] JTNC Catalog of Public Release Approved Standards, Version 1.1, 20 Aug 2015
[JTNC-Ref 23] SCA Specification, Version 4.1, 20 August 2015
[JTNC-Ref 24] SCA Specification, Appendix A: Glossary 20 August 2015, Version: 4.1
[JTNC-Ref 25] SCA Specification, Appendix B: SCA Application Environment Profiles, 20 August 2015, Version: 4.1
[JTNC-Ref 26] SCA Specification, Appendix C: Core Framework Interface Definition Language, 20 August 2015, Version: 4.1
[JTNC-Ref 27] SCA Specification, Appendix D: Platform Specific Model – Domain Profile Descriptor Files, 20 August 2015, Version: 4.1
[JTNC-Ref 28] SCA Specification, Appendix D-1: Platform Specific Model – Document Type Definition Files, 20 August 2015, Version: 4.1 and attachment 1
[JTNC-Ref 29] SCA Specification, Appendix E: Platform Specific Model (PSM) - Transfer Mechanisms and Enabling Technologies, 20 August 2015, Version: 4.1
[JTNC-Ref 30] SCA Specification, Appendix E-1: Application Interface Definition Language Platform Independent Model Profiles, 20 August 2015, Version: 4.1
[JTNC-Ref 31] SCA Specification, Appendix E-2: Platform Specific Model – Common Object Request Broker Architecture, 20 August 2015, Version: 4.1 and attachments 1 & 2
[JTNC-Ref 32] SCA Specification, Appendix E-3: Platform Specific Model – Language Specific Mappings, 20 August 2015, Version: 4.1
[JTNC-Ref 33] SCA Specification, Appendix F: Units of Functionality and Profiles, 20 August 2015, Version: 4.1 and attachment 1

[JTNC-Ref 34] SCA Specification User's Guide, Version: 4.1, 23 February 2016
[JTNC-Ref 35] SCA 2.2.2 to 4.1 Migration Guide Briefing, 26 August 2016, JTNC Standards
[JTNC-Ref 36] SCA Version 4.1 Errata, Version: 1.0, 17 February 2017
[JTNC-Ref 37] SCA 4.1 Features and Benefits, Version: 1.0, 18 January 2018
[JTNC-Ref 38] JTNC Test and Evaluation Laboratory Software Communications Architecture, Version 4.1 Operating Environment Requirements List, Version 0.1, 24 September 2019
[JTNC-Ref 39] JTNC Test and Evaluation Laboratory Software Communications Architecture, Version 4.1 Application Requirements List, Version 1.0, 24 September 2019

JTNC Resource Website: https://www.jtnc.mil/Resources-Catalog/

1.7.2 ESSOR Documents

[ESSOR-Ref 01] ESSOR Architecture Introductory Document, Document Number - CA-PRA-SAS-63996053-549, Issue-AB
[ESSOR-Ref 02] ESSOR Radio Services API Description Document, Document Number CA-PRA-SAS-63996050-549, Issue-AB
[ESSOR-Ref 03] Radio Devices API Description Document, Document Number - CA-PRA-SAS-63996048-549, Issue-AB
[ESSOR-Ref 04] ESSOR DSP AEP and IDL Profile Description Document, Document Number - CA-PRA-SAS-63996052-549, Issue-AB

ESSOR Documents Website: https://www.occar.int/programmes/essor

1.7.3 Wireless Innovation Forum Documents

[WInnF-Ref 01] WInnF SCA Test, Evaluation and Certification Model Realization, Document WINNF-10-P-0012, Version V1.0.0, 27 June 2012
[WInnF-Ref 02] WInnF International Radio Security Services API Specification, Document WINNF-09-S-0011, Version V2.0.1, 13 June 2013
[WInnF-Ref 03] WInnF International Radio Security Services Application Programming Interface Functional Requirements: Analysis, Development and Specification for an International Radio Security Services API Set, Document WINNF-13-S-0004, Version V1.0.1, 13 June 2013
[WInnF-Ref 04] WInnF Lw and ULw POSIX AEPs for Resource Constrained Processors, Document WINNF-14-S-0009, Version V1.0.1, 25 July 2014
[WInnF-Ref 05] WInnF IDL Profiles for Platform-Independent Modeling of SDR Applications, Document WINNF-14-S-0016, Version V2.0.2, 12 June 2015

[WInnF-Ref 06] WInnF SCA 4.1 Requirements Allocation, Objectives, and Verification Criteria, Working Document WINNF-16-P-0025-V1.0.0, 17 March 2017

[WInnF-Ref 07] WInnF SCA 4.1 Test Procedures, Document WINNF-TS-4001, Version V1.0.0, 18 December 2018

[WInnF-Ref 08] WInnF SCA 4.1 Applications Verification Plan, Document WINNF-TS-4002, Version V1.0.0, 18 December 2018

[WInnF-Ref 09] WInnF Principles for WInnForum Facility Standards, Document WINNF-TR-2007, Version V1.0.0, 13 October 2020

[WInnF-Ref 10] WInnF Facilities PSMs Mapping Rules, Document WINNF-TR-2008, Version V1.0.1, 18 January 2022

[WInnF-Ref 11] WInnF Time Service Facility PIM Specification, Document WINNF-TS-3004, Version V1.1.1, 18 January 2022

[WInnF-Ref 12] WInnF Time Service Facility SCA PSM Specification, Document WINNF-TS-3004-App02, Version V1.1.1, 18 January 2022

[WInnF-Ref 13] WInnF Time Service Facility Native C++ PSM Specification, Document WINNF-TS-3004-App01, Version V1.1.1, 18 January 2022

[WInnF-Ref 14] WInnF Time Service Facility FPGA PSM Specification, Document WINNF-TS-3004-App03, Version V1.1.1, 18 January 2022

[WInnF-Ref 15] WInnF Transceiver Facility PIM Specification, Document WINNF-TS-0008, Version V2.1.1, 22 January 2022

[WInnF-Ref 16] WInnF Transceiver Facility SCA PSM specification, Document WINNF-TS-0008-App02, Version V2.1.1, 22 January 2022

[WInnF-Ref 17] WInnF Transceiver Facility Native C++ PSM specification, Document WINNF-TS-0008-App01, Version V2.1.1, 22 January 2022

[WInnF-Ref 18] WInnF Transceiver Facility FPGA PSM Specification, Document WINNF-TS-0008-App03, Version V2.1.1, 22 January 2022

[WInnF-Ref 19] WInnF Transceiver Facility Absolute Time Use Case, Document WINNF-TS-0008-App04, Version V2.1.1, 22 January 2022

WInnF Documents Website: https://sds.wirelessinnovation.org/specifications-and-recommendations

2 Software Communication Architecture (SCA)

SCA 2.2.2 specification from JTNC (earlier JTRS) brought standardization in SDR software architecture and it was adopted by alliance for ESSOR in their SDR program. SCA 4.1 specification is based on earlier version but with improvements to PIM and PSM structure. Numerous improvements [JTNC-Ref 34] [JTNC-Ref 35] are done along with backward compatibility with earlier specification.
This specification is for deployment, management, interconnection and intercommunication of different software components inside the SDR.

2.1 Core Specification

IRSA will also follow OMG model for its specification. It will provide set of APIs independent from any particular technology.

2.1.1 Candidate API Set

- Software Communications Architecture (SCA) Specification, Version 4.1- [JTNC-Ref 23] & Appendix A
- Software Communications Architecture (SCA) Specification, Version 2.2.2- [JTNC-Ref 01] & Appendix A

2.1.2 Selected API Set

- Software Communications Architecture (SCA) Specification, Version 4.1- [JTNC-Ref 23] & Appendix A

2.1.3 Modification

None

2.1.4 Rationale

SCA core specification is by default supported by most of the SDRs in India and has global acceptance. SCA 4.1 provided the necessary change required from basic SCA 2.2.2 and accommodated requirements of various stakeholders [JTNC-Ref 37] . It provides backward compatibility which is a necessary requirement from user perspective.

2.1.5 Conclusion

IRSA is adopting core SCA 4.1 specification without modification.

2.2 Associated Specification

SCA 4.1 is defined with the help of associated appendices.

2.2.1 SCA Application Environment Profile

2.2.1.1 Candidate API Set

- Appendix ‘B’ of Software Communications Architecture (SCA) Specification, Version 4.1 [JTNC-Ref 23]

2.2.1.2 Selected API Set

- Appendix 'B' of Software Communications Architecture (SCA) Specification, Version 4.1 [JTNC-Ref 23]

2.2.1.3 Modification

None

2.2.1.4 Rationale

AEP is closely tied with SCA 4.1 core specification and provides profiles for different form factors; hence no modification is required.

2.2.1.5 Conclusion

IRSA is adopting Appendix 'B' of SCA 4.1 specification without modification.

2.2.2 SCA CF IDL

2.2.2.1 Candidate API Set

- Software Communications Architecture (SCA) Specification, Version 4.1, Appendix-C [JTNC-Ref 26]

2.2.2.2 Selected API Set

- Software Communications Architecture (SCA) Specification, Version 4.1, Appendix-C [JTNC-Ref 26]

2.2.2.3 Modification

Current IDL is not modified for development in line with transfer mechanism described in SCA appendices. An IDL equivalent to Appendix C of SCA specification shall be released. It will be required in case of alternate transfer mechanism PSM.

2.2.2.4 Rationale

Depending on the chosen alternate transfer mechanism the IDL shall possibly be syntactically different from the OMG IDL. In case of radio realization without CORBA, alternate IDL can be utilized.

2.2.2.5 Conclusion

The CF IDL is adopted without modification for use with CORBA PSM. An equivalent IDL, if required, for chosen middleware/transfer mechanism will be created.

2.2.3 Domain Profile

2.2.3.1 Candidate API Set

- Software Communications Architecture (SCA) Specification, Version 4.1, Appendix D [JTNC-Ref 27] and Appendix D-1 [JTNC-Ref 28]

2.2.3.2 Selected API Set

- Software Communications Architecture (SCA) Specification, Version 4.1, Appendix D [JTNC-Ref 27] and Appendix D-1 [JTNC-Ref 28]

2.2.3.3 Modification
None

2.2.3.4 Rationale
XML is widely adopted and can be implemented efficiently and easily. It provides required dynamism required for SDR information utilization. No modification is required in Domain Profile specification of SCA.

2.2.3.5 Conclusion
Adopted without any modifications.

2.2.4 Transfer Mechanism

2.2.4.1 Candidate API Set
- Software Communications Architecture (SCA) Specification, Version 4.1, Appendix-E [JTNC-Ref 29] , Appendix E-1 [JTNC-Ref 30] , Appendix E-2 [JTNC-Ref 31] and Appendix E-3 [JTNC-Ref 32]

2.2.4.2 Selected API Set
- Software Communications Architecture (SCA) Specification, Version 4.1, Appendix-E [JTNC-Ref 29] , Appendix E-1 [JTNC-Ref 30] , Appendix E-2 [JTNC-Ref 31] and Appendix E-3 [JTNC-Ref 32]

2.2.4.3 Modification
None.

2.2.4.4 Rationale
CORBA is well suited to HEDS and therefore no modifications were deemed necessary. However, for the platforms that utilize multi-core SoCs for realization of single channel or single security domain, it is felt that a light-weight and simple alternative to CORBA is required.

2.2.4.5 Conclusion
CORBA PSM has been adopted without modification as one of the possible transfer mechanisms. Other mechanisms shall be explored and additional PSM shall be released in future after reference implementation.

2.2.5 Profiles

2.2.5.1 Candidate API Set
- Software Communications Architecture (SCA) Specification, Version 4.1, Appendix-F [JTNC-Ref 33]

2.2.5.2 Selected API Set

- Software Communications Architecture (SCA) Specification, Version 4.1, Appendix-F [JTNC-Ref 33] with modifications

2.2.5.3 Modification

The SCA Profiles in Section F.8 of the [JTNC-Ref 33] has been modified as follows

2.2.5.3.1 SCA Lightweight Profile

2.2.5.3.1.1 Mandatory Units of Functionality (UoF)

- AEP Provider
- Deployment
- Application Installable
- Management Registration

2.2.5.3.1.2 Optional Units of Functionality (UoF)

None

2.2.5.3.2 SCA Medium Profile

2.2.5.3.2.1 Mandatory Units of Functionality (UoF)

- AEP Provider
- Deployment
- Application Installable
- Management Registration
- AppDeploymentData
- AppReleasable
- DeviceMgrDeploymentData
- EventChannel
- Log Capable
- Log Producer
- Nested Deployment
- PlatformComponentFactoryDeployment

2.2.5.3.2.2 Optional Units of Functionality (UoF)

- CORBA Provider

2.2.5.3.3 SCA Full Profile

2.2.5.3.3.1 Mandatory Units of Functionality (UoF)

- AEP Provider
- Deployment
- Application Installable
- Management Registration

- AppDeploymentData
- AppReleasable
- DeviceMgrDeploymentData
- EventChannel
- Log Capable
- Log Producer
- Nested Deployment
- PlatformComponentFactoryDeployment
- Channel Extension
- Management Un-Registration
- Management Releasable

2.2.5.3.3.2 Optional Units of Functionality (UoF)

- CORBA Provider

2.2.5.4 Rationale

Modifications are required in SCA profile since too many optional UOF may lead to variety of SDR platform implementations for same type of SDR form factors.

2.2.5.5 Conclusion

Adopted with modifications in SCA profile to provide uniformity and consistency in single type of SDR platform by different manufacturers.

3 Platform API Specification

JTRS after release of SCA2.2.2 specification provided different APIs ([JTNC-Ref 14] to [JTNC-Ref 21]) for SDR platform abstraction. Alliance for ESSOR released 2 documents namely Radio Devices API Description Document [ESSOR-Ref 03] and Radio Services API Description Document [ESSOR-Ref 02] covering radio platform APIs used in different ESSOR Architecture based SDRs. WINNF released different APIs and in 2020 release facility framework for all future API specification.

In subsequent section details are provided about facility framework and APIs adopted by IRSA from earlier specification.

Following ESSOR service APIs are not considered in IRSA as they are related to management and control. Security aspect may come into consideration for these APIs.

i. Configuration Service
ii. Fault Management Service
iii. HMI Service
iv. Retransmission Service2

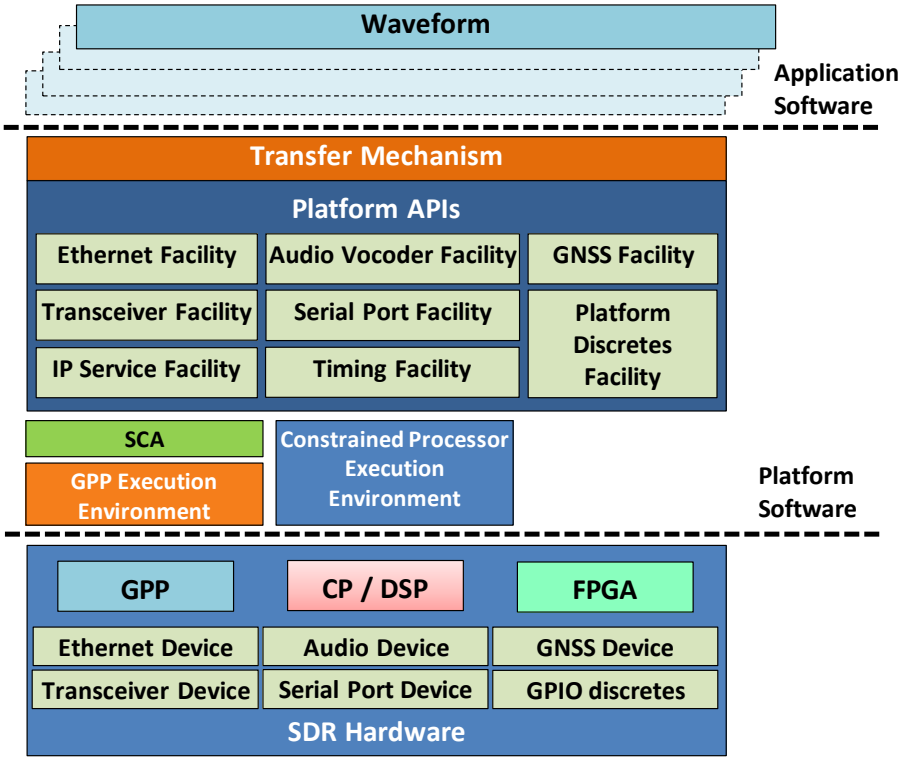

Figure 3.1 : Plaform API

3.1 Facility Framework

3.1.1 Introduction

WINNF facility framework is comprehensive and follows OMG PIM & PSM model in specifying various SDR platform API set. Hardware abstractions (devices) and software abstractions (services) through facility framework assist in meeting objective of waveform portability over different processing environment (GPP, DSP & FPGA).

3.1.2 Rationale

The abstractions for SDR platforms specified by WinnF Facilities ([WInnF-Ref 09], [WInnF-Ref 11], [WInnF-Ref 15]), ESSOR API Description documents [ESSOR-Ref 01], [ESSOR-Ref 02], [ESSOR-Ref 03]) and JTRS Standard Device & Service APIs ([JTNC-Ref 14] to [JTNC-Ref 21]) respectively, were considered for finalization of specification approach for IRSA software and hardware abstractions. The WinnF facility framework specifies the platform abstractions as facilities and addresses all aspects like architectural principles, functional capabilities. It addresses these aspects in terms of 'provides' and 'uses' services, real-time concepts, PIM and PSM specifications for Native C++, SCA & FPGA. Therefore, the WinnF facility framework was most suitable method for specification of platform abstractions.

3.1.3 Approach

The WInnF Facility framework has been adopted for the specification of various software (Services) and hardware (Devices) abstractions in an IRSA compliant SDR platform. Therefore the "Principles for WInnForum Facility Standards" [WInnF-Ref 09] has been followed for the PIM specification of the hardware and software abstractions. Multiple PSMs of the specified PIMs shall be released in due course of time.

Few of the facility specifications have been adopted without any modification. For other facilities either ESSOR or JTRS API has been selected as reference. The modifications in the APIs mainly pertain to consolidation of multiple related APIs and removal of duplicate/redundant/obsolete APIs. Final API set is adapted to the WInnF facility framework template and is specified as IRSA facility.

3.2 Primitive APIs

JTRS had released different primitive APIs with some functionality repeated in different APIs. IRSA platform abstraction facility may be dependent on these APIs. Only selected APIs will be part of any facility under IRSA.

3.2.1 Approach

3.2.1.1 Candidate API Set

Following primitive JTRS APIs are considered, [JTNC-Ref 02] [JTNC-Ref 03] [JTNC-Ref 04] [JTNC-Ref 05] [JTNC-Ref 07] [JTNC-Ref 08] [JTNC-Ref 09] [JTNC-Ref 11] [JTNC-Ref 13]

i. DeviceIo

ii. DeviceIoControl
iii. DeviceIoSignals
iv. DeviceMessageControl
v. DevicePacket
vi. DeviceSimplePacket
vii. DevicePacketSignals
viii. DeviceSimplePacketSignals
ix. Packet

3.2.1.2 Selected API Set

i. DeviceIo [JTNC-Ref 13]
ii. DeviceMessageControl [JTNC-Ref 07]
iii. Packet [JTNC-Ref 08]

3.2.1.3 Modification

Signatures of selected APIs will be retained without modifications.

3.2.1.4 Rationale

These APIs have been utilized only by Serial Port Facility in this document. However, these APIs are provided for utilization by other facilities in future. Other APIs have been removed due to ambiguity, redundancy and obsolescence reasons.

3.2.1.5 Conclusion

Selected primitive APIs are provided in this document for completeness and support for future extensions of facilities.

3.2.2 Primitive API Details

3.2.2.1 DeviceIo

3.2.2.1.1 DeviceIo::DeviceIoControl

3.2.2.1.1.1 enableRtsCts() operation

Refer section A.3.1.1 of [JTNC-Ref 13]

3.2.2.1.1.2 setRts() operation

Refer section A.3.1.2 of [JTNC-Ref 13]

3.2.2.1.2 DeviceIo::DeviceIoSignals

Refer section A.2.3.1.2 of [JTNC-Ref 13]

3.2.2.1.2.1 setCts() operation

Refer section A.3.2.1 of [JTNC-Ref 13]

3.2.2.2 DevMsgCtl

3.2.2.2.1 DevMsgCtl::DeviceMessageControl

3.2.2.2.1.1 rxActive() operation
Refer section A.3.1.2 of [JTNC-Ref 07]

3.2.2.2.1.2 txActive() operation
Refer section A.3.1.3 of [JTNC-Ref 07]

3.2.2.2.1.3 abortTx() operation
Refer section A.3.1.1 of [JTNC-Ref 07]

3.2.2.3 Packet

3.2.2.3.1 Packet::PayloadStatus

3.2.2.3.1.1 getMaxPayloadSize() operation
Refer section A.3.1.1 of [JTNC-Ref 08]

3.2.2.3.1.2 getMinPayloadSize() operation
Refer section A.3.1.2 of [JTNC-Ref 08]

3.2.2.3.1.3 getDesiredPayloadSize() operation
Refer section A.3.1.3 of [JTNC-Ref 08]

3.2.2.3.1.4 getMinOverrideTimeout() operation
Refer section A.3.1.4 of [JTNC-Ref 08]

3.2.2.3.2 Packet::PayloadControl

3.2.2.3.2.1 setMaxPayloadSize() operation
Refer section A.3.2.1 of [JTNC-Ref 08]

3.2.2.3.2.2 setMinPayloadSize() operation
Refer section A.3.2.2 of [JTNC-Ref 08]

3.2.2.3.2.3 setDesiredPayloadSize() operation
Refer section A.3.2.3 of [JTNC-Ref 08]

3.2.2.3.2.4 setMinOverrideTimeout() operation
Refer section A.3.2.4 of [JTNC-Ref 08]

3.2.2.3.3 Packet::FlowSignals

3.2.2.3.3.1 signalResume() operation
Refer section B.3.2.1 of [JTNC-Ref 08]

3.2.2.3.4 Packet::EmptySignals

3.2.2.3.4.1 signalEmpty() operation

Refer section C.3.2.1 of [JTNC-Ref 08]

3.2.2.3.5 Packet::FlowControl

3.2.2.3.5.1 enableFlowResumeSignals() operation

Refer section B.3.1.1 of [JTNC-Ref 08]

3.2.2.3.5.2 spaceAvailable() operation

Refer section B.3.1.2 of [JTNC-Ref 08]

3.2.2.3.6 Packet::FlowOctetStream

3.2.2.3.6.1 pushPacket() operation

Refer section B.3.3.1 of [JTNC-Ref 07]

3.2.2.3.7 Packet::EmptyControl

3.2.2.3.7.1 enableEmptySignals() operation

Refer section C.3.1.1 of [JTNC-Ref 07]

3.3 Ethernet Facility

3.3.1 Approach

3.3.1.1 Candidate API Set

- ESSOR Ethernet Device API - [ESSOR-Ref 03]
- JTRS Ethernet Device API - [JTNC-Ref 06]

3.3.1.2 Selected API Set

ESSOR Ethernet Device API - [ESSOR-Ref 03]

3.3.1.3 Modification

No modification in APIs except exclusion of network management APIs and to adapt in PIM of facility framework, suitable modifications are done to remove any SCA specific terminology.

3.3.1.4 Rationale

Above referenced API is sufficient to cater for requirements of platform and waveform components. ESSOR Ethernet Device API specification was in line with SCA and to create PIM in Facility Framework, suitable modifications are required. The network management interface has been excluded to prohibit access to radio application.

3.3.1.5 Conclusion

ESSOR API is adopted without the network management interface and provided in the form of PIM specification as per Principles for WInnF Facility Standards [WInnF-Ref 09].

PSM Documents (SCA, Native C++, FPGA) corresponding to this facility PIM specification as applicable will be released after their reference implementation.

3.3.2 Introduction

Section 3.3 provides the PIM specification of IRSA Ethernet Facility. The specification of Ethernet facility is applicable to diverse platforms irrespective of underlying hardware and software.

3.3.2.1 Overview

The Ethernet Facility provides a standardized access to the Ethernet hardware for components that require Ethernet protocol to transmit or receive data. It mainly provides the following functionalities.

- Abstracts the native Ethernet port of the host machine.
- Provides services to the IP Service component.
- Sends and receives IP packets to and from the IPService component.
- Packet capture & injection from/to Ethernet device.

This facility also describes the APIs related to the basic Ethernet Device and to the following extensions:

- Mode Configuration Extension
- Multicast Mode Extension
- Promiscuous Mode Extension
- Header Configuration Extension
- MAC Address Extension

3.3.2.2 Definitions

None.

3.3.2.3 Technical Details

Refer 6.1.1 and 6.1.2 of [ESSOR-Ref 03] for usage context and general behavior of Ethernet facility in SCA specific platform.

3.3.3 Services

3.3.3.1 Provide Services

Service Groups / Module	Services / Interfaces	Primitives
EthernetData	EthernetPacketProducer	setMaxPayloadSize() setMinPayloadSize() signalFlowResume()
	EthernetPacketConsumer	pushPacket() getMaxPayloadSize() getMinPayloadSize()

		enableFlowResumeSignals()
EthernetConfig	EthernetAddress	getMacAddress()
	EthernetModeConfig	getMode() setMode()
	EthernetHeaderConfig	getRetainHeader() setRetainHeader()

Table 3.1: Ethernet Facility Provide Services

3.3.3.2 Use Services

Service Groups / Module	Services / Interfaces	Primitives
EthernetData	EthernetPacketProducer	setMaxPayloadSize() setMinPayloadSize() signalFlowResume()
	EthernetPacketConsumer	pushPacket(){sendPacket() in ESSOR} getMaxPayloadSize() getMinPayloadSize() enableFlowResumeSignals()

Table 3.2: Ethernet Facility Use Services

3.3.3.3 State Machines

3.3.3.3.1 Ethernet_SM

Ethernet_SM is specified as the main state machine followed by Ethernet Facility. An instance of Ethernet_SM is followed by each Ethernet instance.

The following figure is the statechart of Ethernet_SM state machine:

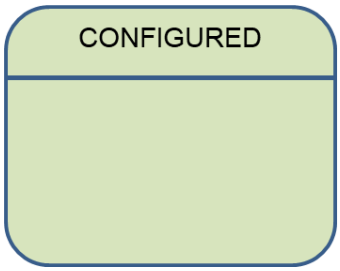

Figure 3.2 : Ethernet_SM statechart

3.3.3.3.1.1 States

3.3.3.3.1.2 CONFIGURED

CONFIGURED is specified as the unique state of Ethernet, during which it is configured according to the requirements of all the radio application to be supported during the CONFIGURED state.

CONFIGURED is reached by Ethernet when it

- Complies with any value specified for a capability or a property,
- Is capable of interacting with radio application according to the service interfaces of its service implementations.

How CONFIGURED is reached is unspecified by the PIM specification, and can be specified by the applied PSM specification.

3.3.3.4 Service groups

3.3.3.4.1 Ethernet::EthernetData

This group provides data exchange capability between radio application and Ethernet.

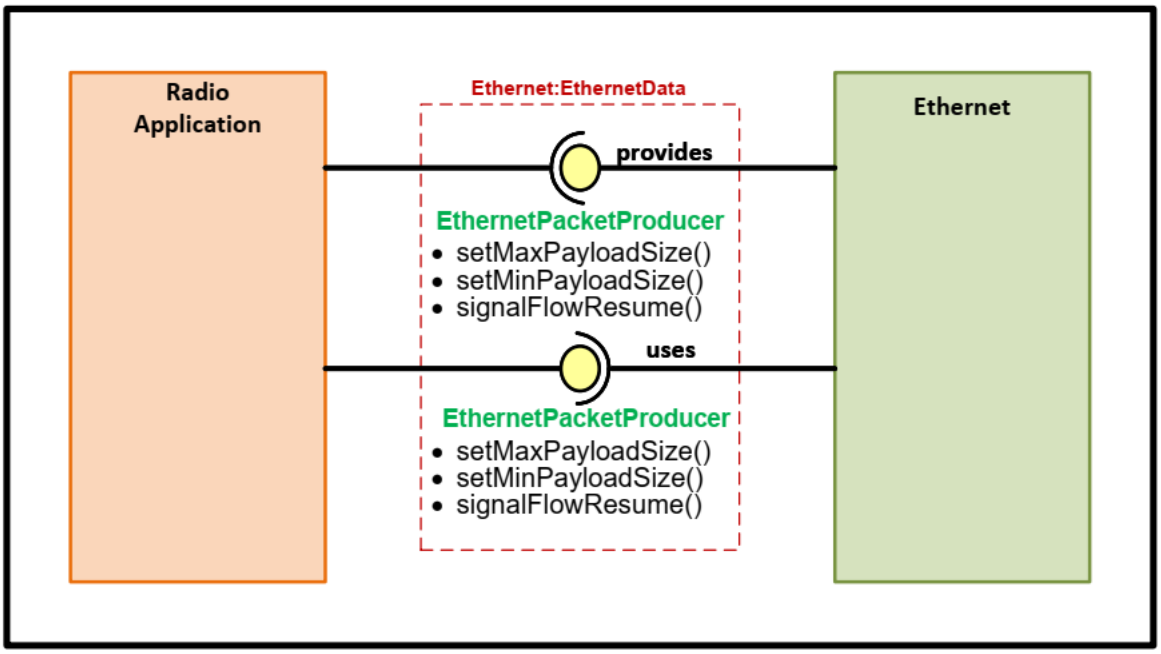

Figure 3.3 : EthernetData (EthernetPacketProducer) services group

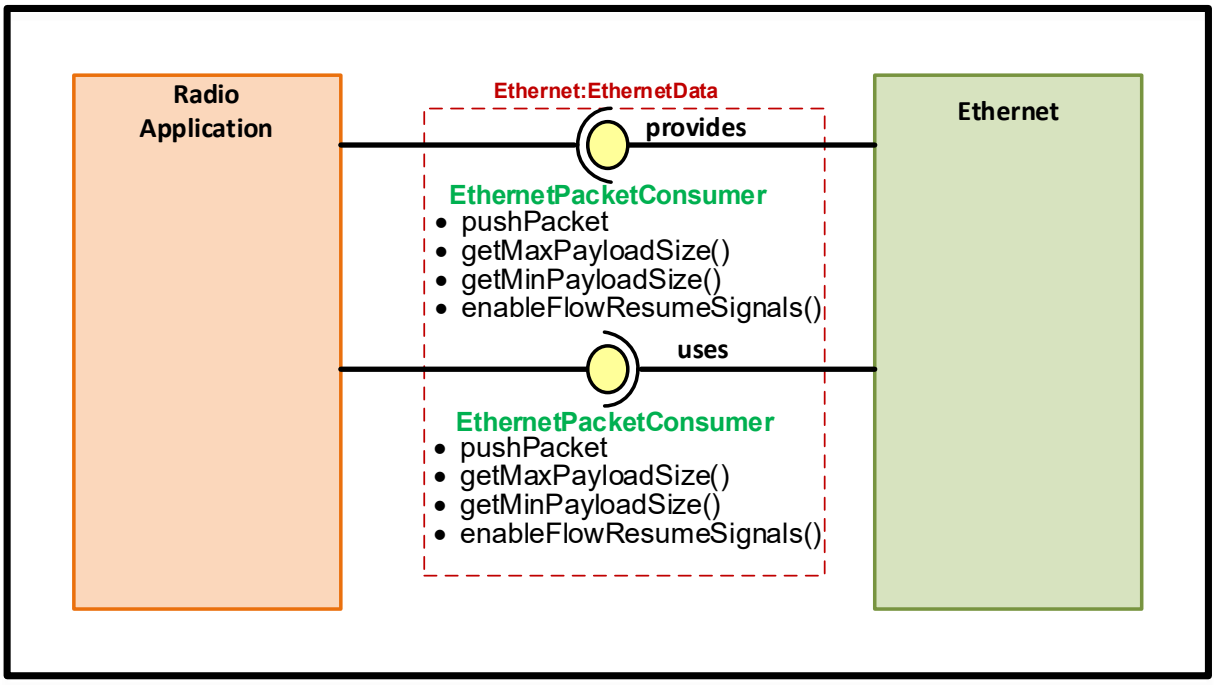

Figure 3.4 : EthernetData (EthernetPacketConsumer) services group

3.3.3.4.2 Ethernet::EthernetConfig

This group enables Radio application to configure Ethernet like header configuration, Mode Configuration and Mac address access.

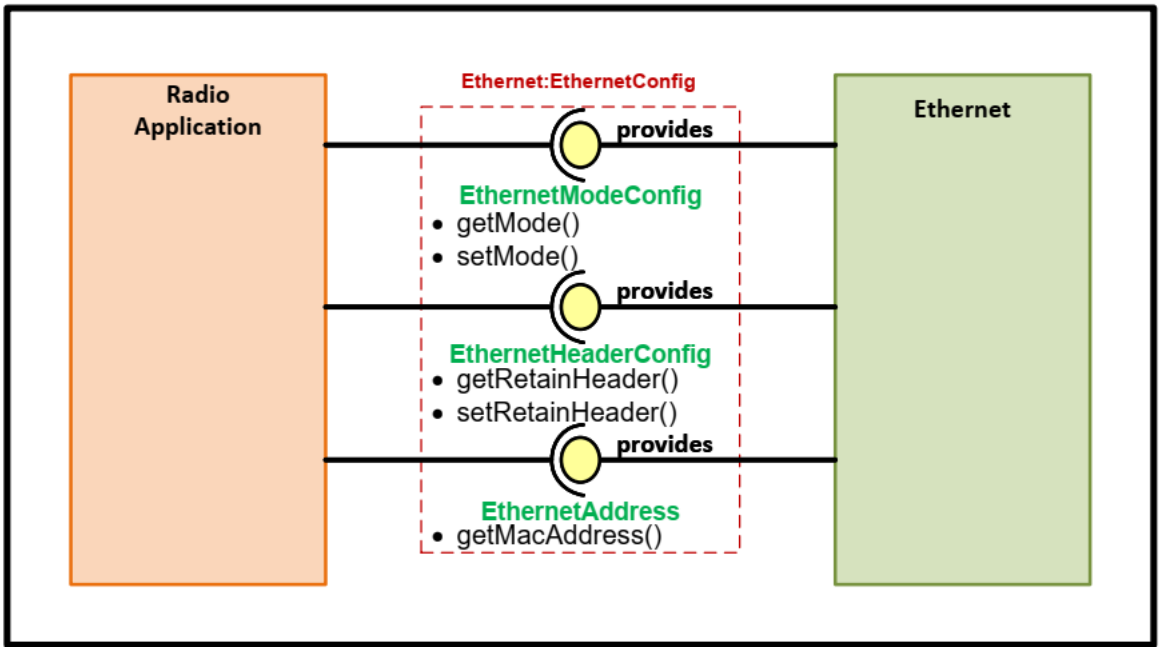

Figure 3.5 : EthernetConfig services group

3.3.4 Service Primitives

This section specifies the primitives of the Ethernet facility services. Each declaration of

a primitive complies with the Full PIM IDL Profile of WInnF IDL profiles for PIM of SDR Applications, specified in [WInnF-Ref 05]. The declaration of each primitive also complies with SCA 4.1 Appendix E-1 [JTNC-Ref 30] .

3.3.4.1 EthernetData::EthernetPacketProducer

3.3.4.1.1 setMaxPayloadSize()

3.3.4.1.1.1 Overview
Refer 6.3.1.3.3.1 of [ESSOR-Ref 03]

3.3.4.1.1.2 Signatures
Refer 6.3.1.3.3.1.1 of [ESSOR-Ref 03]

3.3.4.1.1.3 Parameters
Refer 6.3.1.3.3.1.2 of [ESSOR-Ref 03]

3.3.4.1.1.4 Exceptions
Refer 6.3.1.3.3.1.6 of [ESSOR-Ref 03]

3.3.4.1.1.5 Attributes
None.

3.3.4.1.1.6 Behavior requirements
Refer 6.3.1.3.1 of [ESSOR-Ref 03]

3.3.4.1.2 setMinPayloadSize()

3.3.4.1.2.1 Overview
Refer 6.3.1.3.3.2 of [ESSOR-Ref 03]

3.3.4.1.2.2 Signatures
Refer 6.3.1.3.3.2.1 of [ESSOR-Ref 03]

3.3.4.1.2.3 Parameters
Refer 6.3.1.3.3.2.2 of [ESSOR-Ref 03]

3.3.4.1.2.4 Exceptions
Refer 6.3.1.3.3.2.6 of [ESSOR-Ref 03]

3.3.4.1.2.5 Attributes
None.

3.3.4.1.2.6 Behavior requirements
Refer 6.3.1.3.1 of [ESSOR-Ref 03]

3.3.4.1.3 signalFlowResume()

3.3.4.1.3.1 Overview
Refer 6.3.1.3.3.3 of [ESSOR-Ref 03]

3.3.4.1.3.2 Signatures
Refer 6.3.1.3.3.3.1 of [ESSOR-Ref 03]

3.3.4.1.3.3 Parameters
Refer 6.3.1.3.3.3.2 of [ESSOR-Ref 03]

3.3.4.1.3.4 Exceptions
Refer 6.3.1.3.3.3.6 of [ESSOR-Ref 03]

3.3.4.1.3.5 Attributes
None.

3.3.4.1.3.6 Behavior requirements
Refer 6.3.1.3.1 of [ESSOR-Ref 03]

3.3.4.2 EthernetData::EthernetPacketConsumer

3.3.4.2.1 pushPacket() operation
ESSOR sendPacket() operation has been renamed as pushPacket().

3.3.4.2.1.1 Overview
Refer 6.3.1.2.3.1 of [ESSOR-Ref 03]

3.3.4.2.1.2 Signatures
boolean pushPacket (**in unsigned short** protocolType, **in** CF::OctetSequence address, **in** CF::OctetSequence payload);

3.3.4.2.1.3 Parameters
Refer 6.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.3.4.2.1.4 Exceptions
Refer 6.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.3.4.2.1.5 Attributes
None.

3.3.4.2.1.6 Behavior requirements
Refer 6.3.1.2.1 of [ESSOR-Ref 03]

3.3.4.2.2 getMaxPayloadSize()

3.3.4.2.2.1 Overview
Refer 6.3.1.2.3.3 of [ESSOR-Ref 03]

3.3.4.2.2.2 Signatures
Refer 6.3.1.2.3.3.1 of [ESSOR-Ref 03]

3.3.4.2.2.3 Parameters
Refer 6.3.1.2.3.3.2 of [ESSOR-Ref 03]

3.3.4.2.2.4 Exceptions
Refer 6.3.1.2.3.3.6 of [ESSOR-Ref 03]

3.3.4.2.2.5 Attributes
None.

3.3.4.2.2.6 Behavior requirements
Refer 6.3.1.2.1 of [ESSOR-Ref 03]

3.3.4.2.3 getMinPayloadSize()

3.3.4.2.3.1 Overview
Refer 6.3.1.2.3.4 of [ESSOR-Ref 03]

3.3.4.2.3.2 Signatures
Refer 6.3.1.2.3.4.1 of [ESSOR-Ref 03]

3.3.4.2.3.3 Parameters
Refer 6.3.1.2.3.4.2 of [ESSOR-Ref 03]

3.3.4.2.3.4 Exceptions
Refer 6.3.1.2.3.4.6 of [ESSOR-Ref 03]

3.3.4.2.3.5 Attributes
None.

3.3.4.2.3.6 Behavior requirements
Refer 6.3.1.2.1 of [ESSOR-Ref 03]

3.3.4.2.4 enableFlowResumeSignals()

3.3.4.2.4.1 Overview
Refer 6.3.1.2.3.2 of [ESSOR-Ref 03]

3.3.4.2.4.2 Signatures
Refer 6.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.3.4.2.4.3 Parameters
Refer 6.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.3.4.2.4.4 Exceptions

Refer 6.3.1.2.3.2.6 of [ESSOR-Ref 03]

3.3.4.2.4.5 Attributes
None.

3.3.4.2.4.6 Behavior requirements
Refer 6.3.1.2.1 of [ESSOR-Ref 03]

3.3.4.3 EthernetConfig::EthernetModeConfig

3.3.4.3.1 setMode()

3.3.4.3.1.1 Overview
Refer 6.4.3.1.2.3.1 of [ESSOR-Ref 03]

3.3.4.3.1.2 Signatures
Refer 6.4.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.3.4.3.1.3 Parameters
Refer 6.4.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.3.4.3.1.4 Exceptions
Refer 6.4.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.3.4.3.1.5 Attributes
None.

3.3.4.3.1.6 Behavior requirements
Refer 6.4.3.1.2.1 of [ESSOR-Ref 03]

3.3.4.3.2 getMode()

3.3.4.3.2.1 Overview
Refer 6.4.3.1.2.3.2 of [ESSOR-Ref 03]

3.3.4.3.2.2 Signatures
Refer 6.4.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.3.4.3.2.3 Parameters
Refer 6.4.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.3.4.3.2.4 Exceptions
Refer 6.4.3.1.2.3.2.6 of [ESSOR-Ref 03]

3.3.4.3.2.5 Attributes
None.

3.3.4.3.2.6 Behavior requirements

Refer 6.4.3.1.2.1 of [ESSOR-Ref 03]

3.3.4.4 EthernetConfig::EthernetHeaderConfig

3.3.4.4.1 getRetainHeader()

3.3.4.4.1.1 Overview
Refer 6.7.3.1.2.3.2 of [ESSOR-Ref 03]

3.3.4.4.1.2 Signatures
Refer 6.7.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.3.4.4.1.3 Parameters
Refer 6.7.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.3.4.4.1.4 Exceptions
Refer 6.7.3.1.2.3.2.6 of [ESSOR-Ref 03]

3.3.4.4.1.5 Attributes
None.

3.3.4.4.1.6 Behavior requirements
Refer 6.7.3.1.2.1 of [ESSOR-Ref 03]

3.3.4.4.2 setRetainHeader()

3.3.4.4.2.1 Overview
Refer 6.7.3.1.2.3.1 of [ESSOR-Ref 03]

3.3.4.4.2.2 Signatures
Refer 6.7.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.3.4.4.2.3 Parameters
Refer 6.7.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.3.4.4.2.4 Exceptions
Refer 6.7.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.3.4.4.2.5 Attributes
None.

3.3.4.4.2.6 Behavior requirements
Refer 6.7.3.1.2.1 of [ESSOR-Ref 03]

3.3.4.5 EthernetConfig::EthernetAddress

3.3.4.5.1 getMacAddress()

3.3.4.5.1.1 Overview
Refer 6.8.3.1.2.3.1 of [ESSOR-Ref 03]

3.3.4.5.1.2 Signatures
Refer 6.8.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.3.4.5.1.3 Parameters
Refer 6.8.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.3.4.5.1.4 Exceptions
Refer 6.8.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.3.4.5.1.5 Attributes
None.

3.3.4.5.1.6 Behavior requirements
Refer 6.8.3.1.2.1 of [ESSOR-Ref 03]

3.3.4.6 Exceptions
Refer to sections 6.3.1.3.3.1.6, 6.3.1.3.3.2.6, 6.4.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.3.4.7 Type
None

3.3.5 Ethernet Facility Attributes

3.3.5.1 Capabilities

Name	Type	Description	Range/Values
MODE	unsigned short	The mode configuration supported by the Ethernet device.	MODE_NONE, MODE_STANDARD, MODE_MULTICAST, MODE_PROMISCUOUS
MODE_CONFIGURATION_SUPPORTED	boolean	Indicates whether the mode configuration support is provided or not	TRUE indicates mode configuration is supported, FALSE otherwise.
MULTICAST_MODE_SUPPORTED	boolean	Indicates whether the multicast mode support is provided or not	TRUE indicates multicast mode is supported, FALSE otherwise.
PROMISCUOUS_MODE_SUPPORTED	boolean	Indicates whether the Promiscuous mode support is provided or not	TRUE indicates Promiscuous mode is supported, FALSE otherwise.
HEADERCONFIGURATION_SUPPORTED	boolean	Indicates whether the Ethernet Header Configuration is	TRUE indicates Ethernet Header Configuration is supported, FALSE

		supported or not	otherwise.
MACADDRESS_ACESS_SUPPORTED	boolean	Indicates whether the MAC Address access is provided or not	TRUE indicates MAC Address access is provided, FALSE otherwise.

Table 3.3: Ethernet Facility General Capabilities

3.3.5.2 Properties

Name	Type	Description	Range Value
MAX_PAYLOAD_SIZE	unsigned long	Maximum payload size allowed for a payload.	[512, 16383]
MIN_PAYLOAD_SIZE	unsigned long	Minimum payload size allowed for a payload.	[0, 512]
MODE	enum EthernetMode	The mode to configure the Ethernet device.	As per the enumerations for Ethernet mode.
INTERFACE_NAME	string	A string representing the name of the interface.	N/A
MAC*	Unsigned char (octet sequence)	The MAC address of the Ethernet port.	[0x000000000000, 0xFFFFFFFFFFFF]

* MAC is query-only attribute

Table 3.4: Ethernet Facility Properties

3.3.5.3 Variables

None

3.4 Audio Vocoder Facility

3.4.1 Approach

Both JTNC and ESSOR architectures define the Audio Port Device which is responsible for providing Tone creation, PTT Signaling as well as audio conversion functions (conversion between digital voice and analog voice). Apart from Audio Port Device, JTNC and ESSOR architectures define the vocoder service which performs the audio processing (vocoding) functions on the audio voice samples. Since Audio and vocoder functions are strongly related, IRSA combines the Audio functions and the vocoder functions into a single facility called AudioVocoder Facility (AudVoc)

3.4.1.1 Candidate API Set

- ESSOR Audio Port / Vocoder API - [ESSOR-Ref 03]
- JTNC Audio Port Device API - [JTNC-Ref 14]
- JTNC Vocoder Service API - [JTNC-Ref 21]

3.4.1.2 Selected API Set

ESSOR Audio Port / Vocoder API - [ESSOR-Ref 03].

3.4.1.3 Modifications

IRSA has taken ESSOR APIs used for configuration of a generic vocoder algorithm instead of specific vocoder algorithm configuration APIs. Modifications are done for audio data packet transfer API to make it more specific to the type of data being transferred instead of using generic packet API. Additional APIs are defined for controlling the audio conversion and vocoder functions within the same facility. To support radio platform having multiple audio ports for a single audio channel, additional capabilities are defined to support for multiple audio ports multiplexed/mixed for a single audio channel.

3.4.1.4 Rationale

ESSOR Audio Port/Vocoder API [ESSOR-Ref 03] includes all capabilities/APIs from JTNC audio device API [JTNC-Ref 14] and vocoder service API [JTNC-Ref 21] with additional capabilities and APIs. ESSOR Audio Port/Vocoder API is chosen as reference for IRSA Audio/Vocoder facility. Since audio and vocoder functionality are closely linked, both audio and vocoder functions are combined in a single facility with new attribute to configure the facility for different deployment scenario.

3.4.1.5 Conclusion

ESSOR Audio Port/Vocoder API is chosen as reference for IRSA Audio/Vocoder facility PIM and provided in the form of PIM specification as per Principles for WInnF Facility Standards [WInnF-Ref 09]. Most of the APIs are directly used, with addition of new APIs mainly for audio/vocoder data streams, second audio channel and multiple audio port support. Addition attributes are defined with respect to combining audio and vocoder functionality within a single facility. PSM Documents (SCA, Native C++, FPGA) corresponding to this facility PIM specification as applicable will be released after their reference implementation.

3.4.2 Introduction

3.4.2.1 Overview

AudioVocoder (AudVoc) Facility provides Audio related functions for the radio application. AudioVocoder facility sits between the Audio stream handled by the Radio Application and audio end point (Microphone/Speaker). Figure 3.6 provides the context for the AudioVocoder Facility.

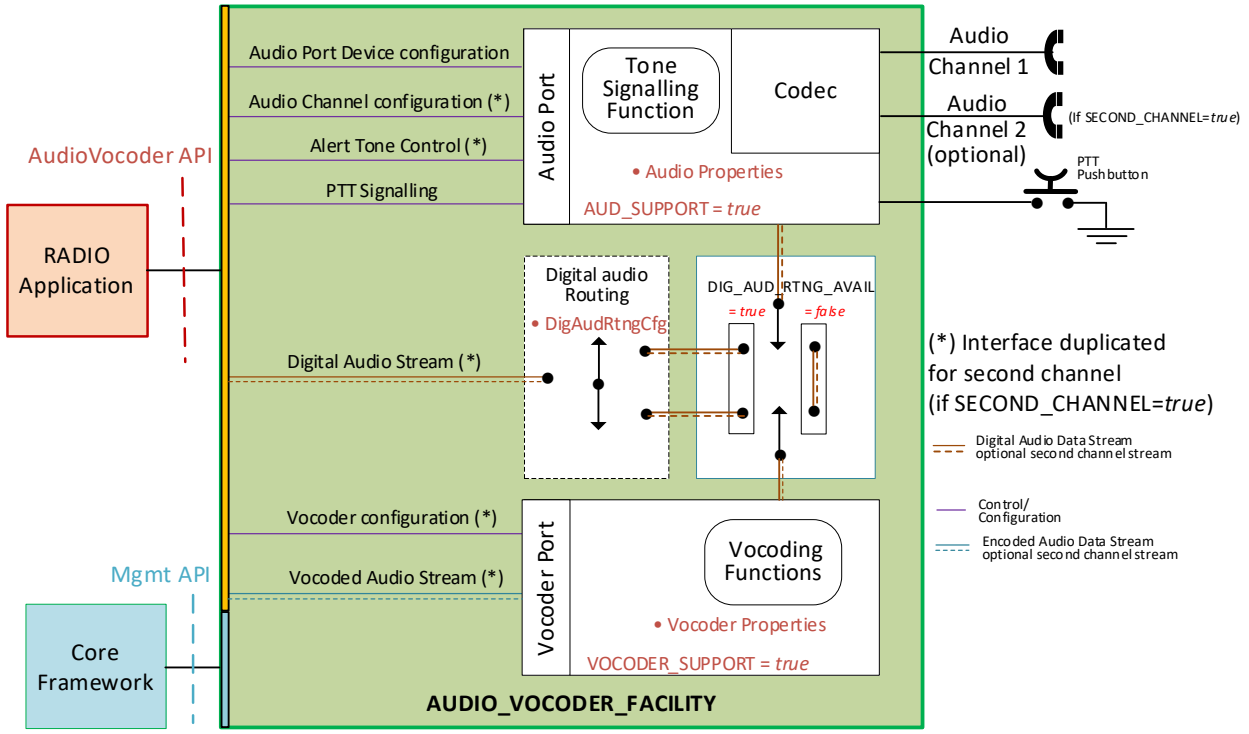

Figure 3.6 : AudioVocoder Context

As special case, AudioVocoder Facility caters to the configuration where radio platform supports multiple audio end points (as different independent audio ports) participating as a single radio. Support of this radio configuration is mentioned as multiple audio port case. Figure 3.7 provides the context for the AudioVocoder facility for multiple audio ports case. Multiple audio port is treated as an optional functionality of AudioVocoder facility which does not affect the other functions of the facility.

3.4.2.2 Definitions

Audio Conversion (CODEC)

- Conversion of analog audio to/from digital samples, basically ADC/DAC functionality.

Codec

- HW performing the audio conversion function.
- Apart from the ADC/DAC functionality, codec can have configurable gain control, AGC function, programable filters, multiple analog input/output, and its routing towards the DAC/ADC. Can have one or two channels (mono or stereo)
- Please refer CODEC definition in section 9.2 of [ESSOR-Ref 03].

Sampling Frequency

- Sampling frequency (rate) is defined as the sampling rate of the digital audio signal, in samples/s.

- Typically for voice, sampling frequency is 8000 (8000 samples/second).

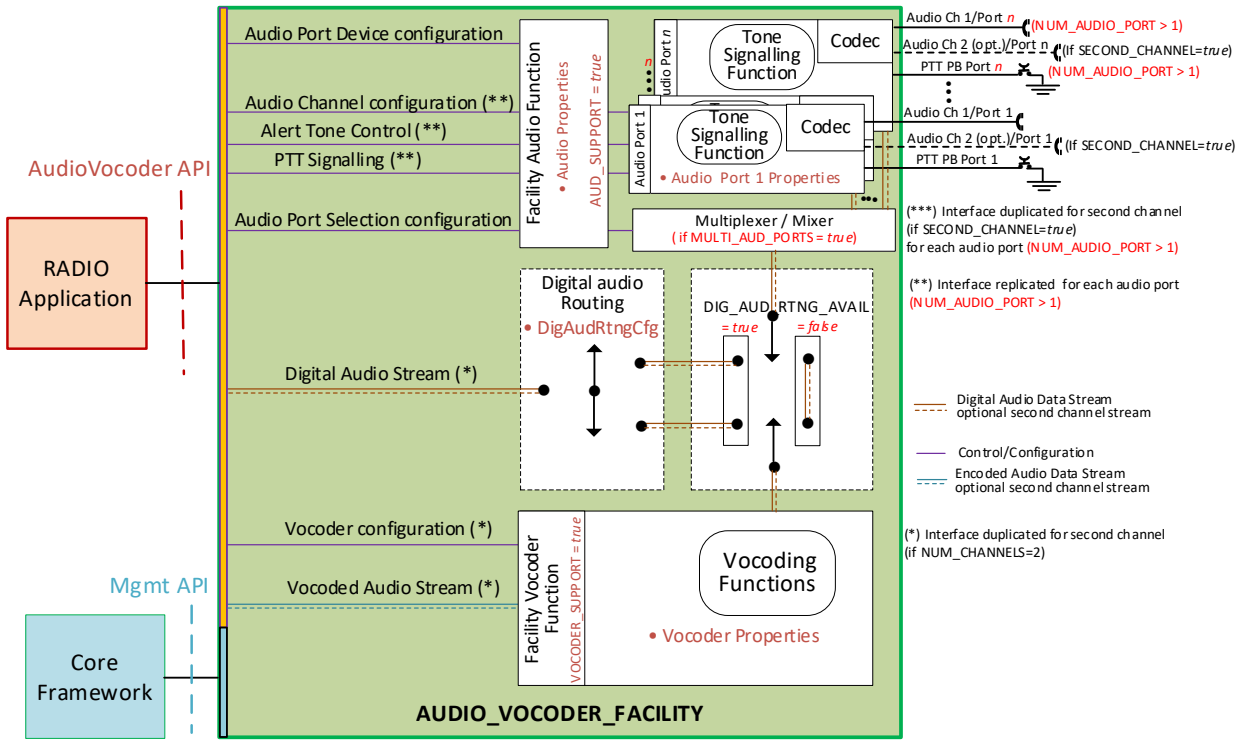

Figure 3.7 : AudioVocoder Facility Context (Multiple Audio Port case)

Sample size

- Size of each sample generated specified in number of bits. Values can be 8,16,20,24 or 32 bits.

Audio Channel

- Please refer 'Audio Channel' definition in sec 9.2 of [ESSOR-Ref 03].

Sidetone

- Please refer sidetone definition in sec 9.2 of [ESSOR-Ref 03].

Audio Compression (ACP function)

- Please refer 'Audio compression' definition in sec 9.2 of [ESSOR-Ref 03].

Voice processing function (vocoder function)

- Voice encoding and decoding function to reduce the baud rate of the digital audio stream.

Vocoder

- HW / FW / SW functions performing the voice processing functions. Decoder takes vocoded data and coverts into raw audio samples. Encoder takes the raw audio samples and converts into encoded audio data.

Digital Audio or Raw audio

- Refers to raw audio samples as per sampling rate and size from/to audio conversion function. Please refer 'Raw voice traffic' definition in sec 9.2 of [ESSOR-Ref 03].

Vocoded Audio or Encoded Audio

- Output Audio bits from vocoder after voice processing function (or to the voice processing function for decoding). Please refer 'Encoded voice traffic' definition in sec 9.2 of [ESSOR-Ref 03].

Tx Vocoder (Encoder)

- Encoding Function converting raw audio samples into encoded audio data.

Rx Vocoder (Decoder)

- Decoding Function converting encoded audio data to raw audio samples.

Continuous Stream Vocoder

- Vocoder Algorithm which generates continuous stream of encoded bits. Please refer Continuous stream vocoder description as part of Appendix C of [ESSOR-Ref 03].

Frame-Based Vocoder

- Vocoder algorithm which generates a frame with well-defined bit structure. The number of bits in the frame is dependent upon the algorithm and the bit rate and may not be multiple of 8. Please refer Frame-based vocoder description as part of Appendix C of [ESSOR-Ref 03].

Audio end point

- Analogue audio termination/origination point basically speaker/microphone/headset or equivalent.

Audio Port

- Indicates Audio conversion and related function where the speaker/Microphone are connected. Basically, it relates to all function between the audio endpoint and digital audio data routing which include audio conversion, PTT signaling and tone generation. In case of Multiple Audio Ports (MULTI_AUDIO_PORTS = true), digital audio channel data from each audio port is connected to Multiplexer/Mixer function. All audio port supports the same number of audio channels (SECOND_CHANNEL)

Multiplexer/Mixer

- Functionality enabled only for multiple audio ports.
- For upstream digital audio, function include selecting/mixing digital audio samples from individual audio ports towards radio application/vocoder function.

- For downstream digital audio, function include routing the digital audio samples from radio application/vocoder function to the audio ports.
- In case support of second audio channel, functionality is handled independent of the first channel.

3.4.2.3 Technical Details

AudioVocoder facility provides functions related to two main categories -

- Audio Port functions
- Vocoder Functions

Some facility implementations may choose to support only one of these functions indicated by capability attributes. In case of facility supporting both audio port and vocoder functions, facility may provide additional Digital Audio Routing configuration, which enables deployment of different type of radio application having different needs from the facility. In case of radio platform supporting multiple audio ports, individual audio port functions are instantiated for each of audio ports. Digital audio data streams from multiple audio ports are multiplexed/mixed into a single digital audio data stream for digital audio routing/vocoder functions. Similarly single digital audio data stream from digital audio routing/vocoder functions is forwarded to a single or all audio ports endpoints. Apart from introduction of multiplexer/mixer functions, other functionality remains same for multiple audio port. So, a radio platform supporting multiple audio port can act as single audio port platform with no support from radio application.

3.4.2.3.1 Audio Port Functions

Audio Port functions of the facility includes-

- PTT signaling indication.
- Tone Generation
- Audio conversion functions

PTT signaling indication function indicates the status of an external push button switch, whether the button is pressed or released. Tone generation function allows the radio application to pre-configure different type of tone profiles and controlling the generation of these tones towards the audio endpoint. Audio conversion function primarily relates to CODEC hardware having ADC and DAC units. Analogue audio from the microphone/headset is converted into digital audio samples stream using the ADC of CODEC. Similarly digital audio samples stream is converted into analogue signal towards speaker/headset using the DAC of CODEC. This creates a single bi-direction audio path with the audio end point. ADC/DAC functionality is controlled by sampling frequency/sample size configuration. Additionally, CODEC can support Audio compressor capabilities, sidetone generation capability and control of input and output gain for the audio channel.

3.4.2.3.2 Vocoder Function

Vocoder Function relates processing of voice processing function (a.k.a vocoder algorithm) primarily aiming at reducing the bandwidth occupied by the signal with change in quality within acceptable range. Tx Vocoder (a.k.a Encoder) function takes digital audio samples as input and produces vocoded audio data. Similarly, Rx Vocoder (a.k.a Decoder) function take vocoded audio data as input and produces digital audio samples. Facility can support multiple standard vocoder algorithm with configuration specific to the algorithm. Facility can choose the implementation of different vocoder algorithm in SW/FW or Hardware with no effect on radio application. Facility supports having different algorithm/configuration for the Rx and Tx vocoder.

Tx vocoder algorithm can support VAD/DTX functions which detects voice activity on the digital audio input. On silence detection, vocoder generates comfort noise/silence detection frames on vocoder output. This can be used by the radio application to use discontinuous transmission to reduce transmission bandwidth/power requirement during silence periods. Rx vocoder can use the Comfort Noise vocoder frame to generate suitable background during the discontinuity period.

Rx Vocoder algorithm can also have packet loss concealment algorithm which produces suitable digital audio sample as to reduce the effect of packet loss (no data at vocoded audio input). Vocoder algorithm implementation within the facility can be –

- HALF_DUPLEX: Only one of Encoder or Decoder vocoder functions can be active for an audio channel at any point of time.
- FULL_DUPLEX: Both Encoder and Decoder vocoder function are simultaneously active for an audio channel.

3.4.2.3.3 Digital Audio Routing Function

Digital Audio Routing Function provides the configurability of facility to use the facility services for different deployment scenarios. DIG_AUD_RTNG_AVAIL capability indicates whether facility provides the Digital audio routing configuration. If facility provides the digital audio routing capability, different deployment scenario for digital audio can be achieved by configuring DIG_AUD_RTNG_CFG property.

a) Basic Analogue PTT = Audio function support capability is required along with DIG_AUD_RTNG_CFG property configured as EXTERNAL_CODEC. In this case digital audio data samples flows between CODEC of the audio port and radio application. Vocoder function of the facility is not available/used.

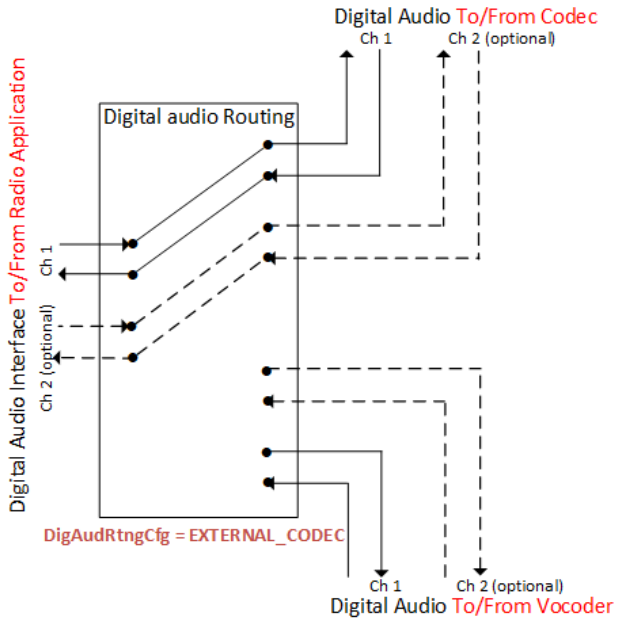

Figure 3.8 : Digital Audio Routing (DigAudRtngCfg=EXTERNAL_CODEC)

b) Vocoder Only = Vocoder function support capability is required along with DIG_AUD_RTNG_CFG property configured as EXTERNAL_VOCODER. Audio function support capability can be used only for PTT signaling and tone generation, if required. Audio conversion function (CODEC) of the audio port is not available/used. Digital audio data samples flow between the radio application and vocoder (digital audio data port).

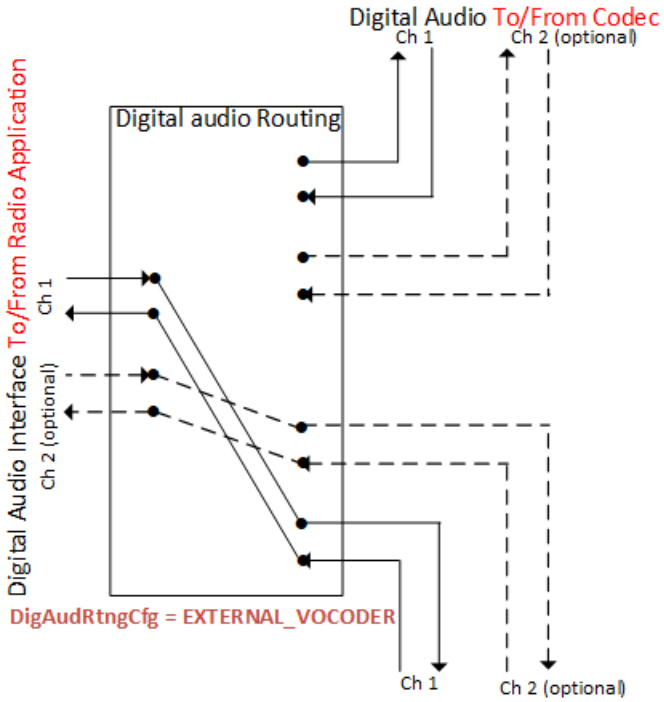

Figure 3.9 : Digital Audio Routing (DigAudRtngCfg = EXTERNAL_VOCODER)

c) Vocoded Audio PTT = Both Audio function and vocoder function support capability is required along with DIG_AUD_RTNG_CFG property configured as INTERNAL. Digital audio data samples flow between CODEC of the audio port and vocoder function internally within the facility. Digital audio data samples are not available to radio application. This configuration is also achieved when both audio function and vocoder function support capability are available with no capability for Digital audio routing configuration (DIG_AUD_RTNG_AVAIL = False). Since CODEC configuration (sample rate/sample size) is dictated by the vocoder algorithm, audio port configuration in this case is done internally by the facility and radio application does not have control on this configuration.

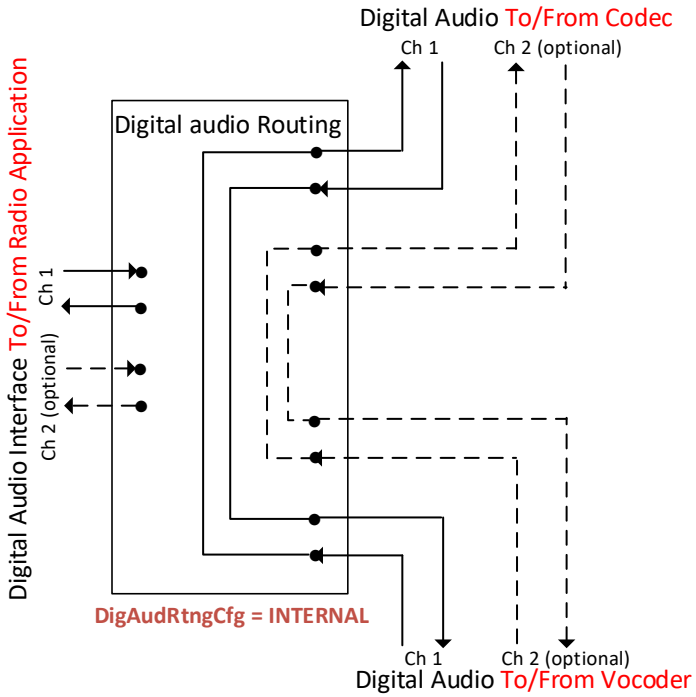

Figure 3.10 : Digital Audio Routing (DigAudRtngCfg = INTERNAL)

d) Vocoder Loopback (Transcoding) = Vocoder function support capability is required along with DIG_AUD_RTNG_CFG property configured as LOOPBACK. Digital audio data samples from output of Rx-vocoder (Decoder) is routed to Digital audio data input for Tx-Vocoder (Encoder). This configuration is useful when radio application transcodes from one vocoder algorithm to another vocoder algorithm. In this case, Tx vocoder and Rx vocoder uses different algorithm.

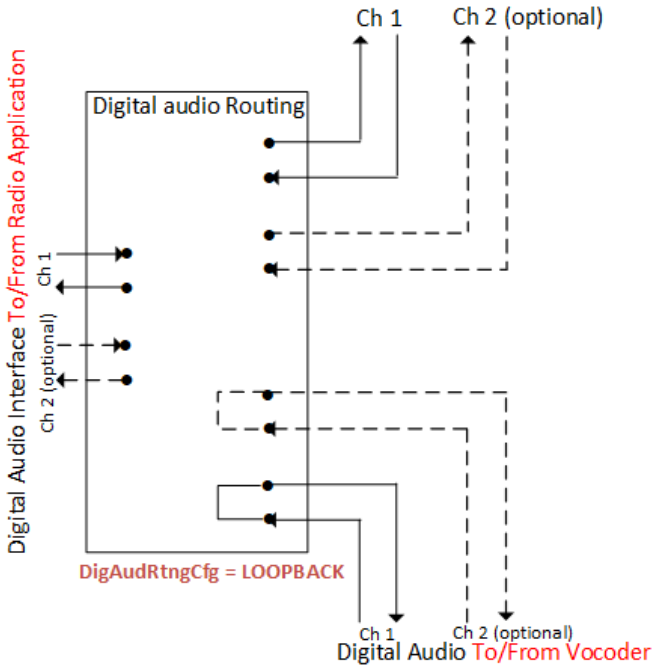

Figure 3.11 : Digital Audio Routing (DigAudRtngCfg = LOOPBACK)

3.4.2.3.4 SECOND_CHANNEL

As Hardware CODEC may support two parallel bidirectional channels, facility may support an independent additional audio channel for same radio application. Facility indicates the availability of an additional audio channel using SECOND_CHANNEL support capability. If SECOND_CHANNEL supported by facility, the additional audio channel shall be supported end-to-end, i.e., both the audio port and vocoder function shall support the second audio channel. In case of multiple audio port support, all audio ports shall support the second channel. Same digital audio routing configuration is applied to both the channels, but the two channels are routed independently to two service instances with radio application or two instances of vocoder algorithm. PTT signalling is common for the two channels.

3.4.2.3.5 MIXER/MULTIPLEXER

For multiple audio port support, additional Mixer/Multiplexer function is used within the facility. Mixer/Multiplexer function handles both the upstream digital audio data (Digital audio data from audio ports) and downstream digital audio data (digital audio data to audio ports). Facility provides the capability for the radio application to configure the mixer/multiplexer for supporting different deployment scenarios. For upstream digital audio data, radio application can configure the mixer/multiplexer function as –

- Radio application explicitly specify the audio port whose digital audio data is sent as upstream digital audio data.
- Digital audio data from audio port whose PTT Pushbutton is pressed, is sent as upstream digital audio data. Priority selection is applied if more than one audio ports have their PTT Pushbutton pressed.

- Digital audio data from all audio ports are mixed and sent as upstream digital audio data.

For downstream digital audio data, radio application can configure the mixer/multiplexer function as –

- Radio application explicitly specify the audio port to whom the downstream digital audio data is sent.
- Downstream digital audio data is sent to same audio port which is selected for the upstream digital audio data.
- Downstream digital audio data is replicated and sent to all audio ports.

In case of second channel support, mixer/multiplexer functions apply the same configuration for both channels, but two channels are handled independently as two upstream digital audio and two downstream digital audio channels.

3.4.3 Services

3.4.3.1 Provide Services

The following table lists the provide services of the API (used by a radio application and provided by a AudioVocoder facility instance):

Service Groups / Module	Services / Interfaces	Primitives
AudibleNotification (Available if AUD_SUPPORT = true)	AudibleAlertsAndAlarms (**)	createTone() startTone() stopTone() stopAllTones() destroyTone()
AudioFunctionConfig (Available If AUD_SUPPORT = true)	AudioPortDeviceConfig	setAudioPortParams() getAudioPortParams() setAudioChannelLinkEnabled() getAudioChannelLinkEnabled()

	ChannelAudioConfig (**)	setAcpProperty() getAcpProperty() setAcpEnabled() getAcpEnabled() setAcpDelayEnabled() getAcpDelayEnabled() setInputGain() getInputGain() setOutputGain() getOutputGain() setSideToneEnabled() getSideToneEnabled() setAudioOutputEnabled() getAudioOutputEnabled() setAudioInputEnabled() getAudioInputEnabled()
VocoderFunctionConfig (Available if VOCODER_SUPPORT = true)	VocoderCtrl (*)	getAlgorithmsSupported() setTxAlgorithm() setRxAlgorithm() configVocoderTxAlgo() getTxAlgorithm() configVocoderRxAlgo() getRxAlgorithm() getVocoderTxAlgoConfig() getVocoderRxAlgoConfig() getVocDigAudioConfig() getVadDtxEnabled() setVadDtxEnabled()
	VocoderRuntimeConfig (*)	setVocoderTxEnabled() setVocoderRxEnabled() getVocoderTxEnabled() getVocoderRxEnabled()
DigitalAudio (Available if DIG_AUD_RTNG_CFG = EXTRENAL_AUDIO or EXTRENAL_VOCODER)	SamplesTransmission (*)	PushRxSamplePacket()
	SampleStreamCtrl (*)	getDesiredNumSamples() setDesiredNumSamples()
VocoderAudio (Available if VOCODER_SUPPOR	VocodedFrameTransmission (*)	PushRxFramePacket()

T= true)	FrameStreamCtrl (*)	getRxVocoderFrameSize() getDesiredNumFrames() setDesiredNumFrames()
AudioPortSelection (Available if AUD_SUPPORT = true And MULTI_AUDIO_PORTS = true)	PortSelectionConfig	setUpstreamPortSelectMode() getUpstreamPortSelectMode() setDownstramPortSelectMode() getDownstreamPortSelectMode()
	PortSelection	setActivePort() getUpstreamActivePort() getDownstreamActivePort()

Table 3.5: Audio Vocoder provide services

(*) = Interface duplicated for second audio channel (if SECOND_CHANNEL = true)
(**) = Interface replicated for each audio channel (if SECOND_CHANNEL=true) for each audio port (if MULTI_AUDIO_PORTS = true)
(***) = Interface replicated for each audio port (if MULTI_AUDIO_PORTS = true)

3.4.3.2 Use Services

The following table lists the use services of the API (provided by a radio application and used by a AudVoc Facility instance):

Service Groups / Module	Services / Interfaces	Primitives
PttSignalling (Available if AUD_SUPPORT = true)	AudioPttSignal (***)	setPtt()
DigitalAudio (Available when DIG_AUD_RTNG_CFG = EXTERNAL_AUDIO or EXTERNAL_VOCODER)	SamplesReception (*)	PushTxSamplePacket()
	SampleStreamCtrl (*)	getDesiredNumSamples() setDesiredNumSamples()
VocoderAudio (Available if VOCODER_SUPPORT = true)	VocodedFrameReception (*)	PushTxFramePacket()
	FrameStreamCtrl (*)	setRxVocoderFrameSize() setDesiredNumFrames() getDesiredNumFrames()

Table 3.6: Audio Vocoder use services

(*) = Interface duplicated for second audio channel (if SECOND_CHANNEL = true)
(**) = Interface replicated for each audio channel (if SECOND_CHANNEL=true) for each

audio port (if MULTI_AUDIO_PORTS = true)
(***) = Interface replicated for each audio port (if MULTI_AUDIO_PORTS = true)

3.4.3.3 Service groups

3.4.3.3.1 AudioVocoder::PttSignalling

PttSignalling service group enables radio applications to receive PTT related signalling from an audio port.

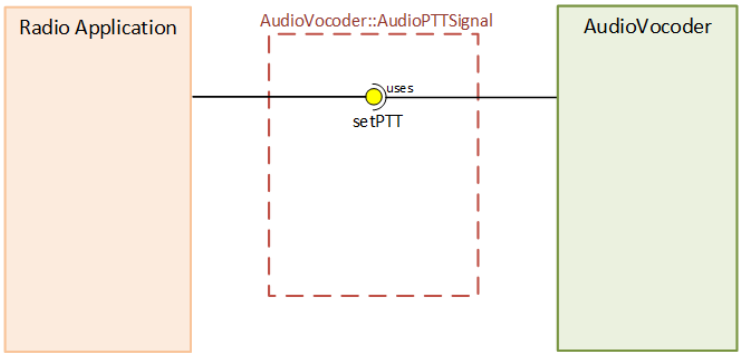

Figure 3.12 : PttSignalling Service Group

AudioPTTSignal service enables radio applications related to get indication related to PTT push button signalling for an audio port.

3.4.3.3.1.1 AudioVocoder::PttSignalling::AudioPttSignal Interface Description

AudioPttSignal interface is composed of SetPtt() operation
SetPtt() enables radio application to receive the status of the PTT push button of the audio port, whenever the push button is pressed or released

3.4.3.3.2 AudioVocoder::AudibleNotification

AudibleNotification service group enables radio applications to control the audible notification (alert tones) function of an audio port channel.

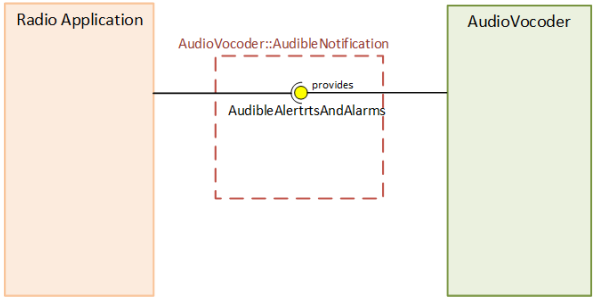

Figure 3.13 : AudibleNotification Service Group

AudibleAlertsAndAlarms service enables radio application for configuring characteristics of the generated tones as well as controlling the generation of audible notifications (alert tones) using the audio hardware of the audio port channel.

3.4.3.3.2.1 AudioVocoder::AudibleNotification::AudibleAlertsAndAlarms interface

Description

AudibleAlertsAndAlarms interface is composed of createTone(), startTone(), stopTone(), stopAllTones(), and destroyTone() operations.

- createTone() enables radio application to define the configuration of various tones which can be generated when required.
- startTone() enables radio application to start the generation of a pre-defined tone via the audio hardware.
- stopTone() enables radio application to stop the generation of the specified tone.
- stopAllTones() enables radio application to stop the generation of all tones currently being played.
- destroyTone() enables the radio application to un-define a previously defined tone. The specified tone is then no longer available for generation.

3.4.3.3.3 AudioVocoder::AudioFunctionConfig

AudioFunctionConfig service group enables radio application to configure various aspects of the audio hardware.

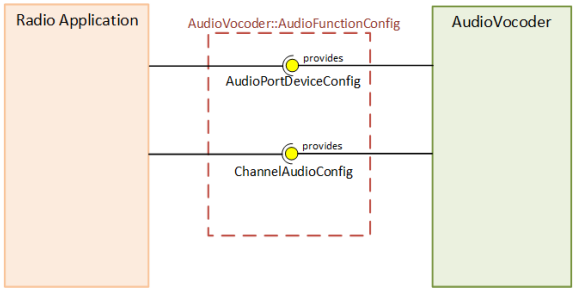

Figure 3.14 : AudioFunctionConfig Service Group

AudioPortDeviceConfig enables radio application to configure the Audio Port Device. Same configuration is applied to all audio ports if the platform supports more than one audio ports (if MULTI_AUDIO_PORTS=true).

ChannelAudioConfig enables radio applications to configure the property of an audio port channel. Both channels (if SECOND_CHANNEL=true) of an audio port can be individually configured if the channel configuration are unlinked. If the channel configuration is linked, then same configuration is applied to both channels of the audio port.

3.4.3.3.3.1 AudioVocoder::AudioFunctionConfig::AudioPortDeviceConfig interface description

AudioPortDeviceConfig interface is composed of setAudioPortParams(), getAudioPortParams(), setAudioChannelLinkEnabled() and getAudioChannelLinkEnabled() operations.

- setAudioPortParams() enables radio application to configure the audio conversion hardware device of audio ports. These configurations include sampling rate, sample size etc.
- getAudioPortParams() enables radio application to get the current setting of the

audio conversion hardware device of audio ports.

- setAudioChannelLinkEnabled() enables radio application to link/unlink configuration of both channels of an audio port. Linking channels of an audio port results in application of same configuration to both channels (if SECOND_CHANNEL = true) of an audio port.
- getAudioChannelLinkEnabled() enables radio application to get the current channel linking status of an audio port.

3.4.3.3.3.2 AudioVocoder::AudioFunctionConfig:: ChannelAudioConfig() interface description

ChannelAudioConfig interface is composed of setAcpProperty(), getAcpProperty(), setAcpEnabled(), getAcpEnabled(), setAcpDelayEnabled(), getAcpDelayEnabled(), setInputGain(), getInputGain(), setOutputGain(), getOutputGain(), setSideToneEnabled(), getSideToneEnabled(), setAudioOutputEnabled(), getAudioOutputEnabled(), setAudioInputEnabled() and getAudioInputEnabled() operations.

- setAcpProperty() enables radio application to set the Audio compression parameters of a channel of an audio port.
- getAcpProperty() enables radio application to get the current audio compression parameters of a channel of an audio port.
- setAcpEnabled() enables radio application to enable/disable the ACP algorithm.
- getAcpEnabled() enables radio application get the current enable/disable status of ACP algorithm.
- setAcpDelayEnabled() enables radio application to enable/disable the delay of ACP output.
- getAcpDelayEnabled() enables radio application to get the enable/disable status of the delay of ACP output.
- setInputGain() enables radio application to set the input gain of a channel of an audio port.
- getInputGain() enables radio application to get the current value of input gain of a channel of an audio port.
- setOutputGain() enables radio application to set the output gain of a channel of an audio port.
- getOutputGain() enables radio application to get the current value of the output gain of a channel of an audio port.
- setSideToneEnabled() enables radio application to enable/disable the sidetone of a channel of an audio port.
- getSideToneEnabled() enables radio application get the enable/disable status of the sidetone of a channel of an audio port.
- setAudioOutputEnabled() enables radio application to enable/disable the audio output of a channel of an audio port.

- getAudioOutputEnabled() enables radio application to get the enable/disable status of the audio output of a channel of an audio port.
- setAudioInputEnabled() enables radio application to enable/disable the audio input of a channel of an audio port.
- getAudioInputEnabled()enables radio application to get the enable/disable status of the audio input of a channel of an audio port.

3.4.3.3.4 AudioVocoder::VocoderFunctionConfig

VocoderFunctionConfig service group enables radio application for configuring the vocoder function of the AudioVocoder Facility.

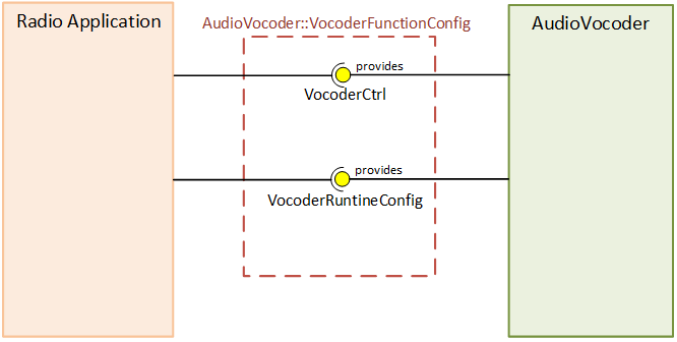

Figure 3.15 : VocoderFunctionConfig Service Group

VocoderCtrl service enables radio application to configure the parameters of vocoder algorithm being configured for a channel.
VocoderRuntimeConfig enables radio application to control vocoder functions during the run time of the device.

3.4.3.3.4.1 AudioVocoder::VocoderFunctionConfig::VocoderCtrl interface description

VocoderCtrl interface is composed of getAlgorithmsSupported(), setTxAlgorithm(), setRxAlgorithm(), configVocoderTxAlgo(), getTxAlgorithm(), configVocoderRxAlgo(), getRxAlgorithm(), getVocoderTxAlgoConfig(), getVocoderRxAlgoConfig(), getVocDigAudioConfig(), getVadDtxEnabled() and setVadDtxEnabled() operations.

- getAlgorithmsSupported() enables radio application to get the list of vocoder algorithm supported by the facility.
- setTxAlgorithm() enables radio application to set the Tx Vocoder Algorithm.
- setRxAlgorithm() enables radio application to set the Rx Vocoder Algorithm
- configVocoderTxAlgo() enables radio application to set and configure the Tx Vocoder (Encoder) algorithm.
- getTxAlgorithm() enables radio application to get current algorithm of the Tx Vocoder.
- configVocoderRxAlgo() enables radio application to set and configure the Rx Vocoder (Decoder) algorithm.
- getRxAlgorithm() enables radio application to get current algorithm of the Rx Vocoder.

- getVocoderTxAlgoConfig() enables radio application to get the current configuration of the Vocoder Tx (Encoder) algorithm.
- getVocoderRxAlgoConfig() enables radio application to get the current configuration of the Vocoder Rx (Decoder) algorithm.
- getVocDigAudioConfig() enables radio application to get the digital audio sample configuration expected by the vocoder.
- getVadDtxEnabled() enables radio application to get the enable/disable status of VAD/DTX of the Tx vocoder (if supported).
- setVadDtxEnabled() enables radio application to enable/disable VAD/DTX function of the Tx vocoder (if supported).

3.4.3.3.4.2 AudioVocoder::VocoderFunctionConfig::VocoderRuntimeConfig interface description

VocoderRuntimeConfig interface is composed of setVocoderTxEnabled(), setVocoderRxEnabled(), getVocoderTxEnabled() and getVocoderRxEnabled() operations.

- setVocoderTxEnabled() enables radio application to enable/disable Vocoder Tx (Encoder). Can be used during Half duplex mode, Push to Talk Mode.
- setVocoderRxEnabled() enables radio application to enable/disable Vocoder Rx (Decoder). Can be used during Half duplex mode, Push to Talk Mode.
- getVocoderTxEnabled() enables radio application to get the enable/disable status of Vocoder Tx (Encoder).
- getVocoderRxEnabled() enables radio application to get the enable/disable status of Vocoder Rx (Decoder).

3.4.3.3.5 AudioVocoder::DigitalAudio

DigitalAudio service group is used when source/sink of digital audio is radio application while other end can be vocoder or the audio conversion function. It enables radio application to send/receive digital audio as well as configuration related to the digital audio data transfer.

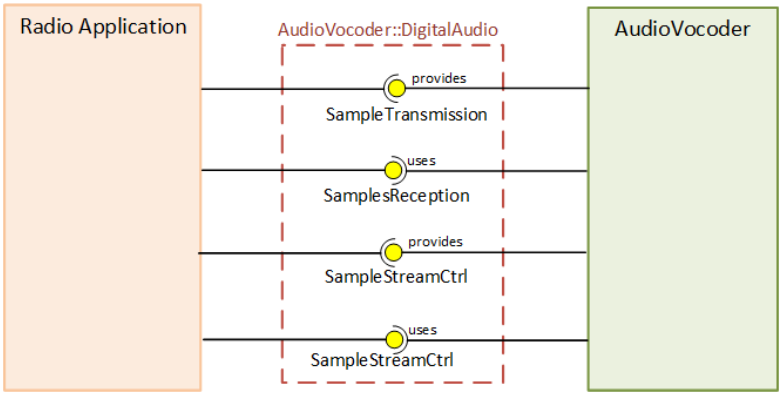

Figure 3.16 : DigitalAudio Service Group

- SamplesTransmission service enables radio application to send digital audio data

to vocoder or audio conversion function of the facility.

- SamplesReception service enables radio application to receive digital audio data from vocoder or audio conversion function of the facility.
- SampleStreamCtrl provide service enables radio application to control the size of digital audio data packets received from facility (when DIG_AUD_RTNG_CFG=EXTERNAL_CODEC). It also enables radio application to get the desired size of digital audio data packet transmitted to facility (when DIG_AUD_RTNG_CFG=EXTERNAL_VOCODER).
- SampleStreamCtrl use service enables facility to control the size of the digital audio data packets transmitted by radio application (when DIG_AUD_RTNG_CFG=EXTERNAL_VOCODER). It also enables facility to get the desired size of digital audio data packet transmitted to radio application (when DIG_AUD_RTNG_CFG=EXTERNAL_CODEC).

3.4.3.3.5.1 AudioVocoder::DigitalAudio::SamplesTransmission interface description

SamplesTransmission interface is composed of PushRxSamplePacket() operation.

- PushRxSamplePacket() enables radio application to push digital audio data to vocoder or audio conversion function of the facility.

3.4.3.3.5.2 AudioVocoder::DigitalAudio::SampleStreamCtrl provide service interface description

SampleStreamCtrl interface is composed of getDesiredNumSamples() and setDesiredNumSamples().

- getDesiredNumSamples() provide service operation enables radio application to get number of samples per packet expected by vocoder or audio conversion function while receiving digital audio data from radio application.
- setDesiredNumSamples() provide service operation enables radio application to set expected number of samples per packet while receiving digital audio data from vocoder or audio conversion function of facility.

3.4.3.3.5.3 AudioVocoder::DigitalAudio::SampleStreamCtrl use service interface description

SampleStreamCtrl interface is composed of getDesiredNumSamples() and setDesiredNumSamples().

- getDesiredNumSamples() use service operation enables radio application to provide expected number of samples per packet while receiving digital audio data from vocoder or audio conversion function of facility.
- setDesiredNumSamples() use service operation enables radio application to get number of samples per packet expected by vocoder or audio conversion function while receiving digital audio data from radio application.

3.4.3.3.5.4 AudioVocoder::DigitalAudio::SamplesReception interface description

SamplesTransmission interface is composed of PushTxSamplePacket() operation.

- PushTxSamplePacket() enables radio application to receive digital audio data from vocoder or audio conversion facility.

3.4.3.3.6 AudioVocoder::VocoderAudio

VocoderAudio service group enables radio application to send/receive vocoded audio as well as configuration related to the vocoded audio data transfer.

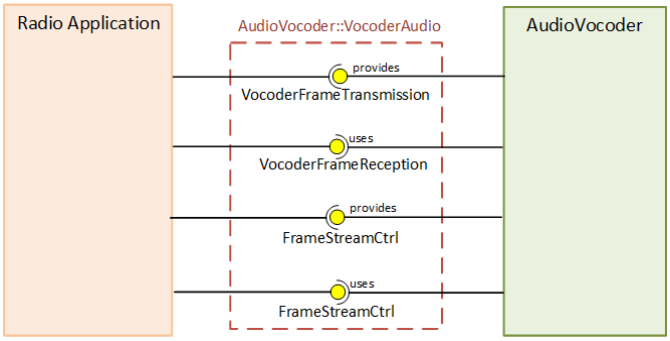

Figure 3.17 : VocoderAudio Service Group

- VocodedFrameTransmission service enables radio application to send vocoded audio data to vocoder function of the facility.
- VocodedFrameReception service enables radio application to receive vocoded audio data from vocoder function of the facility.
- FrameStreamCtrl provide service enables radio application to control the number of vocoded frames per packet received from facility. It also enables radio application to get the desired size of the vocoded data frame as well as to get desired number of vocoded data frame per packet transmitted to facility.
- FrameStreamCtrl use service enables facility to control the size of vocoded frame (for continuous stream Rx vocoder algorithm) transmitted by radio application. It also enables facility to get the desired number of vocoded frames per packet received by the radio application.

3.4.3.3.6.1 AudioVocoder::VocoderAudio::VocodedFrameTransmission interface description

VocodedFrameTransmission interface is composed of PushRxFramePacket() operation.

- PushRxFramePacket() enables radio application to send vocoded audio data packets to vocoder function of the facility.

3.4.3.3.6.2 AudioVocoder::VocoderAudio::VocodedFrameReception interface description

VocodedFrameTransmission interface is composed of PushTxFramePacket() operation.

- PushTxFramePacket() enables radio application to receive vocoded audio data packets from vocoder function of the facility.

3.4.3.3.6.3 AudioVocoder::VocoderAudio::FrameStreamCtrl provide service interface

description

FrameStreamCtrl interface is composed of getRxVocoderFrameSize(), setDesiredNumFrames() and getDesiredNumFrames() operations.

- getRxVocoderFrameSize() enables radio application to get the number of bits per encoded Frame for Rx Vocoder (Decoder).
- setDesiredNumFrames() provide service operation enables radio application to set the desired number of encoded frames in a single vocoded audio data packet expected by radio application while receiving vocoded data packet from Tx vocoder (Encoder).
- getDesiredNumFrames() provide service operation enables radio application to get the desired number of encoded frames in a single vocoded audio data packet, expected by facility while sending vocoded data packet to Rx vocoder (Decoder).

3.4.3.3.6.4 AudioVocoder::VocoderAudio::FrameStreamCtrl use service interface description

FrameStreamCtrl interface is composed of setRxVocoderFrameSize(), setDesiredNumFrames() and getDesiredNumFrames() operations.

- setRxVocoderFrameSize() enables facility to set the number of bits (multiple of 8 bits) per encoded frame for Rx Vocoder (Decoder) in case of continuous bit stream vocoder algorithm.
- setDesiredNumFrames() use service operation enables the facility to set the desired number of encoded frames in a single vocoded audio data packet expected by the Rx vocoder while receiving vocoded data packet from radio application.
- getDesiredNumFrames() use service operation enables facility to get the desired number of encoded frames in a single vocoded audio data packet, expected by radio application while receiving vocoded data packet from Tx vocoder (encoder).

3.4.3.3.7 AudioVocoder:: AudioPortSelection

AudioPortSelection service group is applicable for multiple audio port deployment scenario (MULTI_AUDIO_PORTS = true). AudioPortSelection service group enables radio application for configuring the mixer/multiplexer for audio port priority and selection.

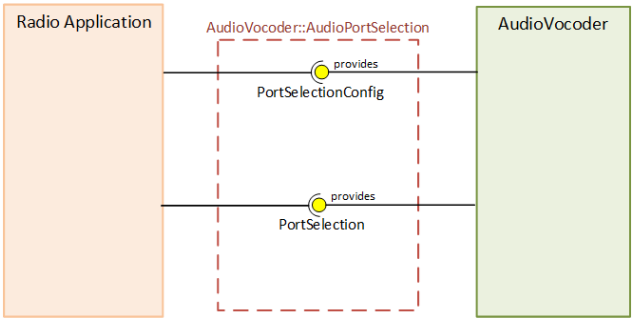

Figure 3.18 : AudioPortSelection

- PortSelectionConfig service enables radio application to configure the port

selection related configuration.

- PortSelection enables radio application to for selection of audio port for audio data input and output.

3.4.3.3.7.1 AudioVocoder:: AudioPortSelection::PortSelectionConfig interface description

PortSelectionConfig interface is composed of setUpstreamPortSelectMode(), getUpstreamPortSelectMode(), getDownstreamPortSelectMode() and setDownstreamPortSelectMode() operations.

- setUpstreamPortSelectMode() enables radio application set the audio port selection mechanism.
- getUpstreamPortSelectMode() enables radio application to get current applied audio port selection mechanism.
- setDownstreamPortSelectMode() enables radio application set the audio port selection mechanism.
- getDownstreamPortSelectMode() enables radio application to get current applied audio port selection mechanism.

3.4.3.3.7.2 AudioVocoder:: AudioPortSelection::PortSelection interface description

PortSelection interface is composed of setActivePort(), getUpstreamActivePort(), and getDownstreamActivePort() operations.

- setActivePort() enables radio application to manually set an audio port for upstream or downstream audio data routing.
- getUpstreamActivePort() enables radio application to get the current active audio port for upstream audio data.
- getDownstreamActivePort() enables radio application to get the current active audio port for downstream audio data.

3.4.4 Service Primitives

This section specifies the primitives of the Audio Vocoder facility services. Each declaration of a primitive complies with the Full PIM IDL Profile of WInnF IDL profiles for PIM of SDR Applications, specified in [WInnF-Ref 05]. The declaration of each primitive also complies with SCA 4.1 Appendix E-1 [JTNC-Ref 30] .

3.4.4.1 AudioVocoder::PttSignalling::AudioPttSignal

3.4.4.1.1 SetPtt() operation

3.4.4.1.1.1 Overview

Refer 3.1.3.1.3.3.1 of [ESSOR-Ref 03]

3.4.4.1.1.2 Signatures

Refer 3.1.3.1.3.3.1.1 of [ESSOR-Ref 03]

3.4.4.1.1.3 Parameters
Refer 3.1.3.1.3.3.1.2 of [ESSOR-Ref 03]

3.4.4.1.1.4 Return Value
Refer 3.1.3.1.3.3.1.3 of [ESSOR-Ref 03]

3.4.4.1.1.5 Exceptions
Refer 3.1.3.1.3.3.1.6 of [ESSOR-Ref 03]

3.4.4.1.1.6 Attributes
None

3.4.4.1.1.7 Behavior requirements
Facility calls this primitive whenever the PB switch status of the corresponding audio port changes.

3.4.4.2 AudioVocoder::AudibleNotification::AudibleAlertsAndAlarms

3.4.4.2.1 createTone() operation

3.4.4.2.1.1 Overview
Refer 3.1.3.1.2.3.1 of [ESSOR-Ref 03]

3.4.4.2.1.2 Signatures
Refer 3.1.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.4.4.2.1.3 Parameters
Refer 3.1.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.4.4.2.1.4 Return Value
Refer 3.1.3.1.2.3.1.3 of [ESSOR-Ref 03]

3.4.4.2.1.5 Exceptions
Refer 3.1.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.4.4.2.1.6 Attributes
- MIN_SIMPLE_TONE_FREQ
- MAX_SIMPLE_TONE_FREQ,
- MIN_MULTI_TONE_FREQ
- MAX_MULTI_TONE_FREQ
- MAX_TONE_SAMPLES_STORAGE

3.4.4.2.1.7 Behavior requirements
On createTone() facility shall check the ToneProfile elements and raise exception, if parameters are invalid/out of range. Facility shall raise exception if no space available for storing the new tone profile data. If no exception is raised, facility shall allocate a unique

(unique across the complete facility) toneId, store the complete tone profile data against the toneId and returns the toneId to radio application. This toneId is used by the radio application for future generation of tones.

3.4.4.2.2 startTone operation

3.4.4.2.2.1 Overview
Refer 3.1.3.1.2.3.2 of [ESSOR-Ref 03]

3.4.4.2.2.2 Signatures
Refer 3.1.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.4.4.2.2.3 Parameters
Refer 3.1.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.4.4.2.2.4 Return Value
Refer 3.1.3.1.2.3.2.3 of [ESSOR-Ref 03]

3.4.4.2.2.5 Exceptions
Refer 3.1.3.1.2.3.2.6 of [ESSOR-Ref 03]

3.4.4.2.2.6 Attributes
None

3.4.4.2.2.7 Behavior requirements
On startTone() operation, facility shall raise exception if the provided toneId is not previously defined. If no exception is raised, facility shall generate tone samples as per the tone profile corresponding to toneId, towards the audio conversion function of the channel of the audio port

3.4.4.2.3 stopTone() operation

3.4.4.2.3.1 Overview
Refer 3.1.3.1.2.3.3 of [ESSOR-Ref 03]

3.4.4.2.3.2 Signatures
Refer 3.1.3.1.2.3.3.1 of [ESSOR-Ref 03]

3.4.4.2.3.3 Parameters
Refer 3.1.3.1.2.3.3.2 of [ESSOR-Ref 03]

3.4.4.2.3.4 Return Value
Refer 3.1.3.1.2.3.3.3 of [ESSOR-Ref 03]

3.4.4.2.3.5 Exceptions
Refer 3.1.3.1.2.3.3.6 of [ESSOR-Ref 03]

3.4.4.2.3.6 Attributes

None

3.4.4.2.3.7 Behavior requirements

On stopTone() operation, facility shall raise exception if the provided toneId is not earlier created using createTone() operation. If no exception is raised, facility shall stop the generation of the tone corresponding to the specified toneId. No action is taken if the specified toneId is already stopped.

3.4.4.2.4 stopAllTones() operation

3.4.4.2.4.1 Overview

Refer 3.1.3.1.2.3.4 of [ESSOR-Ref 03]

3.4.4.2.4.2 Signatures

Refer 3.1.3.1.2.3.4.1 of [ESSOR-Ref 03]

3.4.4.2.4.3 Parameters

Refer 3.1.3.1.2.3.4.2 of [ESSOR-Ref 03]

3.4.4.2.4.4 Return Value

Refer 3.1.3.1.2.3.4.3 of [ESSOR-Ref 03]

3.4.4.2.4.5 Exceptions

Refer 3.1.3.1.2.3.4.6 of [ESSOR-Ref 03]

3.4.4.2.4.6 Attributes

None

3.4.4.2.4.7 Behavior requirements

On stopAllTones() operation, facility shall stop all tones currently being generated for the channel of the audio port.

3.4.4.2.5 destroyTone() operation

3.4.4.2.5.1 Overview

Refer 3.1.3.1.2.3.5 of [ESSOR-Ref 03]

3.4.4.2.5.2 Signatures

Refer 3.1.3.1.2.3.5.1 of [ESSOR-Ref 03]

3.4.4.2.5.3 Parameters

Refer 3.1.3.1.2.3.5.2 of [ESSOR-Ref 03]

3.4.4.2.5.4 Return Value

Refer 3.1.3.1.2.3.5.3 of [ESSOR-Ref 03]

3.4.4.2.5.5 Exceptions
Refer 3.1.3.1.2.3.5.6 of [ESSOR-Ref 03]

3.4.4.2.5.6 Attributes
None

3.4.4.2.5.7 Behavior requirements
On destroyTone() operation, facility shall raise exception if the specified toneId does not exist i.e. not created earlier using createTone(). If no exception is raised, facility shall stop generation of tone corresponding to toneId, if currently started, and remove the complete context associated with toneId including the tone profile. Specified toneId will no longer be available for use in future unless created again.

3.4.4.3 AudioVocoder::AudioFunctionConfig::AudioPortDeviceConfig

3.4.4.3.1 setAudioPortParams() operation

3.4.4.3.1.1 Overview
Refer 3.1.3.1.4.3.1 of [ESSOR-Ref 03]

3.4.4.3.1.2 Signatures
Refer 3.1.3.1.4.3.1.1 of [ESSOR-Ref 03]

3.4.4.3.1.3 Parameters
Refer 3.1.3.1.4.3.1.2 of [ESSOR-Ref 03]

3.4.4.3.1.4 Return Value
Refer 3.1.3.1.4.3.1.3 of [ESSOR-Ref 03]

3.4.4.3.1.5 Exceptions
Refer 3.1.3.1.4.3.1.6 of [ESSOR-Ref 03]

3.4.4.3.1.6 Attributes
- MIN_SAMPLE_RATE
- MAX_SAMPLE_RATE
- SAMPLE_SIZES
- CONNECTION_TYPES

3.4.4.3.1.7 Behavior requirements
On setAudioPortParams() operation, if no exception is raised, facility configures the CODEC of the audio port with the required sampling rate/size and connectivity as specified by the input parameters. Same parameters are applicable for both channels (if SECOND_CHANNEL = true) of the audio port

3.4.4.3.2 getAudioPortParams() operation

3.4.4.3.2.1 Overview
Refer 3.1.3.1.4.3.2 of [ESSOR-Ref 03]

3.4.4.3.2.2 Signatures
Refer 3.1.3.1.4.3.2.1 of [ESSOR-Ref 03]

3.4.4.3.2.3 Parameters
Refer 3.1.3.1.4.3.2.2 of [ESSOR-Ref 03]

3.4.4.3.2.4 Return Value
Refer 3.1.3.1.4.3.2.3 of [ESSOR-Ref 03]

3.4.4.3.2.5 Exceptions
Refer 3.1.3.1.4.3.2.6 of [ESSOR-Ref 03]

3.4.4.3.2.6 Attributes
None

3.4.4.3.2.7 Behavior requirements
On getAudioPortParams() operation, facility shall provide the currently configured parameter of the CODEC (audio conversion function) of the audio port.

3.4.4.3.3 setAudioChannelLinkEnabled() operation

3.4.4.3.3.1 Overview
setAudioChannelLinkEnabled() is used by radio application to link/unlink the configuration of both channels (if SECOND_CHANNEL=true) of an audio port. When two channels of an audio port is linked, configuration done on one channel of the audio port applies to other channel of that audio port. This enables radio application to apply the configuration to only one channel of the audio port. If the channels are unlinked, each channel of the audio port has separate independent configuration and change of configuration of one of the channels does not affect configuration of the other channel. Radio application needs to apply individual configurations to each of the channel. By default, both channels are linked together.

3.4.4.3.3.2 Signatures
void setAudioChannelLinkEnabled(in boolean enable)

3.4.4.3.3.3 Parameters

Name	Description	Type	Range/Values
enable	Value indicating whether the channel configuration needs to be	boolean	TRUE = configuration of both channels of the audio port needs to be linked together FALSE=configuration of channels is unlinked.

	linked or unlinked		DEFAULT = TRUE

Table 3.7: Audio Vocoder setAudioChannelLinkEnabled parameter

3.4.4.3.3.4 Return Value

None

3.4.4.3.3.5 Exceptions

Following exceptions are raised for this operation -

- unsupportedOperation: If the facility does not support additional audio channel(SECOND_CHANNEL=false), facility shall raise exception on this operation.

3.4.4.3.3.6 Attributes

- SECOND_CHANNEL

3.4.4.3.3.7 Behavior requirements

On setAudioChannelLinkEnabled() operation, if the input parameter is TRUE, facility shall apply the configuration of channel 1 to channel 2 of the audio port. Facility shall link the configuration of the two channels i.e., one any future configuration of one of the channel of the audio port, same configuration is applied to other channel of the port simultaneously.

If the input parameter is FALSE, facility shall unlink the two channels i.e., any future configuration applied to one of the channels does not affect the configuration of other channel. Current configuration of the channels is not changed for this case.

3.4.4.3.4 getAudioChannelLinkEnabled() operation

3.4.4.3.4.1 Overview

getAudioChannelLinkEnabled() operation is used by radio application to get the configuration linking status of the two channels of the audio port. This operation does not affect any configuration.

3.4.4.3.4.2 Signatures

void getAudioChannelLinkEnabled(out boolean linking_status)

3.4.4.3.4.3 Parameters

Name	Description	Type	Range/Values
linking_status	Value indicating whether the channel configuration is linked or	boolean	TRUE = configuration of both channels of the audio port are linked together FALSE=configuration of channels is unlinked.

	unlinked		

Table 3.8: Audio Vocoder getAudioChannelLinkEnabled parameter

3.4.4.3.4.4 Return Value

None

3.4.4.3.4.5 Exceptions

Following exceptions are raised for this operation -

- unsupportedOperation: If the facility does not support additional audio channel(SECOND_CHANNEL=false), facility shall raise exception on this operation.

3.4.4.3.4.6 Attributes

- SECOND_CHANNEL

3.4.4.3.4.7 Behavior requirements

On getAudioChannelLinkEnabled() operation, if no exception is raised, facility shall provide the current linking status of the channels of the audio port. Facility shall set the status as true if the configuration of channels is linked and FALSE otherwise.

3.4.4.4 AudioVocoder::AudioFunctionConfig::ChannelAudioConfig

3.4.4.4.1 setAcpEnabled() operation

3.4.4.4.1.1 Overview

Refer 3.1.3.1.5.3.1 of [ESSOR-Ref 03]

3.4.4.4.1.2 Signatures

Refer 3.1.3.1.5.3.1.1 of [ESSOR-Ref 03]

3.4.4.4.1.3 Parameters

Refer 3.1.3.1.5.3.1.2 of [ESSOR-Ref 03]

3.4.4.4.1.4 Return Value

Refer 3.1.3.1.5.3.1.3 of [ESSOR-Ref 03]

3.4.4.4.1.5 Exceptions

Refer 3.1.3.1.5.3.1.6 of [ESSOR-Ref 03]

3.4.4.4.1.6 Attributes

- AUDIO_SERVICES

3.4.4.4.1.7 Behavior requirements

On setAcpEnabled () operation, if no exception is raised, facility configures the CODEC of the audio port for enabling or disabling (as per input parameter) the audio compression

for the channel of the audio port. If the channels of the audio port is linked, same configuration is applied to the other channel of the audio port also.

3.4.4.4.2 setAcpDelayEnabled() operation

3.4.4.4.2.1 Overview
Refer 3.1.3.1.5.3.2 of [ESSOR-Ref 03]

3.4.4.4.2.2 Signatures
Refer 3.1.3.1.5.3.2.1 of [ESSOR-Ref 03]

3.4.4.4.2.3 Parameters
Refer 3.1.3.1.5.3.2.2 of [ESSOR-Ref 03]

3.4.4.4.2.4 Return Value
Refer 3.1.3.1.5.3.2.3 of [ESSOR-Ref 03]

3.4.4.4.2.5 Exceptions
Refer 3.1.3.1.5.3.2.6 of [ESSOR-Ref 03]

3.4.4.4.2.6 Attributes
- AUD_SUPPORT
- AUDIO_SERVICES

3.4.4.4.2.7 Behavior requirements
On setAcpDelayEnabled () operation, if no exception is raised, facility shall configure the CODEC of the audio port for enabling or disabling the delay of the Audio compression (ACP) output of the channel of the audio port. If the channels of the audio ports are linked, same configuration is applied to the other channel of the audio port also.

3.4.4.4.3 setSidetoneEnabled() operation

3.4.4.4.3.1 Overview
Refer 3.1.3.1.5.3.3 of [ESSOR-Ref 03]

3.4.4.4.3.2 Signatures
Refer 3.1.3.1.5.3.3.1 of [ESSOR-Ref 03]

3.4.4.4.3.3 Parameters
Refer 3.1.3.1.5.3.3.2 of [ESSOR-Ref 03]

3.4.4.4.3.4 Return Value
Refer 3.1.3.1.5.3.3.3 of [ESSOR-Ref 03]

3.4.4.4.3.5 Exceptions
Refer 3.1.3.1.5.3.3.6 of [ESSOR-Ref 03]

3.4.4.4.3.6 Attributes
None

3.4.4.4.3.7 Behavior requirements
On setSidetoneEnabled() operation, if no exception is raised, facility shall configure the CODEC of the audio port to enable or disable (as per input parameter) the sidetone of the channel of the audio port. If the channels of the audio ports are linked, same configuration is applied to the other channel of the audio port also.

3.4.4.4.4 setAudioOutputEnabled() operation

3.4.4.4.4.1 Overview
Refer 3.1.3.1.5.3.4 of [ESSOR-Ref 03]

3.4.4.4.4.2 Signatures
Refer 3.1.3.1.5.3.4.1 of [ESSOR-Ref 03]

3.4.4.4.4.3 Parameters
Refer 3.1.3.1.5.3.4.2 of [ESSOR-Ref 03]

3.4.4.4.4.4 Return Value
Refer 3.1.3.1.5.3.4.3 of [ESSOR-Ref 03]

3.4.4.4.4.5 Exceptions
Refer 3.1.3.1.5.3.4.6 of [ESSOR-Ref 03]

3.4.4.4.4.6 Attributes
None

3.4.4.4.4.7 Behavior requirements
On setAudioOutputEnabled() operation, if no exception is raised, if the input parameter is

- FALSE: facility shall configure the CODEC (of the audio port) to disable the audio output of the channel. Basically, it relates to disable the DAC functionality of the CODEC. Digital audio data (received from radio application/vocoder function or queued towards CODEC) is discarded in this state.
- TRUE: facility shall configure the CODEC (of the audio port) to enable the audio output. Basically, it relates to enable the DAC functionality of the CODEC. If the current state of the DAC function is already enabled, no action is taken else Rx queue is initialized and digital audio data from radio application/vocoder function is acted upon and routed to CODEC.

If the channels of the audio ports are linked, same configuration is applied to the other channel of the audio port also.

3.4.4.4.5 setOutputGain() operation

3.4.4.4.5.1 Overview
Refer 3.1.3.1.5.3.5 of [ESSOR-Ref 03]

3.4.4.4.5.2 Signatures
Refer 3.1.3.1.5.3.5.1 of [ESSOR-Ref 03]

3.4.4.4.5.3 Parameters
Refer 3.1.3.1.5.3.5.2 of [ESSOR-Ref 03]

3.4.4.4.5.4 Return Value
Refer 3.1.3.1.5.3.5.3 of [ESSOR-Ref 03]

3.4.4.4.5.5 Exceptions
Refer 3.1.3.1.4.3.1.6 of [ESSOR-Ref 03]

3.4.4.4.5.6 Attributes
- MIN_OUTPUT_GAIN
- MAX_OUTPUT_GAIN
- OUTPUT_GAIN_STEP

3.4.4.4.5.7 Behavior requirements
On setOutputGain() operation, if no exception is raised, facility shall configure the CODEC to set the DAC path gain of the channel of the audio port to the desired value. If the channels of the audio ports are linked, same configuration is applied to the other channel of the audio port also.

3.4.4.4.6 setInputGain() operation

3.4.4.4.6.1 Overview
Refer 3.1.3.1.5.3.6 of [ESSOR-Ref 03]

3.4.4.4.6.2 Signatures
Refer 3.1.3.1.5.3.6.1 of [ESSOR-Ref 03]

3.4.4.4.6.3 Parameters
Refer 3.1.3.1.5.3.6.2 of [ESSOR-Ref 03]

3.4.4.4.6.4 Return Value
Refer 3.1.3.1.5.3.6.3 of [ESSOR-Ref 03]

3.4.4.4.6.5 Exceptions
Refer 3.1.3.1.5.3.6.6 of [ESSOR-Ref 03]

3.4.4.4.6.6 Attributes
- MIN_INPUT_GAIN
- MAX_INPUT_GAIN

- INPUT_GAIN_STEP

3.4.4.4.6.7 Behavior requirements

On setInputGain() operation, if no exception is raised, facility shall configure the codec to set the ADC path gain of the channel of the audio port to desired value. If the channels of the audio ports are linked, same configuration is applied to the ADC path of the other channel of the audio port also.

3.4.4.4.7 getAcpProperties() operation

3.4.4.4.7.1 Overview

Refer 3.1.3.1.5.3.7 of [ESSOR-Ref 03]

3.4.4.4.7.2 Signatures

Refer 3.1.3.1.5.3.7.1 of [ESSOR-Ref 03]

3.4.4.4.7.3 Parameters

Refer 3.1.3.1.5.3.7.2 of [ESSOR-Ref 03]

3.4.4.4.7.4 Return Value

Refer 3.1.3.1.5.3.7.3 of [ESSOR-Ref 03]

3.4.4.4.7.5 Exceptions

Refer 3.1.3.1.5.3.7.6 of [ESSOR-Ref 03]

3.4.4.4.7.6 Attributes

None

3.4.4.4.7.7 Behavior requirements

On getAcpProperties() operation, facility shall return Audio Compression parameter being currently configured for the channel of the audio port. Facility shall return the ACP configuration parameters irrespective of whether ACP function is currently enabled for the channel of the audio port.

3.4.4.4.8 getAcpEnabled() operation

3.4.4.4.8.1 Overview

Refer 3.1.3.1.5.3.8 of [ESSOR-Ref 03]

3.4.4.4.8.2 Signatures

Refer 3.1.3.1.5.3.8.1 of [ESSOR-Ref 03]

3.4.4.4.8.3 Parameters

Refer 3.1.3.1.5.3.8.2 of [ESSOR-Ref 03]

3.4.4.4.8.4 Return Value

Refer 3.1.3.1.5.3.8.3 of [ESSOR-Ref 03]

3.4.4.4.8.5 Exceptions
Refer 3.1.3.1.5.3.8.6 of [ESSOR-Ref 03]

3.4.4.4.8.6 Attributes
None

3.4.4.4.8.7 Behavior requirements
On getAcpEnabled() operation, facility shall provide whether Audio compression function is enabled for the channel of the audio port

3.4.4.4.9 getAcpDelayEnabled() operation

3.4.4.4.9.1 Overview
Refer 3.1.3.1.5.3.9 of [ESSOR-Ref 03]

3.4.4.4.9.2 Signatures
Refer 3.1.3.1.5.3.9.1 of [ESSOR-Ref 03]

3.4.4.4.9.3 Parameters
Refer 3.1.3.1.5.3.9.2 of [ESSOR-Ref 03]

3.4.4.4.9.4 Return Value
Refer 3.1.3.1.5.3.9.3 of [ESSOR-Ref 03]

3.4.4.4.9.5 Exceptions
Refer 3.1.3.1.5.3.9.6 of [ESSOR-Ref 03]

3.4.4.4.9.6 Attributes
None

3.4.4.4.9.7 Behavior requirements
On getAcpDelayEnabled() operation, facility shall provide whether Audio compression delay is enabled for the channel of the audio port.

3.4.4.4.10 getSidetoneEnabled() operation

3.4.4.4.10.1 Overview
Refer 3.1.3.1.5.3.10 of [ESSOR-Ref 03]

3.4.4.4.10.2 Signatures
Refer 3.1.3.1.5.3.10.1 of [ESSOR-Ref 03]

3.4.4.4.10.3 Parameters
Refer 3.1.3.1.5.3.10.2 of [ESSOR-Ref 03]

3.4.4.4.10.4 Return Value
Refer 3.1.3.1.5.3.10.3 of [ESSOR-Ref 03]

3.4.4.4.10.5 Exceptions
Refer 3.1.3.1.5.3.10.6 of [ESSOR-Ref 03]

3.4.4.4.10.6 Attributes
None

3.4.4.4.10.7 Behavior requirements
On getSidetoneEnabled() operation, facility shall return whether the sidetone (CODEC level) is enabled for the channel of the audio port.

3.4.4.4.11 getAudioOutputEnabled() operation

3.4.4.4.11.1 Overview
Refer 3.1.3.1.5.3.11 of [ESSOR-Ref 03]

3.4.4.4.11.2 Signatures
Refer 3.1.3.1.5.3.11.1 of [ESSOR-Ref 03]

3.4.4.4.11.3 Parameters
Refer 3.1.3.1.5.3.11.2 of [ESSOR-Ref 03]

3.4.4.4.11.4 Return Value
Refer 3.1.3.1.5.3.11.3 of [ESSOR-Ref 03]

3.4.4.4.11.5 Exceptions
Refer 3.1.3.1.5.3.11.6 of [ESSOR-Ref 03]

3.4.4.4.11.6 Attributes
None

3.4.4.4.11.7 Behavior requirements
On getAudioOutputEnabled() operation, facility shall return whether audio output (i.e DAC path) is enabled for the channel of the audio port.

3.4.4.4.12 getInputGain() operation

3.4.4.4.12.1 Overview
Refer 3.1.3.1.5.3.12 of [ESSOR-Ref 03]

3.4.4.4.12.2 Signatures
Refer 3.1.3.1.5.3.12.1 of [ESSOR-Ref 03]

3.4.4.4.12.3 Parameters
Refer 3.1.3.1.5.3.12.2 of [ESSOR-Ref 03]

3.4.4.4.12.4 Return Value
Refer 3.1.3.1.5.3.12.3 of [ESSOR-Ref 03]

3.4.4.4.12.5 Exceptions
Refer 3.1.3.1.5.3.12.6 of [ESSOR-Ref 03]

3.4.4.4.12.6 Attributes
None

3.4.4.4.12.7 Behavior requirements
On getInputGain() operation, facility shall read the current configured audio input (ADC path) gain of the channel of the audio port.

3.4.4.4.13 getOutputGain() operation

3.4.4.4.13.1 Overview
Refer 3.1.3.15.3.13 of [ESSOR-Ref 03]

3.4.4.4.13.2 Signatures
Refer 3.1.3.1.5.3.13.1 of [ESSOR-Ref 03]

3.4.4.4.13.3 Parameters
Refer 3.1.3.1.5.3.13.2 of [ESSOR-Ref 03]

3.4.4.4.13.4 Return Value
Refer 3.1.3.1.5.3.13.3 of [ESSOR-Ref 03]

3.4.4.4.13.5 Exceptions
Refer 3.1.3.1.5.3.13.6 of [ESSOR-Ref 03]

3.4.4.4.13.6 Attributes
None

3.4.4.4.13.7 Behavior requirements
On getOutputGain() operation, facility shall read the current configured audio output (DAC path) gain of the channel of the audio port.

3.4.4.4.14 setAcpProperties() operation

3.4.4.4.14.1 Overview
Refer 3.1.3.1.5.3.14 of [ESSOR-Ref 03]

3.4.4.4.14.2 Signatures
Refer 3.1.3.1.5.3.14.1 of [ESSOR-Ref 03]

3.4.4.4.14.3 Parameters
Refer 3.1.3.1.5.3.14.2 of [ESSOR-Ref 03]

3.4.4.4.14.4 Return Value
Refer 3.1.3.1.5.3.14.3 of [ESSOR-Ref 03]

3.4.4.4.14.5 Exceptions

Refer 3.1.3.1.5.3.14.6 of [ESSOR-Ref 03]

3.4.4.4.14.6 Attributes

- AUDIO_SERVICES
- ACP_MODES
- MIN_ACP_NORM_FACTOR_VALUE
- MAX_ACP_NORM_FACTOR_VALUE
- MIN_ACP_THRESHOLD_VALUE
- MAX_ACP _THRESHOLD_VALUE
- MIN_ACP_ATTACK_TIME
- MAX_ACP_ATTACK_TIME
- MIN_ACP_RELEASE_TIME
- MAX_ACP_RELEASE_TIME
- MIN_ACP_HOLD_TIME
- MAX_ACP_HOLD_TIME

3.4.4.4.14.7 Behavior requirements

On setAcpProperties() operation, if no exception is raised, facility shall configure the audio port CODEC with the provided Audio compression parameter for the channel of the audio port. If the channels of the audio ports are linked, same configuration is applied to the other channel of the audio port also. Configuration of ACP parameters does not enable or disable the ACP function which is enabled or disabled via setAcpEnabled() operation.

3.4.4.4.15 setAudioInputEnabled() operation

3.4.4.4.15.1 Overview

setAudioInputEnabled operation is used to enable/disable the audio channel input path. Basically, it relates to enable or disable the ADC functionality of the CODEC.

3.4.4.4.15.2 Signatures

void setAudioInputEnabled(in boolean active);

3.4.4.4.15.3 Parameters

Name	Description	Type	Range/Values
active	Audio Input Enable or Disable	boolean	TRUE= Audio Input Enable FALSE = Audio Input Disable Default = FALSE

Table 3.9: Audio Vocoder setAudioInputEnabled parameter

3.4.4.4.15.4 Return Value

None

3.4.4.4.15.5 Exceptions

Following exceptions are raised for this operation -

- hardwareError: If the facility faces an error while enabling or disabling audio input, it shall raise an hardwareError exception.

3.4.4.4.15.6 Attributes

None

3.4.4.4.15.7 Behavior requirements

On setAudioInputEnabled() operation, if no exception is raised, if the input parameter is

- FALSE: facility shall configure the CODEC (of the audio port) to disable the audio output of the channel. Basically, it relates to disable the ADC functionality of the CODEC. Any digital audio data (received from CODEC or queued towards Radio application/vocoder) is discarded in this case.
- TRUE: facility shall configure the CODEC (of the audio port) to enable the audio output. Basically, it relates to enable the ADC functionality of the CODEC. If the current state of the ADC function is already enabled, no action is taken else Tx queue is initialized and digital audio data from CODEC is acted upon and routed to radio application/vocoder.

If the channels of the audio ports are linked, same configuration is applied to the other channel of the audio port also.

3.4.4.4.16 getAudioInputEnabled() operation

3.4.4.4.16.1 Overview

getAudioInputEnabled() operation is used to read the enable/disable state of the audio channel input path.

3.4.4.4.16.2 Signatures

void getAudioOutputEnabled(out boolean enable);

3.4.4.4.16.3 Parameters

Name	Description	Type	Range/Values
enable	Status whether audio input is enabled or disabled	boolean	TRUE= Audio Input is Enabled FALSE = Audio Input is Disabled

Table 3.10: Audio Vocoder getAudioOutputEnabled parameter

3.4.4.4.16.4 Return Value

None

3.4.4.4.16.5 Exceptions

None

3.4.4.4.16.6 Attributes

None

3.4.4.4.16.7 Behavior requirements

On getAudioInputEnabled() operation, facility shall return whether audio output (i.e ADC path) is enabled for the channel of the audio port.

3.4.4.5 AudioVocoder::VocoderFunctionConfig::VocoderCtrl

3.4.4.5.1 getAlgorithmsSupported() operation

3.4.4.5.1.1 Overview

Refer 3.2.3.1.4.3.1 of [ESSOR-Ref 03]

3.4.4.5.1.2 Signatures

Refer 3.2.3.1.4.3.1.1 of [ESSOR-Ref 03]

3.4.4.5.1.3 Parameters

Refer 3.2.3.1.4.3.1.2 of [ESSOR-Ref 03]

3.4.4.5.1.4 Return Value

Refer 3.2.3.1.4.3.1.3 of [ESSOR-Ref 03]

3.4.4.5.1.5 Exceptions

Refer 3.2.3.1.4.3.1.6 of [ESSOR-Ref 03]

3.4.4.5.1.6 Attributes

- VOCODER_ALGOS

3.4.4.5.1.7 Behavior requirements

On getAlogorithmsSupported() operation, facility shall return the list of all vocoder algorithm supported by the platform.

3.4.4.5.2 setTxAlgorithm() operation

3.4.4.5.2.1 Overview

Refer 3.2.3.1.4.3.6 of [ESSOR-Ref 03]

3.4.4.5.2.2 Signatures

Refer 3.2.3.1.4.3.6.1 of [ESSOR-Ref 03]

3.4.4.5.2.3 Parameters

Refer 3.2.3.1.4.3.6.2 of [ESSOR-Ref 03]

3.4.4.5.2.4 Return Value

Refer 3.2.3.1.4.3.6.3 of [ESSOR-Ref 03]

3.4.4.5.2.5 Exceptions
Refer 3.2.3.1.4.3.6.6 of [ESSOR-Ref 03]

3.4.4.5.2.6 Attributes
- VOCODER_ALGOS

3.4.4.5.2.7 Behavior requirements
On setTxAlgorithm() operation, if no exception is raised, facility shall set the Tx vocoder (Encoder) with the algorithm provided as input parameter. Facility shall reset and initialize the Rx and Tx queues of the Tx vocoder (Encoder).

3.4.4.5.3 setRxAlgorithm() operation

3.4.4.5.3.1 Overview
Refer 3.2.3.1.4.3.7 of [ESSOR-Ref 03]

3.4.4.5.3.2 Signatures
Refer 3.2.3.1.4.3.7.1 of [ESSOR-Ref 03]

3.4.4.5.3.3 Parameters
Refer 3.2.3.1.4.3.7.2 of [ESSOR-Ref 03]

3.4.4.5.3.4 Return Value
Refer 3.2.3.1.4.3.7.3 of [ESSOR-Ref 03]

3.4.4.5.3.5 Exceptions
Refer 3.2.3.1.4.3.7.6 of [ESSOR-Ref 03]

3.4.4.5.3.6 Attributes
- VOCODER_ALGOS

3.4.4.5.3.7 Behavior requirements
On setRxAlgorithm() operation, if no exception is raised, facility shall set the Rx vocoder (Encoder) with the algorithm provided as input parameter. Facility shall reset and initialize the Rx and Tx queues of the Rx vocoder (Decoder).

3.4.4.5.4 configVocoderTxAlgo() operation

3.4.4.5.4.1 Overview
configVocoderTxAlgo() operation is used by the radio application to configure the parameters for the Tx Vocoder Algorithm. Configuration parameters set are as per the Tx Vocoder Algorithm being set.

3.4.4.5.4.2 Signatures
void configVocoderTxAlgo(in AlgorithmConfigurationType config, in Algorithm

idAlgorithm);

3.4.4.5.4.3 Parameters
Refer 3.2.10.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.4.4.5.4.4 Return Value
None

3.4.4.5.4.5 Exceptions
Refer 3.2.10.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.4.4.5.4.6 Attributes
- VOCODER_ALGOS

3.4.4.5.4.7 Behavior requirements
On configVocoderTxAlgo() operation, if no exception is raised, facility shall set the Tx Vocoder Algorithm provided as input parameter and also set related configuration parameters for the algorithm being set. Facility shall initialize the Tx Vocoder context with the new algorithm and its configuration. Any previously set Tx Vocoder context is removed. Facility shall reset and initialize the Rx and Tx queues of the Tx vocoder (Encoder).

3.4.4.5.5 configVocoderRxAlgo() operation

3.4.4.5.5.1 Overview
configVocoderRxAlgo() operation is used by the radio application to configure the parameters for the Rx Vocoder Algorithm. Configuration parameters set are as per the Rx Vocoder Algorithm being set.

3.4.4.5.5.2 Signatures
void configVocoderRxAlgo(in AlgorithmConfigurationType config, in Algorithm idAlgorithm);

3.4.4.5.5.3 Parameters
Refer 3.2.10.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.4.4.5.5.4 Return Value
None

3.4.4.5.5.5 Exceptions
Refer 3.2.10.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.4.4.5.5.6 Attributes
- _ALGOS

3.4.4.5.5.7 Behavior requirements

On configVocoderRxAlgo() operation, if no exception is raised, facility shall set the Rx Vocoder Algorithm provided as input parameter and also set related configuration parameters for the algorithm being set. Facility shall initialize the Rx Vocoder context with the new algorithm and its configuration. Any previously set Rx Vocoder context is removed. Facility shall reset and initialize the Rx and Tx queues of the Rx vocoder (Decoder).

3.4.4.5.6 getTxAlgorithm() operation

3.4.4.5.6.1 Overview
Refer 3.2.3.1.4.3.4 of [ESSOR-Ref 03]

3.4.4.5.6.2 Signatures
Refer 3.2.3.1.4.3.4.1 of [ESSOR-Ref 03]

3.4.4.5.6.3 Parameters
Refer 3.2.3.1.4.3.4.2 of [ESSOR-Ref 03]

3.4.4.5.6.4 Return Value
Refer 3.2.3.1.4.3.4.3 of [ESSOR-Ref 03]

3.4.4.5.6.5 Exceptions
Refer 3.2.3.1.4.3.4.6 of [ESSOR-Ref 03]

3.4.4.5.6.6 Attributes
None

3.4.4.5.6.7 Behavior requirements
On getTxAlgorithm() operation, facility shall return the currently set algorithm id for the Tx Vocoder (Encoder). If no algorithm is currently configured for the Tx Vocoder, ALG_NONE is returned.

3.4.4.5.7 getRxAlgorithm() operation

3.4.4.5.7.1 Overview
Refer 3.2.3.1.4.3.5 of [ESSOR-Ref 03]

3.4.4.5.7.2 Signatures
Refer 3.2.3.1.4.3.5.1 of [ESSOR-Ref 03]

3.4.4.5.7.3 Parameters
Refer 3.2.3.1.4.3.5.2 of [ESSOR-Ref 03]

3.4.4.5.7.4 Return Value
Refer 3.2.3.1.4.3.5.3 of [ESSOR-Ref 03]

3.4.4.5.7.5 Exceptions

Refer 3.2.3.1.4.3.5.6 of [ESSOR-Ref 03]

3.4.4.5.7.6 Attributes

None

3.4.4.5.7.7 Behavior requirements

On getRxAlgorithm() operation, facility shall return the currently set algorithm id for the Rx Vocoder (Decoder). If no algorithm is currently configured for the Rx Vocoder, ALG_NONE is returned.

3.4.4.5.8 setVadDtxEnabled() operation

3.4.4.5.8.1 Overview

setVadDtxEnabled() operation is used to enable or disable the VAD/DTX function for the Tx Vocoder (Encoder)

3.4.4.5.8.2 Signatures

void setVadDtxEnabled(in boolean enable)

3.4.4.5.8.3 Parameters

Name	Description	Type	Range/Values
enable	Enable or disable the VAD/DTX function	boolean	TRUE= Enable VAD/DTX function FALSE = Disable VAD/DTX function Default = FALSE (disabled)

Table 3.11: Audio Vocoder setVadDtxEnabled parameter

3.4.4.5.8.4 Return Value

none

3.4.4.5.8.5 Exceptions

Following exceptions are raised for this operation-

- Unsupported: If VAD/DTX is not supported for the current TX vocoder algorithm facility shall raise Unsupported exception.
- OperationNotAvailable: If no TX vocoder is configured, or VAD/DTX parameters are not configured for the current configured TX vocoder, facility shall raise OperationNotAvaiable exception.

3.4.4.5.8.6 Attributes

- VOCODER_ALGOS

3.4.4.5.8.7 Behavior requirements

On setVadDtxEnabled() operation, if there is no algorithm is currently configured for TX Vocoder or the currently configured Tx Vocoder does not support VAD/DTX function,

exception is raised by facility. If no exception is raised, the facility shall enable or disable the VAD/DTX functionality of the TX Vocoder. VAD/DTX function detect the voice activity for the received digital audio samples, and generates Comfort Noise/Silence packet so that the radio application may choose not to transmit them. Any VAD/DTX related configuration shall be done with other configuration while configuring the TX Vocoder.

3.4.4.5.9 getVadDtxEnabled() operation

3.4.4.5.9.1 Overview

getVadDtxEnabled() operation is used to enable or disable the VAD/DTX function for the Tx Vocoder (Encoder)

3.4.4.5.9.2 Signatures

void getVadDtxEnabled(out boolean enabledStatus)

3.4.4.5.9.3 Parameters

Name	Description	Type	Range/Values
enabledStatus	Provides whether VAD/DTX is currently enabled for the vocoder	boolean	TRUE= VAD/DTX function is currently enabled FALSE = VAD/DTX function is currently disabled

Table 3.12: Audio Vocoder getVadDtxEnabled parameter

3.4.4.5.9.4 Return Value

none

3.4.4.5.9.5 Exceptions

none

3.4.4.5.9.6 Attributes

None

3.4.4.5.9.7 Behavior requirements

On getVadDtxEnabled() operation, facility shall return the current status of the VAD/DTX functionality for the TX Vocoder. If no algorithm is currently configured for the Tx Vocoder or the configured vocoder algorithm does not support VAD/DTX function, FALSE shall be returned by the facility.

3.4.4.5.10 getVocoderTxAlgoConfig()operation

3.4.4.5.10.1 Overview

getVocoderTxAlgoConfig()operation is used to get the current configuration for the Tx Vocoder (Encoder)

3.4.4.5.10.2 Signatures

void getVocoderTxAlgoConfig(out AlgorithmConfigurationType config, out Algorithm idAlgorithm)

3.4.4.5.10.3 Parameters

Name	Description	Type	Name
config	Current configuration of Tx Vocoder Algorithm	AlgorithmConfigurationType	
idAlgorithm	Current configured Tx Vocoder Algorithm	Algorithm	

Table 3.13: Audio Vocoder getVocoderTxAlgoConfig parameter

3.4.4.5.10.4 Return Value

none

3.4.4.5.10.5 Exceptions

None

3.4.4.5.10.6 Attributes

None

3.4.4.5.10.7 Behavior requirements

On getVocoderTxAlgoConfig() operation, facility shall return the current configuration of Tx Vocoder along with the Tx Vocoder algorithm. Structure of the configuration is specific to the algorithm used. If no algorithm is currently configured for the Tx Vocoder, facility shall return ALG_NONE as algorithm (configuration parameter shall be ignored by the radio application in this case)

3.4.4.5.11 getVocoderRxAlgoConfig()operation

3.4.4.5.11.1 Overview

getVocoderRxAlgoConfig()operation is used to get the current configuration for the Rx Vocoder (Decoder)

3.4.4.5.11.2 Signatures

void getVocoderRxAlgoConfig(out AlgorithmConfigurationType config, out Algorithm idAlgorithm)

3.4.4.5.11.3 Parameters

Name	Description	Type	Name
config	Current configuration of Rx Vocoder Algorithm	AlgorithmConfigurationType	
idAlgorithm	Current configured Rx Vocoder Algorithm	Algorithm	

Table 3.14: Audio Vocoder getVocoderRxAlgoConfig parameter

3.4.4.5.11.4 Return Value
none

3.4.4.5.11.5 Exceptions
None

3.4.4.5.11.6 Attributes
None

3.4.4.5.11.7 Behavior requirements
On getVocoderRxAlgoConfig() operation, facility shall return the current configuration of Rx Vocoder along with the Rx Vocoder algorithm. Structure of the configuration is specific to the algorithm used. If no algorithm is currently configured for the Rx Vocoder, facility shall return ALG_NONE as algorithm (configuration parameter shall be ignored by the radio application in this case)

3.4.4.5.12 getVocDigAudioConfig() operation

3.4.4.5.12.1 Overview
getVocDigAudioConfig() is used by the radio application to get the configuration for Digital audio samples required by the vocoder based upon algorithm being set. This is used by the radio application only when DIG_AUD_RTNG_CFG is set as EXTERNAL_VOCODER (i.e. radio application supplies/consumes digital audio samples to/from vocoder)

3.4.4.5.12.2 Signatures
getVocDigAudioConfig(out DigitalAudioConfigType config)

3.4.4.5.12.3 Parameters

Name	Description	Type	Name
config	Configuration (sample rate/sample size) of digital audio samples expected by vocoder	DigitalAudioConfigType	

Table 3.15: Audio Vocoder getVocDigAudioConfig parameter

3.4.4.5.12.4 Return Value
none

3.4.4.5.12.5 Exceptions
none

3.4.4.5.12.6 Attributes
- INIT_SAMPLE_RATE
- INIT_SAMPLE_SIZE

3.4.4.5.12.7 Behavior requirements

On getVocDigAudioConfig(), facility shall return the expected configuration for the digital audio samples to/from radio application based upon vocoder configuration. If no algorithm is currently configured for the vocoder, default/initial values for the digital audio sample configuration is returned by the facility.

3.4.4.6 AudioVocoder::VocoderFunctionConfig::VocoderRuntimeConfig

3.4.4.6.1 setVocoderTxEnabled() operation

3.4.4.6.1.1 Overview

setVocoderTxEnabled() operation is used by radio application to enable or disable the TX vocoder function

3.4.4.6.1.2 Signatures

void setVocoderTxEnabled(in boolean enable)

3.4.4.6.1.3 Parameters

Name	Description	Type	Name
enable	Enable or disable Tx Vocoder	boolean	TRUE= Enable Tx Vocoder FALSE = Disable Tx Vocoder Default = FALSE (disabled)

Table 3.16: Audio Vocoder setVocoderTxEnabled parameter

3.4.4.6.1.4 Return Value

none

3.4.4.6.1.5 Exceptions

Following exceptions are raised for this operation –

- operationNotAvailable:If no algorithm is configured for TX vocoder, facility shall raise operationNotAvailable exception
- invalidParameters: If vocoder is currently configured/support Half-duplex mode, and Rx-vocoder is enabled, while this operation the enable parameter is TRUE (request to enable Tx vocoder), facility shall raise invalidParameters exception.

3.4.4.6.1.6 Attributes

- VOCODER_ALGOS

3.4.4.6.1.7 Behavior requirements

On setVocoderTxEnabled(), if no exception is raised and if the input parameter is

- FALSE: facility shall stop the Tx-vocoder (encoder) function. Any digital audio data is discarded and not given to encoder. Any queued vocoded audio data is also

discarded, and no vocoded frames are sent to radio application. TX-vocoder configuration is retained.

- TRUE: If the current state of TX-encoder is already enabled, no action is taken and call returns. Else facility shall start the Tx-vocoder (encoder) function. Facility shall start acting upon the Digital audio data samples from CODEC/radio application and is provided to Tx-Vocoder. Tx queue towards radio application for vocoded audio data is re-initialized (discarding any old vocoded audio frames).

3.4.4.6.2 setVocoderRxEnabled() operation

3.4.4.6.2.1 Overview

setVocoderRxEnabled() operation is used by radio application to enable or disable the Rx vocoder (Decoder) function.

3.4.4.6.2.2 Signatures

void setVocoderRxEnabled(in boolean enable)

3.4.4.6.2.3 Parameters

Name	Description	Type	Name
enable	Enable or disable Rx Vocoder	boolean	TRUE= Enable Rx Vocoder (Decoder) FALSE = Disable Rx Vocoder Default = FALSE (disabled)

Table 3.17: Audio Vocoder setVocoderRxEnabled parameter

3.4.4.6.2.4 Return Value

none

3.4.4.6.2.5 Exceptions

Following exceptions are raised for this operation –

- operationNotAvailable:If no algorithm is configured for Rx vocoder, facility shall raise operationNotAvailable exception.
- invalidParameters: If vocoder is currently configured/support Half-duplex mode, and Tx-vocoder is enabled, while this operation the enable parameter is TRUE (request to enable Rx vocoder), facility shall raise invalidParameters exception.

3.4.4.6.2.6 Attributes

- VOCODER_ALGOS

3.4.4.6.2.7 Behavior requirements

On setVocoderRxEnabled(), if no exception is raised and if the input parameter is

- FALSE: facility shall stop the Rx-vocoder (decoder) function. Any vocoded audio frames from radio application is discarded and not given to decoder. Any queued

digital audio data samples are discarded, and no digital audio data samples are sent towards CODEC/radio application. RX-vocoder configuration is retained.

- TRUE: If the current state of RX-vocoder (decoder) is already enabled, no action is taken and call returns. Else facility shall start the Rx-vocoder (encoder) function. Facility shall start acting upon the vocoded audio data frames from radio application and is provided to RX-vocoder. Tx queue towards CODEC/radio application for digital audio data samples is re-initialized (discarding any old digital audio data samples).

3.4.4.6.3 getVocoderTxEnabled() operation

3.4.4.6.3.1 Overview

getVocoderTxEnabled() operation is used by radio application to get the enable or disable status of the Tx vocoder function.

3.4.4.6.3.2 Signatures

void getVocoderTxEnabled(out boolean enableStatus)

3.4.4.6.3.3 Parameters

Name	Description	Type	Name
enableStatus	Enable or disable Status of Tx Vocoder	boolean	TRUE= Tx Vocoder is enabled FALSE = Tx Vocoder is disabled

Table 3.18: Audio Vocoder getVocoderTxEnabled parameter

3.4.4.6.3.4 Return Value

none

3.4.4.6.3.5 Exceptions

Following exceptions are raised for this operation –

- operationNotAvailable:If no algorithm is configured for Tx vocoder, facility shall raise operationNotAvailable exception.

3.4.4.6.3.6 Attributes

None

3.4.4.6.3.7 Behavior requirements

On getVocoderTxEnabled() , facility shall provide the current state of Tx-vocoder (configured and enabled) or disabled. This operation does not change any state within the facility.

3.4.4.6.4 getVocoderRxEnabled() operation

3.4.4.6.4.1 Overview

getVocoderRxEnabled() operation is used by radio application to get the enable or disable

status of the Rx vocoder function

3.4.4.6.4.2 Signatures

void getVocoderRxEnabled(out boolean enableStatus)

3.4.4.6.4.3 Parameters

Name	Description	Type	Name
enableStatus	Enable or disable Status of Rx Vocoder	boolean	TRUE= Rx Vocoder is enabled FALSE = Rx Vocoder is disabled

Table 3.19: Audio Vocoder getVocoderRxEnabled parameter

3.4.4.6.4.4 Return Value

none

3.4.4.6.4.5 Exceptions

Following exceptions are raised for this operation –

- operationNotAvailable:If no algorithm is configured for Rx vocoder, facility shall raise operationNotAvailable exception.

3.4.4.6.4.6 Attributes

None

3.4.4.6.4.7 Behavior requirements

On getVocoderRxEnabled() , facility shall provide the current state of Rx-vocoder (configured and enabled) or disabled. This operation does not change any state within the facility.

3.4.4.7 AudioVocoder::DigitalAudio::SampleStreamCtrl

3.4.4.7.1 setDesiredNumSamples() operation

3.4.4.7.1.1 Overview

setDesiredNumSamples() operation is used by radio application or the facility to set the desired number of samples per sample stream packet expected by the caller while receiving digital audio data. Caller (facility or radio application) uses this operation, when it wants to change the desired number of samples per packet for the received digital audio data (samples) stream.

When DIG_AUD_RTNG_CFG = EXTERNAL_CODEC, this API is called only by radio application to set the desired number of audio samples per packet while receiving audio samples packet from facility (audio conversion - CODEC function) (i.e., number of samples per packet received from facility is dictated by radio application.)

When DIG_AUD_RTNG_CFG = EXTERNAL_VOCODER, this API is called only by facility (vocoder function) to set the desired number of audio samples per packet while receiving audio samples packet from radio application. (i.e., number of samples per packet received from radio application is dictated by facility.)

3.4.4.7.1.2 Signatures

void setDesiredNumSamples(in unsigned short numSamples)

3.4.4.7.1.3 Parameters

Name	Description	Type	Range/Values
numSamples	Desired number of samples per Sample Stream Packet	unsigned short	Value shall be within MIN_NUM_AUD_SAMPLE_PKT and MAX_NUM_AUD_SAMPLE_PKT attributes value supported by facility. Default value = INIT_NUM_AUD_SAMPLES_PKT

Table 3.20: Audio Vocoder setDesiredNumSamples parameter

3.4.4.7.1.4 Return Value

none

3.4.4.7.1.5 Exceptions

Following exceptions are raised for this operation

- configError: If the parameters are out of range, facility shall raise configError exception.
- operationNotAvailable:If DIG_AUD_RTNG_AVAIL is false, or DIG_AUD_RTNG_CFG is not EXTERNAL_CODEC, facility shall raise operationNotAvailable exception.

3.4.4.7.1.6 Attributes

- DIG_AUD_RTNG_AVAIL
- DIG_AUD_RTNG_CFG
- MIN_NUM_AUD_SAMPLE_PKT
- MAX_NUM_AUD_SAMPLE_PKT
- INIT_NUM_AUD_SAMPLES_PKT

3.4.4.7.1.7 Behavior requirements

When called by radio application, if no exception is raised, facility shall set the value of applicableTxSampleStreamPacketSize with the provided input value. Further sample stream packet sent by the facility (audio conversion function) to radio application will use the updated packet size. Before this operation is called, applicableTxSampleStreamPacketSize is initialized with

INIT_NUM_AUD_SAMPLES_PKT.

When DIG_AUD_RTNG_CFG = EXTERNAL_VOCODER, facility shall call this operation to set the value of applicableRxSampleStreamPacketSize at radio application. Further sample stream packet sent by radio application to facility (vocoder function) will use the updated packet size. Before this operation is called, applicableRxSampleStreamPacketSize is initialized with INIT_NUM_AUD_SAMPLES_PKT.

3.4.4.7.2 getDesiredNumSamples() operation

3.4.4.7.2.1 Overview

getDesiredNumSamples() operation is used by radio application or the facility to get the expected number of samples per sample stream packet for the packets being sent by the caller.Caller (facility or radio application) calls this API before start of pushing audio sample stream packets, so that it can send the packets with the desired packet size.

When DIG_AUD_RTNG_CFG = EXTERNAL_CODEC, this API is called only by facility (audio conversion function) to get the number of audio samples per packet expected by the radio application while receiving audio samples packet from facility (audio conversion - CODEC function) (i.e., number of samples per packet received from facility is dictated by radio application.)

When DIG_AUD_RTNG_CFG = EXTERNAL_VOCODER, this API is called only by radio application calls to get the number of audio samples per packet expected by facility (vocoder function) while receiving audio samples packet from radio application. (i.e., number of samples per packet received from radio application is dictated by facility.)

3.4.4.7.2.2 Signatures

void getDesiredNumSamples(out unsigned short numSamples)

3.4.4.7.2.3 Parameters

Name	Description	Type	Range/Values
numSamples	Desired number of samples per Sample Stream Packet	unsigned short	Value shall be within MIN_NUM_AUD_SAMPLE_PKT and MAX_NUM_AUD_SAMPLE_PKT attributes value supported by facility

Table 3.21: Audio Vocoder getDesiredNumSamples parameter

3.4.4.7.2.4 Return Value

none

3.4.4.7.2.5 Exceptions

Following exceptions are raised for this operation.

- operationNotAvailable:If DIG_AUD_RTNG_AVAIL is false, or DIG_AUD_RTNG_CFG is not EXTERNAL_VOCODER, facility shall raise operationNotAvailable exception.

3.4.4.7.2.6 Attributes

- DIG_AUD_RTNG_AVAIL
- DIG_AUD_RTNG_CFG

3.4.4.7.2.7 Behavior requirements

When called by radio application, if no exception is raised, facility shall provide the value of applicableRxSampleStreamPacketSize. Further sample stream packet sent by radio application to facility (vocoder function) will use the updated packet size.

When DIG_AUD_RTNG_CFG = EXTERNAL_CODEC, facility shall call this operation to get the value of applicableTxSampleStreamPacketSize from radio application. Further sample stream packets sent by facility (vocoder function) to radio application will use the updated packet size.

3.4.4.8 AudioVocoder::DigitalAudio::SamplesTransmission

3.4.4.8.1 pushRxSamplePacket() operation

3.4.4.8.1.1 Overview

pushRxSamplePacket() operation is used by radio application to send the digital audio samples to the facility (vocoder function when DIG_AUD_RTNG_CFG= EXTERNAL_VOCODER or the audio-conversion function when DIG_AUD_RTNG_CFG= EXTERNAL_CODEC).

3.4.4.8.1.2 Signatures

void pushRxSamplePacket(in StreamControlType control, in audioSamplesPacket samples)

3.4.4.8.1.3 Parameters

Name	Description	Type	Range/Values
control	Meta Data information for Audio Samples stream	streamControlType	
samples	Digital Audio Samples payload	audioSamplesPacket	

Table 3.22: Audio Vocoder pushRxSamplePacket parameters

3.4.4.8.1.4 Return Value

none

3.4.4.8.1.5 Exceptions

Following exceptions are raised for this operation-

- maxLengthExceeded: If the length of the audioSamplesPacket exceeds the maximum allowed length for this interface, facility shall raise maxLengthExceeded exception.
- noSpaceAvailable: If the packet cannot be queued for codec/Tx-vocoder due to no space available, facility shall raise noSpaceAvailable exception.
- operationNotAvailable: If the Tx-Vocoder (in case of DIG_AUD_RTNG_CFG= EXTERNAL_VOCODER) is not configured, facility shall raise operationNotAvailable exception.
- invalidParameters: If the parameters are not valid, facility shall raise invalidParameters exception. This exception is also raised if the length of the audioSamplesPacket is not a valid length.

3.4.4.8.1.6 Attributes

None

3.4.4.8.1.7 Behavior requirements

On pushRxSamplePacket(), if no exception is raised, facility shall store the received packet samples in the Rx queue to be sent to the CODEC or vocoder. Facility shall check the streamId and sequenceNum in the packet before storing the samples in the queue. If it detects a change in the streamId (i.e., new stream has begun), facility shall clear the current RX queue and then put the received samples in the queue. If facility detect, gap in sequence number (based on the sequenceNum in the last packet and number of samples sent in the last packet for the same stream id) indicating lost audio samples, facility shall add dummy audio samples (equal to the lost audio samples) in the RX queue and then then put the received samples in the queue. If endoOfFrame is indicated in the packet (indicating last packet for the stream), facility shall process the current samples in the packet, but any new packet with same streamId (except roll over of streamId) in future are discarded. If purge flag is indicated (along with endOfFrame flag) in the packet, shall clear the current RX queue, and marks the end of the current stream (indicated by streamId in the packet).

For DIG_AUD_RTNG_CFG= EXTERNAL_VOCODER, number of audio samples in each packet is dictated by the facility (vocoder function) using getDesiredNumSamples() and setDesiredNumSamples() operations.

For DIG_AUD_RTNG_CFG= EXTERNAL_CODEC, number of audio samples in each packet is dictated by radio application. Radio application can choose to pack any number of audio samples in the packet subject to MIN_NUM_AUD_SAMPLE_PKT and MAX_NUM_AUD_SAMPLE_PKT.

3.4.4.9 AudioVocoder::DigitalAudio::SamplesReception

3.4.4.9.1 PushTxSamplePacket() operation

3.4.4.9.1.1 Overview

PushTxSamplePacket() operation is used by facility (vocoder function when DIG_AUD_RTNG_CFG= EXTERNAL_VOCODER or the audio-conversion function when DIG_AUD_RTNG_CFG= EXTERNAL_CODEC) to send digital audio samples to the radio application.

3.4.4.9.1.2 Signatures

void pushTxSamplePacket(in StreamControlType control, in audioSamplesPacket samples)

3.4.4.9.1.3 Parameters

Name	Description	Type	Range/Values
control	Meta Data information for Audio Samples stream	streamControlType	
samples	Digital Audio Samples payload	audioSamplesPacket	

Table 3.23: Audio Vocoder pushTxSamplePacket parameters

3.4.4.9.1.4 Return Value

none

3.4.4.9.1.5 Exceptions

None

3.4.4.9.1.6 Attributes

None

3.4.4.9.1.7 Behavior requirements

Facility shall perform PushTxSamplePacket() operation for sending digital audio samples to radio application. Facility shall accumulate number of audio samples, pack them into a packet and send it to radio application. Facility shall maintain the sequence number of audio samples for the stream. Since digital audio stream is streaming data, audio samples are expected to be generated with fixed rate equal to the sampling rate set. Facility shall increment the Sequence number for the audio samples, based upon the expected time for the audio samples, even if audio samples are not available, so as to maintain synchronization of the stream. Facility shall start a new stream when it send the first packet after a gap in the stream (not due to unavailability of samples), e.g. in case of sending first packet after enabling of ADC path. Stream Id for new stream is generated by incrementing the older stream id by 1.

For DIG_AUD_RTNG_CFG= EXTERNAL_VOCODER, number of audio samples in each packet is dictated by the facility (vocoder function). Facility decides this number based

upon vocoder being configured subject to MIN_NUM_AUD_SAMPLE_PKT and MAX_NUM_AUD_SAMPLE_PKT.

For DIG_AUD_RTNG_CFG= EXTERNAL_CODEC, number of audio samples in each packet is dictated by radio application using getDesiredNumSamples() and setDesiredNumSamples() operations.

3.4.4.10 AudioVocoder:: VocoderAudio::FrameStreamCtrl

3.4.4.10.1 setDesiredNumFrames() operation

3.4.4.10.1.1 Overview

setDesiredNumFrames() operation is used by radio application or the facility to set the desired number of vocoder frame per vocoded audio stream packet expected by the caller while receiving vocoded audio data. Caller (facility or radio application) uses this operation, when it wants to change the desired number of frames per packet for the received vocoded audio data stream. This is used only for the frame-based vocoder algorithm. For continuous-stream vocoder algorithm, number of vocoder frames per vocoded audio stream packet is always 1.

3.4.4.10.1.2 Signatures

void setDesiredNumFrames(in unsigned short numFrames)

3.4.4.10.1.3 Parameters

Name	Description	Type	Range/Values
numFrames	Desired number for vocoder frames per vocoded audio stream packet	unsigned short	Value shall be within 1 and MAX_NUM_VOC_FRAME_PKT attributes value supported by facility. Default value is 1

Table 3.24: Audio Vocoder setDesiredNumFrames parameter

3.4.4.10.1.4 Return Value

none

3.4.4.10.1.5 Exceptions

Following exceptions are raised for this operation –

- configError: If the parameters are out of range, facility shall raise configError exception.
- operationNotAvailable: If Tx vocoder algorithm is not configured or algorithm is continuous stream vocoder type, facility shall raise operationNotAvailable exception.

3.4.4.10.1.6 Attributes

- MAX_NUM_VOC_FRAME_PKT

3.4.4.10.1.7 Behavior requirements

When setDesiredNumFrames() operation is called by radio application, if no exception is raised, facility shall set the value of applicableTxVocodedStreamPacketSize with the provided input value. Further vocoded audio stream packet sent by the facility (vocoder function) to radio application will use the updated packet size. Before this operation is called by the radio application, initial value for applicableTxVocodedStreamPacketSize, after a new Tx vocoder algorithm is configured at the facility is 1.

Facility (vocoder function) calls this operation to set the value applicableRxVocodedStreamPacketSize at radio application when a frame-based Rx vocoder algorithm is configured at the facility. Further vocoded audio stream packet sent by radio application to facility (vocoder function) will use the updated packet size. Before this operation is called by the radio application, initial value for applicableRxVocodedStreamPacketSize, after a new Rx vocoder algorithm is configured at the facility is 1.

3.4.4.10.2 getDesiredNumFrames() operation

3.4.4.10.2.1 Overview

getDesiredNumFrames() operation is used by radio application or the facility to get the expected number of vocoded frames per vocoded audio stream packet for the packets being sent by the caller. Caller (facility or radio application) calls this API before start of pushing vocoded audio stream packets, so that it can send the packets with the desired packet size. This is used only for the frame-based vocoder algorithm. For continuous-stream vocoder algorithm, number of vocoder frames per vocoded audio stream packet is always 1.

3.4.4.10.2.2 Signatures

void getDesiredNumFrames(out unsigned short numFrames)

3.4.4.10.2.3 Parameters

Name	Description	Type	Range/Values
numFrames	Desired number of vocoder frames per Vocoded audio Stream Packet	unsigned short	Value shall be within 1 and MAX_NUM_VOC_FRAME_PKT attributes value supported by facility.

Table 3.25: Audio Vocoder getDesiredNumFrames parameter

3.4.4.10.2.4 Return Value

none

3.4.4.10.2.5 Exceptions

Following exceptions are raised for this operation-

- operationNotAvailable: If Rx vocoder algorithm is not configured or algorithm is

continuous stream vocoder type, facility shall raise operationNotAvailable exception.

3.4.4.10.2.6 Attributes
None

3.4.4.10.2.7 Behavior requirements
When called by radio application, facility shall provide the value of applicableRxVocodedStreamPacketSize.

When a new Frame-based Tx Vocoder algorithm is configured, facility shall call this operation to get the value of applicableTxVocodedStreamPacketSize from radio application. Further vocoded audio stream packets sent by facility (vocoder function) to radio application will use the updated packet size.

3.4.4.10.3 setRxVocoderFrameSize() operation

3.4.4.10.3.1 Overview
setRxVocoderFrameSize() operation is called by facility to set the value of expected number of bits per vocoder frame for the vocoded audio packet received from radio application. Facility uses this operation only when a continuous stream Rx vocoder algorithm is configured.

3.4.4.10.3.2 Signatures
void setRxVocoderFrameSize (in unsigned short frameSize);

3.4.4.10.3.3 Parameters

Name	Description	Type	Range/Values
frameSize	Number of bits per vocoder frame	unsigned short	Value shall be less than MAX_VOC_FRAME_SIZE. For continuous stream vocoder algorithm, value shall be in multiple of 8.

Table 3.26: Audio Vocoder setRxVocoderFrameSize parameter

3.4.4.10.3.4 Return Value
none

3.4.4.10.3.5 Exceptions
None

3.4.4.10.3.6 Attributes
None

3.4.4.10.3.7 Behavior requirements
When a continuous stream Rx Vocoder algorithm is configured, facility uses this operation to set the value of applicableRxVocoderFrameSize at the radio application. Future

packets sent by radio application shall contains vocoder frame of size set by this operation.

Facility can call this operation, any time during its operation irrespective of whether RX vocoder is currently active or not. Facility shall not call this operation for any frame-based Rx vocoder algorithm.

3.4.4.10.4 getRxVocoderFrameSize() operation

3.4.4.10.4.1 Overview

getRxVocoderFrameSize() operation is used by radio application to get the number of bits per vocoder frame for the Rx vocoder (decoder). For frame-based Rx vocoder algorithm, number of bits per vocoder frame is dictated by the configured Rx vocoder algorithm and bit rate. Number of bits may or may not be in multiple of 8. For continuous stream Rx vocoder algorithm, number of bits per vocoder frame is dictated by facility implementation and shall always be in multiple of 8.

3.4.4.10.4.2 Signatures

void getRxVocoderFrameSize(out unsigned short frameSize);

3.4.4.10.4.3 Parameters

Name	Description	Type	Range/Values
frameSize	Number of bits per vocoder frame	unsigned short	Value shall be less than MAX_VOC_FRAME_SIZE. For continuous stream vocoder algorithm, value shall be in multiple of 8.

Table 3.27: Audio Vocoder getRxVocoderFrameSize parameter

3.4.4.10.4.4 Return Value

none

3.4.4.10.4.5 Exceptions

Following exceptions are raised for this operation-

- operationNotAvailable: If Rx vocoder algorithm is not configured or algorithm is continuous stream vocoder type, facility shall raise operationNotAvailable exception.

3.4.4.10.4.6 Attributes

None

3.4.4.10.4.7 Behavior requirements

On getRxVocoderFrameSize() operation, if no exception is raised, facility shall provide the value of applicableRxVocoderFrameSize (which is set when a new Rx vocoder algorithm is configured). Facility shall provide valid value when the Rx vocoder algorithm is configured irrespective of whether Rx vocoder (Decoder) is currently active or not.

3.4.4.11 AudioVocoder::VocoderAudio::VocodedFrameTransmission

3.4.4.11.1 pushRxFramePacket() operation

3.4.4.11.1.1 Overview

pushRxFramePacket() operation is used by radio application to send the vocoded audio frames to the facility (vocoder function).

3.4.4.11.1.2 Signatures

void pushRxFramePacket(in StreamControlType control, in vocFramePacket frames)

3.4.4.11.1.3 Parameters

Name	Description	Type	Range/Values
control	Meta Data information for vocoded audio stream packet	streamControlType	
frames	Vocoded Audio frames payload	vocFramePacket	

Table 3.28: Audio Vocoder pushRxFramePacket parameters

3.4.4.11.1.4 Return Value

none

3.4.4.11.1.5 Exceptions

Following exceptions are raised for this operation-

- maxLengthExceeded: If the length of the vocFramePacket exceeds the maximum allowed length for this interface, facility shall raise maxLengthExceeded exception.
- noSpaceAvailable: If the packet cannot be queued for Rx-vocoder due to no space available, facility shall raise noSpaceAvailable exception.
- operationNotAvailable: If the Rx-Vocoder is not configured, facility shall raise operationNotAvailable exception.
- invalidParameters: If the parameters are not valid, facility shall raise invalidParameters exception. This exception is also raised if the length of the vocFramePacket is not a valid length.

3.4.4.11.1.6 Attributes

None

3.4.4.11.1.7 Behavior requirements

On pushRxFramePacket(), if no exception is raised, facility shall store the received packet frames in the Rx queue to be sent to the Rx vocoder. Facility shall check the streamId and sequenceNum in the packet before storing the samples in the queue. If it detects a change in the streamId (i.e., new stream has begun), facility shall clear the current RX queue and then put the received vocoder frames in the queue. If facility detect, gap in sequence number (based on the sequenceNum in the last packet and number of samples

sent in the last packet for the same stream id) indicating lost vocoder frames, facility shall add dummy vocoder frames (with LOST_FRAME indication set) in the RX queue and then then put the received vocoder frames in the queue. If endOfFrame is indicated in the packet (indicating last packet for the stream), facility shall process the current vocoder frames in the packet, but any new packet with same streamId (except roll over of streamId) in future are discarded. If purge flag is indicated (along with endOfFrame flag) in the packet, shall clear the current RX queue, and marks the end of the current stream (indicated by streamId in the packet).

For continuous stream Rx vocoder algorithm, number of vocoder frames in each packet shall be 1, whereas for frame-based Rx vocoder algorithm, number of vocoder frames in each packet is dictated by the facility (vocoder function) using getDesiredNumFrames() and setDesiredNumFrames() operations.

3.4.4.12 AudioVocoder::VocoderAudio::VocodedFrameReception

3.4.4.12.1 PushTxFramePacket() operation

3.4.4.12.1.1 Overview

PushTxFramePacket() operation is used by facility (vocoder function) to send vocoded audio frames to the radio application.

3.4.4.12.1.2 Signatures

void pushTxFramePacket(in StreamControlType control, in audioSamplesPacket samples)

3.4.4.12.1.3 Parameters

Name	Description	Type	Range/Values
Control	Meta Data information for vocoded audio stream packet	streamControlType	
Samples	Vocoded audio frames payload	audioSamplesPacket	

Table 3.29: Audio Vocoder pushTxFramePacket parameters

3.4.4.12.1.4 Return Value

none

3.4.4.12.1.5 Exceptions

None

3.4.4.12.1.6 Attributes

None

3.4.4.12.1.7 Behavior requirements

Facility shall perform PushTxFramePacket() operation for sending vocoded audio frames to radio application. Facility shall accumulate number of vocoder frames, pack them into a packet and send it to radio application. Facility shall maintain the sequence number of vocoder frames for the stream. Since vocoded audio stream is streaming data, vocoded audio frames are expected to be generated with fixed rate based upon configured vocoder algorithm and bit rate. Facility shall increment the Sequence number for the vocoder frames, based upon the expected time for the vocoder frames, even if vocoder frames are not available, to maintain synchronization of the stream. Facility shall start a new stream when it sends the first packet after a gap in the stream (not due to unavailability of frames), e.g. in case of sending first packet after enabling of Tx vocoder (Encoder). Stream Id for new stream is generated by incrementing the older stream id by 1.

For continuous stream Tx vocoder algorithm, number of vocoder frames in each packet shall be 1, whereas for frame-based Tx vocoder algorithm, number of vocoder frames in each packet is dictated by radio application using getDesiredNumFrames() and setDesiredNumFrames() operations.

3.4.4.13 AudioVocoder::AudioPortSelection::PortSelectionConfig

3.4.4.13.1 setUpstreamPortSelectMode() operation

3.4.4.13.1.1 Overview

setUpstreamPortSelectMode() operation is used by the radio application to set the mechanism used to select the audio port whose digital audio data samples is sent upstream towards radio application/vocoder function.

3.4.4.13.1.2 Signatures

Void setUpstreamPortSelectMode(in UpstreamPortSelectType mode)

3.4.4.13.1.3 Parameters

Name	Description	Type	Range/Values
mode	Upstream Digital Audio Selection mode to be used by mixer/multiplexer	UpstreamPortSelectType	EXPLICIT_SELECT, PB_PRIORITY, MIXED.

Table 3.30: Audio Vocoder setUpstreamPortSelectMode parameter

3.4.4.13.1.4 Return Value

none

3.4.4.13.1.5 Exceptions

Following exceptions are raised for this operation-

- unsupportedOperation: If MUTLTI_AUDIO_PORTS is false (i.e. current deployment does not support multiple audio ports, facility shall raise unsupportedOperation

exception.

- Unsupported: If the mode provided by radio application is not supported, facility shall raise Unsupported exception.

3.4.4.13.1.6 Attributes

- MUTLTI_AUDIO_PORTS
- NUM_AUDIO_PORTS
- PORT_UPSTREAM_SELECT_MODES

3.4.4.13.1.7 Behavior requirements

On setUpstreamPortSelectMode() call, if no exception is raised, facility shall

- Take no action if current value of applicableUpstreamPortSelectMode (currently active upstream audio selection mode) is same as mode input parameter of the call. Call returns else shall update the value of applicableUpstreamPortSelectMode with the new mode value.
- If the mode value passed in the call is EXPLICIT_SELECT, value of upstreamActiveAudioPort (audio port whose audio data is currently selected for upstream audio data) remains unchanged (i.e. current active port remains active.) and call returns.
- If the mode value passed in the call is PB_PRIORITY, facility shall evaluate PB switch status along with port priority to determine upstreamActiveAudioPort value.
- If the mode value passed is MIXED, update upstreamActiveAudioPort value to 0.
- If there is change in the upstreamActiveAudioPort value, facility shall clear any accumulated audio data samples received from audio ports. Facility shall set up the mechanism for accumulating and routing digital audio data samples from the selected upstreamActiveAudioPort towards radio application/vocoder function and call returns.
- If there is no change in the upstreamActiveAudioPort value, no further action is taken and call returns.

3.4.4.13.2 getUpstreamPortSelectMode() operation

3.4.4.13.2.1 Overview

getUpstreamPortSelectMode() operation is used by radio application to get the currently active mode used for selection of audio port for upstream digital audio data towards radio application/vocoder function.

3.4.4.13.2.2 Signatures

Void getUpstreamPortSelectMode(out UpstreamPortSelectType mode);

3.4.4.13.2.3 Parameters

Name	Description	Type	Range/Values
mode	Upstream Digital Audio Selection mode currently in	UpstreamPortSelectType	EXPLICIT_SELECT, PB_PRIORITY,

	use by mixer/multiplexer		MIXED

Table 3.31: Audio Vocoder getUpstreamPortSelectMode parameters

3.4.4.13.2.4 Return Value

None

3.4.4.13.2.5 Exceptions

Following exceptions are raised for this operation-

- unsupportedOperation: If MUTLTI_AUDIO_PORTS is false (i.e., current deployment does not support multiple audio ports, facility shall raise unsupportedOperation exception.

3.4.4.13.2.6 Attributes

- MUTLTI_AUDIO_PORTS
- NUM_AUDIO_PORTS

3.4.4.13.2.7 Behavior requirements

On getUpstreamPortSelectMode() call facility shall return current value of applicableUpstreamPortSelectMode (currently active upstream audio selection mode) to radio application.

3.4.4.13.3 setDownstreamPortSelectMode() operation

3.4.4.13.3.1 Overview

setDownstreamPortSelectMode() operation is used by the radio application to set the mechanism used to select the audio port for routing the downstream digital audio data samples from radio application/vocoder function.

3.4.4.13.3.2 Signatures

void setDownstreamPortSelectMode(in DownstreamPortSelectType mode)

3.4.4.13.3.3 Parameters

Name	Description	Type	Range/Values
mode	Downstream Digital Audio routing mode to be used by mixer/multiplexer	DownstreamPortSelectType	EXPLICIT_SELECT, FOLLOW_UPSTREAM_PORT, ALL_PORTS

Table 3.32: Audio Vocoder setDownstreamPortSelectMode parameters

3.4.4.13.3.4 Return Value

none

3.4.4.13.3.5 Exceptions

Following exceptions are raised for this operation-

- unsupportedOperation: If MUTLTI_AUDIO_PORTS is false (i.e. current deployment does not support multiple audio ports, facility shall raise unsupportedOperation exception.
- Unsupported: If the mode provided by radio application is not supported, facility shall raise Unsupported exception.

3.4.4.13.3.6 Attributes

- MUTLTI_AUDIO_PORTS
- NUM_AUDIO_PORTS
- PORT_DOWNSTREAM_SELECT_MODES

3.4.4.13.3.7 Behavior requirements

On setDownstreamPortSelectMode() call, if no exception is raised, facility shall

- Take no action if current value of applicableDownstreamPortSelectMode (currently active downstream audio selection mode) is same as mode input parameter of the call. Call returns else shall update the value of applicableDwosnstreamPortSelectMode with the new mode value.
- If the mode value passed in the call is EXPLICIT_SELECT, value of DownstreamActiveAudioPort (audio port to which the downstream audio is currently routed) remains unchanged (i.e., current active port remains active) and call returns.
- If the mode value passed is FOLLOW_UPSTREAM_PORT, update the value upstreamActiveAudioPort value with the value of downstreamActiveAudioPort.
- If the mode value passed is ALL_PORTS, update upstreamActiveAudioPort value to 0.
- If there is change in the value of upstreamActiveAudioPort then
  - If upstreamActiveAudioPort value is 0, set up the mechanism for replicating downstream digital audio data samples from radio application/vocoder functions to all audio ports.
  - Facility shall clear downstream digital audio data samples towards other audio port. Facility shall set up the mechanism for sending downstream digital audio data samples from radio application/vocoder functions to the port indicated by downstreamActiveAudioPort and the call returns.
- If there is no change in the value of upstreamActiveAudioPort, no further action is taken and the call returns.

3.4.4.13.4 getDownstreamPortSelectMode() operation

3.4.4.13.4.1 Overview

getDownstreamPortSelectMode() operation is used by radio application to get the currently active mode used for selection of audio port for downstream digital audio data from radio application/vocoder functions.

3.4.4.13.4.2 Signatures

void getDownstreamPortSelectMode(out DownstreamPortSelectType mode)

3.4.4.13.4.3 Parameters

Name	Description	Type	Range/Values
mode	Downstream Digital Audio routing mode currently in use by mixer/multiplexer	DownstreamPortSelectType	EXPLICIT_SELECT, FOLLOW_UPSTREAM_PORT, ALL_PORTS

Table 3.33: Audio Vocoder getDownstreamPortSelectMode parameters

3.4.4.13.4.4 Return Value

none

3.4.4.13.4.5 Exceptions

Following exceptions are raised for this operation-

- unsupportedOperation: If MUTLTI_AUDIO_PORTS is false (i.e., current deployment does not support multiple audio ports, facility shall raise unsupportedOperation exception.

3.4.4.13.4.6 Attributes

- MUTLTI_AUDIO_PORTS
- NUM_AUDIO_PORTS

3.4.4.13.4.7 Behavior requirements

On getDownstreamPortSelectMode() call facility shall return current value of applicableDownstreamPortSelectMode (currently active downstream audio selection mode) to radio application.

3.4.4.14 AudioVocoder::AudioPortSelection::PortSelection

3.4.4.14.1 setActivePort() operation

3.4.4.14.1.1 Overview

setActivePort() operation is used by radio application to set the active audio port for port selection during routing of upstream digital audio data or downstream digital audio data. Setting of active port is valid only when either of applicableDownstreamPortSelectMode or applicableDownstreamPortSelectMode is set to EXPLICT_SELECT value.

3.4.4.14.1.2 Signatures

void setActivePort(in octet portId);

3.4.4.14.1.3 Parameters

Name	Description	Type	Range/Values
portId	Identification of audio port to be made active	octet	Valid values between 1 and NUM_AUDIO_PORT

Table 3.34: Audio Vocoder setActivePort parameters

3.4.4.14.1.4 Return Value

none

3.4.4.14.1.5 Exceptions

Following exceptions are raised for this operation-

- unsupportedOperation: If MUTLTI_AUDIO_PORTS is false (i.e., current deployment does not support multiple audio ports, facility shall raise unsupportedOperation exception.
- configError: If portId provided by radio application, is greater than NUM_AUDIO_PORT, facility shall raise configError exception.
- unsupportedOperation: If neither of applicableDownstreamPortSelectMode or applicableDownstreamPortSelectMode is set to EXPLICT_SELECT, facility shall raise unsupportedOperation exception.

3.4.4.14.1.6 Attributes

- MUTLTI_AUDIO_PORTS
- NUM_AUDIO_PORTS

3.4.4.14.1.7 Behavior requirements

On setActivePort() operation, if no exception is raised, facility shall

- If value of applicableUpstreamPortSelect is EXPLICT_SELECT
  - If upstreamActiveAudioPort value is same as portId parameter, no further action is taken and call returns.
  - Else facility shall update value of upstreamActiveAudioPort to portId parameter. Facility shall clear any accumulated audio data samples received from audio ports. Facility shall set up the mechanism for accumulating and routing digital audio data samples from the selected upstreamActiveAudioPort towards radio application/vocoder function and call returns.
- If value of applicableDownstreamPortSelect is EXPLICIT_SELECT
  - If downstreamActivePort value is same as portId parameter, no further action is taken and call returns.
  - Else facility shall update the value of downstreamActivePort to portId parameter. Facility shall clear downstream digital audio data samples towards other audio port. Facility shall set up the mechanism for sending downstream digital audio data samples from radio application/vocoder functions to the port indicated by downstreamActivePort and the call returns.
- If the value of applicableDownstreamPortSelect is FOLLOW_UPSTREAM_PORT,
  - If the value of upstreamActivePort is same as downstreamActivePort, no further action is taken and call returns.
  - Else facility shall update the value of downstreamActivePort with the value of

upstreamActivePort. Facility shall clear downstream digital audio data samples towards other audio port. Facility shall set up the mechanism for sending downstream digital audio data samples from radio application/vocoder functions to the port indicated by downstreamActiveAudioPort and the call returns.

3.4.4.14.2 getUpstreamActivePort() operation

3.4.4.14.2.1 Overview

getUpstreamActivePort() operation is called by radio application to get the current active audio port for upstream audio data routing towards radio application/vocoder functions

3.4.4.14.2.2 Signatures

void getUpstreamActivePort(out octet portId)

3.4.4.14.2.3 Parameters

Name	Description	Type	Range/Values
portId	Active audio port Id for upstream digital audio data routing	octet	Values shall be between 0 and NUM_AUDIO_PORT Value 0 means audio data from all ports mixed for upstream digital audio

Table 3.35: Audio Vocoder getUpstreamActivePort parameters

3.4.4.14.2.4 Return Value

none

3.4.4.14.2.5 Exceptions

Following exceptions are raised for this operation-

- unsupportedOperation: If MUTLTI_AUDIO_PORTS is false (i.e., current deployment does not support multiple audio ports, facility shall raise unsupportedOperation exception.

3.4.4.14.2.6 Attributes

- MUTLTI_AUDIO_PORTS
- NUM_AUDIO_PORTS

3.4.4.14.2.7 Behavior requirements

On getUpstreamActivePort() call, if no exception is raised, facility shall provide the value of upstreamActivePort to radio application.

3.4.4.14.3 getDownstreamActivePort() operation

3.4.4.14.3.1 Overview

getDownstreamActivePort() operation is called by radio application to get the current

active audio port for routing of downstream audio data routing from radio application/vocoder function.

3.4.4.14.3.2 Signatures

void getDownstreamActivePort(out octet portId)

3.4.4.14.3.3 Parameters

Name	Description	Type	Range/Values
portId	Active audio port Id for downstream digital audio data routing	octet	Values shall be between 0 and NUM_AUDIO_PORT Value 0 means downstream audio data samples from radio application/vocoder function is replicated to all audio ports.

Table 3.36: Audio Vocoder getDownstreamActivePort parameters

3.4.4.14.3.4 Return Value

None

3.4.4.14.3.5 Exceptions

Following exceptions are raised for this operation-

- unsupportedOperation: If MUTLTI_AUDIO_PORTS is false (i.e., current deployment does not support multiple audio ports, facility shall raise unsupportedOperation exception.

3.4.4.14.3.6 Attributes

- MUTLTI_AUDIO_PORTS
- NUM_AUDIO_PORTS

3.4.4.14.3.7 Behavior requirements

On getDownstreamActivePort() call, if no exception is raised, facility shall provide the value of downstreamActivePort to radio application.

3.4.4.15 Exceptions

3.4.4.15.1 Facility Specific

3.4.4.15.1.1 AudioVocoder::InvalidToneProfile

Refer 3.1.3.1.2.2.1.1 of [ESSOR-Ref 03]

3.4.4.15.1.2 AudioVocoder::InvalidToneId

Refer 3.1.3.1.2.2.1.2 of [ESSOR-Ref 03]

3.4.4.15.2 Generic Exceptions

3.4.4.15.2.1 hardwareError

This exception is raised when called operation failed due to problem in hardware.

3.4.4.15.2.2 configError
This exception is raised when the parameters supplied during the called operation are out of supported range values for the parameter (Value is either lower than minimum supported value or higher than the maximum supported value)

3.4.4.15.2.3 invalidParameters
This exception is raised when the parameters provided supplied during the called operation are not valid values.

3.4.4.15.2.4 unsupported
This exception is raised when the algorithm or mode is not supported by the facility

3.4.4.15.2.5 unsupportedOperation
This exception is raised when the called operation is not supported by the facility for the current deployment.

3.4.4.15.2.6 OperationNotAvailable
This exception is raised when the called operation is not available in the current state of the facility.

3.4.4.15.2.7 noSpaceAvailable
This exception is raised when there is no space to store the data received along with the operation.

3.4.4.15.2.8 maxLengthExceeded
This exception is raised when the length of the data receiving along with the operation exceeds the maximum length valid for the operation.

3.4.4.15.2.9 noSpaceAvailable
This exception is raised when there is no space to store the data received along with the operation.

3.4.4.16 Types
The specification complies with the Full PIM IDL Profile of WInnF IDL profiles for PIM of SDR Applications, specified in [WInnF-Ref 05].

3.4.4.16.1 AudioVocoder::ConnectionType
Refer 3.1.3.1.1.3.1 of [ESSOR-Ref 03]

3.4.4.16.2 AudioVocoder::AcpModeType
Refer 3.1.3.1.1.3.2 of [ESSOR-Ref 03]

3.4.4.16.3 AudioVocoder::AcpProperties
Refer 3.1.3.1.1.4.1 of [ESSOR-Ref 03]

3.4.4.16.4 AudioVocoder::AudioParams
Refer 3.1.3.1.1.4.2 of [ESSOR-Ref 03]

3.4.4.16.5 AudioVocoder::AcpProperties
Refer 3.1.3.1.1.4.1 of [ESSOR-Ref 03]

3.4.4.16.6 AudioVocoder::AcpProperties
Refer 3.1.3.1.1.4.1 of [ESSOR-Ref 03]

3.4.4.16.7 AudioVocoder::ToneProfileType
Refer 3.1.3.1.2.2.2.1 of [ESSOR-Ref 03]

3.4.4.16.8 AudioVocoder::ToneSequence
Refer 3.1.3.1.2.2.2.2 of [ESSOR-Ref 03]

3.4.4.16.9 AudioVocoder::ToneDesciminator
Refer 3.1.3.1.2.2.3.1 of [ESSOR-Ref 03]

3.4.4.16.10 AudioVocoder::SimpleToneProfile
Refer 3.1.3.1.2.2.4.1 of [ESSOR-Ref 03]

3.4.4.16.11 AudioVocoder::ComplexToneProfile
Refer 3.1.3.1.2.2.4.2 of [ESSOR-Ref 03]

3.4.4.16.12 AudioVocoder::ToneElement
Refer 3.1.3.1.2.2.4.3 of [ESSOR-Ref 03]

3.4.4.16.13 AudioVocoder::MultiToneProfile
Refer 3.1.3.1.2.2.4.4 of [ESSOR-Ref 03]

3.4.4.16.14 AudioVocoder:: AlgorithmConfigurationType
Refer 3.2.10.3.1.1.2.1 of [ESSOR-Ref 03]

3.4.4.16.15 AudioVocoder:: Algorithm
Refer 3.2.3.1.1.2.1 of [ESSOR-Ref 03]
Different supported algorithm needs to be defined.

3.4.4.16.16 AudioVocoder:: AlgorithmSequence
Refer 3.2.3.1.1.2.2 of [ESSOR-Ref 03]

3.4.4.16.17 AudioVocoder:: DigitalAudioConfigType
DigitalAudioConfigType type specified as structure containing Sample Rate and sample size for the audio samples used for the digital audio.

```
typedef struct {
        unsigned short sampleRate;  // Sample rate in samples/second
        unsigned short sampleSize    // Sample size in bits
          } DigitalAudioConfigType;
```

3.4.4.16.18 AudioVocoder::audioSample

audioSamples Type is used to define a single audio sample for the digital audio. Representation of audioSample depends upon the sampleSize used for the digital audio.
if sampleSize is 8 bits - typedef octet audioSample;
if sampleSize is 16 bits - typedef unsigned short audioSample;
if sampleSize is 20 bits, 24 bits or 32 bits - typedef unsigned long audioSample;
If sampleSize is less than 32 bits (i.e., 20bit or 24 bits), lower significant bits contain the valid value.

3.4.4.16.19 AudioVocoder::audioSamplesPacket

audioSamplesPacket is specified as sequence of audioSample and represent the payload of digital audio packet.
typedef sequence <audioSamples> audioSamplesPacket;

3.4.4.16.20 AudioVocoder::streamControlType

streamControlType provides additional information for the stream data payload in the packet.

```
typedef struct {
        boolean         endOfStream;
        boolean         purge;
        unsigned short streamId;
        unsigned long SequenceNum;
} streamControlType;
```

Name	Type	Description	Range/Value
endOfStream	boolean	Indicates the last packet of the current stream	TRUE/FALSE
purge	boolean	Purge flag is set along with endOfStream set and zero-byte payload. Indicates that current stream shall be purged (any outstanding packets in the receiver queue shall be purged)	TRUE/FALSE
streamId	unsigned short	Indicates stream for the current packet. Stream id is monotonously increased (with roll over) for each new stream. Change of streamId indicates new stream. New stream shall be started when new stream packets are sent after a gap (not due to loss of packets) e.g., in case of PTT every new talkburst is started with new streamId.	

sequenceNum	unsigned long	Indicates the sequence number of the first sample/frame within the packet. First sample/frame within the first packet of a stream is assigned the sequence number of 0. Sequence number is monotonically increased for each sample/frame being sent irrespective of which packet it is part of. E.g SequenceNum field in the first packet of a stream is 0. If 160 samples are packed within the packet, the SequenceNum field of the next packet will be 0+160=160 and so on for the subsequent packet of the stream.	

Table 3.37: Audio Vocoder streamControlType

3.4.4.16.21 AudioVocoder::vocFlagType

vocFlagType provides information about the vocoder Frame payload type and is specified as set of enumerated value.

```
typedef enum {
        VOICE_FRAME;
        CN_SID_FRAME;
        LOST_FRAME;
} vocFlagType;
```

Name	Type	Description	Range/Value
VOICE_FRAME	Enum literal	In case of VAD/DTX enabled, this value indicates the associated payload contains valid vocoded voice information. In case of VAD/DTX disabled (or not supported), this value indicates associated payload contains valid vocoded audio bits.	
CN_SID_FRAME	Enum literal	Used in case of VAD/DTX enabled. For TX vocoder (encoder), this value indicates, the associated payload bit contains comfort noise/silence detection information. Radio application can choose to apply DTX for the same. For RX vocoder (decoder), the associated payload bits shall be treated as comfort noise / silence indication.	
LOST_FRAME	Enum literal	Used only for RX vocoder (Decoder). This value indicates associated	

		payload is invalid and vocoder may apply Packet Loss Concealment (PLC) function.	

Table 3.38: Audio Vocoder vocFlagType

3.4.4.16.22 AudioVocoder::vocFrame

vocFrame represent a single Vocoder Frame and is specified as a structure containing vocoded audio frame bits and associated metadata related to the payload of the frame.

```
typedef struct {
        vocFlagType payloadFlag;
        unsigned short      padBits;
        sequence <octet> framePayload;
} vocFrame;
```

Name	Type	Description
payloadFlag	vocFlagType	Metadata information for the associated payload bits
padBits	unsigned short	Number of padded bits in the last octet of payload
framePayload	Octet sequence	Vocoder Frame bits. Packing of vocoder frame bits into octet sequence shall be as per Appendix C of [ESSOR-Ref 03]

Table 3.39: Audio Vocoder vocFrame

3.4.4.16.23 AudioVocoder::vocFramesPacket

vocFramesPacket is specified as sequence of vocFrame and represent the payload of vocoded audio packet.

```
typedef sequence <vocFrame> vocFramesPacket;
```

3.4.4.16.24 AudioVocoder::UpstreamPortSelectType

UpstreamPortSelectType represent the mechanism for selection of digital audio data from audio ports to be sent upstream towards vocoder functions/Radio application. UpstreamPortSelectType is specified as enum as below-

```
typedef enum {
        EXPLICIT_SELECT,
        PB_PRIORITY,
        MIXED,
} UpstreamPortSelectType;
```

Name	Description
EXPLICIT_SELECT	Explicit API call is used by radio application to select the audio Port whose digital audio data is sent upstream towards radio application/vocoder function.
PB_PRIORITY	Digital audio data from the audio Port whose associated PTT Push Button press is active is sent upstream towards radio application/vocoder function. If more than one audio port has its Push Button pressed, digital

	audio data from the highest priority audio port among them is sent upstream towards radio application/vocoder function. Identification number of audio port (value between 1 and NUM_AUDIO_PORT) is taken as priority. Lower number indicates higher priority.
MIXED	Digital audio data from all audio ports are mixed together and sent upstream towards radio application/vocoder function.

Table 3.40: Audio Vocoder UpstreamPortSelectType

3.4.4.16.25 AudioVocoder::DownstreamPortSelectType

DownstreamPortSelectType represent the mechanism for distribution of downstream digital audio data from radio application/vocoder function towards the audio ports. DownstreamPortSelectType is specified as enum as below-

```
typedef enum {
        EXPLICIT_SELECT,
        FOLLOW_UPSTREAM_PORT
        ALL_PORTS,
} DownstreamPortSelectType;
```

Name	Description
EXPLICIT_SELECT	Explicit API call is used by radio application to select the audio Port to which downstream digital audio data from radio application/vocoder function is sent.
FOLLOW_UPSTREAM_PORT	Downstream digital audio data from radio application/vocoder function is sent to same audio port whose digital audio data is sent upstream.
ALL_PORTS	Downstream digital audio data from radio application/vocoder function is sent to all audio ports.

Table 3.41: Audio Vocoder DownstreamPortSelectType

3.4.5 Facility Attributes

3.4.5.1 Capabilities

3.4.5.1.1 Services Capability

Name	Type	Description	Range/Values
AUD_SUPPORT	boolean	Indicates whether facility support the Audio port functionality.	TRUE = facility supports Audio conversion functionality including PTT signalling FALSE= CODEC HW absent
MULTI_AUDIO_PORTS	boolean	Indicates whether facility/radio platform supports multiple audio ports scenario.	TRUE = facility supports multiple audio ports scenario FALSE= only single audio port present in the facility Used only when AUD_SUPPORT is TRUE

VOCODER_SUPPORT	boolean	Indicates whether facility supports vocoder functionality.	TRUE = facility supports at least one vocoder algorithm implementation (can be HW or FW implementation) FALSE=no vocoder function available
SECOND_CHANNEL	boolean	Indicates whether facility supports an additional audio channel.	TRUE = facility supports additional audio channel facility support FALSE = only one audio channel present in the facility. If additional channel support is indicated both audio conversion (per Audio port) and vocoder functions support two audio channels
DIG_AUD_RTNG_AVAIL	boolean	Indicates whether facility support routing control for the Digital Audio (raw samples).	TRUE = Support of Digital Audio routing configuration via configuring DIG_AUD_RTNG_CFG attribute is available. FALSE = digital audio (raw samples) are directly connected between the audio conversion functions and vocoder function (equivalent to implicit connection between vocoder and audio port in [ESSOR-Ref 03]) Value is always TRUE if any of the AUD_SUPPORT or VOCODER_SUPPORT is FALSE

Table 3.42: Audio Vocoder services capability

3.4.5.1.2 Features Capability

Name	Type	Description	Range/ Values
AUDIO_SERVICES	structure (see below)	Structure indicating different audio services supported by the facility. Valid only when AUD_SUPPORT is true.	
NUM_AUDIO_PORT	unsigned short	Number of Audio ports supported by the facility. Considered only when MULTI_AUDIO_PORTS is true.	
VOCODER_ALGOS	sequence of structure	Sequence of supported vocoder algorithms.	

		Valid only when VOCODER_SUPPORT is true.	
DIGITAL_AUD_RTNG_ MODES	structure	Each Boolean field in the structure indicates whether the corresponding Digital Audio Routing Mode is supported by the facility. Valid only when DIG_AUD_RTNG_AVAIL is true	
PORT_UPSTREAM_S ELECT_MODES	structure	Each Boolean field in the structure indicates whether the corresponding Port Select Mode is supported by the facility. Valid only when MULTI_AUDIO_PORTS is true.	
PORT_DOWNSTREAM _SELECT_MODES	structure (see below)	Each Boolean field in the structure indicates whether the corresponding Port Select Mode is supported by the facility. Valid only when MULTI_AUDIO_PORTS is true.	

Table 3.43: Audio Vocoder features capability

```
typedef boolean isSupported ;
AUDIO_SERVICES is specified as
struct AUDIO_SERVICES {
        isSupported pttPushButton,
        isSupported simpleToneProfile,
        isSupported complexToneProfile,
        isSupported multiToneProfile,
        isSupported audioConversion,
        isSupported channelAcpFunction,
} ;

VOCODER_ALGOS is specified as
typedef struct {
        Algorithm      vocoderAlgoId;
        isSupported   vadDtx;
        isSupported    fullDuplex;  // If Full duplex is not supported, vocoder is half-duplex
} vocoderAlgo;

sequence <vocoderAlgo> VOCODER_ALGOS;
DIGITAL_AUD_RTNG_MODES is specifies as
struct DIGITAL_AUD_RTNG_MODES {
        isSupported  internalRtng,        // Valid only when both AUD_SUPPORT and
```

VOCODER_SUPPORT are true.
isSpported externalCodecRtng, // Valid only when AUD_SUPPORT is true.
isSupported externalVocoderRtng, //Valid only when VOCODER_SUPPORT is true.
isSupported loopbackRtng, // Valid only when VOCODER_SUPPORT is true.
};

PORT_UPSTREAM_SELECT_MODES is specified as
struct PORT_UPSTREAM_SELECT_MODES {
isSupported explicitSelection,
isSupported pbPrioritySelection,
isSupported mixedSelection
};

PORT_DOWNSTREAM_SELECT_MODES is specified as
struct PORT_DOWNSTREAM_SELECT_MODES {
isSupported explictSelection,
isSupported followUpstreamSelection,
isSupported allPortsSelection
};

3.4.5.1.3 Parameters Validity

Name	Type	Description	Range/Values (Recommended)
MIN_SIMPLE_TONE _FREQ MAX_SIMPLE_TONE_FREQ	unsigned short	Minimum and maximum frequency (in HZ) for Simple tone Profile supported by the facility.	50-4000 Hz
MIN_MULTI_TONE_FREQ MAX_MULTI_TONE_FREQ	unsigned short	Minimum and maximum frequency (in HZ) for Multi tone Profile supported by facility.	50-4000 Hz
MIN_ACP_HOLD_TIME MAX_ACP_HOLD_TIME	unsigned long	Minimum and maximum Hold time (in msec) for ACP properties.	5-5000 msec.
MIN_ACP_RELEASE_TIME MAX_ACP_RELEASE_TIME	unsigned long	Minimum and maximum Release time (in msec) for ACP properties.	5-500 msec.

MIN_ACP_ATTACK_TIME MAX_ACP_ATTACK_TIME	unsigned short	Minimum and maximum Attack time (in msec) for ACP properties.	5-50 msec.
MIN_ACP_THRESHOLD_VALUE MAX_ACP _THRESHOLD_VALUE	float	Minimum and maximum values of input signal minimum threshold (in dBm) for ACP properties.	-30 - +10 dBm
MIN_ACP_NORM_FACTOR_VALUE MAX_ACP_NORM_FACTOR_VALUE	float	Minimum and maximum values of normalisation factor for ACP properties.	0 – 1
ACP_MODES	structure (see below)	Each Boolean field in the structure indicates whether the corresponding ACP mode is supported by the facility.	
MIN_ NUM_AUD_SAMPLE_PKT MAX_NUM_AUD_SAMPLE_PKT	unsigned short	Minimum and maximum number of samples per sample packet (carrying digital audio data) supported by the facility	64-320
MAX_NUM_VOC_FRAME_PKT	unsigned short	Maximum number of vocoder frame per vocoder packet (carrying vocoded audio data) supported by the facility. Minimum value is 1 vocoder frame per packet	
MAX_VOC_FRAME_SIZE	unsigned short	Maximum size (in bits) of vocoder frame supported by facility. Used only for continuous bitstream vocoder algorithm.	
MIN_INPUT_GAIN MAX_INPUT_GAIN	float	Minimum and maximum values for input gain (in dB) configuration	
INPUT_GAIN_STEP	float	Steps size for input gain (in dB)	

		configuration	
MIN_OUTPUT_GAIN MAX_OUTPUT_GAIN	float	Minimum and maximum values for output gain (in dB) configuration	
OUTPUT_GAIN_STEP	float	Steps size for output gain (in dB) configuration	
MAX_TONE_SAMPLES_STORAGE	unsigned long	Maximum storage capacity (in bytes) available for storing all complex tone profiles samples	3000000
MIN_SAMPLE_RATE MAX_SAMPLE_RATE	unsigned short	Minimum and maximum sample rates (in samples/second) supported by the facility.	8000 - 48000
SAMPLE_SIZES	structure (see below)	Each Boolean field in the structure indicates whether the corresponding sample size (bits) is supported by the facility.	
CONNECTION_TYPES	structure (see below)	Each Boolean field in the structure indicates whether the corresponding Connection type is supported by the facility. The types are valid only if SECOND_CHANNEL is true.	

Table 3.44: Audio Vocoder parameters validity

Supported ACP modes is specified as

```
struct   ACP_MODES {
        isSupported acpModePeak,
        isSupported acpModeAverage
};
```

Supported SAMPLE_SIZES is specified as

```
struct   SAMPLE_SIZES {
        isSupported 8bitSample,
        isSupported 16bitSample,
        isSupported 20bitSample,
        isSupported 24bitSample,
        isSupported 32bitSample
};
```

Supported CONNECTION_TYPES is specified as

```
struct   CONNECTION_TYPES {
        isSupported directConnection,
        isSupported crossedConnection
};
```

3.4.5.2 Properties

3.4.5.2.1 Behavior

Name	Type	Description	Range/Values
DIG_AUD_RTNG_CFG	Enum (see below)	Controls the routing of Digital Audio Samples (raw) for the vocoder and codec. Same configuration applies to both channels (if SECOND_CHANNEL = true). Valid only if DIG_AUD_RTNG_AVAIL = true.	INTERNAL, EXTERNAL_CODEC, EXTERNAL_VOCODER, LOOPBACK. When only AUD_SUPPORT is true, only valid value is EXTERNAL_CODEC When only VOCODER_SUPPORT is true, only valid values are EXTERNAL_VOCODER (default value) and LOOPBACK. When both AUD_SUPPORT and VOCODER_SUPPORT is true, all values are valid. Default value is INTERNAL.

Table 3.45: Audio Vocoder behavior properties

DIG_AUD_RTNG_CFG is specified as

```
enum DIG_AUD_RTNG_CFG {INTERNAL, EXTERNAL_CODEC, EXTERNAL_VOCODER, LOOPBACK};
```

3.4.5.2.2 Initialization

Name	Type	Description	Range/Values

			(Recommended)
INIT_SAMPLE_RATE	unsigned short	Initial for the sample rate (in samples/sec) for codec of audio port	8000 samples/sec
INIT_SAMPLE_SIZE	unsigned short	Initial sample size (in bits) for codec of audio port	16 bits
INIT_CONNECTION_TYPE	enum	Initial value for the connection type	DIRECT
INIT_NUM_AUD_SAMPLES_PKT	unsigned short	Initial value for the samples per digital audio packet	160 samples
INIT_NUM_VOC_FRAME_PKT	unsigned short	Initial value of number of vocoder frame per vocoder packet (carrying vocoded audio data).	1
INIT_VOC_FRAME_SIZE	unsigned short	Initial size (in bits) of vocoder frame (only for continuous bitstream vocoders).	
INIT_AUD_INPUT_GAIN	float	Initial value for input gain (in dB) for Audio channel	
INIT_AUD_OUTPUT_GAIN	float	Initial value output gain (in dB) for Audio channel	
INIT_ACP_HOLD_TIME	unsigned long	Initial value for the Hold time (in msec) for ACP properties	1000 msec.
INIT_ACP_RELEASE_TIME	unsigned long	Initial value for the Release Time (in msec) for ACP properties	100 msec.
INIT_ACP_ATTACK_TIME	unsinged short	Initial value for the Attack time (in msec) for ACP properties	5 msec
INIT_ACP_THRESHOLD_VALUE	float	Initial value for input signal minimum threshold (in dBm) for ACP properties	-15 dBm
INIT_ACP_NORM_FACTOR_VALUE	float	Normalization factor for ACP properties	
INIT_ACP_MODE	enum	Initial value of ACP mode	

INIT_UPSTREAM_SELECT_MODE (if MULTI_AUDIO_PORTS = true)	enum	Initial value audio port selection mode for upstream digital audio	PB_PRIORITY
INIT_DOWNSTREAM_SELECT_MODE (if MULTI_AUDIO_PORTS = true)	enum	Initial value audio port selection mode for downstream digital audio	ALL

Table 3.46: Audio Vocoder initialization properties

3.5 IP Service Facility

3.5.1 Approach

3.5.1.1 Candidate API Set

ESSOR IP Service API - [ESSOR-Ref 02]

3.5.1.2 Selected API Set

ESSOR IP Service API - [ESSOR-Ref 02]

3.5.1.3 Modification

Modification in API for exclusion of network management APIs and to adapt in PIM of facility framework, suitable modifications are done to remove any SCA specific terminology. Modifications are also performed for providing IP-V6 support.

3.5.1.4 Rationale

Above referenced API is sufficient to cater requirement of platform and waveform components. ESSOR IP Service API specification was in line with SCA and to create PIM in Facility Framework, suitable modifications are required. The network management interface has been excluded to prohibit access to radio application. Modifications are required for catering the IP addressing formats for IP_V6 support.

3.5.1.5 Conclusion

ESSOR API is adopted without the network management interface and provided in the form of PIM specification as per Principles for WInnF Facility Standards [WInnF-Ref 09] along with modifications for supporting IP-V6 addressing formats. PSM Documents (SCA, Native C++, FPGA) corresponding to this facility PIM specification as applicable will be released after their reference implementation.

3.5.2 Introduction

Section 0 provides the PIM specification of IRSA IP Service Facility. The specification of IP Service facility is applicable to diverse platforms irrespective of underlying hardware and software.

3.5.2.1 Overview

Refer 8.1 of [ESSOR-Ref 02] for basic details. IP Service Facility provides following

capabilities.

- TCP/IP stack encapsulation for the Platform & WF.
- TCP/IP based data exchange service to WF.
- Interface for routing configuration.
- Interface for network interface management.
- Implements the mechanism for routing IP packets received from the WF component.
- Implements and manages the routing table to obtain the next hop router information to route IP packets to the subnets connected to the gateway node.
- Implements and manages the ARP table to route IP packets to the subnets connected to the Radio locally.

3.5.2.2 Technical Details

Refer 8.1.1 and 8.1.2 of [ESSOR-Ref 02] for usage context and general behavior of IP Service facility in SCA specific platform.

3.5.3 Services

3.5.3.1 Provide Services

Service Group/Module	Interface Service	Primitives/Operations
IPSData	IPServicePacketProducer	setMaxPayloadSize() setMinPayloadSize() signalFlowResume()
	IPServicePacketConsumer	pushPacket() getMaxPayloadSize() getMinPayloadSize()
IPSConfig	IPServiceRoutingConfig	addRoute() deleteRoute() assocArp() readRouteTable() readARPTable()

Table 3.47: IP Service provide services

3.5.3.2 Use Services

Service Group/Module	Interface Service	Primitives/Operations
IPSData	IPServicePacketConsumer	pushPacket() getMaxPayloadSize() getMinPayloadSize()
	IPServicePacketproducer	setMaxPayloadSize() setMinPayloadSize() signalFlowResume()

Table 3.48: IP Service use services

3.5.3.3 State Machines

3.5.3.3.1 IPService_SM

IPService_SM is specified as the main state machine followed by IP Service Facility. An instance of IPService_SM is followed by each IP Service instance.

The following figure is the statechart of IPService_SM state machine:

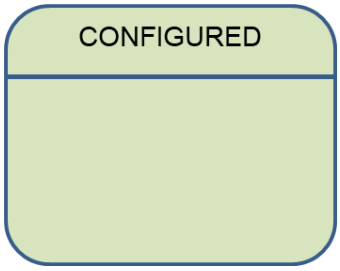

Figure 3.19 : IPService_SM statechart

3.5.3.3.1.1 States

3.5.3.3.1.2 CONFIGURED

CONFIGURED is specified as the unique state of IP Service, during which it is configured according to the requirements of all the radio application to be supported during the CONFIGURED state.
CONFIGURED is reached by IP Service when it

- Complies with any value specified for a capability or a property,
- Is capable of interacting with radio application according to the service interfaces of its service implementations.

How CONFIGURED is reached is unspecified by the PIM specification, and can be specified by the applied PSM specification.

3.5.3.4 Service groups

3.5.3.4.1 IPService::IPSData

This group provides data handling capability for IP Service Facility.

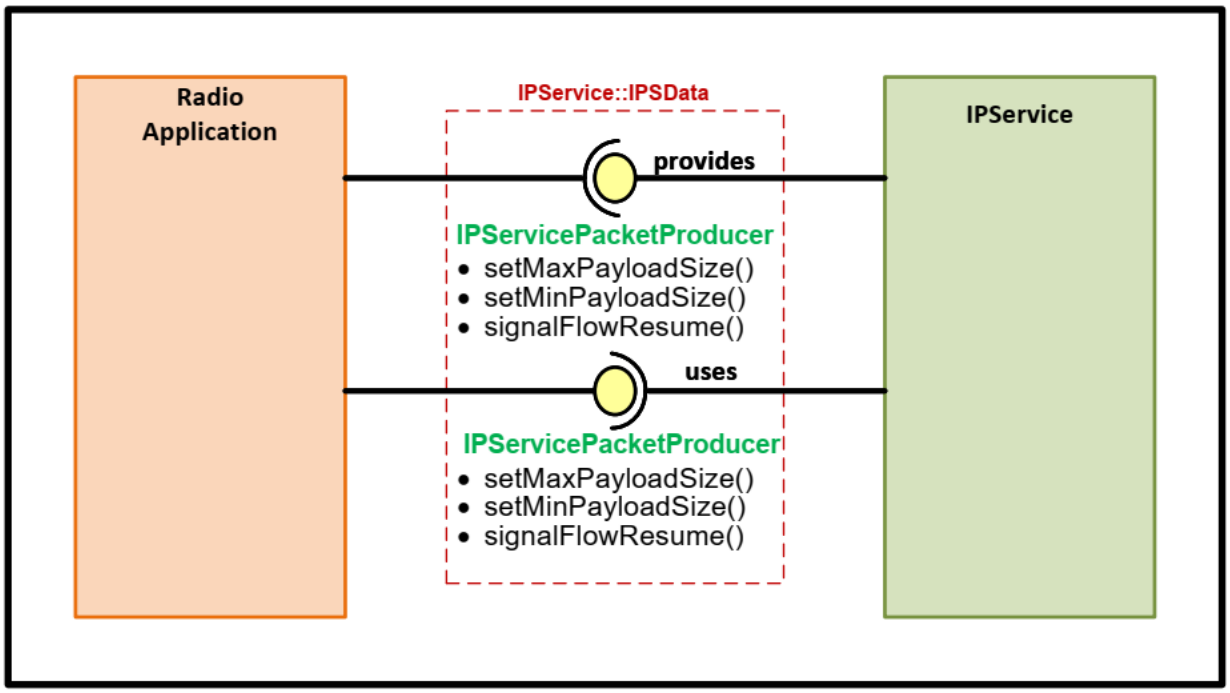

Figure 3.20 : IPSData (IPServicePacketProducer) services group

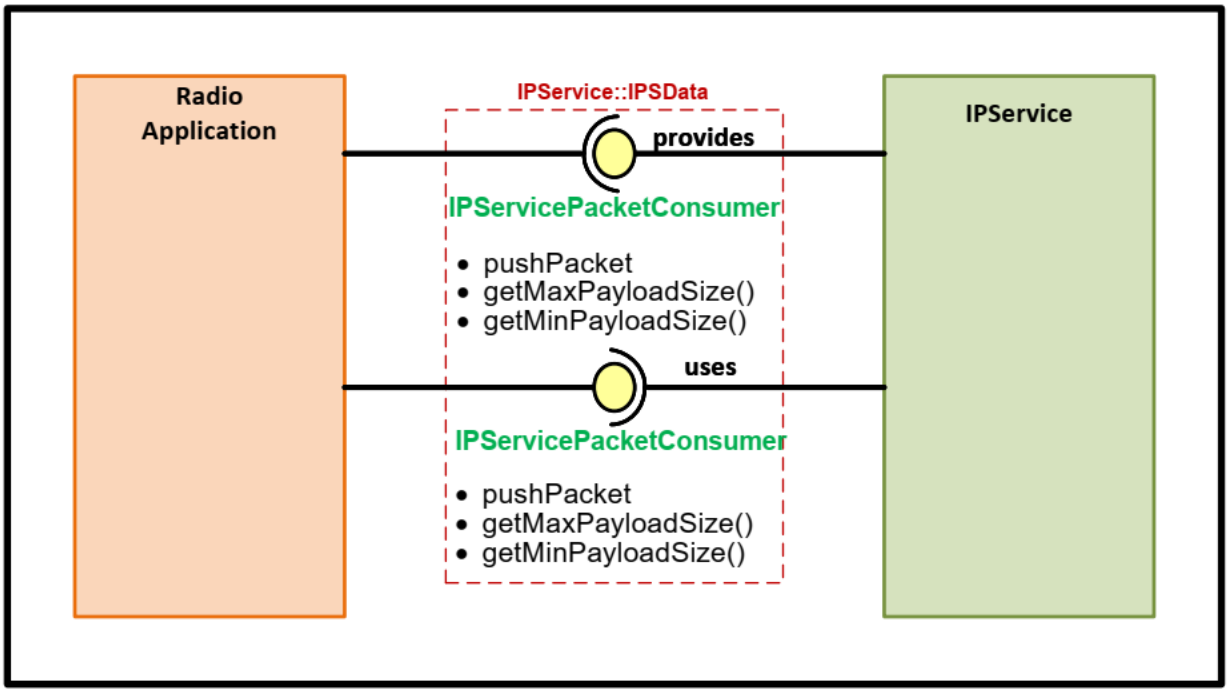

Figure 3.21 : IPSData (IPServicePacketConsumer) services group

3.5.3.4.2 IPService::IPSConfig

This group provides primitives to get the values of minimum and maximum payload size and for management of routing and ARP tables like add, delete routes and reading the routing table.

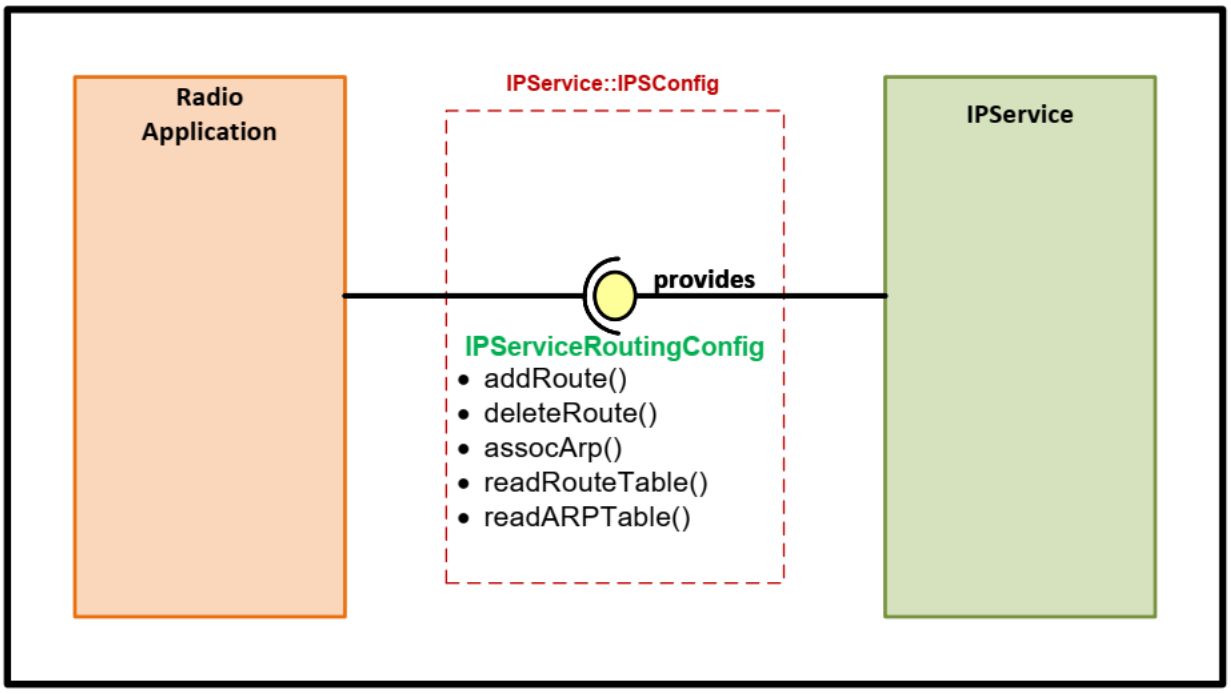

Figure 3.22 : IPSConfig services group

3.5.4 Service Primitives

This section specifies the primitives of the IP Service facility services. Each declaration of a primitive complies with the Full PIM IDL Profile of WInnF IDL profiles for PIM of SDR Applications, specified in [WInnF-Ref 05]. The declaration of each primitive also complies with SCA 4.1 Appendix E-1 [JTNC-Ref 30] .

3.5.4.1 IPSData::IPServicePacketProducer

3.5.4.1.1 setMaxPayloadSize()

3.5.4.1.1.1 Overview

Refer 8.1.7.3.3.1 of [ESSOR-Ref 02]

3.5.4.1.1.2 Signatures

Refer 8.1.7.3.3.1.1 of [ESSOR-Ref 02]

3.5.4.1.1.3 Parameters

Refer 8.1.7.3.3.1.2 of [ESSOR-Ref 02]

3.5.4.1.1.4 Exceptions

Refer 8.1.7.3.3.1.6 of [ESSOR-Ref 02]

3.5.4.1.1.5 Attributes

None.

3.5.4.1.1.6 Behavior requirements
Refer 8.1.7.3.1 of [ESSOR-Ref 02]

3.5.4.1.2 setMinPayloadSize()

3.5.4.1.2.1 Overview
Refer 8.1.7.3.3.2 of [ESSOR-Ref 02]

3.5.4.1.2.2 Signatures
Refer 8.1.7.3.3.2.1 of [ESSOR-Ref 02]

3.5.4.1.2.3 Parameters
Refer 8.1.7.3.3.2.2 of [ESSOR-Ref 02]

3.5.4.1.2.4 Exceptions
Refer 8.1.7.3.3.2.6 of [ESSOR-Ref 02]

3.5.4.1.2.5 Attributes
None.

3.5.4.1.2.6 Behavior requirements
Refer 8.1.7.3.1 of [ESSOR-Ref 02]

3.5.4.1.3 signalFlowResume()

3.5.4.1.3.1 Overview
Refer 8.1.7.3.3.3 of [ESSOR-Ref 02]

3.5.4.1.3.2 Signatures
Refer 8.1.7.3.3.3.1 of [ESSOR-Ref 02]

3.5.4.1.3.3 Parameters
Refer 8.1.7.3.3.3.2 of [ESSOR-Ref 02]

3.5.4.1.3.4 Exceptions
Refer 8.1.7.3.3.3.5 of [ESSOR-Ref 02]

3.5.4.1.3.5 Attributes
None.

3.5.4.1.3.6 Behavior requirements
Refer 8.1.7.3.1 of [ESSOR-Ref 02]

3.5.4.2 IPSData::IPServicePacketConsumer

3.5.4.2.1 pushPacket()

3.5.4.2.1.1 Overview

Refer 8.1.7.2.3.1 of [ESSOR-Ref 02]

3.5.4.2.1.2 Signatures
Refer 8.1.7.2.3.1.1 of [ESSOR-Ref 02]

3.5.4.2.1.3 Parameters
Refer 8.1.7.2.3.1.2 of [ESSOR-Ref 02]

3.5.4.2.1.4 Exceptions
Refer 8.1.7.2.3.1.6 of [ESSOR-Ref 02]

3.5.4.2.1.5 Attributes
None.

3.5.4.2.1.6 Behavior requirements
Refer 8.1.7.2.1 of [ESSOR-Ref 02]

3.5.4.2.2 getMaxPayloadSize()

3.5.4.2.2.1 Overview
Refer 8.1.7.2.3.2 of [ESSOR-Ref 02]

3.5.4.2.2.2 Signatures
Refer 8.1.7.2.3.2.1 of [ESSOR-Ref 02]

3.5.4.2.2.3 Parameters
Refer 8.1.7.2.3.2.2 of [ESSOR-Ref 02]

3.5.4.2.2.4 Exceptions
Refer 8.1.7.2.3.2.6 of [ESSOR-Ref 02]

3.5.4.2.2.5 Attributes
None.

3.5.4.2.2.6 Behavior requirements
Refer 8.1.7.2.1 of [ESSOR-Ref 02]

3.5.4.2.3 getMinPayloadSize()

3.5.4.2.3.1 Overview
Refer 8.1.7.2.3.3 of [ESSOR-Ref 02]

3.5.4.2.3.2 Signatures
Refer 8.1.7.2.3.3.1 of [ESSOR-Ref 02]

3.5.4.2.3.3 Parameters
Refer 8.1.7.2.3.3.2 of [ESSOR-Ref 02]

3.5.4.2.3.4 Exceptions
Refer 8.1.7.2.3.3.6 of [ESSOR-Ref 02]

3.5.4.2.3.5 Attributes
None.

3.5.4.2.3.6 Behavior requirements
Refer 8.1.7.2.1 of [ESSOR-Ref 02]

3.5.4.3 IPSConfig:: IPServiceRoutingConfig

3.5.4.3.1 addRoute()

3.5.4.3.1.1 Overview
Refer 8.1.7.5.3.1 of [ESSOR-Ref 02]

3.5.4.3.1.2 Signatures
Refer 8.1.7.5.3.1.1 of [ESSOR-Ref 02]

3.5.4.3.1.3 Parameters
Refer 8.1.7.5.3.1.2 of [ESSOR-Ref 02]

3.5.4.3.1.4 Exceptions
Refer 8.1.7.5.3.1.6 of [ESSOR-Ref 02]

3.5.4.3.1.5 Attributes
None.

3.5.4.3.1.6 Behavior requirements
Refer 8.1.7.5.1 of [ESSOR-Ref 02]

3.5.4.3.2 deleteRoute()

3.5.4.3.2.1 Overview
Refer 8.1.7.5.3.2 of [ESSOR-Ref 02]

3.5.4.3.2.2 Signatures
Refer 8.1.7.5.3.2.1 of [ESSOR-Ref 02]

3.5.4.3.2.3 Parameters
Refer 8.1.7.5.3.2.2 of [ESSOR-Ref 02]

3.5.4.3.2.4 Exceptions
Refer 8.1.7.5.3.2.6 of [ESSOR-Ref 02]

3.5.4.3.2.5 Attributes
None.

3.5.4.3.2.6 Behavior requirements
Refer 8.1.7.5.1 of [ESSOR-Ref 02]

3.5.4.3.3 assocArp()

3.5.4.3.3.1 Overview
Refer 8.1.7.5.3.3 of [ESSOR-Ref 02]

3.5.4.3.3.2 Signatures
Refer 8.1.7.5.3.3.1 of [ESSOR-Ref 02]

3.5.4.3.3.3 Parameters
Refer 8.1.7.5.3.3.2 of [ESSOR-Ref 02]

3.5.4.3.3.4 Exceptions
Refer 8.1.7.5.3.3.6 of [ESSOR-Ref 02]

3.5.4.3.3.5 Attributes
None.

3.5.4.3.3.6 Behavior requirements
Refer 8.1.7.5.1 of [ESSOR-Ref 02]

3.5.4.3.4 readRouteTable()

3.5.4.3.4.1 Overview
Refer 8.1.7.5.3.4 of [ESSOR-Ref 02]

3.5.4.3.4.2 Signatures
Refer 8.1.7.5.3.4.1 of [ESSOR-Ref 02]

3.5.4.3.4.3 Parameters
Refer 8.1.7.5.3.4.2 of [ESSOR-Ref 02]

3.5.4.3.4.4 Exceptions
Refer 8.1.7.5.3.4.6 of [ESSOR-Ref 02]

3.5.4.3.4.5 Attributes
None.

3.5.4.3.4.6 Behavior requirements
Refer 8.1.7.5.1 of [ESSOR-Ref 02]

3.5.4.3.5 readARPTable()

3.5.4.3.5.1 Overview
Refer 8.1.7.5.3.5 of [ESSOR-Ref 02]

3.5.4.3.5.2 Signatures
Refer 8.1.7.5.3.5.1 of [ESSOR-Ref 02]

3.5.4.3.5.3 Parameters
Refer 8.1.7.5.3.5.2 of [ESSOR-Ref 02]

3.5.4.3.5.4 Exceptions
Refer 8.1.7.5.3.5.6 of [ESSOR-Ref 02]

3.5.4.3.5.5 Attributes
None.

3.5.4.3.5.6 Behavior requirements
Refer 8.1.7.5.1 of [ESSOR-Ref 02]

3.5.4.4 Exceptions
Refer sections 8.1.7.3.2.1.1 and 8.1.7.5.2.1.1 of [ESSOR-Ref 02]

3.5.4.5 Type
The specification complies with the Full PIM IDL Profile of WInnF IDL profiles for PIM of SDR Applications, specified in [WInnF-Ref 05].

3.5.4.5.1 IPVer

```
typedef enum {
        IP_V4,
        IP_V6
} IPVer;
```

3.5.4.5.2 Ipv4RoutingProtocols

```
typedef sequence <string> ipv4RoutingProtocols;
```

3.5.4.5.3 Ipv6RoutingProtocols

```
typedef sequence <string> ipv6RoutingProtocols;
```

3.5.5 IP Service Facility Attributes
The format for IP address shall follow the dot [.] notation for IP_V4 and colon [:] notation for IP_V6 wherever applicable.

3.5.5.1 Capabilities

Name	Type	Description	Range/Values
IP_VERSION	enum IPVer	Defines the IP versions supported.	IP_V4, IP_V6

IPV4_MULTICAST_SUPPORTED	boolean	Indicates whether the IPV4 multicast support is provided or not	TRUE indicates IPV4 multicast mode is supported, FALSE otherwise.
IPV4_BROADCAST_SUPPORTED	boolean	Indicates whether the IPV4 broadcast support is provided or not	TRUE indicates IPV4 broadcast mode is supported, FALSE otherwise.
SUPPORTED_IPV4_ROUTING_PROTOCOLS	ipv4RoutingProtocols	Defines all the IPV4 routing protocols supported	Sequence of IPV4 routing protocols
SUPPORTED_IPV6_ROUTING_PROTOCOLS	ipv6RoutingProtocols	Defines all the IPV6 routing protocols supported	Sequence of IPV6 routing protocols

Table 3.49: IP Service general capabilities

3.5.5.2 Properties

Name	Type	Description	Range/Values
MAXIMUM_SIZE	unsigned long	[46, 16383]	Maximum payload size allowed for a payload.
MINIMUM_SIZE	unsigned long	[46, 16383] as per ESSOR [20, 16383] to cater for smallest possible IPV4 packet.	Minimum payload size allowed for a payload.
IP_ADDRESS	string	N/A	IP address in dot (.) notation for IP_V4 | in colon (:) notation for IP_V6
NETMASK	string	N/A	Netmask [in dot (.) notation for IP_V4 | in colon (:) notation for IP_V6].
INTERFACE_NAME	string	N/A	A string representing the name of the interface.
IP_VERSION	enum IPVer	As per the enumerations for IPVer.	Defines the IP versions supported.
ROUTE_TABLE_SEQ	struct RouteTableEntry	N/A	Sequence of routing table entries.
ARP_TABLE_SEQ	Struct ArpTableEntry	N/A	Sequence of ARP table entries.

Table 3.50: IP Service properties

3.5.5.3 Variables

None.

3.6 GNSS Facility

3.6.1 Approach

3.6.1.1 Candidate API Set

- JTRS Standard Global Positioning System API - [JTNC-Ref 20]
- GNSS Device API - [ESSOR-Ref 03]

3.6.1.2 Selected API Set

GNSS Device API - [ESSOR-Ref 03]

3.6.1.3 Modification

ESSOR GNSS Device API specification was in line with SCA and to create PIM in Facility Framework, suitable modifications are required. The primitives from PNT Extension of ESSOR GNSS Device API pertaining to timing service have been selectively included in the GNSS Facility. Modifications are also required to provide supporting Global navigation satellite systems including India's NAVIC.

3.6.1.4 Rationale

ESSOR GNSS Device API does not have APIs for selecting the GNSS Constellation selection. Therefore, APIs have been added for constellation selection.

3.6.1.5 Conclusion

ESSOR API is extended and provided in the form of PIM specification as per Principles for WInnF Facility Standards [WInnF-Ref 09]. Constellation selection APIs have been added. PSM Documents (SCA, Native C++, FPGA) corresponding to this facility PIM specification as applicable will be released after their reference implementation.

3.6.2 Introduction

Section 3.6 provides the PIM specification of IRSA GNSS Facility. The specification of GNSS facility is applicable to diverse platforms irrespective of underlying hardware and software.

3.6.2.1 Overview

GNSS Facility supports methods and attributes that are specific to Global Navigation Satellite System receiver hardware device. It supports single/multiple constellations of navigation satellites to provide the position, velocity and time (PVT) data to Radio application.

GNSS Facility extends ESSOR GNSS Device API [ESSOR-Ref 03] with following additional support.

i. Constellation Selection
    a. Request for Supported Constellations
    b. Get Current Constellation

c. Select Current Constellation for getting PVT

Refer 7.1 of [ESSOR-Ref 03] for basic details.

3.6.2.2 Technical Details

Refer 7.1.1 of [ESSOR-Ref 03] for usage context of GNSS facility in SCA specific platform. Radio application can additionally fetch the supported constellations, get the constellation currently in use, and set current constellation to one of the supported constellations.

3.6.3 Services

3.6.3.1 Provide Services

Service Group/Module	Interface Service	Primitives/Operations
LatLong	LatLongDataProducer	readLatLong()
MGRS	MgrsDataProducer	readMgrs()
PNT	PNTProducer	readGeoPosition() readUTMPosition() readMGRSPosition() readTimeTfom() readPNT()
SatInfo	SatInfoProducer	readSatInfo()
Constellation	ConstellationHandling	getSupportedConstellations() getConstellationConfiguration() setConstellationConfiguration()

Table 3.51: GNSS provide services

3.6.3.2 Use Services

Service Group/Module	Interface Service	Primitives/Operations
LatLong	LatLong1PpsConsumer	pushLatLong1Pps()
MGRS	Mgrs1PpsConsumer	pushMgrs1Pps()
PNT	PNTConsumer	pushTimeTfom() pushPNT()
SatInfo	SatInfoConsumer	pushSatInfo()

Table 3.52: GNSS use services

3.6.3.3 State Machines

3.6.3.3.1 GNSS_SM

GNSS_SM is specified as the main state machine followed by GNSS Facility. An instance of GNSS_SM is followed by each GNSS instance.

The following figure is the statechart of GNSS_SM state machine:

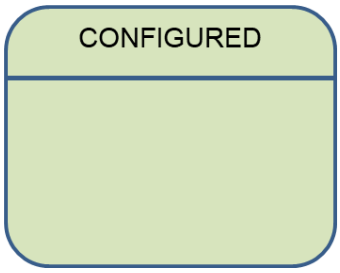

Figure 3.23 : GNSS_SM statechart

3.6.3.3.1.1 States

3.6.3.3.1.2 CONFIGURED

CONFIGURED is specified as the unique state of GNSS, during which it is configured according to the requirements of all the radio application to be supported during the CONFIGURED state.
CONFIGURED is reached by GNSS when it

- Complies with any value specified for a capability or a property,
- Is capable of interacting with radio application according to the service interfaces of its service implementations.

How CONFIGURED is reached is unspecified by the PIM specification, and can be specified by the applied PSM specification.

3.6.3.4 Service groups

3.6.3.4.1 GNSS::LatLong

This group provides producer service primitives for Radio Application to get the PVT data from the GNSS Facility and consumer service primitives for GNSS Facility to push PVT data to Radio Application when a 1pps event occurs.

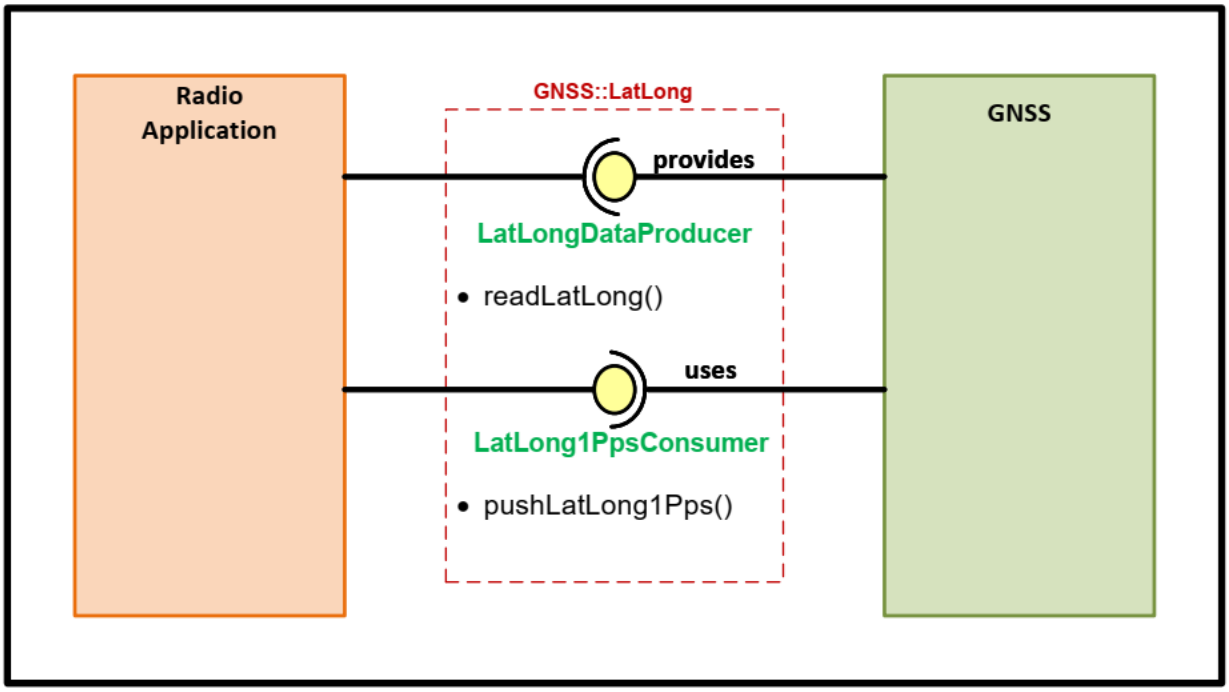

Figure 3.24 : LatLong services group

3.6.3.4.2 GNSS::MGRS

This service group provides producer service primitives for Radio Application to get the MGRS coordinates-based PVT data from the GNSS Facility and consumer service primitives for GNSS Facility to push MGRS coordinates-based PVT data to Radio Application when a 1pps event occurs.

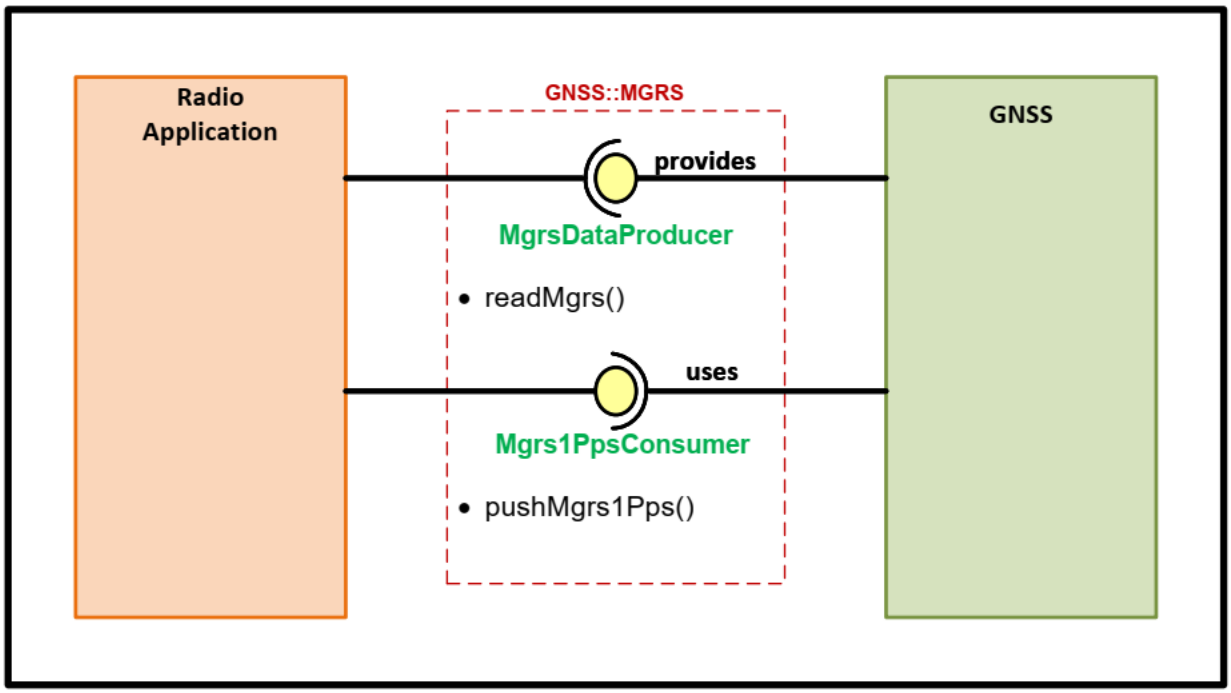

Figure 3.25 : MGRS services group

3.6.3.4.3 GNSS::PNT

This service group provides producer service primitives for Radio Application to get data relative to time, cartographic position and navigation. It also provides consumer service primitives for GNSS Facility to push time and PNT data to radio application.

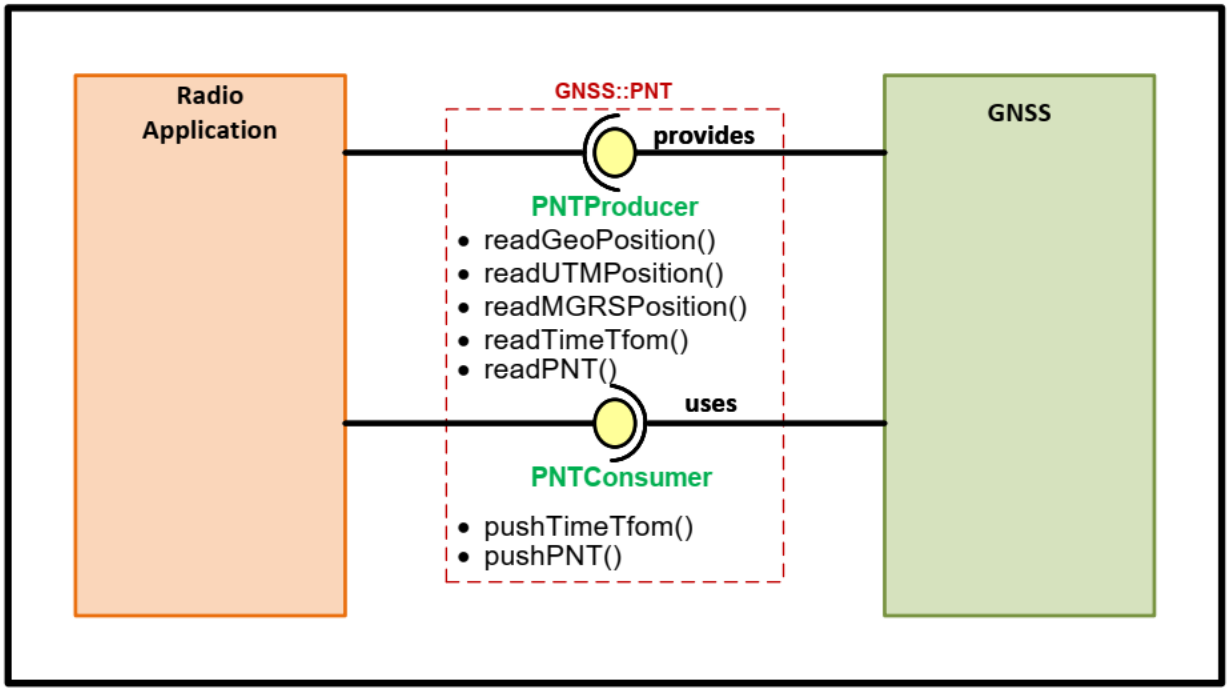

Figure 3.26 : PNT services group

3.6.3.4.4 GNSS::SatInfo

This group provides producer service primitives for Radio Application to get satellites information from the GNSS Facility and consumer service primitives for GNSS Facility to push satellites information to Radio Application.

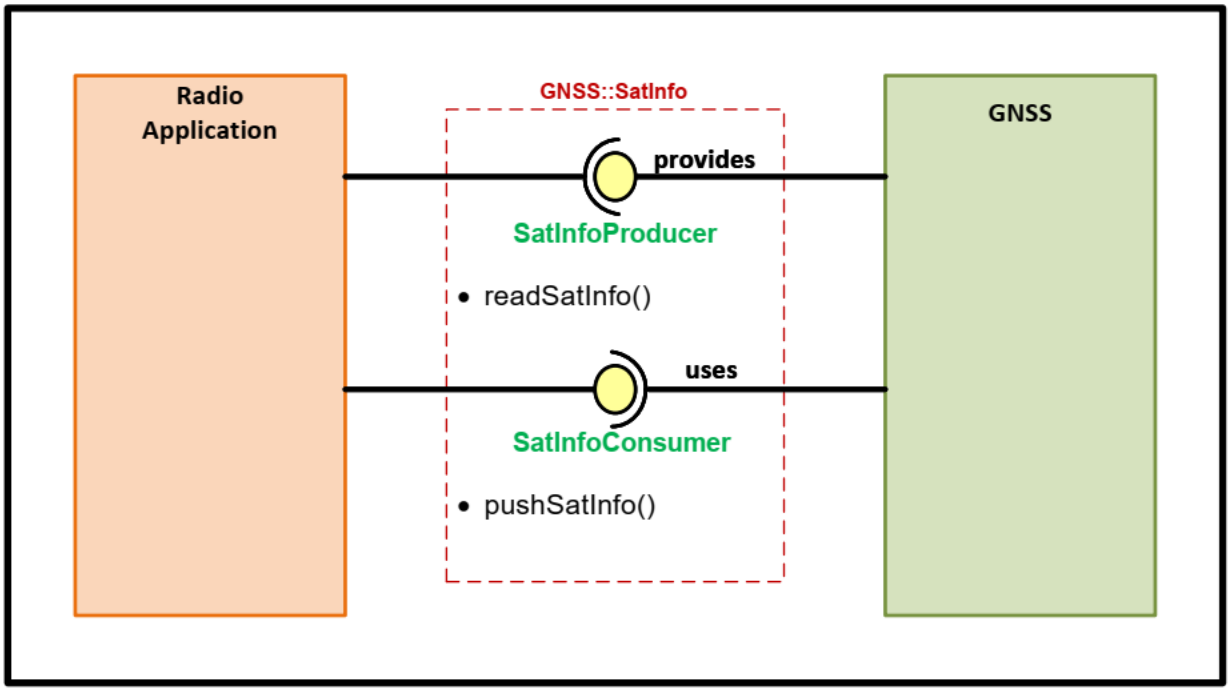

Figure 3.27 : SatInfo services group

3.6.3.4.5 GNSS::Constellation

This service group provides constellation handling service primitives to Radio Application.

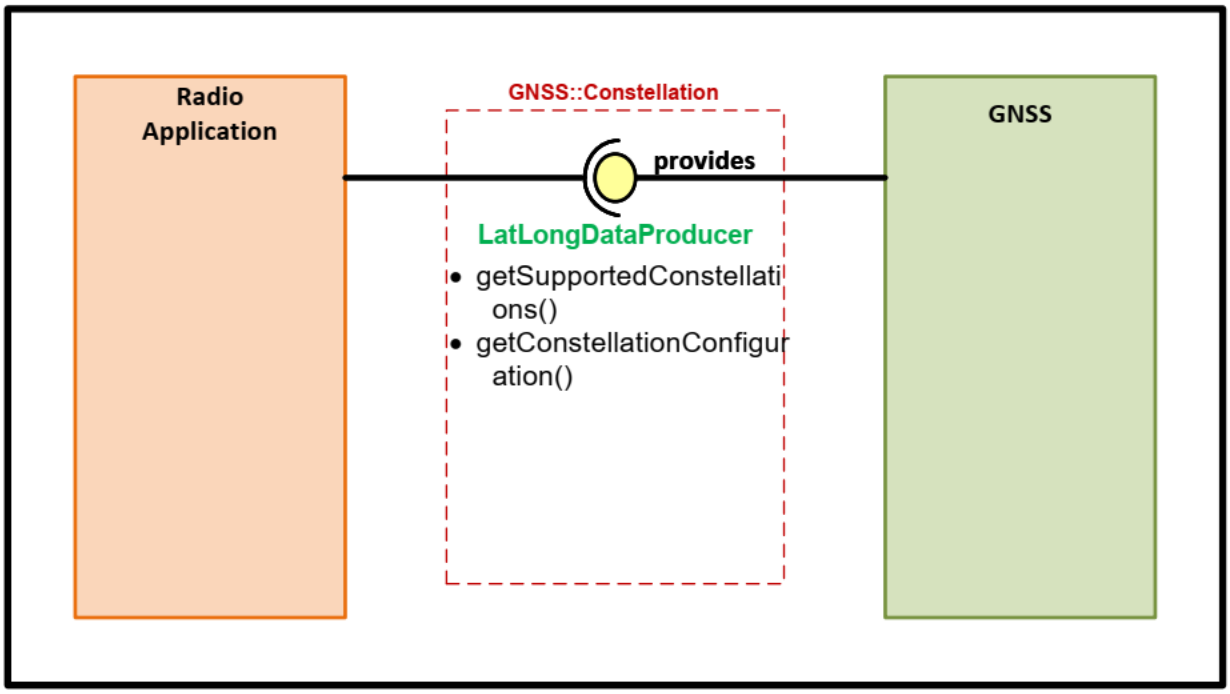

Figure 3.28 : Constellation services group

3.6.4 Service Primitives

This section specifies the primitives of the GNSS facility services. Each declaration of a primitive complies with the Full PIM IDL Profile of WInnF IDL profiles for PIM of SDR

Applications, specified in [WInnF-Ref 05]. The declaration of each primitive also complies with SCA 4.1 Appendix E-1 [JTNC-Ref 30] .

3.6.4.1 GNSS::LatLong::LatLongDataProducer

3.6.4.1.1 readLatLong() operation

3.6.4.1.1.1 Overview
Refer section 7.4.4.1.2.3.1 of [ESSOR-Ref 03]

3.6.4.1.1.2 Signatures
Refer section 7.4.4.1.2.3.1.1 of [ESSOR-Ref 03]

3.6.4.1.1.3 Parameters
Refer sections 7.4.4.1.2.3.1.2 and 7.4.4.1.2.3.1.3 of [ESSOR-Ref 03]

3.6.4.1.1.4 Exceptions
Refer section 7.4.4.1.2.3.1.7 of [ESSOR-Ref 03]

3.6.4.1.1.5 Attributes
None.

3.6.4.1.1.6 Behavior requirements
None.

3.6.4.2 GNSS::LatLong::LatLong1PpsConsumer

3.6.4.2.1 pushLatLong1Pps() operation

3.6.4.2.1.1 Overview
Refer section 7.4.4.1.3.3.1 of [ESSOR-Ref 03]

3.6.4.2.1.2 Signatures
Refer section 7.4.4.1.3.3.1.1 of [ESSOR-Ref 03]

3.6.4.2.1.3 Parameters
Refer section 7.4.4.1.3.3.1.2 of [ESSOR-Ref 03]

3.6.4.2.1.4 Exceptions
Refer section 7.4.4.1.3.3.1.7 of [ESSOR-Ref 03]

3.6.4.2.1.5 Attributes
None.

3.6.4.2.1.6 Behavior requirements
None.

3.6.4.3 GNSS::MGRS::MgrsDataProducer

3.6.4.3.1 readMgrs() operation

3.6.4.3.1.1 Overview
Refer section 7.5.4.1.2.3.1 of [ESSOR-Ref 03]

3.6.4.3.1.2 Signatures
Refer section 7.5.4.1.2.3.1.1 of [ESSOR-Ref 03]

3.6.4.3.1.3 Parameters
Refer section 7.5.4.1.2.3.1.2 of [ESSOR-Ref 03]

3.6.4.3.1.4 Exceptions
Refer section 7.5.4.1.2.3.1.7 of [ESSOR-Ref 03]

3.6.4.3.1.5 Attributes
None.

3.6.4.3.1.6 Behavior requirements
None.

3.6.4.4 GNSS::MGRS::Mgrs1PpsConsumer

3.6.4.4.1 pushMgrs1Pps() operation

3.6.4.4.1.1 Overview
Refer section 7.5.4.1.3.3.1 of [ESSOR-Ref 03]

3.6.4.4.1.2 Signatures
Refer section 7.5.4.1.3.3.1.1 of [ESSOR-Ref 03]

3.6.4.4.1.3 Parameters
Refer section 7.5.4.1.3.3.1.2 of [ESSOR-Ref 03]

3.6.4.4.1.4 Exceptions
Refer section 7.5.4.1.3.3.1.7 of [ESSOR-Ref 03]

3.6.4.4.1.5 Attributes
None.

3.6.4.4.1.6 Behavior requirements
None.

3.6.4.5 GNSS::PNT::PNTProducer

3.6.4.5.1 readGeoPosition() operation

3.6.4.5.1.1 Overview
Refer section 7.6.3.1.3.3.1 of [ESSOR-Ref 03]

3.6.4.5.1.2 Signatures
Refer section 7.6.3.1.3.3.1.1 of [ESSOR-Ref 03]

3.6.4.5.1.3 Parameters
Refer section 7.6.3.1.3.3.1.2 of [ESSOR-Ref 03]

3.6.4.5.1.4 Exceptions
Refer section 7.6.3.1.3.3.1.7 of [ESSOR-Ref 03]

3.6.4.5.1.5 Attributes
None.

3.6.4.5.1.6 Behavior requirements
None.

3.6.4.5.2 readUTMPosition() operation

3.6.4.5.2.1 Overview
Refer section 7.6.3.1.3.3.2 of [ESSOR-Ref 03]

3.6.4.5.2.2 Signatures
Refer section 7.6.3.1.3.3.2.1 of [ESSOR-Ref 03]

3.6.4.5.2.3 Parameters
Refer section 7.6.3.1.3.3.2.2 of [ESSOR-Ref 03]

3.6.4.5.2.4 Exceptions
Refer section 7.6.3.1.3.3.2.7 of [ESSOR-Ref 03]

3.6.4.5.2.5 Attributes
None.

3.6.4.5.2.6 Behavior requirements
None.

3.6.4.5.3 readMGRSPosition() operation

3.6.4.5.3.1 Overview
Refer section 7.6.3.1.3.3.3 of [ESSOR-Ref 03]

3.6.4.5.3.2 Signatures
Refer section 7.6.3.1.3.3.3.1 of [ESSOR-Ref 03]

3.6.4.5.3.3 Parameters
Refer section 7.6.3.1.3.3.3.2 of [ESSOR-Ref 03]

3.6.4.5.3.4 Exceptions

Refer section 7.6.3.1.3.3.3.7 of [ESSOR-Ref 03]

3.6.4.5.3.5 Attributes
None.

3.6.4.5.3.6 Behavior requirements
None.

3.6.4.5.4 readTimeTfom() operation

3.6.4.5.4.1 Overview
Refer section 7.6.3.1.3.3.4 of [ESSOR-Ref 03]

3.6.4.5.4.2 Signatures
Refer section 7.6.3.1.3.3.4.1 of [ESSOR-Ref 03]

3.6.4.5.4.3 Parameters
Refer section 7.6.3.1.3.3.4.2 of [ESSOR-Ref 03]

3.6.4.5.4.4 Exceptions
Refer section 7.6.3.1.3.3.4.7 of [ESSOR-Ref 03]

3.6.4.5.4.5 Attributes
None.

3.6.4.5.4.6 Behavior requirements
None.

3.6.4.5.5 readPNT() operation

3.6.4.5.5.1 Overview
Refer section 7.6.3.1.3.3.5 of [ESSOR-Ref 03]

3.6.4.5.5.2 Signatures
Refer section 7.6.3.1.3.3.5.1 of [ESSOR-Ref 03]

3.6.4.5.5.3 Parameters
Refer section 7.6.3.1.3.3.5.2 of [ESSOR-Ref 03]

3.6.4.5.5.4 Exceptions
Refer section 7.6.3.1.3.3.5.7 of [ESSOR-Ref 03]

3.6.4.5.5.5 Attributes
None.

3.6.4.5.5.6 Behavior requirements
None.

3.6.4.6 GNSS::PNT::PNTConsumer

3.6.4.6.1 pushTimeTfom() operation

3.6.4.6.1.1 Overview
Refer section 7.6.3.1.4.2.1 of [ESSOR-Ref 03]

3.6.4.6.1.2 Signatures
Refer section 7.6.3.1.4.2.1.1 of [ESSOR-Ref 03]

3.6.4.6.1.3 Parameters
Refer section 7.6.3.1.4.2.1.2 of [ESSOR-Ref 03]

3.6.4.6.1.4 Exceptions
Refer section 7.6.3.1.4.2.1.7 of [ESSOR-Ref 03]

3.6.4.6.1.5 Attributes
None.

3.6.4.6.1.6 Behavior requirements
None.

3.6.4.6.2 pushPNT() operation

3.6.4.6.2.1 Overview
Refer section 7.6.3.1.4.2.2 of [ESSOR-Ref 03]

3.6.4.6.2.2 Signatures
Refer section 7.6.3.1.4.2.2.1 of [ESSOR-Ref 03]

3.6.4.6.2.3 Parameters
Refer section 7.6.3.1.4.2.2.2 of [ESSOR-Ref 03]

3.6.4.6.2.4 Exceptions
Refer section 7.6.3.1.4.2.2.7 of [ESSOR-Ref 03]

3.6.4.6.2.5 Attributes
None.

3.6.4.6.2.6 Behavior requirements
None.

3.6.4.7 GNSS::SatInfo::SatInfoProducer

3.6.4.7.1 readSatInfo() operation

3.6.4.7.1.1 Overview
Refer section 7.7.3.2.6.1 of [ESSOR-Ref 03]

3.6.4.7.1.2 Signatures
Refer section 7.7.3.2.6.1.1 of [ESSOR-Ref 03]

3.6.4.7.1.3 Parameters
Refer section 7.7.3.2.6.1.2 of [ESSOR-Ref 03]

3.6.4.7.1.4 Exceptions
Refer section 7.7.3.2.6.1.7 of [ESSOR-Ref 03]

3.6.4.7.1.5 Attributes
None.

3.6.4.7.1.6 Behavior requirements
None.

3.6.4.8 GNSS::SatInfo::SatInfoConsumer

3.6.4.8.1 pushSatInfo() operation

3.6.4.8.1.1 Overview
Refer section 7.7.3.3.6.1 of [ESSOR-Ref 03]

3.6.4.8.1.2 Signatures
Refer section 7.7.3.3.6.1.1 of [ESSOR-Ref 03]

3.6.4.8.1.3 Parameters
Refer section 7.7.3.3.6.1.2 of [ESSOR-Ref 03]

3.6.4.8.1.4 Exceptions
Refer section 7.7.3.3.6.1.7 of [ESSOR-Ref 03]

3.6.4.8.1.5 Attributes
None

3.6.4.8.1.6 Behavior requirements
None

3.6.4.9 GNSS::Constellation::ConstellationHandling

3.6.4.9.1 getSupportedConstellations() operation

3.6.4.9.1.1 Overview
This operation provides the radio application with the list of supported constellations.

3.6.4.9.1.2 Signatures
void getSupportedConstellations (out Constellations supportedConstellations);

3.6.4.9.1.3 Parameters

Name	Type	Description	Range/Values
supportedConstellations	Constellations	List of ConstellationType populated with the constellations supported by the Facility.	Each element of list is a ConstellationType. If only one constellation is supported, then list shall have one element. If multiple constellations are supported, then the list shall have multiple elements specifying all the supported constellations.

Table 3.53: GNSS getSupportedConstellations parameters

3.6.4.9.1.4 Exceptions

None.

3.6.4.9.1.5 Attributes

None.

3.6.4.9.1.6 Behavior requirements

None.

3.6.4.9.2 getConstellationConfiguration() operation

3.6.4.9.2.1 Overview

This operation provides the radio application with the constellation(s) currently in use.

3.6.4.9.2.2 Signatures

void getConstellationConfiguration(out Constellations specificConstellations);

3.6.4.9.2.3 Parameters

Name	Type	Description	Range/Values
specificConstellations	Constellations	List of ConstellationType populated with the constellation or combination of constellations currently in use.	Each element of list is a ConstellationType. If one constellation is currently in use then list shall have one element. If a combination of constellations is in use then the list shall have multiple elements specifying the combination in use.

Table 3.54: GNSS getConstellationConfiguration parameters

3.6.4.9.2.4 Exceptions

None.

3.6.4.9.2.5 Attributes

None.

3.6.4.9.2.6 Behavior requirements

None.

3.6.4.9.3 setConstellationConfiguration() operation

3.6.4.9.3.1 Overview

This operation allows the radio application to change the constellation(s) currently in use to the values specified by the input the parameter.

3.6.4.9.3.2 Signatures

void setConstellationConfiguration(in Constellations specificConstellations);

3.6.4.9.3.3 Parameters

Name	Type	Description	Range/Values
specificConstellations	Constellations	List of ConstellationType populated with the required constellation types.	Each element of list is a ConstellationType. If one constellation is to be configured then list shall have one element. If a combination of constellations is to be configured then the list shall have multiple elements specifying the desired combination.

Table 3.55: GNSS setConstellationConfiguration parameters

3.6.4.9.3.4 Exceptions

This operation can raise exception of type InvalidCostellationConfiguration if an invalid combination of constellations is specified in the input parameter.

3.6.4.9.3.5 Attributes

None.

3.6.4.9.3.6 Behavior requirements

None.

3.6.4.10 Exceptions

Name	Applies To	Description
InvalidCostellationConfiguration	setConstellationConfiguration	The specificConstellations parameter contains an invalid combination of supported constellations.

Table 3.56: GNSS Exceptions

3.6.4.11 Type

The specification complies with the Full PIM IDL Profile of WInnF IDL profiles for PIM of SDR Applications, specified in [WInnF-Ref 05].

3.6.4.11.1 ConstellationType

```
typedef enum {
        Gps,
        Navic,
        Glonass,
        Galileo,
        Beidou,
        Qzss
} ConstellationType;
```

3.6.4.11.2 Constellations

typedef sequence <ConstellationType> Constellations;

3.6.5 GNSS Attributes

3.6.5.1 Capabilities

Name	Type	Description	Range/Values
SUPPORTED_CONSTELLATIONS	Constellations	A sequence of all supported constellation types.	Constellations object which is a sequence of ConstellationType whose elements specify which all constellations are supported. If the Facility supports GPS, NAVIC, GLONASS then this object shall have three elements populated with values Gps, Navic and Glonass.
MULTI_CONSTELLATION_SUPPORTED	boolean	Indicates whether combination of constellations can be configured or not.	TRUE value indicates combination of constellations can be configured. FALSE value indicates combination of constellations cannot be configured.
AGPS_SUPPORTED	boolean	Indicates whether Assisted GPS is supported or not.	TRUE value indicates Assisted GPS support is present in the Facility. FALSE value indicates Assisted GPS support is not present in the Facility.

NAVIC_RS_SUPPORTED	Boolean	Indicates whether NAVIC RS is supported or not.	TRUE value indicates NAVIC RS support is present in the Facility. FALSE value indicates NAVIC RS support is not present in the Facility.
INTEGRITY_SUPPORTED	boolean	Indicates whether position Integrity is supported or not. Note: Integrity is supported only the underlying GNSS device SBAS support (GAGAN in India, WASS in USA)	TRUE value indicates position integrity support is present in the Facility. FALSE value indicates position integrity support is not present in the Facility.
PPS_SUPPORTED	boolean	Indicates whether PPS is available as discrete output or not.	TRUE indicates PPS available as discrete output. FALSE indicates PPS not available as discrete output.
MGRS_SUPPORTED	boolean	Indicates whether MGRS output is available or not.	TRUE indicates MGRS output is available. FALSE indicates MGRS output is not available.
PNT_SUPPORTED	boolean	Indicates whether PNT support is available or not.	TRUE indicates PNT support is available. FALSE indicates PNT support is not available.
SATINFO_SUPPORTED	boolean	Indicates whether SatInfo support is available or not.	TRUE indicates SatInfo support is available. FALSE indicates SatInfo support is not available.

Table 3.57: GNSS general capabilities

3.6.5.2 Properties

Configurable parameters whose value needs to be set before the facility goes into CONFIGURED state.

Name	Type	Description	Range/Values
ENABLE_MULTI_CONSTELLATIONS	boolean	Indicates whether multi constellation support is currently enabled or not. MULTI_CONSTELLATION_SUPPORTED should be TRUE.	TRUE value indicates multi constellation support is currently enabled. FALSE value indicates multi constellation support is currently disabled.

ENABLE_INTEGRITY_SUPPORT	boolean	Indicates whether Integrity support is currently enabled or not. INTEGRITY_SUPPORTED should be TRUE.	TRUE value indicates position integrity support is currently enabled FALSE value indicates position integrity support is currently disabled
ENABLE_AGPS	boolean	Indicates whether Assisted GPS is currently enabled or not. AGPS_SUPPORTED should be TRUE.	TRUE value indicates Assisted GPS support is currently enabled. FALSE value indicates Assisted GPS support is currently disabled
ENABLE_NAVIC_RS	Boolean	Indicates whether NAVIC RS is currently enabled or not. NAVIC_RS_SUPPORTED should be TRUE.	TRUE value indicates NAVIC RS support is currently enabled. FALSE value indicates NAVIC RS support is currently disabled
ENABLE_MGRS	boolean	Indicates whether MGRS output is available or not. MGRS_SUPPORTED should be TRUE.	TRUE indicates MGRS position can be provided FALSE indicates otherwise.

Table 3.58: GNSS properties

3.6.5.3 Variables

None

3.7 Transceiver Facility

3.7.1 Approach

3.7.1.1 Candidate API Set

- WInnF Transceiver Facility PIM - [WInnF-Ref 15]
- WInnF Transceiver Facility SCA PSM - [WInnF-Ref 16]
- WInnF Transceiver Facility Native C++ PSM - [WInnF-Ref 17]
- WInnF Transceiver Facility FPGA PSM - [WInnF-Ref 18]
- WInnF Transceiver Facility Absolute Time Use case - [WInnF-Ref 19]

3.7.1.2 Selected API Set

- WInnF Transceiver Facility PIM - [WInnF-Ref 15]

- WInnF Transceiver Facility SCA PSM - [WInnF-Ref 16]
- WInnF Transceiver Facility Native C++ PSM - [WInnF-Ref 17]
- WInnF Transceiver Facility FPGA PSM - [WInnF-Ref 18]
- WInnF Transceiver Facility Absolute Time Use case - [WInnF-Ref 19]

3.7.1.3 Modification

Nil

3.7.1.4 Rationale

WInnF Transceiver Facility documents follows the facility framework that is chosen for the IRSA standard. It caters for both full duplex and half duplex communications. This standard is found to be comprehensive and almost all conceivable scenarios are addressed by the list of APIs. Though certain APIs may not be of immediate interest in the Indian context, they are retailed anticipating futuristic needs. This would not be an issue sine a platform needs to support an API if and only if the corresponding facility is offered by the platform. Hence no modification is warranted. The document, at places, could be described as cryptic. Hence, the following points are put on record for the purpose of articulation.

1. It is assumed that the Radio Applications that directly access TXCVR facility reside entirely on the black side.
2. The WInnF standard considers Channelization properties as part of ***TuningPreset***. This is to be considered as a capability attribute for the given instance of TXCVR facility. Radio Application needs to respect these presets and need to choose a preset from the available set, for each instance of ***InitialTuning***.
3. The WInnF standard considers the provision for queuing of tuning and burst scheduling calls. This may have implications in platform implementation. Nevertheless, these APIs may be retained and could be made applicable to that class of transceivers that support this feature.
4. Absolute time TXCVRs may be considered as a separate class (as is the case with WInnF). The related APIs may be made applicable only to that class.
5. All services given in the PIM specifications may not be compulsorily mapped into all PSM documents. For example, In FPGA PSM, the “PushTxPacket()” and “PushRxPacket()” services are replaced by “PushTxBlock()” and “PushRxBlock()” services respectively. It supports only streamed transfer, with flow control signalling.

On Half Duplex Operation & Burst Scheduling

6. Radios, by default, will be in Rx state (idle or active Rx). Rx bursts are unscheduled. Burst scheduling would be applicable only to Tx burst.
7. A Tx burst would be scheduled by the Radio Application, based on a Tx requirement. It could be ***DirectCreation***, ***RelativeCreation***, ***AbsoluteCreation*** or ***StrobedCreation***.

8. The TXCVR would switch to Tx mode during the Tx burst activation and switch back to Rx mode during the Tx burst termination. No separate API or service is requied for this purpose. It can be subsumed in the Tx burst processing.
9. No Tx is permitted during an active receive.
10. It is felt that the ***StrobedCreation*** method offers the possibility of using existing method of accessing TXCVR services with minimal changes. The ***ApplicationStrobe*** service can be used for this purpose.
11. ***StrobedCreation*** or ***DirectCreation*** may be used for facilitating continuous transmission.

3.7.1.5 Conclusion

For Transceiver Facility, it is proposed to adopt Transceiver Facility PIM Specification [WInnF-Ref 15] with corresponding PSM Documents as listed below.

1. Transceiver Facility SCA PSM specification - [WInnF-Ref 16]
2. Transceiver Facility Native C++ PSM specification - [WInnF-Ref 17]
3. Transceiver Facility FPGA PSM Specification - [WInnF-Ref 18]
4. Transceiver Facility Absolute Time Use Case Document - [WInnF-Ref 19]

No modifications are envisaged at this point of time. Modification may be made in future, if deemed necessary, based on feedback from reference implementation trials.

3.8 Time Service Facility

3.8.1 Approach

3.8.1.1 Candidate API Set

- WInnF Time Service Facility [WInnF-Ref 11]
- WInnF Time Service Facility SCA PSM - [WInnF-Ref 12]
- WInnF Time Service Facility Native C++ PSM - [WInnF-Ref 13]
- WInnF Time Service Facility FPGA PSM - [WInnF-Ref 14]

3.8.1.2 Selected API Set

- WInnF Time Service Facility [WInnF-Ref 11]
- WInnF Time Service Facility SCA PSM - [WInnF-Ref 12]
- WInnF Time Service Facility Native C++ PSM - [WInnF-Ref 13]
- WInnF Time Service Facility FPGA PSM - [WInnF-Ref 14]

3.8.1.3 Modification

NIL

3.8.1.4 Rationale

WInnF Time Service Facility documents comply the facility framework that is chosen for the IRSA standard. It allows radio platforms to provide radio applications with knowledge

of time. It supports the portability of radio applications and hospitality of radio platforms, through a generic specification of the time service capability, with the API and associated attributes. This standard is considered exhaustive, and almost all eligible scenarios are covered by the list of APIs. The document mainly supports APIs for Provide services, Use services, and Service level conformance. All APIs are retained anticipating current and futuristic requirements. Hence, no modification is warranted.

3.8.1.5 Conclusion

For Time Service Facility It is proposed to adopt Time Service Facility PIM Specification [WInnF-Ref 11] with corresponding PSM Documents as listed below.

1. Time Service Facility SCA PSM specification - [WInnF-Ref 12]
2. Time Service Facility Native C++ PSM Specification - [WInnF-Ref 13]
3. Time Service Facility FPGA PSM Specification - [WInnF-Ref 14]

No modifications are currently being considered. Modifications may be made in the future, if deemed necessary, based on input received during reference implementation trials and testing.

3.9 Serial Port Facility

3.9.1 Approach

3.9.1.1 Candidate API Set

- ESSOR Serial Port Device API - [ESSOR-Ref 03]
- JTNC Serial Port Device API - [JTNC-Ref 18]

3.9.1.2 Selected API Set

ESSOR Serial Port Device API - [ESSOR-Ref 03]

3.9.1.3 Modification

ESSOR Serial Port Facility API specification was in line with SCA and to create PIM in Facility Framework, suitable modifications are required.

3.9.1.4 Rationale

Above referenced API is sufficient to cater requirement of platform and waveform components. Its specification was in line with SCA and to create PIM in Facility Framework, suitable modifications are required.

3.9.1.5 Conclusion

ESSOR API is adopted and provided in the form of PIM specification as per Principles for WInnF Facility Standards [WInnF-Ref 09]. PSM Documents (SCA, Native C++, FPGA) corresponding to this facility PIM specification as applicable will be released after their reference implementation.

3.9.2 Introduction

3.9.2.1 Overview

Refer 5.1 of [ESSOR-Ref 03]

3.9.2.2 Definitions

None.

3.9.2.3 Technical Details

Refer 5.1.1 of [ESSOR-Ref 03] for usage context of Serial Port facility in SCA specific platform.

3.9.3 Services

3.9.3.1 Provide Services

The following table lists the *provide services* of the API (used by a radio application or radio platform and provided by Serial Port facility):

Service Group/Module	Interface Service	Primitives/Operations
SerialPortData	SerialPortPacketProducer Packet::PayloadControl, Packet::FlowSignals, Packet::EmptySignals, DeviceIo::DeviceIoSignals	setMaxPayloadSize() setMinPayloadSize() setDesiredPayloadSize() setMinOverrideTimeout() signalResume() signalEmpty() setCts()
	SerialPortPacketConsumer Packet::FlowControl Packet::FlowOctetStream Packet::EmptyControl Packet::PayloadStatus DeviceIo::DeviceIoControl	pushPacket() getMaxPayloadSize() getMinPayloadSize() getDesiredPayloadSize() getMinOverrideTimeout() enableFlowResumeSignals() spaceAvailable() enableEmptySignals() enableRtsCts() setRts()
SerialPortManagement	SerialPortConfiguration	setSynchronizationConfig() getSynchronizationConfig() getSynchronizationConfigsSupported() setLoopback() getLoopback() setProtocolType() getProtocolType()
	SerialMessageControl	rxActive()

Service Group/Module	Interface Service	Primitives/Operations
	DevMsgCtl::DeviceMessageControl	txActive() abortTx()
	DynamicBaud	setBaudRate() getEffectiveBaudRate()
	SerialPortTxRxDataPolarity	setTxDataPolarity() getTxDataPolarity() setRxDataPolarity() getRxDataPolarity()
	SerialPortTxRxClockManagement	setTxClockStatus() getTxClockStatus() setRxClockStatus() getRxClockStatus()
	SerialPortDataRateSynchManagement	setDataRateSynch() getDataRateSynch ()
	SerialPortDcdManagement	setDcd()
	SerialPortDtrDsrManagement	setDsr() getDtr()
	SerialPortHwRtsCtsManagement	setHwCts() getHwRts()
SerialAsync	Async	setAsyncConfig() getAsyncConfig()
SerialSync	SyncRaw	setSyncRawConfig() getSyncRawConfig()
	SyncHdlc	setSyncHdlcConfig() getSyncHdlcConfig()

Table 3.59: Serial Port Provide Services

3.9.3.2 Use Services

The following table lists the *used services* of the API (provided by a radio application or radio platform and used by Serial Port Facility

Service Group/Module	Interface Service	Primitives/Operations
SerialPortData	SerialPortPacketProducer Packet::PayloadControl, Packet::FlowSignals, Packet::EmptySignals, DeviceIo::DeviceIoSignals	setMaxPayloadSize() setMinPayloadSize() setDesiredPayloadSize() setMinOverrideTimeout() signalResume() signalEmpty() setCts()
	SerialPortPacketConsumer Packet::FlowControl Packet::FlowOctetStream	pushPacket() getMaxPayloadSize() getMinPayloadSize()

	Packet::EmptyControl Packet::PayloadStatus DeviceIo::DeviceIoControl	getDesiredPayloadSize() getMinOverrideTimeout() enableFlowResumeSignals() spaceAvailable() enableEmptySignals() enableRtsCts() setRts()
SerialPortNotification	SerialPortNotifyTxRxClock	notifyTxClockStatus() notifyRxClockStatus()
	SerialPortNotifyDataRateSynch	notifyDataRateSynch()
	SerialPortNotifyDtr	notifyDtr()
	SerialPortNotifyHwRts	notifyHwRts()

Table 3.60: Serial Port use services

3.9.3.3 State Machines

3.9.3.3.1 SerialPort_SM

SerialPort_SM is specified as the main state machine followed by Serial Port Facility. An instance of SerialPort_SM is followed by each SerialPort instance.

The following figure is the statechart of SerialPort_SM state machine:

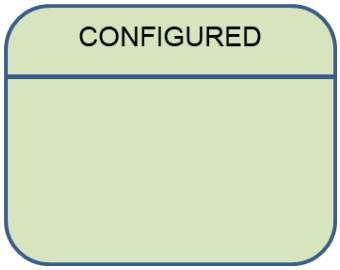

Figure 3.29 : SerialPort_SM statechart

3.9.3.3.1.1 States

3.9.3.3.1.2 CONFIGURED

CONFIGURED is specified as the unique state of SerialPort, during which it is configured according to the requirements of all the radio application to be supported during the CONFIGURED state.
CONFIGURED is reached by SerialPort when it

- Complies with any value specified for a capability or a property,
- Is capable of interacting with radio application according to the service interfaces of its service implementations.

How CONFIGURED is reached is unspecified by the PIM specification, and can be specified by the applied PSM specification

3.9.3.4 Service groups

3.9.3.4.1 SerialPort::SerialPortData

This service group contains interfaces SerialPortPacketProducer which inherits from PayloadControl, FlowSignals, EmptySignals interfaces from JTRS Packet API and DeviceIoSignals from JTRS DeviceIo API. It also contains SerialPortPacketConsumer interface which inherits from FlowControl, FlowOctetStream, EmptyControl, PayloadStatus interfaces from JTRS Packet API and DeviceIoControl from JTRS DeviceIo API.

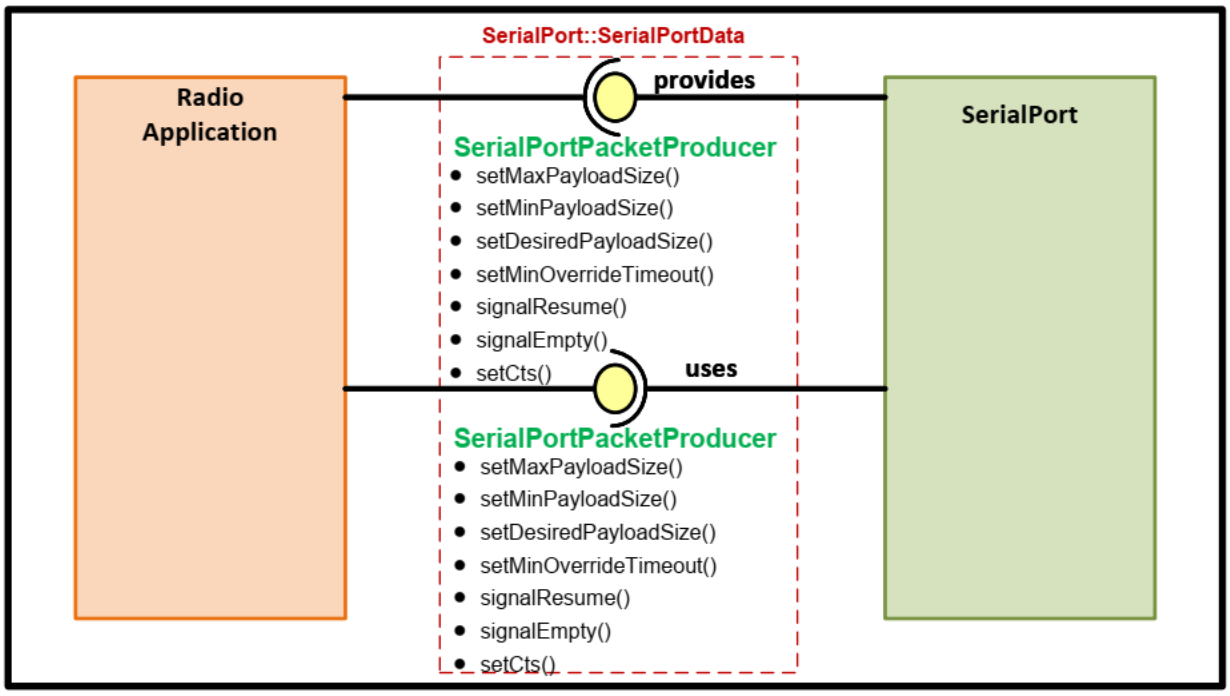

Figure 3.30 : SerialPortData (SerialPortPacketProducer) services group

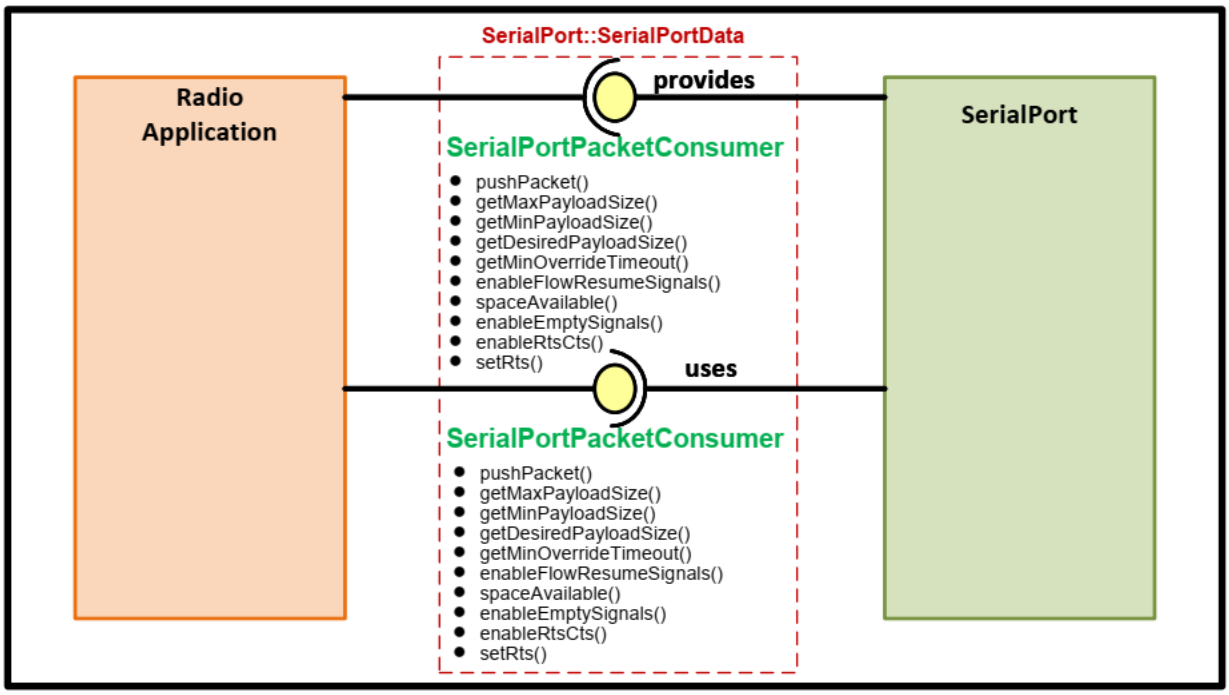

Figure 3.31 : SerialPortData (SerialPortPacketConsumer) services group

3.9.3.4.2 SerialPort::SerialPortManagement

This service group contains the following interfaces.

1. SerialPortConfiguration interface having primitives pertaining to synchronization configuration, loopback and protocol type. SerialMessageControl interface which inherits from JTRS DeviceMessageControl API. DynamicBaud interface having primitives pertaining to baud configuration.

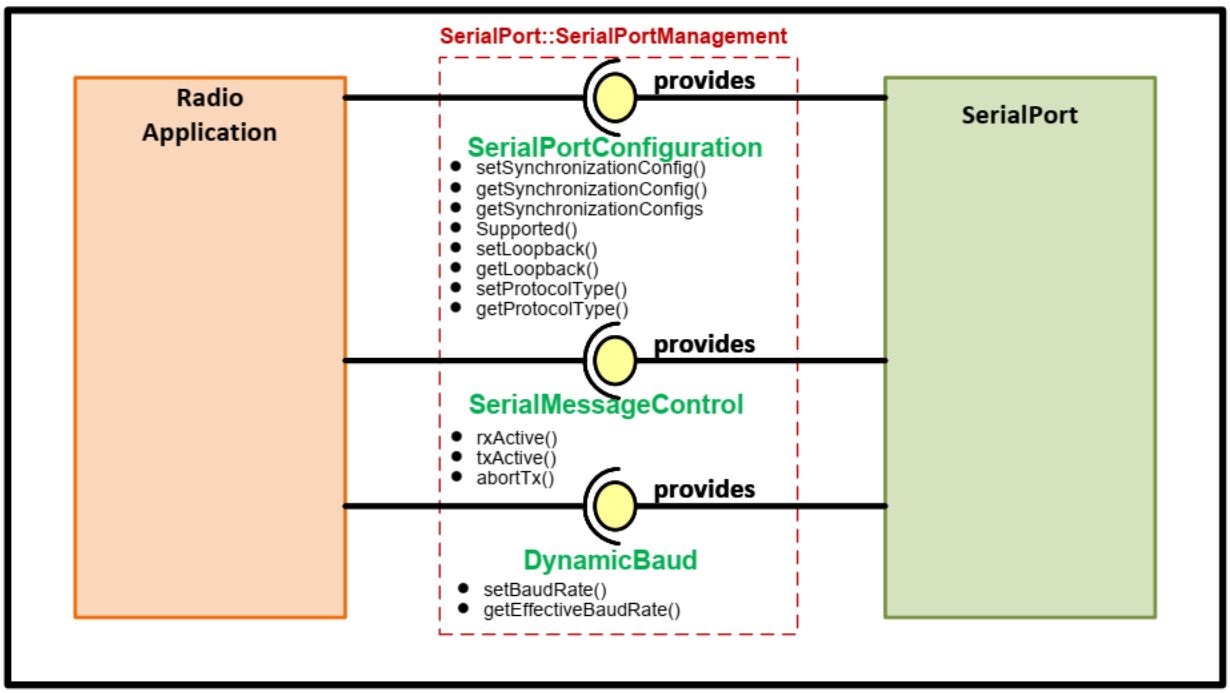

Figure 3.32 : SerialPortManagement-1 services group

2. Interfaces SerialPortTxRxDataPolarity, SerialPortTxRxClockManagement, SerialPortTxRxClockManagement, SerialPortDataRateSynchManagement, SerialPortDcdManagement, SerialPortDtrDsrManagement and SerialPortHwRtsCtsManagement. These interfaces provide primitives pertaining data polarity, clock status, data rate synchronization, DCD, DSR, DTR and hardware RTS/CTS

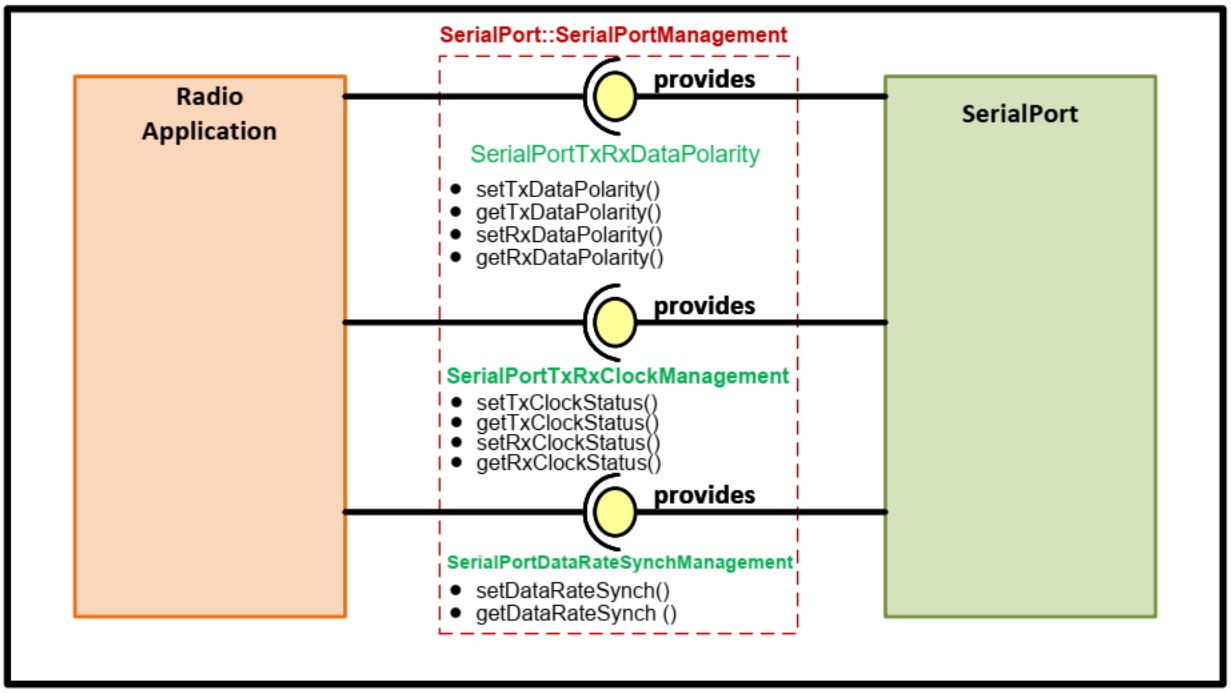

Figure 3.33 : SerialPortManagement-2 services group

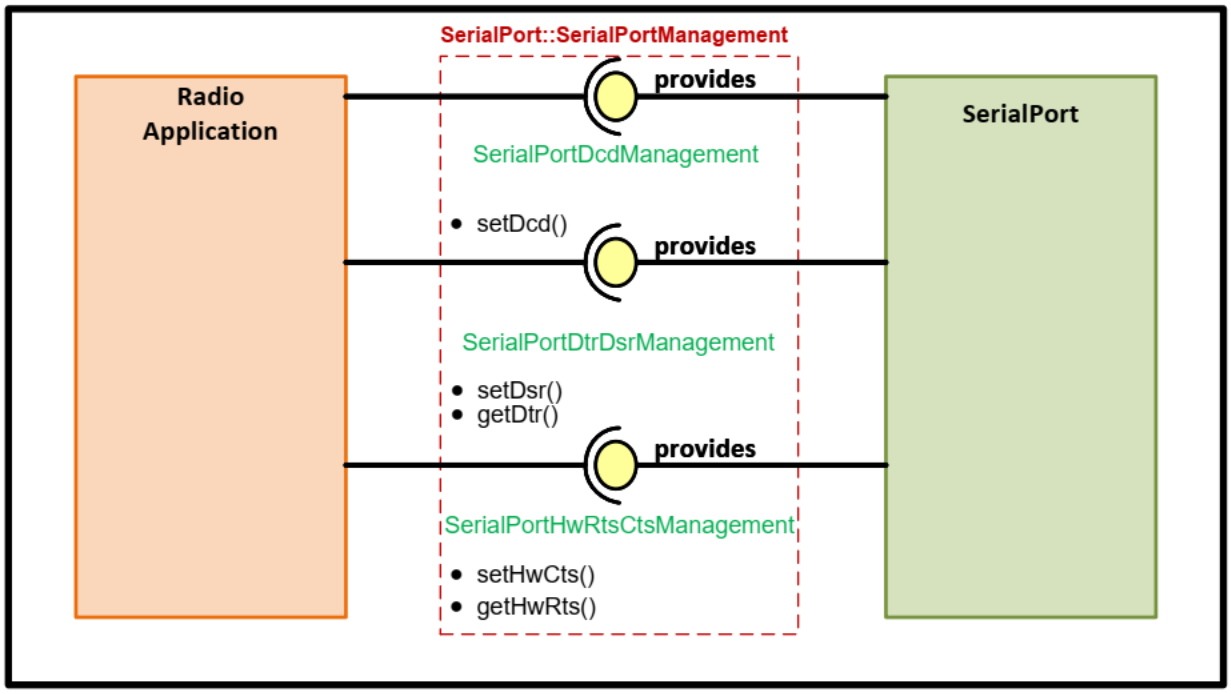

Figure 3.34 : SerialPortManagement-3 services group

3.9.3.4.3 SerialPort::SerialAsync

This service group contains interface Async having primitives for asynchronous serial configuration.

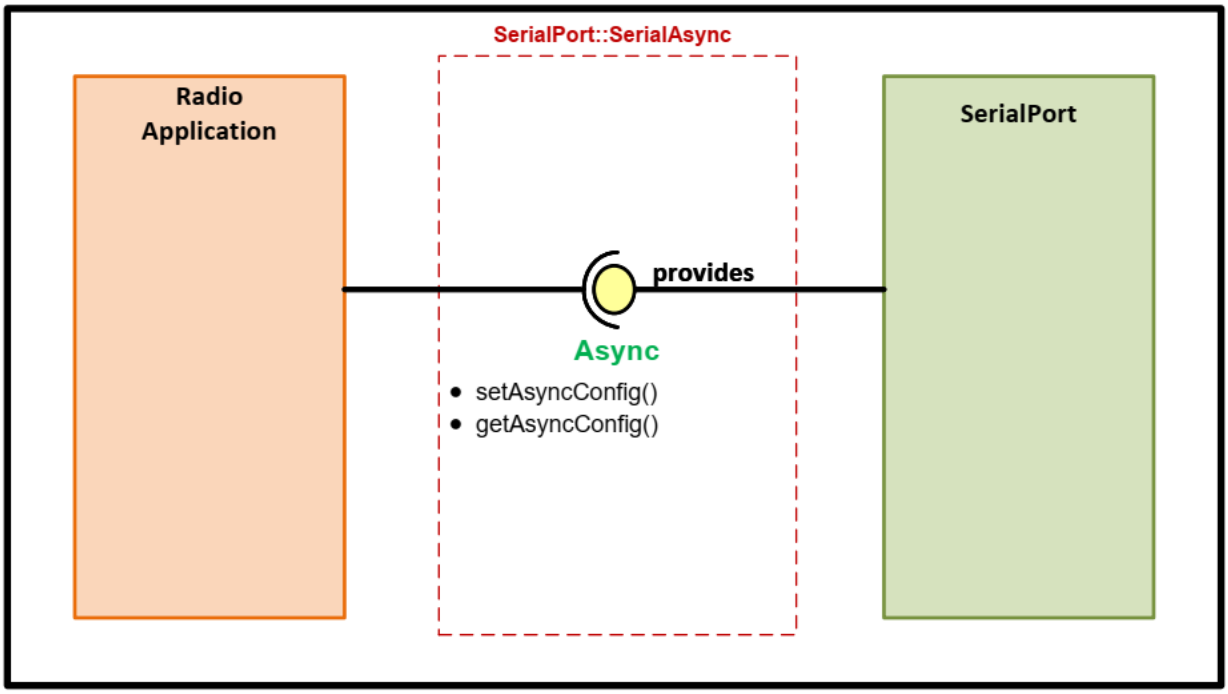

Figure 3.35 : SerialAsync services group

3.9.3.4.4 SerialPort::Sync

This service group contains interfaces SyncRaw and SyncHdlc having primitives

pertaining to the synchronous raw serial configuration and to configure the synchronous serial port using HDLC.

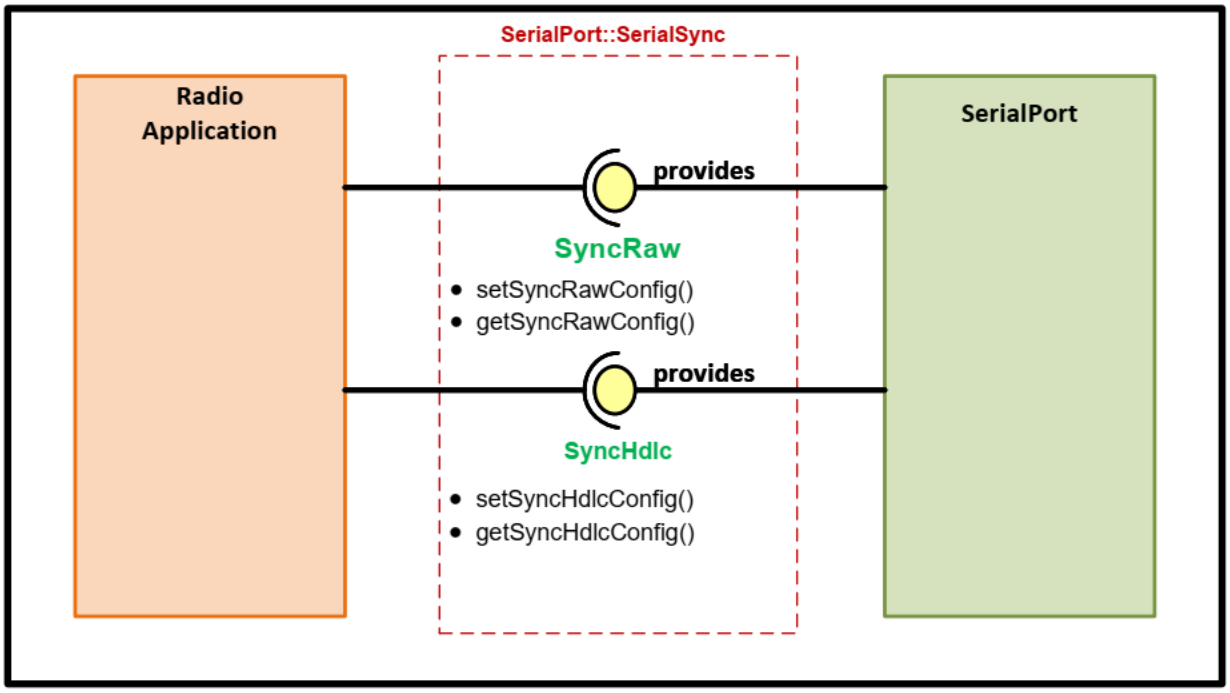

Figure 3.36 : SerialSync services group

3.9.3.4.5 SerialPort::SerialPortNotification

This service group contains interfaces SerialPortNotifyTxRxClock, SerialPortNotifyDataRateSynch, SerialPortNotifyDtr and SerialPortNotifyHwRts having notification primitives pertaining to Tx/Rx clock status, data rate synchronization, DTR and hardware RTS.

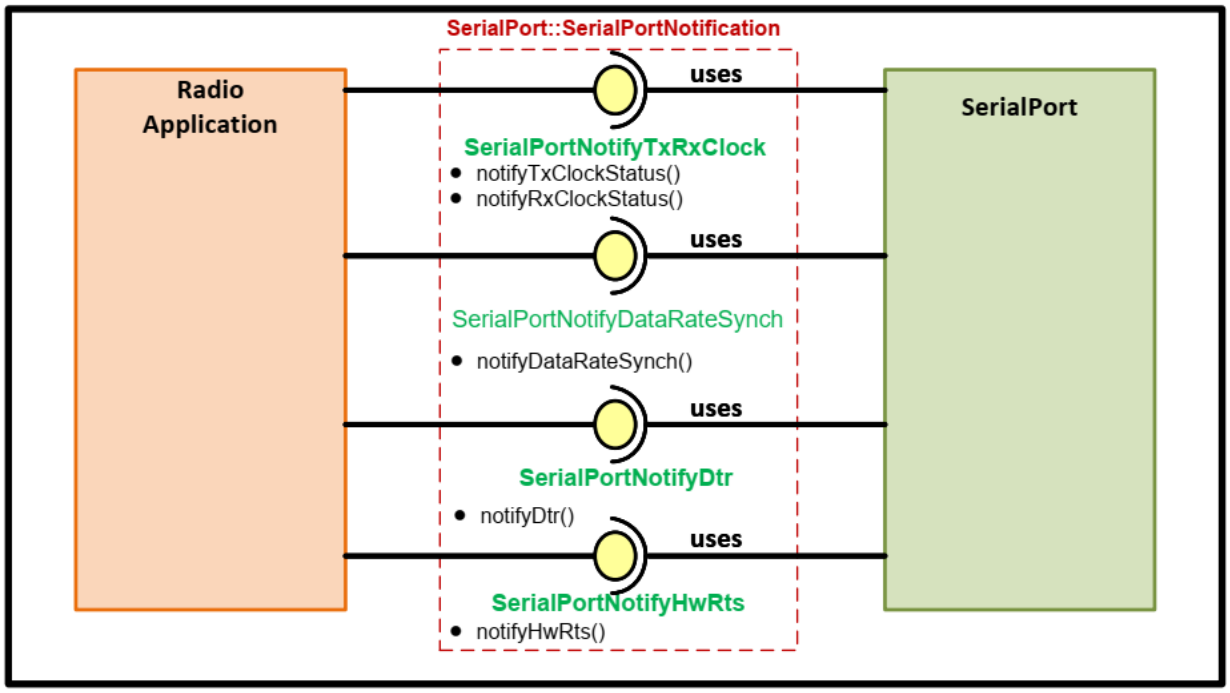

Figure 3.37 : SerialPortNotification services group

3.9.4 Service Primitives

This section specifies the primitives of the Serial Port facility services. Each declaration of a primitive complies with the Full PIM IDL Profile of WInnF IDL profiles for PIM of SDR Applications, specified in [WInnF-Ref 05]. The declaration of each primitive also complies with SCA 4.1 Appendix E-1 [JTNC-Ref 30] .

3.9.4.1 SerialPort::SerialPortData::SerialPortPacketProducer

Refer to sections 3.2.2.3.2, 3.2.2.3.3, 3.2.2.3.4 and 3.2.2.1.2 of this document for the primitives inherited from Packet API interfaces and DeviceIo API interfaces viz. Packet::PayloadControl, Packet::FlowSignals, Packet::EmptySignals, and DeviceIo::DeviceIoSignals respectively.

3.9.4.2 SerialPort::SerialPortData::SerialPortPacketConsumer

Refer to sections 3.2.2.3.5, 3.2.2.3.6, 3.2.2.3.7, 3.2.2.3.1 and 3.2.2.1.1 of this document for the primitives inherited from Packet API interfaces and DeviceIo API interfaces viz. Packet::FlowControl, Packet::FlowOctetStream, Packet::EmptyControl, Packet::PayloadStatus, DeviceIo::DeviceIoControl respectively.

3.9.4.3 SerialPort::SerialPortManagement::SerialPortConfiguration

3.9.4.3.1 setSynchronizationConfig() operation

3.9.4.3.1.1 Overview

Refer section 5.3.1.5.3.1 of [ESSOR-Ref 03]

3.9.4.3.1.2 Signatures
Refer section 5.3.1.5.3.1.1 of [ESSOR-Ref 03]

3.9.4.3.1.3 Parameters
Refer section 5.3.1.5.3.1.2 of [ESSOR-Ref 03]

3.9.4.3.1.4 Exceptions
Refer section 5.3.1.5.3.1.6 of [ESSOR-Ref 03]

3.9.4.3.1.5 Attributes
None.

3.9.4.3.1.6 Behavior requirements
None

3.9.4.3.2 getSynchronizationConfig() operation

3.9.4.3.2.1 Overview
Refer section 5.3.1.5.3.2 of [ESSOR-Ref 03]

3.9.4.3.2.2 Signatures
Refer section 5.3.1.5.3.2.1 of [ESSOR-Ref 03]

3.9.4.3.2.3 Parameters
Refer section 5.3.1.5.3.2.2 of [ESSOR-Ref 03]

3.9.4.3.2.4 Exceptions
Refer section 5.3.1.5.3.2.6 of [ESSOR-Ref 03]

3.9.4.3.2.5 Attributes
None.

3.9.4.3.2.6 Behavior requirements
None

3.9.4.3.3 getSynchronizationConfigsSupported() operation

3.9.4.3.3.1 Overview
Refer section 5.3.1.5.3.3 of [ESSOR-Ref 03]

3.9.4.3.3.2 Signatures
Refer section 5.3.1.5.3.3.1 of [ESSOR-Ref 03]

3.9.4.3.3.3 Parameters
Refer section 5.3.1.5.3.3.2 of [ESSOR-Ref 03]

3.9.4.3.3.4 Exceptions

Refer section 5.3.1.5.3.3.6 of [ESSOR-Ref 03]

3.9.4.3.3.5 Attributes
None.

3.9.4.3.3.6 Behavior requirements
None

3.9.4.3.4 setLoopback() operation

3.9.4.3.4.1 Overview
Refer section 5.3.1.5.3.4 of [ESSOR-Ref 03]

3.9.4.3.4.2 Signatures
Refer section 5.3.1.5.3.4.1 of [ESSOR-Ref 03]

3.9.4.3.4.3 Parameters
Refer section 5.3.1.5.3.4.2 of [ESSOR-Ref 03]

3.9.4.3.4.4 Exceptions
Refer section 5.3.1.5.3.4.6 of [ESSOR-Ref 03]

3.9.4.3.4.5 Attributes
None.

3.9.4.3.4.6 Behavior requirements
None

3.9.4.3.5 getLoopback() operation

3.9.4.3.5.1 Overview
Refer section 5.3.1.5.3.5 of [ESSOR-Ref 03]

3.9.4.3.5.2 Signatures
Refer section 5.3.1.5.3.5.1 of [ESSOR-Ref 03]

3.9.4.3.5.3 Parameters
Refer section 5.3.1.5.3.5.2 of [ESSOR-Ref 03]

3.9.4.3.5.4 Exceptions
Refer section 5.3.1.5.3.5.6 of [ESSOR-Ref 03]

3.9.4.3.5.5 Attributes
None.

3.9.4.3.5.6 Behavior requirements
None

3.9.4.3.6 setProtocolType() operation

3.9.4.3.6.1 Overview
Refer section 5.3.1.5.3.6 of [ESSOR-Ref 03]

3.9.4.3.6.2 Signatures
Refer section 5.3.1.5.3.6.1 of [ESSOR-Ref 03]

3.9.4.3.6.3 Parameters
Refer section 5.3.1.5.3.6.2 of [ESSOR-Ref 03]

3.9.4.3.6.4 Exceptions
Refer section 5.3.1.5.3.6.6 of [ESSOR-Ref 03]

3.9.4.3.6.5 Attributes
None.

3.9.4.3.6.6 Behavior requirements
None

3.9.4.3.7 getProtocolType() operation

3.9.4.3.7.1 Overview
Refer section 5.3.1.5.3.7 of [ESSOR-Ref 03]

3.9.4.3.7.2 Signatures
Refer section 5.3.1.5.3.7.1 of [ESSOR-Ref 03]

3.9.4.3.7.3 Parameters
Refer section 5.3.1.5.3.7.2 of [ESSOR-Ref 03]

3.9.4.3.7.4 Exceptions
Refer section 5.3.1.5.3.7.6 of [ESSOR-Ref 03]

3.9.4.3.7.5 Attributes
None.

3.9.4.3.7.6 Behavior requirements
None

3.9.4.4 SerialPort::SerialPortManagement::SerialMessageControl
Refer to section 3.2.2.2 of this document for the primitives inherited from DeviceMessageControl API interfaces viz. DevMsgCtl::DeviceMessageControl.

3.9.4.5 SerialPort::SerialPortManagement::DynamicBaud

3.9.4.5.1 setBaudRate() operation

3.9.4.5.1.1 Overview
Refer section 5.7.3.1.2.3.1 of [ESSOR-Ref 03]

3.9.4.5.1.2 Signatures
Refer section 5.7.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.9.4.5.1.3 Parameters
Refer section 5.7.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.9.4.5.1.4 Exceptions
Refer section 5.7.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.9.4.5.1.5 Attributes
None.

3.9.4.5.1.6 Behavior requirements
None

3.9.4.5.2 getEffectiveBaudRate() operation

3.9.4.5.2.1 Overview
Refer section 5.7.3.1.2.3.2 of [ESSOR-Ref 03]

3.9.4.5.2.2 Signatures
Refer section 5.7.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.9.4.5.2.3 Parameters
Refer section 5.7.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.9.4.5.2.4 Exceptions
Refer section 5.7.3.1.2.3.2.6 of [ESSOR-Ref 03]

3.9.4.5.2.5 Attributes
None.

3.9.4.5.2.6 Behavior requirements
None

3.9.4.6 SerialPort::SerialPortManagement::SerialPortTxRxDataPolarity

3.9.4.6.1 setTxDataPolarity() operation

3.9.4.6.1.1 Overview
Refer section 5.8.3.1.2.3.1 of [ESSOR-Ref 03]

3.9.4.6.1.2 Signatures
Refer section 5.8.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.9.4.6.1.3 Parameters
Refer section 5.8.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.9.4.6.1.4 Exceptions
Refer section 5.8.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.9.4.6.1.5 Attributes
None.

3.9.4.6.1.6 Behavior requirements
None

3.9.4.6.2 getTxDataPolarity() operation

3.9.4.6.2.1 Overview
Refer section 5.8.3.1.2.3.2 of [ESSOR-Ref 03]

3.9.4.6.2.2 Signatures
Refer section 5.8.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.9.4.6.2.3 Parameters
Refer section 5.8.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.9.4.6.2.4 Exceptions
Refer section 5.8.3.1.2.3.2.6 of [ESSOR-Ref 03]

3.9.4.6.2.5 Attributes
None.

3.9.4.6.2.6 Behavior requirements
None

3.9.4.6.3 setRxDataPolarity() operation

3.9.4.6.3.1 Overview
Refer section 5.8.3.1.2.3.3 of [ESSOR-Ref 03]

3.9.4.6.3.2 Signatures
Refer section 5.8.3.1.2.3.3.1 of [ESSOR-Ref 03]

3.9.4.6.3.3 Parameters
Refer section 5.8.3.1.2.3.3.2 of [ESSOR-Ref 03]

3.9.4.6.3.4 Exceptions
Refer section 5.8.3.1.2.3.3.6 of [ESSOR-Ref 03]

3.9.4.6.3.5 Attributes

None.

3.9.4.6.3.6 Behavior requirements
None

3.9.4.6.4 getRxDataPolarity() operation

3.9.4.6.4.1 Overview
Refer section 5.8.3.1.2.3.4 of [ESSOR-Ref 03]

3.9.4.6.4.2 Signatures
Refer section 5.8.3.1.2.3.4.1 of [ESSOR-Ref 03]

3.9.4.6.4.3 Parameters
Refer section 5.8.3.1.2.3.4.2 of [ESSOR-Ref 03]

3.9.4.6.4.4 Exceptions
Refer section 5.8.3.1.2.3.4.6 of [ESSOR-Ref 03]

3.9.4.6.4.5 Attributes
None.

3.9.4.6.4.6 Behavior requirements
None

3.9.4.7 SerialPort::SerialPortManagement::SerialPortTxRxClockManagement

3.9.4.7.1 setTxClockStatus() operation

3.9.4.7.1.1 Overview
Refer section 5.9.3.1.2.3.1 of [ESSOR-Ref 03]

3.9.4.7.1.2 Signatures
Refer section 5.9.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.9.4.7.1.3 Parameters
Refer section 5.9.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.9.4.7.1.4 Exceptions
Refer section 5.9.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.9.4.7.1.5 Attributes
None.

3.9.4.7.1.6 Behavior requirements
None

3.9.4.7.2 getTxClockStatus() operation

3.9.4.7.2.1 Overview
Refer section 5.9.3.1.2.3.2 of [ESSOR-Ref 03]

3.9.4.7.2.2 Signatures
Refer section 5.9.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.9.4.7.2.3 Parameters
Refer section 5.9.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.9.4.7.2.4 Exceptions
Refer section 5.9.3.1.2.3.2.6 of [ESSOR-Ref 03]

3.9.4.7.2.5 Attributes
None.

3.9.4.7.2.6 Behavior requirements
None

3.9.4.7.3 setRxClockStatus() operation

3.9.4.7.3.1 Overview
Refer section 5.9.3.1.2.3.3 of [ESSOR-Ref 03]

3.9.4.7.3.2 Signatures
Refer section 5.9.3.1.2.3.3.1 of [ESSOR-Ref 03]

3.9.4.7.3.3 Parameters
Refer section 5.9.3.1.2.3.3.2 of [ESSOR-Ref 03]

3.9.4.7.3.4 Exceptions
Refer section 5.9.3.1.2.3.3.6 of [ESSOR-Ref 03]

3.9.4.7.3.5 Attributes
None.

3.9.4.7.3.6 Behavior requirements
None

3.9.4.7.4 getRxClockStatus() operation

3.9.4.7.4.1 Overview
Refer section 5.9.3.1.2.3.4 of [ESSOR-Ref 03]

3.9.4.7.4.2 Signatures
Refer section 5.9.3.1.2.3.4.1 of [ESSOR-Ref 03]

3.9.4.7.4.3 Parameters

Refer section 5.9.3.1.2.3.4.2 of [ESSOR-Ref 03]

3.9.4.7.4.4 Exceptions
Refer section 5.9.3.1.2.3.4.6 of [ESSOR-Ref 03]

3.9.4.7.4.5 Attributes
None.

3.9.4.7.4.6 Behavior requirements
None

3.9.4.8 SerialPort::SerialPortManagement::SerialPortDataRateSynchManagement

3.9.4.8.1 setDataRateSynch() operation

3.9.4.8.1.1 Overview
Refer section 5.10.3.1.2.3.1 of [ESSOR-Ref 03]

3.9.4.8.1.2 Signatures
Refer section 5.10.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.9.4.8.1.3 Parameters
Refer section 5.10.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.9.4.8.1.4 Exceptions
Refer section 5.10.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.9.4.8.1.5 Attributes
None.

3.9.4.8.1.6 Behavior requirements
None

3.9.4.8.2 getDataRateSynch() operation

3.9.4.8.2.1 Overview
Refer section 5.10.3.1.2.3.2 of [ESSOR-Ref 03]

3.9.4.8.2.2 Signatures
Refer section 5.10.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.9.4.8.2.3 Parameters
Refer section 5.10.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.9.4.8.2.4 Exceptions
Refer section 5.10.3.1.2.3.2.6 of [ESSOR-Ref 03]

3.9.4.8.2.5 Attributes

None.

3.9.4.8.2.6 Behavior requirements

None

3.9.4.9 SerialPort::SerialPortManagement::SerialPortDcdManagement

3.9.4.9.1 setDcd() operation

3.9.4.9.1.1 Overview

Refer section 5.11.3.1.2.3.1 of [ESSOR-Ref 03]

3.9.4.9.1.2 Signatures

Refer section 5.11.3.1.2.3.1 of [ESSOR-Ref 03]

3.9.4.9.1.3 Parameters

Refer section 5.11.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.9.4.9.1.4 Exceptions

Refer section 5.11.3.1.2.3.1.5 of [ESSOR-Ref 03]

3.9.4.9.1.5 Attributes

None.

3.9.4.9.1.6 Behavior requirements

None

3.9.4.10 SerialPort::SerialPortManagement::SerialPortDtrDsrManagement

3.9.4.10.1 setDsr() operation

3.9.4.10.1.1 Overview

Refer section 5.12.3.1.2.3.1 of [ESSOR-Ref 03]

3.9.4.10.1.2 Signatures

Refer section 5.12.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.9.4.10.1.3 Parameters

Refer section 5.12.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.9.4.10.1.4 Exceptions

Refer section 5.12.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.9.4.10.1.5 Attributes

None.

3.9.4.10.1.6 Behavior requirements

None

3.9.4.10.2 getDtr() operation

3.9.4.10.2.1 Overview
Refer section 5.12.3.1.2.3.2 of [ESSOR-Ref 03]

3.9.4.10.2.2 Signatures
Refer section 5.12.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.9.4.10.2.3 Parameters
Refer section 5.12.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.9.4.10.2.4 Exceptions
Refer section 5.12.3.1.2.3.2.3 of [ESSOR-Ref 03]

3.9.4.10.2.5 Attributes
None.

3.9.4.10.2.6 Behavior requirements
None

3.9.4.11 SerialPort::SerialPortManagement::SerialPortHwRtsCtsManagement

3.9.4.11.1 setHwCts() operation

3.9.4.11.1.1 Overview
Refer section 5.13.3.1.2.3.1 of [ESSOR-Ref 03]

3.9.4.11.1.2 Signatures
Refer section 5.13.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.9.4.11.1.3 Parameters
Refer section 5.13.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.9.4.11.1.4 Exceptions
Refer section 5.13.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.9.4.11.1.5 Attributes
None.

3.9.4.11.1.6 Behavior requirements
None

3.9.4.11.2 getHwRts() operation

3.9.4.11.2.1 Overview
Refer section 5.13.3.1.2.3.2 of [ESSOR-Ref 03]

3.9.4.11.2.2 Signatures

Refer section 5.13.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.9.4.11.2.3 Parameters
Refer section 5.13.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.9.4.11.2.4 Exceptions
Refer section 5.13.3.1.2.3.2.6 of [ESSOR-Ref 03]

3.9.4.11.2.5 Attributes
None.

3.9.4.11.2.6 Behavior requirements
None

3.9.4.12 SerialPort::SerialAsync::Async

3.9.4.12.1 getAsyncConfig() operation

3.9.4.12.1.1 Overview
Refer section 5.4.3.1.2.3.2 of [ESSOR-Ref 03]

3.9.4.12.1.2 Signatures
Refer section 5.4.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.9.4.12.1.3 Parameters
Refer section 5.4.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.9.4.12.1.4 Exceptions
Refer section 5.4.3.1.2.3.2.6 of [ESSOR-Ref 03]

3.9.4.12.1.5 Attributes
None.

3.9.4.12.1.6 Behavior requirements
None

3.9.4.12.2 setAsyncConfig() operation

3.9.4.12.2.1 Overview
Refer section 5.4.3.1.2.3.1 of [ESSOR-Ref 03]

3.9.4.12.2.2 Signatures
Refer section 5.4.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.9.4.12.2.3 Parameters
Refer section 5.4.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.9.4.12.2.4 Exceptions

Refer section 5.4.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.9.4.12.2.5 Attributes
None.

3.9.4.12.2.6 Behavior requirements
None

3.9.4.13 SerialPort::Sync::SyncRaw

3.9.4.13.1 setSyncRawConfig() operation

3.9.4.13.1.1 Overview
Refer section 5.5.3.1.2.3.1 of [ESSOR-Ref 03]

3.9.4.13.1.2 Signatures
Refer section 5.5.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.9.4.13.1.3 Parameters
Refer section 5.5.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.9.4.13.1.4 Exceptions
Refer section 5.5.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.9.4.13.1.5 Attributes
None.

3.9.4.13.1.6 Behavior requirements
None

3.9.4.13.2 getSyncRawConfig() operation

3.9.4.13.2.1 Overview
Refer section 5.5.3.1.2.3.2 of [ESSOR-Ref 03]

3.9.4.13.2.2 Signatures
Refer section 5.5.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.9.4.13.2.3 Parameters
Refer section 5.5.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.9.4.13.2.4 Exceptions
Refer section 5.5.3.1.2.3.2.6 of [ESSOR-Ref 03]

3.9.4.13.2.5 Attributes
None.

3.9.4.13.2.6 Behavior requirements

None

3.9.4.14 SerialPort::Sync::SyncHdlc

3.9.4.14.1 setSyncHdlcConfig() operation

3.9.4.14.1.1 Overview
Refer section 5.6.3.1.2.3.1 of [ESSOR-Ref 03]

3.9.4.14.1.2 Signatures
Refer section 5.6.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.9.4.14.1.3 Parameters
Refer section 5.6.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.9.4.14.1.4 Exceptions
Refer section 5.6.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.9.4.14.1.5 Attributes
None.

3.9.4.14.1.6 Behavior requirements
None

3.9.4.14.2 getSyncHdlcConfig() operation

3.9.4.14.2.1 Overview
Refer section 5.6.3.1.2.3.2 of [ESSOR-Ref 03]

3.9.4.14.2.2 Signatures
Refer section 5.6.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.9.4.14.2.3 Parameters
Refer section 5.6.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.9.4.14.2.4 Exceptions
Refer section 5.6.3.1.2.3.2.6 of [ESSOR-Ref 03]

3.9.4.14.2.5 Attributes
None.

3.9.4.14.2.6 Behavior requirements
None

3.9.4.15 SerialPort::SerialPortNotification::SerialPortNotifyTxRxClock

3.9.4.15.1 notifyTxClockStatus() operation

3.9.4.15.1.1 Overview
Refer section 5.9.3.1.3.3.1 of [ESSOR-Ref 03]

3.9.4.15.1.2 Signatures
Refer section 5.9.3.1.3.3.1.1 of [ESSOR-Ref 03]

3.9.4.15.1.3 Parameters
Refer section 5.9.3.1.3.3.1.2 of [ESSOR-Ref 03]

3.9.4.15.1.4 Exceptions
Refer section 5.9.3.1.3.3.1.6 of [ESSOR-Ref 03]

3.9.4.15.1.5 Attributes
None.

3.9.4.15.1.6 Behavior requirements
None

3.9.4.15.2 notifyRxClockStatus() operation

3.9.4.15.2.1 Overview
Refer section 5.9.3.1.3.3.2 of [ESSOR-Ref 03]

3.9.4.15.2.2 Signatures
Refer section 5.9.3.1.3.3.2.1 of [ESSOR-Ref 03]

3.9.4.15.2.3 Parameters
Refer section 5.9.3.1.3.3.2.2 of [ESSOR-Ref 03]

3.9.4.15.2.4 Exceptions
Refer section 5.9.3.1.3.3.2.6 of [ESSOR-Ref 03]

3.9.4.15.2.5 Attributes
None.

3.9.4.15.2.6 Behavior requirements
None

3.9.4.16 SerialPort::SerialPortNotification::SerialPortNotifyDataRateSynch

3.9.4.16.1 notifyDataRateSynch() operation

3.9.4.16.1.1 Overview
Refer section 5.10.3.1.3.3.1 of [ESSOR-Ref 03]

3.9.4.16.1.2 Signatures
Refer section 5.10.3.1.3.3.1.1 of [ESSOR-Ref 03]

3.9.4.16.1.3 Parameters
Refer section 5.10.3.1.3.3.1.2 of [ESSOR-Ref 03]

3.9.4.16.1.4 Exceptions
Refer section 5.10.3.1.3.3.1.6 of [ESSOR-Ref 03]

3.9.4.16.1.5 Attributes
None.

3.9.4.16.1.6 Behavior requirements
None

3.9.4.17 SerialPort::SerialPortNotification::SerialPortNotifyDtr

3.9.4.17.1 notifyDtr() operation

3.9.4.17.1.1 Overview
Refer section 5.12.3.1.3.3.1 of [ESSOR-Ref 03]

3.9.4.17.1.2 Signatures
Refer section 5.12.3.1.3.3.1.1 of [ESSOR-Ref 03]

3.9.4.17.1.3 Parameters
Refer section 5.12.3.1.3.3.1.2 of [ESSOR-Ref 03]

3.9.4.17.1.4 Exceptions
Refer section 5.12.3.1.3.3.1.6 of [ESSOR-Ref 03]

3.9.4.17.1.5 Attributes
None.

3.9.4.17.1.6 Behavior requirements
None

3.9.4.18 SerialPort::SerialPortNotification::SerialPortNotifyHwRts

3.9.4.18.1 notifyHwRts() operation

3.9.4.18.1.1 Overview
Refer section 5.13.3.1.3.3.1 of [ESSOR-Ref 03]

3.9.4.18.1.2 Signatures
Refer section 5.13.3.1.3.3.1.1 of [ESSOR-Ref 03]

3.9.4.18.1.3 Parameters
Refer section 5.13.3.1.3.3.1.2 of [ESSOR-Ref 03]

3.9.4.18.1.4 Exceptions

Refer section 5.13.3.1.3.3.1.3 of [ESSOR-Ref 03]

3.9.4.18.1.5 Attributes
None.

3.9.4.18.1.6 Behavior requirements
None

3.9.4.19 Exceptions
Refer section 5.3.1.5.3.1.6, 5.3.1.5.3.4.6, 5.3.1.5.3.6.6, 5.7.3.1.2.3.1.6, 5.5.3.1.2.3.2.6, 5.6.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.9.4.20 Type
The specification complies with the Full PIM IDL Profile of WInnF IDL profiles for PIM of SDR Applications, specified in [WInnF-Ref 05].

3.9.4.20.1 SerialPort::SynchronizationConfig
Refer sections 5.3.1.1.2.1, 5.4.3.1.1.2, 5.4.3.1.1.2 and 5.5.3.1.1.2 of [ESSOR-Ref 03]

3.9.4.20.2 configSupported
```
typedef enum {
        ASYNC,
        SYNC,
        RAW,
        HDLC
} configSupported;
```

3.9.4.20.3 configSupportedList
```
typedef sequence < configSupported > configSupportedList
```

3.9.4.20.4 protocolSupported
```
typedef enum {
        232,
        422,
        485
} protocolSupported;
```

3.9.4.20.5 protocolSupportedList
```
typedef sequence < protocolSupported > protocolSupportedList
```

3.9.4.20.6 asyncBaudRates
```
typedef sequence <unsigned long > asyncBaudRates;
```

3.9.4.20.7 asyncBitNum
```
typedef sequence <unsigned short > asyncBitNum;
```

3.9.4.20.8 asyncParity

```
typedef enum {
        NONE,
        ODD,
        EVEN
} asyncParity;
```

3.9.4.20.9 asyncStopBit

```
typedef enum {
        1,
        1.5,
        2
} asyncStopBit;
```

3.9.4.20.10 rawClockSource

```
typedef enum {
        DCE,
        DTE
        } rawClockSource;
```

3.9.4.20.11 rawClockControl

```
typedef enum {
        RTS,
        CTS,
        FREE
        } rawClockControl;
```

3.9.4.20.12 rawBaudRates

```
typedef sequence <unsigned long > rawBaudRates;
```

3.9.4.20.13 hdlcClockSource

```
typedef enum {
        DCE,
        DTE
        } hdlcClockSource;
```

3.9.4.20.14 hdlcClockControl

```
typedef enum {
        RTS,
        CTS,
        FREE
        } hdlcClockControl;
```

3.9.4.20.15 hdlcBaudRates

```
typedef sequence <unsigned long > hdlcBaudRates;
```

3.9.4.20.16 baudDirection

```
typedef enum {
        TX,
        RX,
        Both
        } baudDirection;
```

3.9.4.20.17 txBaudRates

```
typedef sequence <unsigned long> txBaudRates;
```

3.9.4.20.18 rxBaudRates

```
typedef sequence <unsigned long > rxBaudRates;
```

3.9.4.20.19 txPolarities

```
typedef sequence <string> txPolarities;
```

3.9.4.20.20 rxPolarities

```
typedef sequence <string> rxPolarities;
```

3.9.5 Serial Port Facility Attributes

3.9.5.1 Capabilities

Name	Type	Description	Range/Values
CONFIG_SUPPORTED	sequence of enum configSupported	List of configurations	Sequence of configuration (asynchronous, synchronous raw, synchronous HDLC)
PROTOCOL_SUPPORTED	sequence of enum protocolSupported	List of protocols	Sequence of protocols (232,422,485)
ASYNC_CONFIGURATION_SUPPORTED	boolean	Indicates whether the asynhronous configuration support is provided or not	TRUE indicates asynchronous configuration is supported, FALSE otherwise.
SYNC_RAW_CONFIGURATION_SUPPORTED	boolean	Indicates whether the synhronous raw configuration support is provided or not	TRUE indicates synchronous raw configuration is supported, FALSE otherwise.
SYNC_HDLC_CONFIGURATION_SUPPORTED	boolean	Indicates whether the synhronous HDLC configuration support is provided or not	TRUE indicates synchronous HDLC configuration is supported, FALSE otherwise.
DYNAMIC_BAUD_SUPPORTED	boolean	Indicates whether the dynamic baud support is provided or not	TRUE indicates dynamic baud is supported, FALSE otherwise.
RTS_CTS_SUP	boolean	Indicates whether the	TRUE indicates RTS/CTS

PORTED		RTS/CTS Signals support is provided or not	Signals supported, FALSE otherwise.
DTR_DSR_SUPPORTED	boolean	Indicates whether the DTR/DSR Signals support is provided or not	TRUE indicates DTR/DSR Signals supported, FALSE otherwise.
DCD_SUPPORTED	boolean	Indicates whether the DCD Signals support is provided or not	TRUE indicates DCD Signals is supported, FALSE otherwise.
DATA_RATE_SYNC_SUPPORTED	boolean	Indicates whether the data rate synchronization support is provided or not	TRUE indicates data rate synchronization is supported, FALSE otherwise.
SYNC_CLOCK SUPPORTED	boolean	Indicates whether the data rate synchronous clock activation support is provided or not	TRUE indicates data rate synchronous clock activation is supported, FALSE otherwise.
DATA_POLARITY_SUPPORTED	boolean	Indicates whether the data polarity support is provided or not	TRUE indicates data rate data polarity is supported, FALSE otherwise.
ASYNC_BAUD_RATES_SUPPORTED	sequence of unsigned long asyncBaudRates	Set of baud rates supported by the serial port for ASYNC configuration	Sequence of baud rates (numbers) for ASYNC configuration
ASYNC_DATA_BITS_SUPPORTED	sequence of unsigned short asyncBitNum	Set of data bits numbers that can be configured supported by the serial port for ASYNC configuration	Sequence of data bit numbers (numbers) for ASYNC configuration
ASYNC_PARITY_SUPPORTED	enum asyncParity	Set of parity bits supported by the serial port for ASYNC configuration	Enumeration of parity bit (NONE,ODD, EVEN)
ASYNC_STOP_BIT_SUPPORTED	enum asyncStopBit	Set of stop bit numbers supported by AYNC configuration	Enumeration of async stop bit source(1,1.5,2)
HW_FLOW_CONTROL_SUPPORTED	boolean	Indicates whether the hardware flow control support is provided or not	TRUE indicates hardware flow control is supported, FALSE otherwise.
SW_FLOW_CONTROL_SUPP	boolean	Indicates whether the software flow control	TRUE indicates software flow control is supported,

ORTED		support is provided or not	FALSE otherwise.
DMA_ACCESS_SUPPORTED	boolean	Indicates whether the DMA access support is provided or not	TRUE indicates DMA access is supported, FALSE otherwise.
RAW_CLOCK_SOURCE_SUPPORTED	enum rawClockSource	Set of clock source supported by the serial port for RAW configuration	Enumeration of RAW clock source(DCE,DTE)
RAW_CLOCK_CONTROL_SUPPORTED	enum rawClockControl	Set of clock control supported by the serial port for RAW configuration	Enumeration of RAW clock controls(RTS, CTS, free running)
RAW_BAUD_RATES_SUPPORTED	sequence of unsigned long hdlcBaudRates	Set of baud rates supported by the serial port for RAW configuration	Sequence of baud rates (numbers) for RAW configuration
HDLC_CLOCK_SOURCE_SUPPORTED	enum hdlcClockSource	Set of clock source supported by the serial port for HDLC configureation	Enumeration of HDLC clock source(DCE,DTE)
HDLC_CLOCK_CONTROL_SUPPORTED	enum hdlcClockControl	Set of clock control supported by the serial port for HDLC configuration	Enumeration of HDLC clock controls(RTS, CTS, free running)
HDLC_BAUD_RATES_SUPPORTED	sequence Sequence of unsigned long hdlcBaudRates	Set of baud rates supported by the serial port for HDLC configuration	Sequence of baud rates (numbers) for HDLC configuration
BAUD_DIRECTION_SUPPORTED	enum baudDirection	Set of direction supported by the serial port for dynamic baud API	Enumeration of baud direction(TX, RX, BOTH)
TX_BAUD_RATES_SUPPORTED	Sequence of unsigned long txBaudRates	Set of baud rates supported by the serial port for transmission	Sequence of TX baud rates (numbers)
RX_BAUD_RATES_SUPPORTED	Sequence of unsigned long rxBaudRates	Set of baud rates supported by the serial port for reception	Sequence of RX baud rates (numbers)
TX_POLARITY_SUPPORTED	sequence of string txPolarities	Set of polarities supported by the serial port for	Sequence of TX polarities

		transmission	
RX_POLARITY_SUPPORTED	sequence of string rxPolarities	Set of polarities supported by the serial port for reception	Sequence of RX polarities

Table 3.61: Serial Port general capabilities

3.9.5.2 Properties

Name	Type	Description	Range/Values
MAX_PAYLOAD_SIZE	unsigned long	Maximum payload size allowed for a payload.	
MIN_PAYLOAD_SIZE	unsigned long	Minimum payload size allowed for a payload.	
DESIRED_PALOAD_SIZE	unsigned long	Desired payload size allowed for a payload.	

Table 3.62: Serial Port properties

3.9.5.3 Variables

None

3.10 Platform Discretes Facility

3.10.1 Approach

3.10.1.1 Candidate API Set

Platform Discretes Device API - [ESSOR-Ref 03]

3.10.1.2 Selected API Set

Platform Discretes Device API - [ESSOR-Ref 03]

3.10.1.3 Modification

No modification in API but to adapt in PIM of facility framework, suitable modifications are done to remove any SCA specific terminology.

3.10.1.4 Rationale

Above referenced API is sufficient to cater requirement of platform and waveform components. Its specification was in line with SCA and to create PIM in Facility Framework, suitable modifications are required.

3.10.1.5 Conclusion

ESSOR API is adopted and provided in the form of PIM specification as per Principles for WInnF Facility Standards [WInnF-Ref 09]. PSM Documents (SCA, Native C++, FPGA) corresponding to this facility PIM specification as applicable will be released after their reference implementation.

3.10.2 Introduction

3.10.2.1 Overview

Refer 4.1 of [ESSOR-Ref 03]

3.10.2.2 Definitions

None

3.10.2.3 Technical Details

Refer 4.1.1 and 4.1.2 of [ESSOR-Ref 03] for usage context and general behavior of Platform Discretes facility in SCA specific platform.

3.10.3 Services

3.10.3.1 Provide Services

The following table lists the provide services of the API (used by a radio application or radio platform and provided by platform discretes facility):

Service Group/Module	Interface Service	Primitives/Operations
Discretes_Control	DiscretesManagment	getDiscretesProfile() getDiscretesStatus() setDiscreteEnabledStatus() getDiscreteEnabledStatus() getDiscreteSignalStatus() setDiscreteSignalStatus() registerComponent() unregisterComponent()

Table 3.63: Platform Discretes Provide Services

3.10.3.2 Use Services

The following table lists the used services of the API (provided by a radio application or radio platform and used by platform discretes facility):

Service Group/Module	Interface Service	Primitives/Operations
Discretes_Async_Status	DiscretesStatusNotification	notifyDiscretesStatusChanges()

Table 3.64: Platform Discretes Use Services

3.10.3.3 State Machines

Platform Discrete Facility will not go for any state change during any operation.

3.10.3.3.1 PlatformDiscrete_SM

PlatformDiscrete_SM is specified as the main state machine for Platform Discrete Facility. An instance of PlatformDiscrete_SM is followed by each Platform Discrete instance.

The following figure is the statechart of PlatformDiscrete_SM state machine:

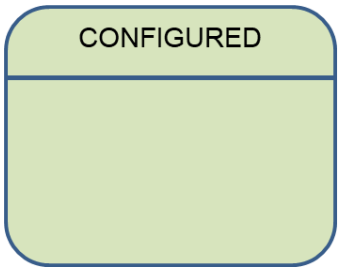

Figure 3.38 : PlatformDiscrete_SM statechart

3.10.3.3.1.1 States

3.10.3.3.1.2 CONFIGURED

CONFIGURED is specified as the unique state, during which a Platform Discrete Facility is configured according to the needs of all the radio applications to be supported during the CONFIGURED state. CONFIGURED is reached by Platform Discrete Facility when it

- Complies with any value specified for a capability or a property,
- Is capable of interacting with radio application according to the service interfaces of its service implementations.

How CONFIGURED is reached is unspecified by the PIM specification, and can be specified by the applied PSM specification.

3.10.3.4 Service groups

3.10.3.4.1 Platform_Discrete::Discretes_Control

Discretes_Control service group enables management of platform discretes and provide access to different components.

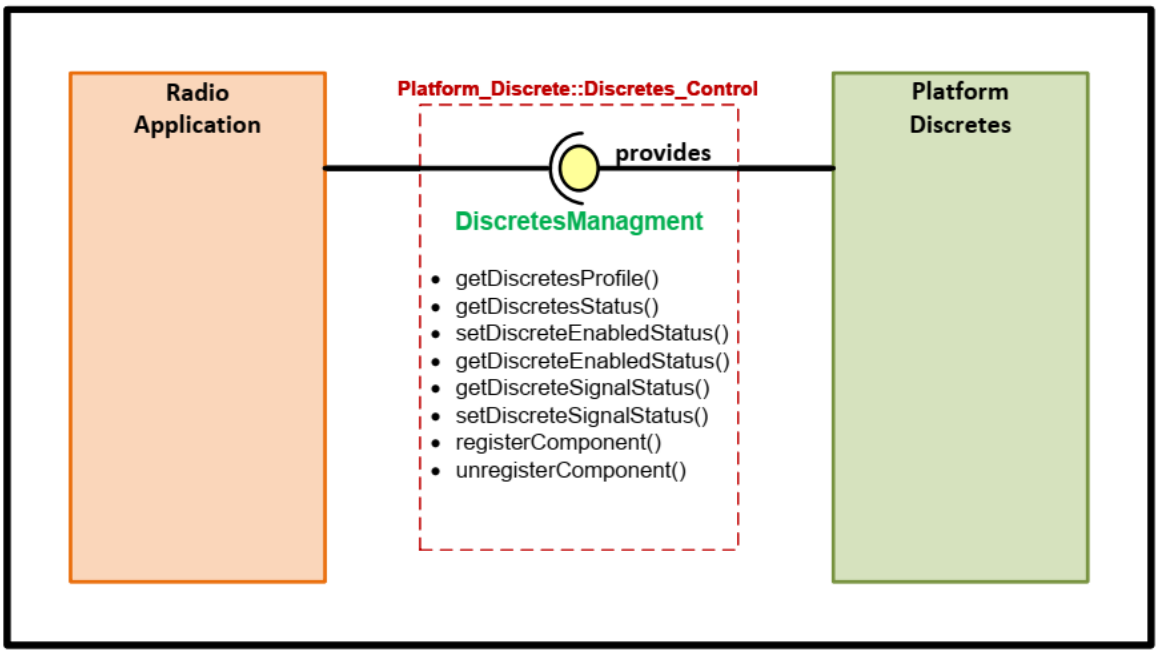

Figure 3.39 : Discretes_Control service group

3.10.3.4.2 Platform_Discrete::Discretes_Async_Status

Discretes_Async_Status service group enables immediate notifications on discretes status change to all components granted access.

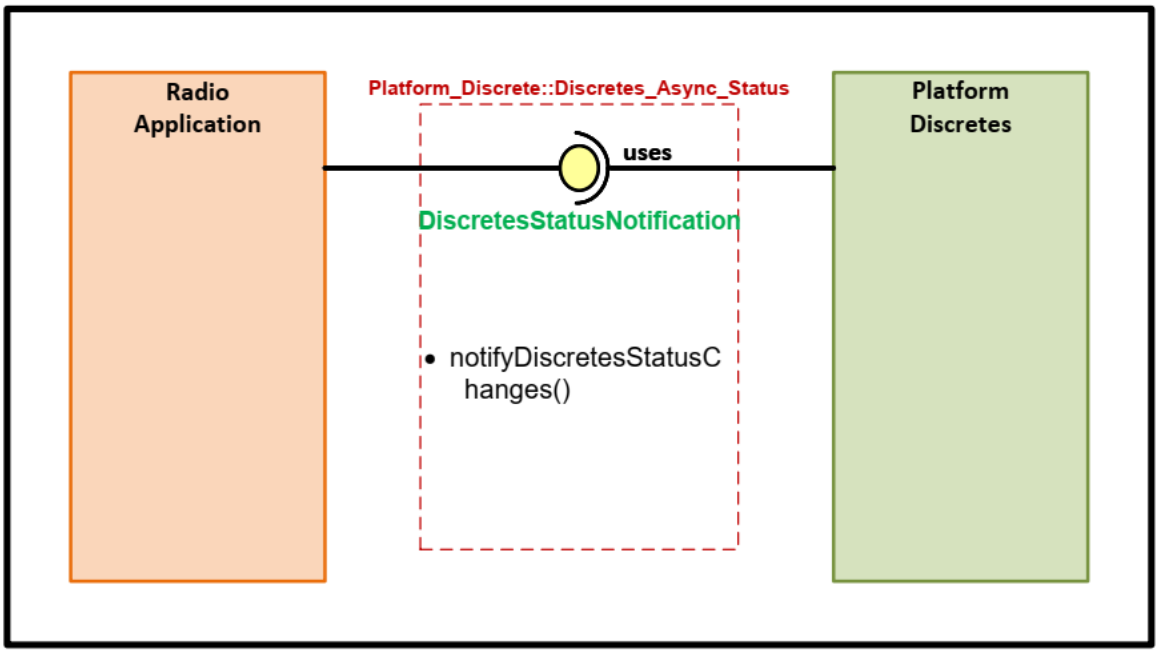

Figure 3.40 : Discretes_Async_Status services group

3.10.4 Service Primitives

This section specifies the primitives of the Platform Discretes facility services. Each declaration of a primitive complies with the Full PIM IDL Profile of WInnF IDL profiles for PIM of SDR Applications, specified in [WInnF-Ref 05]. The declaration of each primitive

also complies with SCA 4.1 Appendix E-1 [JTNC-Ref 30] .

3.10.4.1 Platform_Discrete::Discretes_Control::DiscretesManagment

3.10.4.1.1 getDiscretesProfile() operation

3.10.4.1.1.1 Overview
Refer 4.3.1.2.3.1 of [ESSOR-Ref 03]

3.10.4.1.1.2 Signatures
Refer 4.3.1.2.3.1.1 of [ESSOR-Ref 03]

3.10.4.1.1.3 Parameters
Refer 4.3.1.2.3.1.2 of [ESSOR-Ref 03]

3.10.4.1.1.4 Exceptions
Refer 4.3.1.2.3.1.6 of [ESSOR-Ref 03]

3.10.4.1.1.5 Attributes
None.

3.10.4.1.1.6 Behavior requirements
This operation provides the capability to get the profile of the discretes.

3.10.4.1.2 getDiscretesStatus() operation

3.10.4.1.2.1 Overview
Refer 4.3.1.2.3.2 of [ESSOR-Ref 03]

3.10.4.1.2.2 Signatures
Refer 4.3.1.2.3.2.1 of [ESSOR-Ref 03]

3.10.4.1.2.3 Parameters
Refer 4.3.1.2.3.2.2 of [ESSOR-Ref 03]

3.10.4.1.2.4 Exceptions
Refer 4.3.1.2.3.2.6 of [ESSOR-Ref 03]

3.10.4.1.2.5 Attributes
None.

3.10.4.1.2.6 Behavior requirements
This operation provides the capability to set the profile of the discretes.

3.10.4.1.3 setDiscreteEnabledStatus() operation

3.10.4.1.3.1 Overview
Refer 4.3.1.2.3.3 of [ESSOR-Ref 03]

3.10.4.1.3.2 Signatures
Refer 4.3.1.2.3.3.1 of [ESSOR-Ref 03]

3.10.4.1.3.3 Parameters
Refer 4.3.1.2.3.3.2 of [ESSOR-Ref 03]

3.10.4.1.3.4 Exceptions
Refer 4.3.1.2.3.3.6 of [ESSOR-Ref 03]

3.10.4.1.3.5 Attributes
None.

3.10.4.1.3.6 Behavior requirements
This operation provides the capability to get the status of the discretes.

3.10.4.1.4 getDiscreteEnabledStatus() operation

3.10.4.1.4.1 Overview
Refer 4.3.1.2.3.4 of [ESSOR-Ref 03]

3.10.4.1.4.2 Signatures
Refer 4.3.1.2.3.4.1 of [ESSOR-Ref 03]

3.10.4.1.4.3 Parameters
Refer 4.3.1.2.3.4.2 of [ESSOR-Ref 03]

3.10.4.1.4.4 Exceptions
Refer 4.3.1.2.3.4.6 of [ESSOR-Ref 03]

3.10.4.1.4.5 Attributes
None.

3.10.4.1.4.6 Behavior requirements
This operation provides the capability to get the status of the discretes.

3.10.4.1.5 setDiscreteSignalStatus()operation

3.10.4.1.5.1 Overview
Refer 4.3.1.2.3.5 of [ESSOR-Ref 03]

3.10.4.1.5.2 Signatures
Refer 4.3.1.2.3.5.1 of [ESSOR-Ref 03]

3.10.4.1.5.3 Parameters
Refer 4.3.1.2.3.5.2 of [ESSOR-Ref 03]

3.10.4.1.5.4 Exceptions

Refer 4.3.1.2.3.5.6 of [ESSOR-Ref 03]

3.10.4.1.5.5 Attributes

None.

3.10.4.1.5.6 Behavior requirements

This operation provides the capability to set the status of the discretes.

3.10.4.1.6 getDiscreteSignalStatus()operation

3.10.4.1.6.1 Overview

Refer 4.3.1.2.3.6 of [ESSOR-Ref 03]

3.10.4.1.6.2 Signatures

Refer 4.3.1.2.3.6.1 of [ESSOR-Ref 03]

3.10.4.1.6.3 Parameters

Refer 4.3.1.2.3.6.2 of [ESSOR-Ref 03]

3.10.4.1.6.4 Exceptions

Refer 4.3.1.2.3.6.6 of [ESSOR-Ref 03]

3.10.4.1.6.5 Attributes

None.

3.10.4.1.6.6 Behavior requirements

This operation provides the capability to get the status of the discretes.

3.10.4.1.7 registerComponent() operation

3.10.4.1.7.1 Overview

Refer 4.3.1.2.3.7 of [ESSOR-Ref 03]

3.10.4.1.7.2 Signatures

Refer 4.3.1.2.3.7.1 of [ESSOR-Ref 03]

3.10.4.1.7.3 Parameters

Refer 4.3.1.2.3.7.2 of [ESSOR-Ref 03]

3.10.4.1.7.4 Exceptions

Refer 4.3.1.2.3.7.6 of [ESSOR-Ref 03]

3.10.4.1.7.5 Attributes

None.

3.10.4.1.7.6 Behavior requirements

This operation enables different WF/PF components get the status update of the

discretes.

3.10.4.1.8 unregisterComponent() operation

3.10.4.1.8.1 Overview
Refer 4.3.1.2.3.8 of [ESSOR-Ref 03]

3.10.4.1.8.2 Signatures
Refer 4.3.1.2.3.8.1 of [ESSOR-Ref 03]

3.10.4.1.8.3 Parameters
Refer 4.3.1.2.3.8.2 of [ESSOR-Ref 03]

3.10.4.1.8.4 Exceptions
Refer 4.3.1.2.3.8.6 of [ESSOR-Ref 03]

3.10.4.1.8.5 Attributes
None.

3.10.4.1.8.6 Behavior requirements
This operation disables different WF/PF components get the status update of the discretes.

3.10.4.2 Platform_Discrete:: Discretes_Async_Status:: DiscretesStatusNotification

3.10.4.2.1 notifyDiscretesStatusChanges() operation

3.10.4.2.1.1 Overview
Refer 4.3.1.3.3.1 of [ESSOR-Ref 03]

3.10.4.2.1.2 Signatures
Refer 4.3.1.3.3.1.1 of [ESSOR-Ref 03]

3.10.4.2.1.3 Parameters
Refer 4.3.1.3.3.1.2 of [ESSOR-Ref 03]

3.10.4.2.1.4 Exceptions
Refer 4.3.1.3.3.1.6 of [ESSOR-Ref 03]

3.10.4.2.1.5 Attributes
None.

3.10.4.2.1.6 Behavior requirements
None.

3.10.4.3 Exceptions

3.10.4.3.1 Platform_Discrete::InvalidInputData

An exception is an abnormal situation related to the calling context or to parameters values, detected during execution of a called primitive.

Exceptions are only specified for provide services.

3.10.4.3.1.1 Specification

General exceptions are specified by the following table:

S. No.	Name	Applies To	Description
1.	InvalidInputData	setDiscreteEnabledStatus() getDiscreteEnabledStatus() getDiscreteSignalStatus() setDiscreteSignalStatus() registerComponent() unregisterComponent()	Refer 4.3.1.1.1.1 of [ESSOR-Ref 03]

Table 3.65: Platform Discretes general exceptions

3.10.4.3.1.2 Associated capabilities

The exceptionsActive capability is specified as a boolean that indicates if the Platform Discretes Facility raises exceptions.

The supportedExceptions capability is specified as a set of booleans indicating, for each specified exception, if it is raised by the Platform Discretes Facility.

3.10.4.4 Type

The specification complies with the Full PIM IDL Profile of WInnF IDL profiles for PIM of SDR Applications, specified in [WInnF-Ref 05].

3.10.4.4.1 Platform_Discrete:: DiscreteId

Refer 4.3.1.1.2.1 of [ESSOR-Ref 03]

3.10.4.4.2 Platform_Discrete:: DiscresteId

Refer 4.3.1.1.2.2 of [ESSOR-Ref 03]

3.10.4.4.3 Platform_Discrete:: ComponentId

Refer 4.3.1.1.2.3 of [ESSOR-Ref 03]

3.10.4.4.4 Platform_Discrete:: DiscreteType

Refer 4.3.1.1.2.4 of [ESSOR-Ref 03]

3.10.4.4.5 Platform_Discrete:: DiscretesProfile

Refer 4.3.1.1.2.5 of [ESSOR-Ref 03]

3.10.4.4.6 Platform_Discrete:: DiscretesStatus

Refer 4.3.1.1.2.6 of [ESSOR-Ref 03]

3.10.4.4.7 Platform_Discrete:: discreteStatus

Refer 4.3.1.1.4.1 of [ESSOR-Ref 03]

3.10.4.4.8 Platform_Discrete:: discreteProfile

Refer 4.3.1.1.4.2 of [ESSOR-Ref 03]

3.10.5 Facility Attributes

3.10.5.1 Capabilities

The following table lists the general capabilities of a platform discretes facility:

Name	Type	Range	Description
DISCRETE_COUNT	unsigned short	Platform Dependent	Number of Discretes supported by the facility
IN_DISCRETE_LIST	sequence of string	Platform Dependent	List of incoming discretes available to WF to acquire the status set by another PF/WF component on the corresponding outgoing discrete.
OUT_DISCRETE_LIST	sequence of string	Platform Dependent	List of outgoing discretes available to PF/WF component to signal an event to another WF component. Its status is stored until the next update of the discrete signal.
Exceptions Active	Boolean	True/False	Indicates if the Platform Discrete Service raises exceptions.
Supported Exceptions	Sequence of Boolean	Sequence of True / False	Set of booleans indicating, for each specified exception, if it is raised by the Platform Discrete Service.

Table 3.66: Platform Discretes general capabilities

3.10.5.2 Properties

Nil

3.10.5.3 Variables

Name	Type	Range	Description
IN_DISCRETE_#_VAL	uint_8	0/1	Incoming discrete and outgoing discrete work in pair and values will be same.

OUT_DISCRETE_#_VAL	uint_8	0/1	Incoming discrete and outgoing discrete work in pair and values will be same.

Table 3.67: Platform Discretes variables

4 Execution Environment

4.1 GPP Execution Environment

4.1.1 GPP Deployment Mechanism

SCA Full Profile as per section 2.2.5.3.3

4.1.2 GPP AEP

SCA AEP [JTNC-Ref 25] is accepted without modification.

4.2 Constrained Processor Execution Environment

4.2.1 Constrained Processor Deployment Mechanism

4.2.1.1 Approach

4.2.1.1.1 Candidate API Set

- Section 4.3.3 of ESSOR Architecture Introductory Document - [ESSOR-Ref 01]
- Section 4.4.3 of ESSOR Architecture Introductory Document - [ESSOR-Ref 01]

4.2.1.1.2 Selected API Set

Section 4.4.3 of ESSOR Architecture Introductory Document - [ESSOR-Ref 01]

4.2.1.1.3 Modification

None

4.2.1.1.4 Rationale

First candidate API set assumes the ‘CORBA everywhere’ approach which is not preferred in IRSA. Therefore, the MHAL based deployment mechanism is adopted.

4.2.1.1.5 Conclusion

Section 4.4.3 of ESSOR Architecture Introductory Document - [ESSOR-Ref 01] has been adopted without any modification.

4.2.2 Compact AEP Profile for Constrained Processors

AEP defines the execution environment for the waveform components running on contrained a processing element. AEP is a set of Operating System APIs, which a complaint radio platform is guaranteed to provide, and a complaint waveform component restricts its usage within the defined set, for ease of portability. The compact AEP profile for constrained processors defined herein, relates to the environment presented to waveform components running on constrained processor e.g., DSP, real-time processors with OS.

4.2.2.1 Approach

Compact AEP Profile is applicable to a processor which works under following constraints

or conditions.

- No Process Concept, all are threads/tasks.
- Complete memory shared across all threads/tasks (single address space).
- No file system concept.
- RTOS/microkernel for thread scheduling.
- Applications running are hard real-time applications with strict timing deadline.

4.2.2.1.1 Candidate API Set

- JTNC SCA - APPENDIX B: SCA APPLICATION ENVIRONMENT PROFILES [JTNC-Ref 25]
- WINNF Lw and ULw POSIX AEPs for Resource Constrained Processors [WInnF-Ref 04]
- ESSOR DSP AEP and IDL Profile Description Document [ESSOR-Ref 04]

4.2.2.1.2 Selected API Set

- Mainly WINNF Lw and ULw POSIX AEPs for Resource Constrained Processors [WInnF-Ref 04]
- Some APIs from ESSOR DSP AEP and IDL Profile Description Document [ESSOR-Ref 04]

4.2.2.1.3 Modifications

IRSA defines the compact AEP profile for constrained processors using the Lw profile from WInnF specification [WInnF-Ref 04] completely along with optional APIs defined for the Lw profile. Additional APIs from ULw profile defined in WInnF specification [WInnF-Ref 04] and timed version of API from ESSOR DSP AEP specification [ESSOR-Ref 04] are taken as optional group of APIs for the compact AEP profile for constrained processors. C language library support from Lw Profile from SCA AEP specification [JTNC-Ref 25] is adopted in IRSA.

4.2.2.1.4 Rationale

JTNC, ESSOR and WInnF have published DSP AEP as part of their specifications for waveform component usage. DSP AEP published by three specifications are harmonized with each other with minimal difference between them. All three specifications use restricted set of POSIX APIs.

Since WInnF DSP AEP specification [WInnF-Ref 04] is most elaborate one, IRSA uses this as the base specification.

WInnF Specification defines Lw and ULw AEP profiles for DSP which is adapted by JTNC specification. ESSOR specifies only one DSP AEP profile. Most of the APIs (as part of base specification) which are waiting for event to happen (like message in message queue, semaphore, mutex etc.) are blocking and wait indefinitely till the required event happens. ESSOR DSP AEP specification includes timed version of APIs (which blocks

for specified time). If radio application uses polling mechanism, it can be useful for application to time out and take any error action instead of blocking indefinitely, but due to increase usage of processing elements for implementing these as part of platform, IRSA has included timed version of APIs as optional group of APIs.

JTNC Specification additionally standardizes set of C language support functions (delivered as C library support) as part of Lw AEP whereas WInnF and ESSOR do not cover C language support function as part of AEP. It will be useful to standardize these APIs for enhanced portability of radio application. This will ensure availability of these functions across all platforms.

4.2.2.1.5 Conclusion

IRSA defines compact AEP profile for constrained processors taking complete Lw AEP profile defined in WInnF DSP AEP specification. Additionally, timed version of these APIs is grouped as additional optional API group apart from the optional API groups already defined as part of WInnF DSP AEP specification.

Set of C language library support APIs taken from Lw AEP profile defined in SCA AEP specification [JTNC-Ref 25] is included as part of compact profile for constrained processor AEP.

4.2.2.2 Introduction

IRSA defines Compact AEP Profile for use in constrained processor environment. As part of AEP, a restricted set of POSIX™ complaint APIs are specified, which a complaint radio platform operating system shall support for use by the radio application.

IRSA Compact AEP Profile consists of an essential sub-profile specification which defines set of POSIX APIs related to a set of functional group, which a complaint radio platform shall mandatorily support.

Apart from the functional groups specified in the essential profile, a set of three functional groups is specified as optional sub-profiles, whose compliance is specified by radio platform. These sub-profiles augment the functional groups defined in essential sub-profile. These groups are specified as follows-

- Optional Sub-profile A: This sub-profile contains POSIX APIs for ‘get’ functions related to functional groups defined in the essential profile. These can be used for error detection in radio application code.
- Optional Sub-profile B: This sub-profile contains POSIX APIs for ‘delete’ functions related to functional groups defined in the essential profile. This enables to clean up the operating system resources when another application is installed.
- Optional Sub-profile C: This sub-profile contains POSIX APIs for ‘timed-wait’ functions (which blocks for specified time) related to functional groups defined in

the essential profile. These can be used by radio application to apply poll method instead of blocking indefinitely.

4.2.2.3 Essential Sub-Profile

The Essential sub-profile of the IRSA Compact AEP Profile consists of POSIX API related to following functional groups –

a) Thread/Scheduling functional group
b) Semaphore functional group
c) Mutex functional group
d) Timer functional group
e) Message Queue functional group
f) C language library functional group
g) Conditional Variable functional group (Optional)

4.2.2.3.1 Thread/Scheduling Functional Group

POSIX APIs for this functional group caters to functionality related to management of Threads (or Tasks) and its allocation to CPU (scheduling).
Threads (or Tasks) are independent flow of control within the system.

- Single point of entry for the thread/task when it is created. Implicit deletion of thread happens when it runs to completion without blocking.
- Mainly have three states – Running (currently allocated CPU), blocked (waiting for some resource to become available) and ready to run (ready and waiting for the CPU)
- Scheduling functions manages the allocation of CPU to a thread, based upon the scheduling policy and priority of the thread.

Priority based scheduling (SCHED_FIFO scheduling policy)

- Separate list of ‘ready to run’ thread for each priority.
- List is used to determine the next thread to be allocated to CPU (head of the highest priority non-empty list).
- List is evaluated every time when any thread goes into ready to run state or the current running thread blocks.
- A higher priority thread which has moved into ready to run state preempts currently running lower priority thread.
- Within a single priority, CPU is provided to thread which is waiting in ‘ready to run‘ state for the longest duration.

Summary table for supported POSIX APIs for this functional group is specified below

POSIX API	Function	Restriction
pthread_create()	Thread creation	
pthread_self()	Retrieval of self-thread id	

pthread_attr_init()	Initialization of Thread Attributes object used during thread creation	
pthread_attr_destroy()	Deletion of Thread Attributes Object	
pthread_attr_setdetachstate()	Setting of detach State as part of Thread Attribute object	Only PTHREAD_CREATE_DETACHED value allowed
pthread_attr_setinheritsched()	Setting of Inherit State as part of Thread Attribute object	Only PTHREAD_EXPLICIT_SCHED value allowed
pthread_attr_setschedparam()	Setting of Schedule Param (thread priority) as part of Thread Attribute Object	
pthread_attr_setschedpolicy()	Setting of Schedule Policy as part of Thread Attribute object	Only SCHED_FIFO Value allowed
pthread_attr_setstack()	Setting of Thread Stack Address and Size as part of Thread Attribute Object	
pthread_attr_setstacksize()	Setting of Thread Stack size as part of Thread Attribute Object	

Table 4.1: Thread/Scheduling functional group

Refer section 2.2.2 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group and section 6.2.1 of [WInnF-Ref 04] for rationale of selected POSIX APIs.

4.2.2.3.2 Semaphore Functional Group

POSIX APIs for this functional group relates to semaphore functionality which is shared resource used for synchronization across multiple threads. Semaphore is initialized with a value, which is decremented in each semaphore lock operation (unless it is zero before

the lock operation) and incremented in each semaphore unlock operation (unless some threads are already blocked on the semaphore before the unlock operation). Thread blocks if semaphore value was zero before the lock operation. Similarly, one of the blocked threads (if any) unblocks, during the unlock operation. Unlike mutex, there is no ownership of semaphore, any thread can initiate lock and unlock operation. Support of only unnamed semaphore i.e., reference to the semaphore is already known to all relevant threads using proprietary mechanism.

Summary table for supported POSIX APIs for this functional group is specified below.

POSIX API	Function	Restriction
sem_init()	Creation and initialization of semaphore with initial value	
sem_post()	Semaphore Unlocking operation	
sem_wait()	Semaphore Locking operation	

Table 4.2: Semaphore functional group

Refer section 2.3.2 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group and section 6.2.2 of [WInnF-Ref 04] for rationale of selected POSIX APIs.

4.2.2.3.3 Mutex Functional Group

POSIX APIs for this functional group relates to Mutex (Mutual Exclusion). Mutex is defined to protect simultaneous access of a shared data by different threads. A thread takes the mutex (lock) before accessing the shared resource and then release (unlock) the mutex after completion. Thus, mutex can be used to serialize the access to shared data across multiple threads. A mutex can be owned by only one thread at any point of time, other threads get blocked waiting for the mutex to be available. Only the thread which owns the mutex can release (unlock) the mutex, in which case the highest priority thread waiting (blocked) for the mutex becomes the owner of the mutex and unblocks. Only PTHREAD_MUTEX_NORMAL type non-robust mutex is supported i.e., no counting mutex is available and deadlock can happen if a thread terminates while owning the mutex. To avoid priority inversion (when a mutex is owned by lower priority thread and higher priority thread is blocked on the same mutex), one of the PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT shall be used for mutex protocol attribute.

Summary table for supported POSIX APIs for this functional group is specified below.

POSIX API	Function	Restriction
pthread_mutex_init()	Mutex creation	

pthread_mutexattr_init()	Initialization of Mutex Attributes object used during mutex creation	
pthread_mutexattr_destroy()	Deletion of Mutex attribute object	
pthread_mutexattr_setprotocol()	Setting of protocol value as part of Mutex Attribute object	Only PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT values supported
pthread_mutexattr_setprioceiling()	Setting of priority value as part of Mutex Attribute object	Valid only in case of PTHREAD_PRIO_PROTECT
pthread_mutexattr_settype()	Setting of type value as part of Mutex Attribute object	Only PTHREAD_MUTEX_NORMAL value is supported
pthread_mutex_lock()	Locking the mutex i.e., request to acquire the mutex. Calling thread blocks if mutex not available	
pthread_mutex_unlock()	Unlocking the mutex i.e. releasing the mutex (only by the thread which is currently owning the mutex	

Table 4.3: Mutex functional group

Refer section 2.4.2 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group and section 6.2.4 of [WInnF-Ref 04] for rationale of selected POSIX APIs.

4.2.2.3.4 Message Queue Functional Group

POSIX APIs for this functional group relates to Message Queue which is used for sending message buffers between threads. Message queue handle obtained during this process is used by the thread to send/receive message on that queue. While sending/queuing the message in the message queue, the sender specifies the message buffer pointer, message length and the priority of the message. Message is inserted into the queue at

the position indicated by message priority if space is available in the message queue. While receiving/dequeuing the message from message queue, the message from the head of the queue is copied to the buffer provided by the receiver. Since both during queueing/dequeuing operation, complete message buffer is copied, to avoid huge copy operation, application can just pass the pointer of the actual message buffer as message and sizeof(pointer) as message length during send/receive operation. In this case only the pointer is copied during send/receive. Actual message buffer can be accessed using the copied pointer.

Summary table for supported POSIX APIs for this functional group is specified below.

POSIX API	Function	Restriction
mq_open()	Open the message queue with the given name for sending/receiving on the message queue. It provides the calling thread with the message descriptor of the message queue, which is used for sending/receiving on that queue.	• mode parameter is ignored. • O_NONBLOCK flag is never set. • Message queue name is treated as simple string without any meaning attached to <slash> character.
mq_receive()	Retrieve message from head of the message queue by copying message from the queue to the supplied buffer. Thread blocks if no message present in the queue, till message arrives on that message queue	
mq_send()	Queue the message to the message queue by the copying the supplied message buffer to the message queue position indicated by supplied message priority. Thread blocks if no space in the message queue till space available	Blocking call since O_NONBLOCK flag is not set during message queue creation

Table 4.4: Message Queue functional group

4.2.2.3.4.1 POSIX API: mq_open()

POSIX standard is fully applicable with following restriction/clarification –

- While creating the message queue, mode parameter is ignored, since there is no access checking mechanism.
- As an option radio platform can chose to restrict the value of mq_msgsize parameter (as part of mq_attr for the message queue) equal to sizeof(void *). Radio Platform vendor can specify whether radio platform enforce the restriction or not.
- O_NONBLOCK flag is never set and ignored by the platform.

- Since there is no filesystem support, message queue name is treated as simple string without any meaning attached to <slash>

4.2.2.3.4.2 POSIX API: mq_receive()

POSIX standard is fully applicable with following restriction/clarification-

- Blocking call since O_NONBLOCK flag is not set during message queue creation.

4.2.2.3.4.3 POSIX API: mq_send()

POSIX standard is fully applicable with following restriction/clarification-

- Blocking call since O_NONBLOCK flag is not set during message queue creation.
- As an option radio platform may choose to omit handling of message priority. Radio platform in this case ignores the msg_prio field. Radio Platform vendor can specify whether radio platform implements the option or not.

4.2.2.3.5 Timer Functional Group

POSIX APIs for this functional group relates to timer function which is used to timing event for the thread. Timer is created by the thread, specifying the clock to be used for timing base. Timerid created during this process is used by subsequent starting/stopping of timer. Application starts the created timer specifying either interval time or the absolute time for the expiry. Application can also specify whether timer shall be a periodic timer or one-shot timer. At timer expiry, callback function (provided during timer creation) is called from the context of a system thread (created during timer creation). Platform vendor shall publish the priority of the system thread, so to determine preemption of currently running thread. Platform vendor can publish any other mechanism used, to indicate timer expiry.

Summary table for supported POSIX APIs for this functional group is specified below.

POSIX API	Function	Restriction
timer_create()	Timer creation by the thread. Timerid provided during this operation is used for subsequent starting / stopping of timer. Application provides the notification function and its argument value, to be called by the system, on timer expiry	For *clockid* parameter value , only CLOCK_REALTIME is used For *sigev_notify* member of *evp* param (sigevent struct), only value of SIGEV_THREAD is used i.e., a notification function is called in the context of a system thread.
timer_settime()	To start(arm)/stop (disarm) a timer earlier created.	
clock_getres()	To get the resolution of the clock for the timer create	Same value of *clockid* parameter as supported during timer create can be used

clock_gettime()	To get the current time of the clock	For *clockid* parameter value, only CLOCK_REALTIME is used.

Table 4.5: Timer functional group

Refer section 2.6.1, section 2.6.2, section 2.6.3 and section 2.6.4 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group.

4.2.2.3.6 C Language Library Functional Group

Set of POSIX API part of C language library are included in this profile. Platform may provide additional C language library functions and radio application may use them, with a potential risk of future portability to another platform.

4.2.2.3.6.1 C Language-Specific Support Services Function

All functions of mentioned as Mandatory (MAN) for the LwAEP profile in Table 22 of [JTNC-Ref 25] are included in essential sub-profile of the IRSA compact AEP profile for constrained processors.

4.2.2.3.6.2 C Language-Specific Mathematical Function

All functions of mentioned as Mandatory (MAN) for the LwAEP profile in Table 23 of [JTNC-Ref 25] are included in essential sub-profile of the IRSA compact AEP profile for constrained processors.

4.2.2.3.7 Optional Functional group

Apart from the functional groups defined above, radio platform can optionally support following functional group, support for which is specified by platform vendor –

- Conditional Variable Functional Group

4.2.2.3.7.1 Conditional Variable Functional Group

POSIX APIs for this functional group relates to condition variable used for synchronization between multiple threads when a particular condition is met. A thread blocks while calling the ‘wait’ function (waiting for condition to be met), other thread can unblock the threads by calling ‘signal’ function (indicating condition has been met). A condition variable is dynamically associated with a mutex. The calling thread needs to acquire the mutex before calling the wait/signal functions. The mutex is automatically released when thread blocks on wait function and get re-acquired when it comes out of the wait function.

Summary table for supported POSIX APIs for this functional group is specified below.

POSIX API	Function	Restriction
pthread_cond_init()	Initializing the condition variable before its usage	Vaue of *attr* parameter (pthread_condattr_t *) while using the function shall be NULL
pthread_cond_broadcast()	Signal condition has been met. Wake up all	

	waiting threads	
pthread_cond_signal()	Signals condition has been met. Wake up one of the waiting threads.	
pthread_cond_wait()	Waiting for condition to be met. Calling thread blocks	

Table 4.6: Conditional Variable functional group

Refer section 2.7.1, section 2.7.2, and section 2.7.3 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group.

4.2.2.4 Optional Sub-profile A

POSIX APIs for sub-profile A consists of 'get' APIs related to functional groups which are part of essential sub-profile. Please refer to section 6.3 of [WInnF-Ref 04] for rationale for sub-profile A.

4.2.2.4.1 Thread/Scheduling Functional Group

Summary table for supported POSIX APIs for sub-profile A - Thread/Scheduling Functional Group is specified below –

POSIX API	Function	Restriction
pthread_attr_getdetachstate()	Get the value of detachstate as part of Thread attribute object	
pthread_attr_getinheritsched()	Get the value of inheritsched as part of Thread attribute object	
pthread_attr_gettschedparam()	Get the value of schedparam as part of Thread attribute object	
pthread_attr_gettschedpolicy()	Get the value of schedpolicy as part of Thread attribute object	
pthread_attr_getstacksize()	Get the value of stack size as part of Thread attribute object	
pthread_attr_getstackaddr()	Get the value of stack address as part of Thread attribute object	
pthread_attr_getguardsize()	Get the value of Guard size as part of Thread attribute object	
pthread_attr_setguardsize()	Set the Guard size as part of Thread attribute object	

Table 4.7: Sub-profile A - Thread/Scheduling functional group

Refer section 3.2.1 of [WInnF-Ref 04] and 3.2.3 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group.

4.2.2.4.2 Semaphore Functional Group

Summary table for supported POSIX APIs for sub-profile A– Semaphore Functional Group is specified below-

POSIX API	Function	Restriction
sem_getvalue()	Get the current value of semaphore without effecting its value (no lock/unlock operation)	

Table 4.8: Sub-profile A - Semaphore functional group

Refer section 3.2.4 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group.

4.2.2.4.3 Mutex Functional Group

Summary table for supported POSIX APIs for sub-profile A – Mutex Functional Group is specified below-

POSIX API	Function	Restriction
pthread_mutexattr_getprotocol()	Get the value of protocol as part of Mutex attribute object	
pthread_mutexattr_getprioceiling()	Get the value of priority ceiling as part of Mutex attribute object	
pthread_mutexattr_gettype()	Get the value of mutex type as part of Mutex attribute object	

Table 4.9: Sub-profile A - Mutex functional group

Refer section 3.2.2 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group.

4.2.2.4.4 Message Queue Functional Group

Summary table for supported POSIX APIs for sub-profile A – Message Queue Functional Group is specified below-

POSIX API	Function	Restriction
mq_getattr()	Get the current value message queue attribute for the current message queue. The attribute can be used to get the message size parameter	

Table 4.10: Sub-profile A - Message Queue functional group

4.2.2.4.4.1 POSIX API: mq_getattr()

POSIX standard is fully applicable.

Radio application uses this API to get the message size attribute (mq_msgsize) of the message queue. Buffer size provided by the application to receive the message shall be at least equal to the message size attribute of the message queue.

4.2.2.4.5 Timer Functional Group

Summary table for supported POSIX APIs for sub-profile A - Timer Functional Group is specified below-

POSIX API	Function	Restriction
timer_gettime()	To get the current status of timer (timerid) – armed/not armed and time to expiry, if armed	

Table 4.11: Sub-profile A - Timer functional group

Refer section 3.2.5 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group.

4.2.2.5 Optional Sub-profile B

POSIX APIs for sub-profile B consists of ‘delete/close’ APIs related to functional groups which are part of the essential sub-profile. Please refer to section 6.4 of [WInnF-Ref 04] for rationale for this sub-profile. APIs defined for this sub-profile are used for clearing of the resources when radio application is inactive. In case of support of optional conditional variable functional group, sub-profile B includes the ‘delete’ API related to Conditional variable functional group.

4.2.2.5.1 Thread/Scheduling Functional Group

Summary table for supported POSIX APIs for sub-profile B - Thread/Scheduling Functional Group is specified below-

POSIX API	Function	Restriction
pthread_exit()	To cleanup and delete the calling thread	

Table 4.12: Sub-profile B - Thread/Scheduling functional group

Refer section 4.2.1 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group.

4.2.2.5.2 Semaphore Functional Group

Summary table for supported POSIX APIs for sub-profile B - Semaphore Functional Group is specified below-

POSIX API	Function	Restriction

sem_destroy()	Delete/Destroy the semaphore. Semaphore cannot be used unless initialized again	

Table 4.13: Sub-profile B - Semaphore functional group

Refer section 4.2.2 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group.

4.2.2.5.3 Mutex Functional Group

Summary table for supported POSIX APIs for sub-profile B - Mutex Functional Group is specified below-

POSIX API	Function	Restriction
pthread_mutex_destroy()	Deleting/ Destroying the mutex and releasing the associated OS resources	

Table 4.14: Sub-profile B - Mutex functional group

Refer section 4.2.4 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group.

4.2.2.5.4 Message Queue Functional Group

Summary table for supported POSIX APIs for sub-profile B - Message queue Functional Group is specified below-

POSIX API	Function	Restriction
mq_close()	Close the message queue	

Table 4.15: Group B - Message Queue functional group

Refer section 4.2.3 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group.

4.2.2.5.5 Timer Functional Group

Summary table for supported POSIX APIs for sub-profile B - Timer Functional Group is specified below-

POSIX API	Function	Restriction
timer_delete()	To destroy the earlier created timerid. If the timer is armed (running), it will automatically be stopped	

Table 4.16: Sub-profile B - Timer functional group

Refer section 4.2.5 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group.

4.2.2.5.6 Conditional Variable Functional Group

Summary table for supported POSIX APIs for sub-profile B - Conditional Variable Group

is specified below-

POSIX API	Function	Restriction
pthread_cond_destroy()	Destroy the condition variable	

Table 4.17: Sub-profile B - Conditional Variable functional group

Refer section 4.2.6 of [WInnF-Ref 04] for details of selected POSIX APIs support for this functional group. This API is part of sub-profile B profile only if CONDITIONAL VARIABLE functional group is supported.

4.2.2.6 Optional Sub-profile C

POSIX APIs for sub-profile C consists of 'timed wait' APIs related to functional groups which are part of essential sub-profile. Sub-profile C APIs are used by application to specify time out for blocking calls, so that application is not indefinitely blocked.

4.2.2.6.1 Semaphore Functional Group

Summary table for supported POSIX APIs for sub-profile C - Semaphore Functional Group is specified below-

POSIX API	Function	Restriction
sem_trywait()	If the semaphore is not already locked, it is locked, else thread returns without any effect on semaphore value	
sem_timedwait()	Request for locking the semaphore. Thread gets blocked till the specified time if semaphore cannot be locked.	

Table 4.18: Sub-profile C - Semaphore functional group

4.2.2.6.1.1 POSIX API: sem_trywait()

POSIX standard is fully applicable.

4.2.2.6.1.2 POSIX API: sem_timedwait()

POSIX standard is fully applicable.

4.2.2.6.2 Mutex Functional Group

Summary table for supported POSIX APIs for sub-profile C - Mutex Functional Group is specified below-

POSIX API	Function	Restriction
pthread_mutex_trylock()	Non-blocking version of Mutex lock. Thread acquires mutex, if available, else function returns without blocking	
pthread_mutex_timedlock()	Request to acquire the mutex. Thread gets blocked till the specified time, and unblocks if the mutex is not available within the time	

Table 4.19: Sub-profile C - Mutex functional group

4.2.2.6.2.1 POSIX API: pthread_mutex_trylock ()

POSIX standard is fully applicable.

4.2.2.6.2.2 POSIX API: pthread_mutex_timedlock ()

POSIX standard is fully applicable.

4.2.2.6.3 Message Queue Functional Group

Summary table for supported POSIX APIs for sub-profile C - Message Queue Functional Group is specified below-

POSIX API	Function	Restriction
mq_timedreceive()	Retrieve message from message queue. Thread blocks till the specified time, if no message in the message queue arrives	

Table 4.20: Sub-profile C - Message Queue functional group

4.2.2.6.3.1 POSIX API: mq_timedreceive()

POSIX standard is fully applicable.

4.2.2.7 Capabilities

Set of capability attributes are defined for the Constrained Processor AEP, which specifies the radio platform support for different optional features specified for Constrained Processor AEP.

Attribute Name	Description
SUBPROFILE_A_SUPPORT	Platform supports POSIX APIs specified as part of optional Sub-profile A
SUBPROFILE _B_SUPPORT	Platform supports POSIX APIs specified as part of optional Sub-profile B
SUBPROFILE _C_SUPPORT	Platform supports POSIX APIs specified as part of optional Sub-profile C
MQ_MSG_SIZE_RESTRICTION	Platform restricts mq_msgsize = sizeof(void *) for message queue functional group
MQ_PRIORITIES_SUPPORT	Platform support message priorities for Message queue
COND_VARIABLE_SUPPORT	Platform supports Conditional Variable functional group. If conditional variable functional group is supported, then support of sub-profile B indicates support of the corresponding group of APIs for conditional variable functional group also.

Table 4.21: Constrained Processor AEP capabilities

4.2.3 Tiny AEP Profile for Constrained Processors

AEP defines the execution environment for the waveform components running on

contrained a processing element. AEP is a set of APIs, which a complaint radio platform is guaranteed to provide, and a complaint waveform component restricts its usage within the defined set, for ease of portability. The tiny AEP profile for constrained processors defined herein, relates to the environment presented to waveform components running on constrained processor e.g., DSP, real-time processors without OS.

4.2.3.1 Approach

For Ultra-Constrained processors, where having RTOS/microkernel is an overhead, application uses bare-metal implementation. Baremetal application can be viewed as set of event handler functions, which are called from the main loop. Different task functions which are triggered/scheduled sequentially in a tight loop, can be viewed as tasks that run to completion without blocking, hereinafter, referred to as Run-To-Completion (RTC) tasks. To ease porting of such applications across different SDRs, IRSA introduces a reduced set of APIs under the tiny AEP profile. This profile is applicable to a processor which works under following constraints/conditions.

- No Process Concept, all are Run-To-Completion (RTC) tasks.
- Complete memory shared across all RTC tasks (single address space).
- No file system concept.
- No RTOS/microkernel for thread scheduling.
- Applications running are hard real-time applications with strict timing deadline.

4.2.3.1.1 Candidate API Set

None

4.2.3.1.2 Selected API Set

Not Applicable

4.2.3.1.3 Modifications

Not Applicable

4.2.3.1.4 Rationale

Due to complexity the waveform development for SDR and requirements of ultra-low latency & real-time performance, it is possible that waveform developers choose to deploy some critical components on a bare-metal/OS-less environment trading off portability for performance. In such scenarios it will be useful to define a set of APIs to abstract the waveform component from the underlying hardware even in the absence of OS/microkernel to achieve some level of portability. These APIs should have negligible penalty in terms of latency and performance so that the real-time performance is not impacted.

4.2.3.1.5 Conclusion

IRSA introduces tiny AEP profile to ease porting of applications across for constrained processors having no OS/mircrokernel.

4.2.3.2 Introduction

IRSA defines Tiny AEP Profile for use in constrained processor environment consisting of restricted API related to following functional groups –

a) Thread/Scheduling functional group
b) Message Queue functional group
c) Timer functional group
d) C language library functional group

The names of various APIs under the above function groups are same as POSIX APIs, despite the modified semantics as specified in subsequent sub-sections to enhance portability.

The application entry point shall be a function named application_main that is always created and has the highest priority. It shall be called by the main event loop by default. The main_applicatin function should distinguish between first invocation from subsequent ones so that it can do the registration of RTCs only once.

4.2.3.3 Thread/Scheduling Functional Group

The RTC tasks can be registered with the main loop using API from this functional group, so that they will be called from the main loop. Application needs to ensure that these functions do not block and hold processor resources (relinquish control after completion). Prioriy determines the relative order of their invocation from main loop. The application_main function controls registration of other RTC tasks.

Summary table for supported APIs for this functional group is specified below

API	Function	Restriction
pthread_create()	Registration of the specified RTC task with the main loop	
pthread_attr_init()	Initialization of Thread Attributes object used during thread creation	Only Priority attributes is used
pthread_attr_setschedparam()	Setting of Schedule Param (thread priority) as part of Thread Attribute Object	
pthread_kill()	De-registration of the specified RTC task from	

	the main loop	

Table 4.22: Thread/Scheduling Functional Group APIs

4.2.3.4 Message Queue Functional Group

APIs for this functional group relate to Message Queue which is used for sending message buffers between RTC tasks. These can be used to create the message queues during the initialization. The message descriptor can be shared across sender and receiver RTC tasks. Message queue handle obtained during this process is used by the RTC tasks to send/receive message on that queue. While sending/queuing the message in the message queue, the sender specifies the message buffer pointer, message length and the priority of the message. Message is inserted into the queue at the position indicated by message priority if space is available in the message queue.

Since both during queueing/dequeuing operation, complete message buffer is copied, to avoid huge copy operation, application can just pass the pointer of the actual message buffer as message and sizeof(pointer) as message length during send/receive operation. In this case only the pointer is copied during send/receive. Actual message buffer can be accessed using the copied pointer.

Summary table for supported APIs for this functional group is specified below

API	Function	Restriction
mq_open()	Open the message queue with the given name for sending/receiving on the message queue. It provides the calling function handler with the message descriptor of the message queue, which is used for sending/receiving on that queue.	• mode parameter is ignored. • mq_msgsize parameter as part of mq_attr for the message queue shall be fixed and equal to sizeof(void *) • O_NONBLOCK flag is always set, since all task runs in non blocking manner • Message queue name is treated as simple string without any meaning attached to <slash> character.
mq_receive()	Retrieve message from head of the message queue by copying message from the queue to the supplied buffer.	Always a non blocking call since O_NONBLOCK flag is always set during message queue creation
mq_send()	Queue the message to the message queue by copying the supplied	No priority support. Priority parameter is ignored by the

	message buffer to the message queue	platform Always a non-blocking call since O_NONBLOCK flag is always set during message queue creation

Table 4.23: Message Queue Functional Group APIs

4.2.3.5 Timer Functional Group

APIs in this functional group relates to timer functions used for timing events for RTC tasks. One or many timers are created by the application_main function, specifying the clock to be used for timing base. Timerid created during this process is used by subsequent starting/stopping of timer. RTC task starts/stops the created timer specifying either interval time or the absolute time for the expiry. Application can also specify whether timer shall be a periodic timer or one-shot timer. At timer expiry, callback function (provided during timer creation) is called from the main loop. Calback function is called as the highest priority after the application_main function.

Summary table for supported APIs for this functional group is specified below.

API	Function	Restriction
timer_create()	Timer creation during initialization. Timerid provided during this operation is used for subsequent starting / stopping of timer. Application provides the notification function and its argument value, to be called by the main loop, on timer expiry	For *clockid* parameter value , only CLOCK_REALTIME is used For *sigev_notify* member of *evp* param (sigevent struct), only value of SIGEV_THREAD is used i.e., a notification function is called in the context of a system
timer_settime()	To start(arm)/stop (disarm) a timer earlier created.	
clock_getres()	To get the resolution of the clock for the timer create	Same value of *clockid* parameter as supported during timer create can be used
clock_gettime()	To get the current time of the clock	For *clockid* parameter value, only CLOCK_REALTIME is used.

Table 4.24: Timer Functional Group APIs

4.2.3.6 C Language Library Functional Group

A Set of POSIX APIs that are part of C language library have been included in the tiny AEP profile. Platform may provide additional C language library functions and radio application may use them, with a potential impact on portability to another platform in future. Following subset of categories as used in the compact AEP profile is included in tiny AEP Profile

- Memory Allocation and manipulation functions as a subset of those specified

in section 4.2.2.3.6.1
- Mathematical functions, as specified in 4.2.2.3.6.2
- rand() and srand()
- Any other utility functions as optional

4.2.4 Resource Support

This capability enables any waveform component deployed on CP (CP Resource) to be controlled by CF or other platform components via the SCA Base Application interfaces.

4.2.4.1.1 Approach

4.2.4.1.1.1 Candidate API Set

Section 4.4.4 of ESSOR Architecture Introductory Document - [ESSOR-Ref 01]

4.2.4.1.1.2 Selected API Set

Section 4.4.4 of ESSOR Architecture Introductory Document - [ESSOR-Ref 01]

4.2.4.1.1.3 Modification

Modifications to replace SCA CF::Resource interface with SCA BaseApplication interfaces as per SCA 4.1 have been carried out. Moreover, explicit specification of port connection support has been dropped because the CF::PortAccessor interface is already part of Base Application Interfaces inherited and implemented by ApplicationComponent.

4.2.4.1.1.4 Rationale

The modifications were carried out to align the approach with the SCA 4.1 specification.

4.2.4.1.1.5 Conclusion

Section 4.4.4 of ESSOR Architecture Introductory Document - [ESSOR-Ref 01] has been adopted with modifications. This capability is realized through a SCA ApplicationComponent proxy deployed on GPP which forwards/receives the requests/responses to/from the CP resource via CP execution environment as per the SCA BaseApplication interfaces.

4.3 Platform API Access From Same Processing Node

The mechanism to access the platform APIs by application component when the platform service/facility and application component are deployed on the same processing node.

4.3.1.1 Approach

4.3.1.1.1 Candidate API Set

- Facility PSMs [WInnF-Ref 09],[WInnF-Ref 10]

4.3.1.1.2 Selected API Set

Facility PSMs [WInnF-Ref 09],[WInnF-Ref 10]

4.3.1.1.3 Modification
None

4.3.1.1.4 Rationale
None

4.3.1.1.5 Conclusion
Mechanism of access of platform APIs by the application components has been adopted without any modification.

4.4 Platform API Access From Different Processing Node

The mechanism to access the platform APIs by application component when the platform service/facility and application component are deployed on the different processing node.

4.4.1.1 Approach

4.4.1.1.1 Candidate API Set
- Section 4.3.3 of ESSOR Architecture Introductory Document - [ESSOR-Ref 01]
- Section 4.4.5 of ESSOR Architecture Introductory Document - [ESSOR-Ref 01] considered for access by constrained processors
- WInnF Facility PSMs [WInnF-Ref 09],[WInnF-Ref 10]

4.4.1.1.2 Selected API Set
WInnF Facility PSMs [WInnF-Ref 09],[WInnF-Ref 10]

4.4.1.1.3 Modification
None

4.4.1.1.4 Rationale
First candidate API set assumes the ‘CORBA everywhere’ approach which is not preferred in IRSA. From the second and third candidate, the PSM approach has been adopted because of its straight forward nature and availability of full set of services provided by facility using this approach. Moreover, it does not involve exchanging MHAL messages by the application component.

4.4.1.1.5 Conclusion
WInnF Facility PSMs [WInnF-Ref 09],[WInnF-Ref 10] has been adopted without any modification.

5 Transfer mechanism

5.1 CORBA based Middleware

As per section 2.2.4 CORBA PSM has been adopted by IRSA without any modifications.

5.2 Middleware 2

The SCA 4.1 has given core specification as PIM and separated the technology specific models into various appendices viz. CF interfaces defined using CORBA IDL, PSM for SCA Domain Profiles using XML DTD files, and platform specific transport and technology mode using CORBA. SCA is accommodating as far as choosing the implementation technology is concerned provided the interface definitions are preserved across the realization technologies. Various possible technologies for transfer mechanisms have also been provided by the SCA 4.1 specification but the PSM has only been provided for CORBA. CORBA is well suited for Heterogenous Embedded Distributed Systems (HEDS). However, with the advancements in semi-conductor technology the world of computing is increasingly moving towards multi-core SoCs which are packing more and more computing power on a single chip. For dedicated hardware like SDR, the OEMs also prefer realizing single channel or single security domain using one SoC due to various reasons pertaining to hardware design. Technically, it is feasible to realize all security domains on one SoC. There is very low probability of having multiple separate processors on one security domain in SDR hardware in future. Therefore, it is imperative to have light-weight and relatively simple alternative to CORBA for multi-core processors. The light-weight transfer mechanism is even more justified for SWaP constrained form-factors of SDR like manpack, handheld etc.

IRSA has defined different profiles for different types of SDRs and not all of them are HEDS or are not using multiple GPPs in one security domain. Alternative transfer mechanism can be utilized in those SDRs. IRSA shall release PSM for the alternate transfer mechanism after reference implementation and its performance evaluation.

5.3 Raw Middleware

5.3.1 Modem Hardware Abstraction Layer

5.3.1.1 Approach

5.3.1.1.1 Candidate API Set

JTNC Standard MHAL API [JTNC-Ref 17]

5.3.1.1.2 Selected API Set

Section A to D of JTNC Standard MHAL API [JTNC-Ref 17] Section E titled “MHAL RF CHAIN COORDINATOR API EXTENSION” has been excluded.

5.3.1.1.3 Modification

No modification except exclusion of section E.

5.3.1.1.4 Rationale

Above referenced API is sufficient to cater requirement of platform and waveform

components. Section E titled “MHAL RF CHAIN COORDINATOR API EXTENSION” has been excluded because similar functionality has been adopted from WInnF Transceiver Facility, [WInnF-Ref 15] to [WInnF-Ref 19].

5.3.1.1.5 Conclusion

JTNC Standard MHAL API is adopted except the MHAL RF Chain Coordinator API Extension.

5.3.2 MHAL On Chip Bus

5.3.2.1 Approach

5.3.2.1.1 Candidate API Set

JTNC Standard MHAL on Chip Bus API [JTNC-Ref 16]

5.3.2.1.2 Selected API Set

Section A to D of JTNC Standard MHAL on Chip Bus API [JTNC-Ref 16] Section E titled “MOCB RF CHAIN COORDINATOR (RFC) API EXTENSION” has been excluded.

5.3.2.1.3 Modification

No modification except exclusion of section E.

5.3.2.1.4 Rationale

Above referenced API is sufficient to cater requirement of platform and waveform components. Section E titled “MOCB RF CHAIN COORDINATOR (RFC) API EXTENSION” has been excluded because similar functionality has been adopted from WInnF Transceiver Facility, [WInnF-Ref 15] to [WInnF-Ref 19].

5.3.2.1.5 Conclusion

JTNC Standard MHAL on Chip Bus API is adopted except the MHAL RF Chain Coordinator (RFC) API Extension.

6 IRSA Radio Profiles

The following profiles have been created considering different form-factor constraints. These profiles shall ensure same environment over a similar form-factor SDRs from different OEMs. This standardized environment consists of SCA framework, platform APIs, execution environment and transfer mechanism. The objective of profiles is to provide seamless portability of waveforms across same/similar form-factor SDRs. These profiles are applicable only to the computing resources of the SDR under consideration.

6.1 Heterogenous SDR Profile

Heterogenous profile is applicable to those categories of SDRs wherein a security domain or a channel within an SDR is realized using multiple general-purpose processors (single or multi-core) along with constrained processor(s) and FPGA(s). A security domain or a channel in an SDR hardware can be realized by utilizing any combination of the processing elements GPP, CP & FPGA. Figure 6.1 depicts one security domain or one channel in a sample hardware configuration.

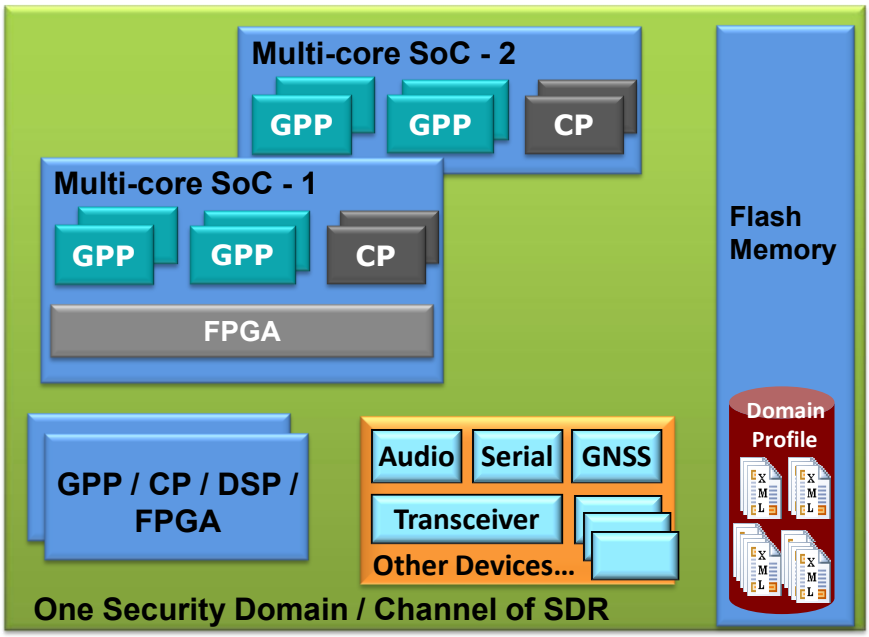

Figure 6.1 : Heterogenous SDR Profile – Sample Hardware Configuration

Following PIM and PSM components of IRSA specification are mandatory for Heterogenous SDR Profile.

6.1.1 PIM Components

1. Modified SCA full profile as per section 2.2.5.3.3
2. Platform API – As per hardware resources present
3. Execution Environment
    a. Each of the processing elements can have any of the following AEP profiles

[JTNC-Ref 25] depending on the resource capability
   i. SCA AEP profile
   ii. Constrained Processor Execution Environment as per section 4.2
4. Transfer mechanism PIM
   a. Full PIM IDL Profile[JTNC-Ref 30] [WInnF-Ref 05]
   b. MHAL/MOCB for communication among software components of GPP, constrained processors and FPGA

6.1.2 PSM Components

1. Platform API PSM
   a. SCA
   b. Native C++
   c. FPGA
2. Transfer mechanism PSM
   a. Full CORBA Profile [JTNC-Ref 31]
   b. Alt-M
   c. MHAL/MOCB or both

Figure 6.2 highlights the various interfaces and PSMs applicable for different processing elements like GPP, CP & FPGAs in this SDR profile. It also depicts various possible scenarios pertaining to applicable interfaces between processing elements in a sample hardware configuration where the Heterogenous SDR Profile shall be applicable.

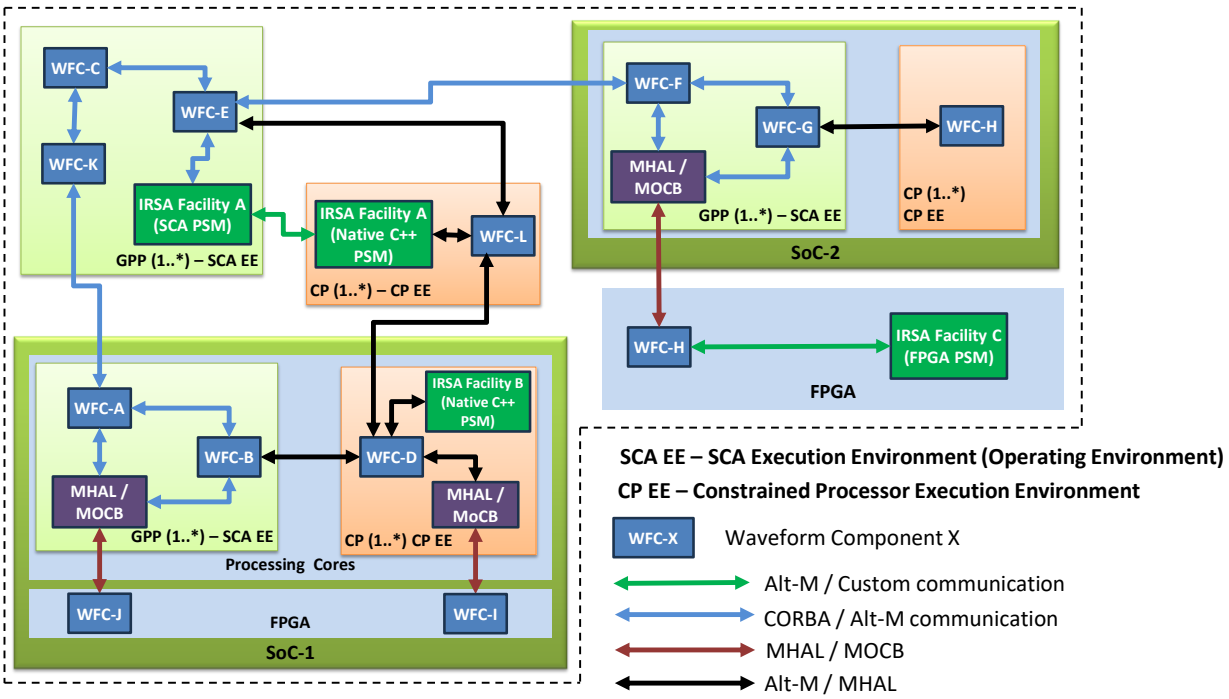

Figure 6.2 : Heterogenous SDR Profile – Sample Hardware Configuration Interface View

6.1.3 Conformance

"SDR *xxx* is conformant to IRSA Heterogeneous SDR profile with the presence of

following components conforming to corresponding IRSA criteria”

6.2 Homogenous SDR Profile

Homogenous profile is applicable to those categories of SDRs wherein a security domain or a channel within an SDR is realized using one general-purpose processor (single or multi-core) along with constrained processor(s) and FPGA(s). This profile is also applicable to those multi-channel SDRs where each channel is independent and fulfills above criteria. Figure 6.3 depicts one security domain or one channel in three sample hardware configurations.

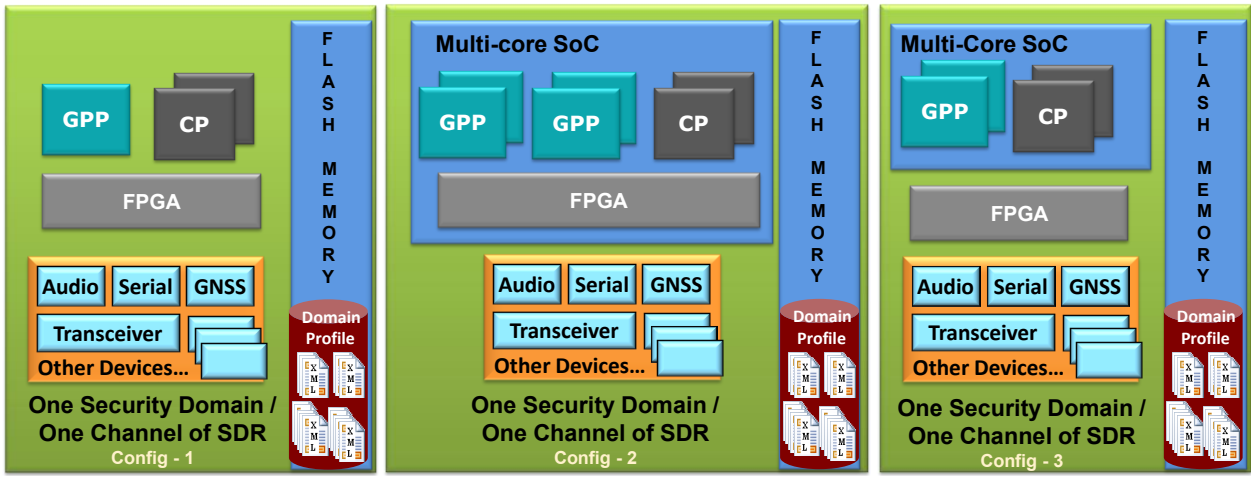

Figure 6.3 : Homogenous SDR Profile – Three Sample Hardware Configurations

Following PIM and PSM components of IRSA specification are mandatory for Homogenous SDR Profile.

6.2.1 PIM Components

1. Modified SCA Medium profile as per section 2.2.5.3.2
2. Platform API – As per hardware resources present
3. Execution Environment
    a. SCA AEP profile
    b. Constrained Processor Execution Environment as per section 4.2
4. Transfer mechanism
    a. Alt-M for communication among software components executing in OS/RTOS environment
    b. Alt-M/MHAL/MOCB for communication among software components of GPP, constrained processors and FPGA

6.2.2 PSM Components

1. Platform API PSM
    a. SCA
    b. Native C++
    c. FPGA

2. Transfer mechanism PSM
    a. Alt-M
    b. MHAL/MOCB or both

Figure 6.4 highlights the various interfaces and PSMs applicable for different processing elements like GPP, CP & FPGAs in this SDR profile. It also depicts various possible scenarios pertaining to applicable interfaces between processing elements in three sample hardware configurations where the Homogenous SDR Profile shall be applicable.

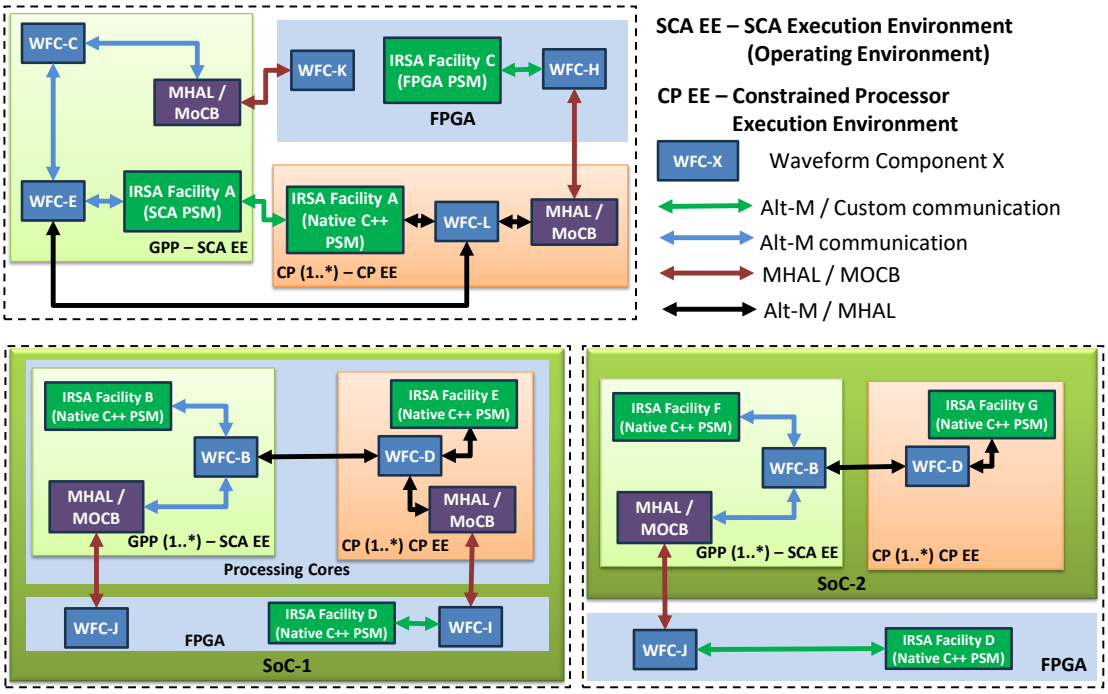

Figure 6.4 : Homogenous SDR Profile - Three Sample Hardware Configurations' Interface View

6.2.3 Conformance

"SDR *xxx* is conformant to IRSA homogeneous SDR profile with the presence of following components conforming to corresponding IRSA criteria"

6.3 Single Processor SDR profile

Single Processor profile is applicable to those categories of SDRs wherein entire SDR hardware is realized using one general-purpose processor (single or multi-core) along with constrained processor(s) and FPGA(s). Figure 6.5 depicts entire SDR computing resources in three sample hardware configurations. The figure is same as that of homogenous SDR profile but in this case, it represents all processing elements available in the SDR instead of one security domain or one channel.

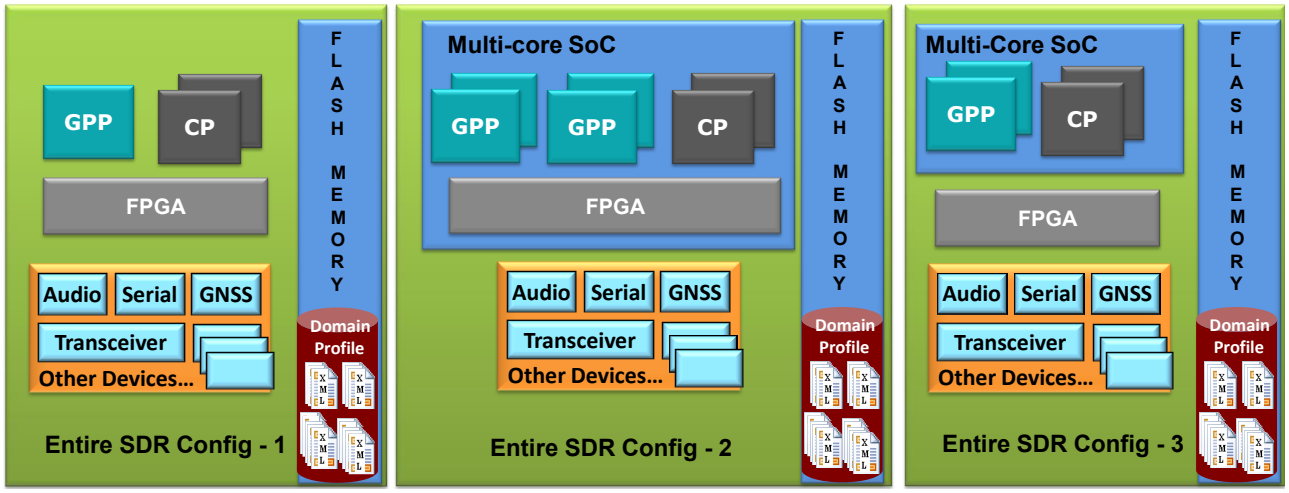

Figure 6.5 : Single Processor SDR Profile – Three Sample SDR Configurations

Following PIM and PSM components of IRSA specification are mandatory for single processor SDR Profile.

6.3.1 PIM Components

1. Modified SCA Lightweight profile as per section 2.2.5.3.1
2. Platform API – As per hardware resources present
3. Execution Environment
    a. SCA AEP profile
    b. Constrained Processor Execution Environment as per section 4.2
4. Transfer mechanism
    a. Alt-M for communication among software components executing in OS/RTOS environment
    b. Alt-M/MHAL for communication among software components of GPP, constrained processors and FPGA

6.3.2 PSM Components

2. Platform API PSM
    a. SCA
    b. Native C++
    c. FPGA
3. Transfer mechanism PSM
    a. Alt-M
    b. MHAL

Figure 6.6 highlights the various interfaces and PSMs applicable for different processing elements like GPP, CP & FPGAs in this SDR profile. It also depicts various possible scenarios pertaining to applicable interfaces between processing elements in three sample hardware configurations where the Single Processor SDR Profile shall be applicable. The figure is same as that of homogenous SDR profile but in this case, it

represents all processing elements available in SDR instead of one security domain or a channel.

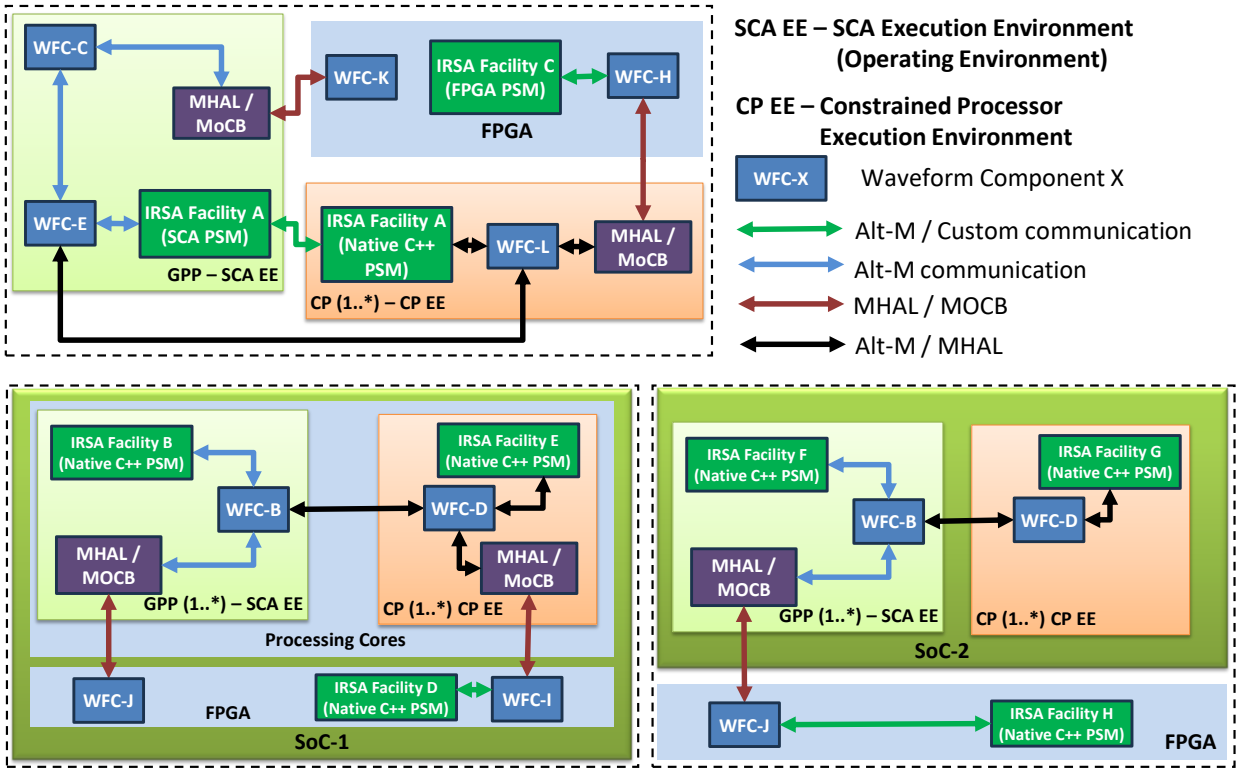

Figure 6.6 : Single Processor SDR Profile – Three Sample SDR Configurations' Interface View

6.3.3 Conformance

"SDR *xxx* is conformant to IRSA Single Processor SDR profile with the presence of following components conforming to corresponding IRSA criteria"

7 Waveform Portability

7.1 Portability Metrics

Utlizing the concepts of portability and hospitality defined in [WInnF-Ref 09], IRSA introduces two quantitative metrics for portability namely Waveform Portability Index (WPI) and Platform Hospitality Index (PHI) defined as follows.

7.1.1 Waveform Portability Index (WPI)

Waveform Portability Index represents the degree/extent to which a waveform conforms to the IRSA specification. It measures the reduction in effort required to port a waveform from one IRSA compliant SDR system to another. A higher WPI indicates higher portability (less porting effort), while a lower WPI suggests a lower portability (higher porting effort) of a waveform. It is a weighted sum of compliance to different constituents (parameters) of IRSA specified in this document.

In due course of time, the names of various parameters shall be defined along with their weights and criterion for assigning values to them. A table in following format for the calculation of WPI shall be incorporated in IRSA.

Table 7.1 : Parameters of Waveform Portability Index (WPI)

S.No.	Parameter	Description	Weight (%)	Value (0-10)
1.				
2.				
:				
:				
N				
		Total	100	

The WPI will be determined using the following formula:

WPI = ∑ (Parameter Value x Parameter Weight) / 100

The final index ranges from 0 (hard to port) to 10 (highly portable)

7.1.2 Platform Hospitality Index (PHI)

Platform Hospitality Index represents the degree/extent to which a SDR platform conforms to the IRSA specification. It measures the reduction in effort required to port a waveform from one IRSA compliance SDR system to another. A higher PHI indicates higher hospitality (less porting effort), while a lower PHI suggests lower hospitality

(greater porting effort) on a platform. It is a weighted sum of compliance to different constituents (parameters) of IRSA specified in this document.

In due course of time, the names of various parameters shall be defined along with their weights and criterion for assigning values to them. A table in following format for the calculation of PHI shall be incorporated in IRSA.

Table 7.2 : Parameters of Platform Hospitality (PHI) Index

S.No.	Parameter	Description	Weight (%)	Value (0-10)
1.				
2.				
:				
:				
N				
		Total	100	

The PHI will be determined using the following formula:

PHI = ∑ (Parameter Value x Parameter Weight) / 100

The final score ranges from 0 (least hospitable platform) to 10 (highly hospitable platform)

7.2 Waveform Portability Considerations Across SDR Profiles

The IRSA SDR profiles ensure same environment over a similar form-factor SDRs to maximize the waveform portability. The heterogenous SDR profile is a super set of the three, homogenous is super set of single processor profile. So, there will be seamless portability of a waveform from single processor profile to the other two profiles, however, the portability effort will increase across profiles.

Figure 7.1 : Waveform Porting Effort Across SDR Profiles

8 Glossary

8.1 Abbreviations and Acronyms

Acronyms	Description
ACP	Audio ComPressor
ADC	Analog to Digital Converter
AEP	Application Environment Profile
AGPS	Assisted GPS
API	Application Program Interface
ARP	Address Resolution Protocol
BeiDou	Big Dipper, Chinese satellite navigation system
CF	Core Framework
CP	Constrained Processor
CODEC	Coder-Decoder
CORBA	Common Object Request Broker Architecture
COTS	Commercial-Off-The-Shelf
CTS	Clear-To-Send
DAC	Digital to Analog Converter
DSP	Digital Signal Processor
DSR	Data Set Ready
DTD	Document Type Definition
DTR	Data Terminal Ready
DTX	Discontinuous Transmission
EE	Execution Environment
ESSOR	European Secure SOftware defined Radio
FF	Facility Framework

Acronyms	Description
FPGA	Field Programmable Gate Array
GAGAN	GPS Aided GEO Augmented Navigation
GEO	Geostationary Earth Orbit
GLONASS	Global Navigation Satellite System
GNSS	Global Navigation Satellite System
GPIO	General Purpose Input Output
GPP	General Purpose Processor
GPS	Global Positioning System
HEDS	Heterogenous Embedded Distributed Systems
HDLC	High-Level Data Link Control
HW	Hardware
ID	Identifier
IDL	Interface Definition Language
IO	Input / Ouput
IP	Internet Protocol
IPS	IP Service
IRSA	Indian Radio Software Architecture
JTRS	Joint Tactical Radio System
JTNC	Joint Tactical Networking Centre
MAC	Media Access Control
MGRS	Military Grid Reference System
MHAL	Modem Hardware Abstraction Layer
MoCB	MHAL on Chip Bus
NAVIC	Navigation with Indian Constellation
NAVIC RS	Navigation with Indian Constellation Restricted Service
OE	Operating Environment
OMG	Object Model Group
ORB	Object Request Broker
OS	Operating System
PHI	Platform Hospitality Index
PIM	Platform Independent Model
POSIX	Portable Operating System Interface
PNT	Positioning, Navigation, and Timing
PPS	Pulse-per-second
PSM	Platform Specific Model
PTT	Push To Talk
PVT	Positioning, Velocity, and Timing
QZSS	Quasi-Zenith Satellite System, Japanese Regional Satellite Navigation System
RTC	Run-To-Completion
RTOS	Real-Time Operating System
RTS	Request-To-Send
Rx / RX	Receiver / Reception
SBAS	Satellite-Based Augmentation System
SCA	Software Communication Architecture

Acronyms	Description
SDR	Software Define Radio
SW	Software
TCP	Transmission Control Protocol
TXCVR	Transceiver
XCVR	Transceiver
Tx / TX	Transmitter / Transmission
ULw / ULW	Ultra-Light Weight
UML	Unified Modelling Language
UTM	Universal Transverse Mercator
VAD	Voice Activity Detection
WASS	Wide Area Augmentation System, SBAS developed by US Govt.
WCET	Worst-Case Execution Time
WF	Waveform
WInnF	Wireless Innovation Forum
WInnForum	Wireless Innovation Forum
WPI	Waveform Portability Index
XML	eXtensible Markup Language

Table 8.1 : Abbreviations and Acronyms

8.2 Definitions

Component / Software Component

A Component is a modular, reusable unit of software that provides a specific set of services and it can also use a set of services provided by other components. Its services are specified as well-defined interfaces. Components are combined and integrated with other components to create a complete application.

Constrained Processor (CP)

A processor that operates under resource constraints such as limited processing power, memory or energy. It operates with or without OS and where single address space is shared among the OS kernel (if present) and application component(s). There is no process creation. Only thread creation is possible, that too if OS is present.

Digital Signal Processor (DSP)

A DSP in the context of JTNC & WInnF refers to a processing element which hosts small footprint kernel/OS. This kind of processing element is a subset of constrained processors (CP) defined by IRSA. It may have small footprint kernel / OS supporting only thread/task creation or it run non-OS / bare-metal environment.

General Purpose Processor (GPP)

A processor that is capable of executing a broad range of instructions and running multiple software programs simultaneously. It hosts an operating system (Real-time or general) having separate address space for each process and the kernel. Each process can have

multiple threads.

Operating Environment (OE)
Execution Environment (RTOS, SCA CF) and transport mechanism are collectively referred to as Operating Environment

Platform APIs
Set of standardized interfaces which abstract the underlying hardware and software from waveform software with the objective of providing waveform portability across various hardware platforms. These APIs are logically grouped on the basis of their functionality and specified as Facilities with well-defined interfaces.

Platform Independent Model (PIM)
A model of a subsystem that contains no information specific to the platforms or technologies that can potentially be used to realize it.

Platform Specific Model (PSM)
A model of a system that contains technology specific information which can be used for its realization on a specific platform.

Transfer Mechanism
The specific standard or technology whereby various software components deployed over same or different processing elements and execution environments communicate with each other.

Waveform / Radio Application
A software realized by creating an assembly of various components to achieve radio functions. Components are combined and integrated with other components to create a complete application.

Appendix-A: Feedback Form

IRSA Specification Document: Feedback Form

Contact Information

Name & Designation:

Organization:

Ph No./ Mob:

E-mail:

Feedback Information

Feedback Type: Positive ☐ Constructive ☐ Negative ☐

Feedback Category:

Content

Design ☐

Completeness ☐

Any Other ☐

Technical

Functionality ☐ Consistency ☐

Clarity ☐ Accuracy ☐

Performance ☐ Any Other ☐

Document Section No:

Suggestion (Attach Sheet if Required):