

Specification for PIM and PSM for SWRADIO Components

Proposal to the OMG Telecom RFP: PIM and PSM for SWRADIO Components

Joint Revised Submission

Submitters:

Mercury Computer Systems

Raytheon

Thales

With support and collaboration from:

BAE Systems

Boeing

Carleton University

École de Technologie Supérieure
General Dynamics Decision Systems

Harris

ISR Technologies

L-3 Communications Analytics Corporation

MITRE

Mobile Smarts

Rockwell Collins

SCA Technica

Space Coast Communication Systems

Spectrum Signal Processing

Virginia Tech University

OMG document swradio/2004-01-01

Copyright 2004 by Mercury Computer Systems

Copyright 2004 by Raytheon Corp.

Copyright 2004 by THALES.

The submitting and supporting companies listed above have all contributed to this revised submission.

The companies listed above hereby grant a royalty-free license to the Object Management Group, Inc. (OMG) and Software Defined Radio Forum (SDRF) for worldwide distribution of this document or any derivative works thereof, so long as the OMG and SDRF reproduces the copyright notices and the below paragraphs on all distributed copies. The material in this document is submitted to the OMG and SDRF for evaluation. Submission of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE

ACCURATE, THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The companies listed above shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material. The information contained in this document is subject to change without notice.

This document contains information that is protected by copyright. All Rights Reserved. Except as otherwise provided herein, no part of this work may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems— without the permission of one of the copyright owners. All copies of this document must include the copyright and other information contained on this page.

The copyright owners grant member companies of the OMG and SDRF permission to make a limited number of copies of this document for their internal use as part of the OMG and SDRF evaluation process. Communication channel concepts presented herein are covered by a U.S. patent application by Raytheon Company. If adopted as part of an OMG standard, Raytheon will license the technology on a no fee basis for use in implementing the standard.

RESTRICTED RIGHTS LEGEND. Use, duplication, or disclosure by government is subject to restrictions as set forth in subdivision © (1) (ii) of the Right in Technical Data and Computer Software Clause at DFARS 252.227.7013.

CORBA, Object Request Broker, UML, and OMG are trademarks of Object Management Group.

Contents

Page

0	Foreword.....	vi
0.1	Copyright Waiver.....	vi
0.2	Co-Submitting Companies and Supporters.....	vi
0.3	Guide to the Submission	vii
0.4	Submission Contact Points.....	vii
0.5	Statement of Proof of Concept	vii
0.6	Resolution of RFP Mandatory and Optional Requirements.....	viii
0.6.1	General Requirements.....	viii
0.6.2	Specific Requirements	x
0.7	Responses To RFP Issues to be Discussed.....	xii
0.8	Introduction.....	xiii
1	Scope	3
2	Conformance.....	4
3	Normative references.....	5
4	Terms and definitions	5
5	Symbols (and abbreviated terms)	7
6	UML Profile for Software Radio (SWRADIO).....	9
1.1	Applications and Devices.....	10
6.1.1	Base Types.....	11
6.1.2	Properties.....	21
6.1.3	Resources and Components.....	35
6.1.4	Devices.....	53
6.1.5	Applications.....	69
6.2	Communication Equipment.....	81
6.2.1	CommEquipmentCommunicationPath	86
6.2.2	CommEquipementConnector	87
6.2.3	Property	88
6.2.4	Port	90
6.2.5	CommEquipment.....	94
6.3	Infrastructure	10
6.3.1	Radio Services.....	120
6.3.2	Communication Channel.....	123
6.3.3	Radio Management.....	134
6.3.4	SWRADIO Deployment	153
7	Platform Independent Model (PIM)	153
1.2	Common Radio Facilities	154
7.1.1	File Facilities.....	154
7.2	Common Layer Facilities.....	170
7.2.1	Quality of Service Building Block.....	171
7.2.2	Flow Control Building Block	178
7.2.3	Measurement Building Block	185
7.2.4	Error Control Building Block.....	187
7.2.5	Transmission Building Block	189
7.2.6	Packet Data Unit Building Block.....	191
7.2.7	Stream Building Block.....	193
7.2.8	Service Access Point Building Block	198
7.3	Data Link Layer Facilities.....	200
7.3.1	Link Layer Control Facilities	200
7.3.2	MAC Facilities	216

7.4	IO Facilities	223
7.4.1	IO Profile	224
7.4.2	IO Devices.....	226
7.4.3	IO Interfaces.....	229
7.5	Physical Layer Facilities.....	Error! Bookmark not defined.
7.5.2	Modem Facilities.....	Error! Bookmark not defined.
7.5.3	RF/IF Facilities.....	Error! Bookmark not defined.
7.6	Radio Control Facilities.....	1
7.6.1	Alarms and Alerts Facility	1
7.6.2	Audio Facility	1
7.6.3	Radio Management Facility	Error! Bookmark not defined.
8	Platform Specific Model (PSM).....	18
	Annex A (informative) Core Framework CORBA IDL	2
	A.1 Base Types Interfaces.....	2
	A.2 Resource Interfaces.....	3
	A.3 ResourceFactory Interfaces.....	5
	A.4 Device Interfaces.....	6
	A.5 File Services Interfaces.....	8
	A.6 DeviceManager Interfaces.....	11
	A.7 DomainManager Interfaces.....	12
	Annex B (normative) Common Layer Facilities CORBA IDL.....	1
	B.1 TBD	1
	B.2 TBD	1
	Annex C (normative) Common Radio Facilities CORBA IDL	1
	C.1 File Facilities Interfaces	1
	C.1.1 File Interface.....	1
	C.1.2 FileSystem Interface.....	1
	C.1.3 FileManager Interface.....	1
	Annex D (normative) Data Link Layer Facilities CORBA IDL	1
	D.1 Link Layer Interfaces.....	1
	D.2 MAC Interfaces.....	1
	Annex E (normative) Input/Output Facilities CORBA IDL	1
	E.1 Serial Interfaces	1
	E.2 Audio Interfaces.....	1
	Annex F (normative) Physical Layer Facilities CORBA IDL	1
	F.1 RF/IF Interfaces	1
	Annex G (normative) Radio Control Facilities CORBA IDL	2
	G.1 Radio Management Interfaces.....	2
	Annex H (normative) Operating System Profile.....	1
	H.1 TBD	Error! Bookmark not defined.
	Bibliography.....	3

0 Foreword

0.1 Copyright Waiver

The companies listed above hereby grant a royalty-free license to the Object Management Group, Inc. (OMG) for worldwide distribution of this document or any derivative works thereof, so long as the OMG reproduces the copyright notices and the below paragraphs on all distributed copies. The material in this document is submitted to the OMG for evaluation. Submission of this document does not represent a commitment to implement any portion of this specification in the products of the submit-ters. The copyright owners grant member companies of the OMG permission to make a limited number of copies of this document (up to fifty copies) for their internal use as part of the OMG evaluation process.

0.2 Co-Submitting Companies and Supporters

The following companies are pleased to submit this specification in response to the PIM and PSM for SWRADIO Components (swradio/02-06-02):

- Mercury Computer Systems
- Raytheon
- THALES

In addition, the following organizations are pleased to support this submission:

- BAE Systems
- Boeing
- Carleton University
- École de Technologie Supérieure
- General Dynamics Decision Systems
- Harris
- ISR Technologies
- L-3 Communications Analytics Corporation
- MITRE
- Mobile Smarts
- Rockwell Collins
- SCA Technica

- Space Coast Communication Systems
- Spectrum Signal Processing
- Virginia Tech University

0.3 Guide to the Submission

This submission presents the Platform Independent Model (PIM), as well as, the CORBA Platform Specific Model (PSMs) along with its IDL specified interfaces of services. It was prepared as a response to the RFP issued by the OMG, swradio/02-06-02. The first five sections follow the ISO specification format: Scope, Conformance, Normative References, Terms, and Symbols. The remaining outline for this submission is as follows:

- Section 6 UML Profile for Software Radio – defines the UML profile concepts for a SDR such as communication channel, Radio Frequency (RF) channel, communication equipment (e.g., antenna, amplifier), applications and devices, and infrastructure.
- Section 7 Platform Independent Model (PIM) – defines the PIM definition for the SDR facilities.
- Section 8 CORBA Platform Specific Model (PSM) – defines the CORBA interface definitions for the SDR facilities and their mappings to the PIM.
- Annexes contain the CORBA IDL

A separate document, Software Defined Radio Use Cases (swradio/2003-05-02), supplements this submission. This document captures the use cases that were used for the SDR PIM definition.

0.4 Submission Contact Points

All questions about this submission should be directed to:

Jeff Smith
Mercury Computer Systems, Inc.
199 Riverneck Road
Chelmsford, MA 01824-2820 USA
Tel: +1 978-967-1760
Fax: +1 987-256-3599
email: jesmith@mc.com

Gerald Bickle
Raytheon Company
1010 Production Driver
Fort Wayne, IN 46808, USA
Tel: +1 269 429-6280
Fax: +1 260 429-5060
email: Gerald_L_Bickle@raytheon.com

Francois-Xavier LEBAS
THALES Communications France
160 Boulevard de Valmy, BP 82, 92704 Colombes Cedex, FRANCE
Tel: +33.(0)1.41.30.24.20
Fax: +33 (0)1.41.30.35.05
email: francois-xavier.lebas@fr.thalesgroup.com

0.5 Statement of Proof of Concept

This submission is derived from the following design and implementation efforts:

- Digital Modular Radio (DMR) Program
- Joint Tactical Radio System (JTRS) Step 2 (A, B, C)
- JTRS Joint Program Office (JPO) Waveform Programs
- JTRS Cluster 1 Program
- Software Defined Radio Forum (SDRF)

0.6 Resolution of RFP Mandatory and Optional Requirements

0.6.1 General Requirements

0.6.1.1 OMG Modeling Language

Submitters are encouraged to express models via OMG modeling languages such as UML, MOF, CWM and SPEM (subject to any further constraints on the types of the models and modeling technologies specified in Chapter 6 of this RFP). Submissions containing models expressed via OMG modeling languages shall be accompanied by an OMG XMI [XMI] representation of the models (including a machine-readable copy). A best effort should be made to provide an OMG XMI representation even in those cases where models are expressed via non-OMG modeling languages.

All model expression is done using an OMG modeling language.

0.6.1.2 Behavior

Chapter 6 of this RFP specifies whether PIM(s), PSM(s), or both are being solicited. If proposals specify a PIM and corresponding PSM(s), then the rules specifying the mapping(s) between the PIM and PSM(s) shall either be identified by reference to a standard mapping or specified in the proposal. In order to allow possible inconsistencies in a proposal to be resolved later, proposals shall identify whether the mapping technique or the resulting PSM(s) are to be considered normative.

The submission provides both PIM and PSM mappings as described in section 9.

0.6.1.3 Completeness

Proposals shall be precise and functionally complete. All relevant assumptions and context required for implementing the specification shall be provided.

This submission is believed to be precise and functionally complete.

0.6.1.4 Compliance Points

Proposals shall specify compliance points that clearly state what features all implementations must support and which features (if any) may optionally be supported.

This submission clearly identifies mandatory requirements over those, which are optional. Refer to Section 3 for more discussions on compliances.

0.6.1.5 Reuse of Existing OMG Specifications

Proposals shall reuse existing OMG and other standard specifications in preference to defining new models to specify similar functionality.

This submission is believed to have maximized its reuse of existing OMG specifications.

0.6.1.6 Changes or Extensions

Proposals shall justify and fully specify any changes or extensions required to existing OMG specifications. In general, OMG favors proposals that are upwards compatible with existing standards and that minimize changes and extensions to existing specifications.

TBD

0.6.1.7 Functional Factoring

Proposals shall factor out functionality that could be used in different contexts and specify their models, interfaces, etc. separately. Such minimalism fosters re-use and avoids functional duplication.

The interfaces in this submission are believed to be optimally factored.

0.6.1.8 Duplication

Proposals shall use or depend on other specifications only where it is actually necessary. While re-use of existing specifications to avoid duplication will be encouraged, proposals should avoid gratuitous use.

It is believed that there are no unnecessary dependencies on other specifications.

0.6.1.9 Compatibility

Proposals shall be compatible with and usable with existing specifications from OMG and other standards bodies, as appropriate. Separate specifications offering distinct functionality should be usable together where it makes sense to do so.

The interfaces in this submission are believed to be compatible with other OMG specifications.

0.6.1.10 Implementation Flexibility

Proposals shall preserve maximum implementation flexibility. Implementation descriptions should not be included and proposals shall not constrain implementations any more than is necessary to promote intreroperability.

Implementation flexibility has been preserved by keeping this submission free of implementation description.

0.6.1.11 Interoperability

Proposals shall allow independent implementations that are substitutable and interoperable. An implementation should be replaceable by an alternative implementation without requiring changes to any client.

The only interfaces required for interoperability are specified in IDL.

0.6.1.12 ISO Compatibility

Proposals shall be compatible with the architecture for system distribution defined in ISO's Reference Model of Open Distributed Processing [RM-ODP]. Where such compatibility is not achieved, or is not appropriate, the response to the RFP must include reasons why compatibility is not appropriate and an outline of any plans to achieve such compatibility in the future.

This submission does not change the overall CORBA services architecture.

0.6.1.13 Security

In order to demonstrate that the specification proposed in response to this RFP can be made secure in environments requiring security, answers to the following questions shall be provided:

What, if any, are the security sensitive elements that are introduced by the proposal?

TBD

Which accesses to security-sensitive elements must be subject to security policy control?

TBD

Does the proposed service or facility need to be security aware?

TBD

What default policies (e.g., for authentication, audit, authorization, message protection etc.) should be applied to the security sensitive elements introduced by the proposal? Of what security considerations must the implementers of your proposal be aware?

TBD

0.6.1.14 Internationalization

Proposals shall specify the degree of internationalization support that they provide.

TBD

0.6.2 Specific Requirements

0.6.2.1 Mandatory Requirements

0.6.2.1.1 Format of Proposals

- *Proposals shall use UML and MOF for this RFP to express all concepts of the solicited PIM and the solicited PSM(s).*

This submission uses UML and MOF to express the software radio facilities PIM and PSM.

- *Proposals shall use CORBA IDL and the UML Profile for CORBA, as outlined in section 5.1 of this document, to express the concepts of the solicited PSM.*

This submission uses CORBA IDL as shown in Annex B and UML profile for CORBA as depicted in section 9, CORBA Platform Specific Model.

0.6.2.1.2 Platform Independent Model

- *Proposals shall define interfaces between waveform components and communication channels, radio devices and radio services.*

This submission defines interfaces between the various different components within a radio as specified in sections 7 UML Profile for SWRADIO and 8 PIM.

- *Proposals shall define data transfer interfaces between waveform components.*

TBD. The revised submission will define data transfer interfaces between waveform components within a radio as specified in section 8, Platform Independent Model (PIM).

- *Proposals shall define interfaces for a Data Link Provider Interface (DLPI) specification. The DLPI interfaces map to the ISO Data Link Service Definition IS 8886 and Logical Link Control IS 8802-2 (LLC). Where the two standards conflict, DIS 8886 prevails.*

This submission defines Data Link Layer interfaces that map to the DLPI concepts, which are specified in section 8.3.1 Link Layer Facilities.

- *Proposals shall define interfaces for standardized Media Access Control (MAC) application program interfaces that can be utilized in defining waveform components.*

This submission defines MAC interfaces that can be utilized in defining waveform components, which are specified in section 8.3.2 MAC Facilities.

- *Proposals shall define SDR device interfaces and radio services in the PIM. The device interfaces shall include interfaces for, as a minimum, audio and modem devices.*

This submission defines audio interfaces as stated in section 8.4 I/O Facilities and modem interfaces as stated in section **Error! Reference source not found.**, Physical Layer Facilities.

- *Proposals shall define Operating System interfaces and services that provide portability for waveform components.*

TBD.

- *Proposals shall define communication channel creation and management interfaces.*

TBD. The revised submission will define *communication channel creation and management* as specified in section **Error! Reference source not found.** Radio Management.

0.6.2.1.3 Platform Specific Model

- *Proposals shall provide a PSM(s) that defines a realization of the above-defined PIM using the CORBA as the middleware and XML as configuration descriptor.*

The mapping to PIM to PSM is described in section 9 PSM. Annexes contain the actual CORBA IDL as result of this mapping.

- *Operating system interfaces shall conform to the POSIX C APIs.*

TBD

0.6.2.1.4 Optional Requirements

- *Proposals may provide PSM(s) for other technologies than CORBA by specializing the PIM of section 8.*

This submission provides no other PSM other than CORBA.

- *The PIM may provide interfaces for management of radio communications.*

Section 8.6, Radio Control Facilities, describes interfaces for radio management.

- *The PIM may provide interfaces for radio services.*

Other services are specified in section 7, UML Profile for SWRADIO and in sections 7.1 (Common Radio Facilities) and 8.2 (Common Layer Facilities).

0.7 Responses To RFP Issues to be Discussed

- *Proposals shall demonstrate interoperability between waveforms components.*

Interoperability is achieved by standard interfaces

- *Proposals shall demonstrate how software portability across hardware platforms and reuse in different waveforms is supported.*

Standard I/O interfaces, OS profile, ...

- *Proposals shall demonstrate how through the PIM independence of waveform components from specific hardware solutions can be achieved.*

MDA, waveform definition can be as a PIM mapping to various PSM technologies

- *Proposals shall discuss the feasibility of interface commonality across waveforms, especially at the physical layer.*

TBD

- *Proposals shall demonstrate how the PIM addresses scalability from low-capability (mobile phones) to high-capability radios (e.g., basestations, multi-channel radios, etc.).*

TBD

- *Proposals shall demonstrate how the management of communication channels can be supported by the proposed PIM.*

TBD

- *Describe the relationship to expected submissions against the Deployment and Configuration of Component based Distributed Applications RFP (orbos/2002-01-19) as it relates to communication channel management.*

TBD

- *Describe the relationship to expected submissions against the UML 2.0 Superstructure (ad/00-09-02) as it relates to data flow.*

The UML Profile for SWRADIO extends the interface, component, and device constructs within the UML 2.0 Specification.

1 Introduction

Software Defined Radio (SDR) terminology tells everybody HOW things are internally arranged inside a radio set rather than WHICH services are effectively provided to end users.

Indeed, SDR is not just another way to build Radios with same functions but supports a lot of new critical needs. For example, SDR supports several waveforms inside the same box, eases bug fixing, supports reconfigurability, brings support for digitized, IP-based, data transmissions and improves security. Those improvements were not only expected within the radio military market but also meet emerging and actual needs within civil radio market.

Supporting multiple concurrent waveforms inside a single radio set is more critical for the military market where numerous waveforms co-exist rather than for the civil market. But civil market changes, and future civil radios, are likely to support at least both a cell waveform and a high bandwidth local waveform on the same hardware. Moreover, future radio nodes equipments may have to concurrently support multiple cell waveforms such as Global System for Mobiles (GSM), General Packet radio Services (GPRS) and Universal Mobile Telecommunications System (UMTS) and high throughput waveforms such as Bluetooth and WiFi.

Bug fixing is still a nightmare for many radio manufacturers since this may involve the return thousands of radio sets to factories. Download of new software is intended to fix software bugs or to workaround hardware bugs and is already a key need for tomorrow. This need begins to be addressed on a waveform-by-waveform basis (3GPP) but this does not solve the problem for future multi-waveform radios. Software Radio addresses this need.

Enabling cost-effective technology insertion is a strong motivation for many manufacturers suddenly faced with a more volatile market than the past and where tomorrow's standards are unclear. The life cycle of new radio sets become so short that the return-on investments (ROI) cannot be ensured. Enabling cost-effective technology insertion is also a concern for many operators and with customers faced with exploding costs.

Reconfigurability is vital within military markets not only to tailor radio systems for versatile missions but also to gain increasing importance within civil systems. Downloading new services and setting new parameters are requirements for both markets.

Since one century, radio was mainly used to transmit human voice. In the future, radios will transmit digitized data rather than analog voice. This is a major shift. For example, radio that was mainly focused on point-to-point links will have to support new services such as networking and used to manage the seamless qualities of radio function challenges. Radio is sometimes viewed not only as being the last hop inside a network, but it is likely to be used as a flexible wireless backbone in more and more cases.

Radio security has changed. Security functions cannot be frozen for an entire radio system life. Security functions must be able to evolve to counter new and evolving threats and to keep the security chain safe. This is especially important when coupling radios with information systems becomes the rule.

In addition to the functional improvements described above, SDR targets also bring all those new functions within an architecture that supports a cost-effective engineering approach. As a matter of fact, this submission path points the way towards a new era within radio system engineering that noticeably enables cost-effective technology insertion.

Designing a model for SDR make sense only if this model fulfils actual radio needs and requirements. When applying UML based analysis and design, this requires a prior functional analysis based on Use Cases and sets of Scenarios models as sequence diagrams.

These two analysis models can be summarized as:

- The first level (UML Profile) addresses the Radio System, Radio Set and Communication Channel concepts. This analysis model provides hardware and software abstractions of components, devices and resources as well as providing an extension to the UML specification for the software radio domain.

- The second level (SDR Use Cases/swradio-03-05-02) addresses the operational Radio Set (Radio Equipment) operational functions and scenarios. This also encompasses waveform analysis based upon waveform characteristics.

Finally, the whole analysis model is mapped to the design model as a proof-of-concept.

1.1 Architectural Issues

The Open System Model (OSI) elaborated and promoted by the International Standard Organization (ISO) is probably the most important standard of the telecommunications domain. The OSI model influenced a great number of people around the world and had a strong impact on software industry.

As a matter of fact, the OSI model refined and formalized several major concepts introduced in previous models such as the Layers concept. Also, the OSI model introduced the new concepts of Open Systems and Services:

- The Open Systems concept opened the way to distributed computing,
- The Services concept formed the basis for the popular Client/Server Architecture.

A full description of the OSI model can be found inside the ISO IS 7498. For convenience, a short summary is proposed in this section. The OSI model assumes that the structure of the communications functions located on a network node should be structured into a stack of 7 Layers where:

- a Layer talks with its counterpart located on another node,
- the conversation between corresponding Layers is ruled by a Protocol which exchanges Protocol Data Units (PDU),
- a Layer supply Services to upper Layers through Service Access Points (SAP),
- a Layer act as a PDU consumer and/or provider for upper and lower layers.

Note that some communications links do not always require all Layers and that some Layers may be empty inside a communications stack. When expressed in UML, the core of the OSI could be modeled as in Figure 2. In this figure, a sublayer is a subdivision of a layer.

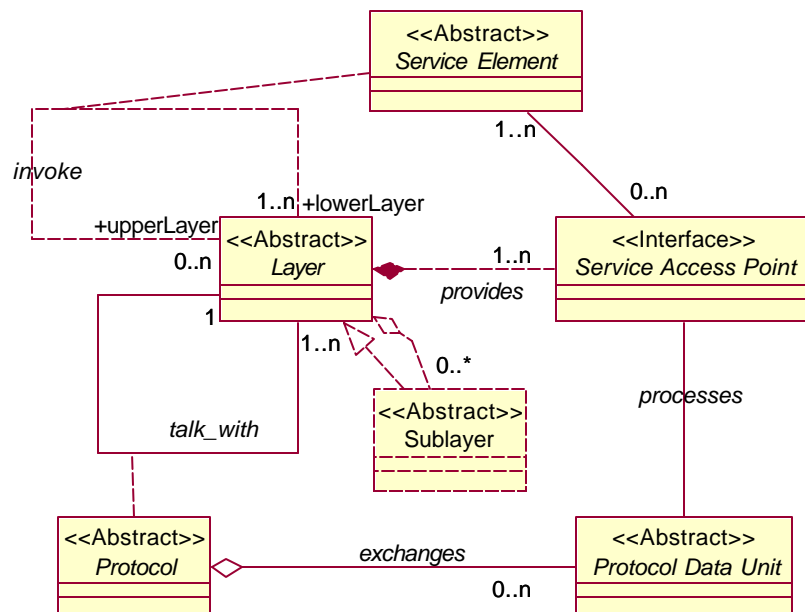


Figure 1: Abstract OSI Model Core

Ironically, and despite its importance, the OSI model was almost never strictly applied mainly for performances reasons. More, with the worldwide adoption of IP, the “old” layered model on which IP is built superseded the OSI model.

Also, the OSI model enforces a strict partitioning between layers that is not always easily applicable: for example, presentation services (level 7) are required by lower layers to communicate.

Nevertheless, the OSI model which have evolved since its inception, is not at all obsolete and remain a reference for the whole telecom industry. As an example, the Internet Engineering Task Force (IETF) which is in charge of adopting new Internet-related standards often used the OSI-related standards and other related ISO standards as a guideline for new Internet standards and many of the most recent and advanced Internet standards (RFC) are based on ISO subsets¹.

This submission recognizes that the OSI communication model support remains a desirable goal, but conformance with this model is not a must for any telecommunications equipment due to design and performance constraints. Even though the OSI model is the preferred waveform-layering model for software radio platforms, the proposed architecture is able to support other in-use or next-to-come models as well as non-standard waveform layering possibilities through the usage of building blocks as a part of waveform layers.

This submission apply the Extended OSI model shown below:

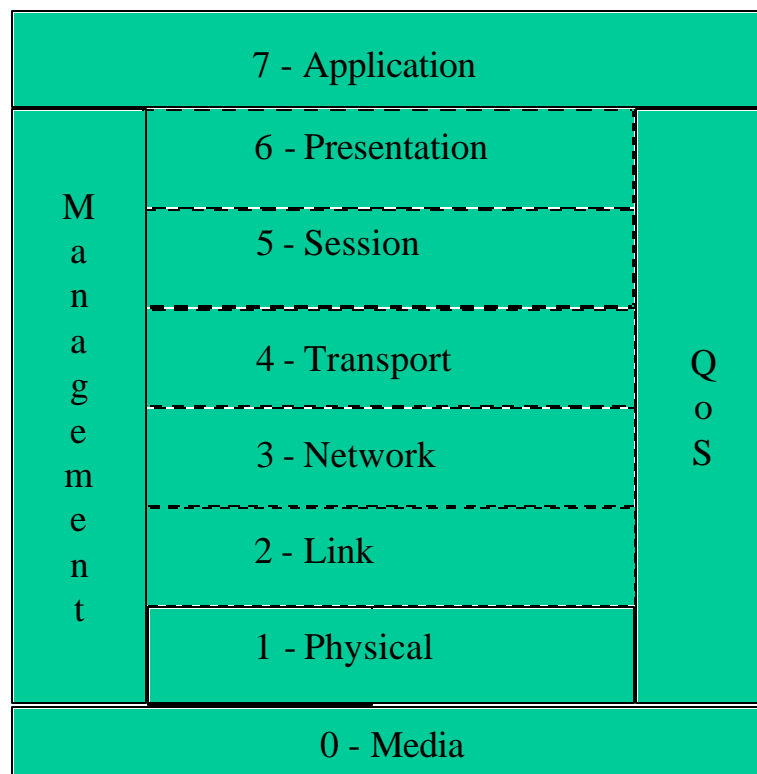


Figure 2 : Extended OSI Model

The QoS layer stands for Quality of Service layer and includes Security.

¹ This is mostly the case for management and security-related RFC which are more and more conformant to the OSI standards .

1.2 Platform and Waveform Issues

The Platform concept used here refers to a composite product infrastructure that supports a product line approach, which allows for various dedicated configurations such as radio nodes, radio terminals and/or other radio gateways. In some way, this Platform concept extends the Platform concept used within OMG To promote an open standard, this submission supports a Platform/Waveform approach where :

- the Platform provides a **standardized** yet **extensible** set of software services that abstracts hardware dependencies and support waveforms as well as other applications types such as management applications,
- one or several Waveforms can be developed and ported onto various Platforms,
- management and/or end-user applications can also be developed and ported onto various Platforms.

Such a Platform/Waveform approach opens the way to an open market where waveforms providers can be independent of platform providers. Platform services splits into:

- Management services that supports Deployment & Configuration, Control, Fault ...
- WaveForm Application (WFA) services that supports flow processing (coding, decimating, etc.).

This submission defines a set of Platform-Independent Interfaces and do not make any assumptions on how those interfaces are supported without making any assumptions about the internal Platform architecture.

1.3 Software Communications Architecture

The Software Communications Architecture, SCA in short, is today the most significant manifestation and realization of the Software Defined Radio concept. As a matter of fact, the SCA has already been adopted on significant projects.

The SCA provides a framework that supports an industrial resource-based and component-based approach to build versatile radiosets, each offering several configurations. Furthermore, some of the waveform related APIs are specified in the SCA API supplement document. This document, together with radio management and control related facilities in the SCA specification provides a basis for this submission. However, it must be recognized that the SCA is more focused on management and control facilities than to provide radio business services for waveforms developers and the API supplement is far from being complete.. Also, the SCA model is not platform-independent and is intimately embedded with CORBA. Finally, the SCA is not fully conformant to OMG's existing and/or on-going standards. This submission recognizes the SCA as a major milestone for Software Radio to settle on and targets to extend the SCA capabilities in several ways:

1. To provide additional higher level waveform-oriented services while preserving backward compatibility with SCA
2. To abstract the SCA UML model from its CORBA Platform and provide what is called an UML Profile for Software Radio,
3. To link this PIM with other related OMG standards. (Lightweight services, Deployment and Configuration, Lightweight Logging, Lightweight CCM, UML 2.0, etc.)

It is worth noticing that the SCA concepts and even that the CCM concepts gracefully matches the OSI ones in several ways:

- the OSI Service Access Point can be mapped upon the SCA and CCM Port concept,
- the Application Service Elements can be mapped upon the SCA and CCM Interfaces concept,

- SCA packet and payload concepts in the waveform supplement maps to PDU (Packet Data Unit) and SDU (Service Data Unit) in OSI model,
- There is also a many-to-many relationship between the component concept in SCA and layer concept in the OSI model in the sense that a single layer may comprise of multiple components, or a single component may be deployed as a part of a waveform layer. Both SCA components and OSI waveform layering together with SAPs provide functional boundaries and standard access to waveform facilities.

1.4 Microframeworks

This submission introduces the concept of Microframework. A Microframework is a set of high-level services dedicated to process a waveform function (e.g: a layer) and that ease the development of a new waveform by providing not only a library of dedicated services objects but also provides a default behavior.

The idea of the microframework came from the fact that layers of the same level belonging to distinct waveforms sometimes share the same set of states and transitions and that they apply similar type of processings in the same order.

From an object-oriented design point of view, this obviously translate into that some layers could be simply derived from already existing layers at the price of overloading few operation(s) in a derived class.

Microframeworks could be defined to support several situations. This submission will define microframeworks for OSI conformant layers. Additional microframeworks could be added in future to support non-OSI conformant models.

1.5 Information Flows

A radio has to deal with 2 kinds of information flows:

- Data flows that contains informations that are just transported and considered as neutral (e.g: uninterpreted payload) when flooding through radioset,
- Control flows that instruct the radioset about the way that data has to be processed.

Data flows may require differenciated processing policies depending of their respective nature:

- Streaming data flows (such as voice or video) must provide a guaranteed throughput rate and require timelined real-time processing,
- Segmented data flows (such as software protocols) that only require statistical real-time processing,
- Some flows need to be protected (confidentiality, integrity) while others do not.

Control flow itself splits into 2 Control subflows:

- In Band Control is the control information that is embedded within the data flow. In Band Control provides a convenient way to synchronize control with following data,
- Out of Band Control is the Control information that can be processed at any time (asynchronously) during data flow processing or even when no data flow is processed.

Data and In Band Control flows are transported and processed through a Channel. In some cases, several separate flows may have to be multiplexed into a single Channel. Channels themselves sometimes need to be multiplexed.

1.6 Security

Not all radio communications require the same (high) level of security but all communications require some level of security against accidental as well as intentional threats. Basic security functions that have to be supported in some way are:

- Confidentiality to protect against unauthorized reading,
- Integrity to protect against unauthorized change,
- Authentication to protect against unauthorized access,
- Protection against denial-of-service.

Depending of the threats analysis, but also on regulations, configurations, etc. many security TRANSEC and COMSEC mechanisms such as ciphering, checksumming, signing, and frequency hopping could be implemented and enforced. But considering that:

- Platforms and Waveforms can possibly be arranged into unattended unique configurations that defeat today's security-conformance strategies,
- Platforms and Waveforms should embed their own security mechanisms since they are possibly provided by separate manufacturers and cannot rely on each other,
- Platform and Waveform should not provide API to manage and/or configure their internal security.

This submission does not enforce any public Security API and assumes that:

- Platforms ensure their own security level with private security mechanisms transparent to Waveforms,
- Waveforms ensure their own security level with private security mechanisms.

Common security mechanisms using standardized API may be provided to Platform and Waveform by some general-purpose security device.

2 Scope

A Software Defined Radio (SDR) is a wireless communication system (low-capability mobile phones to high-capability multi-channel radios), in which the particular communication and transmission characteristics are realized through specialized software running on flexible signal processing hardware. This is very different from the traditional approach of using specialized hardware and has the benefit of instant reuse or sharing of a single system platform for multiple communication purposes. Within the physical limits of the underlying hardware, virtually any communication task can be realized instantaneously through a software load, including the ability of extensive field-upgrades and maintenance.

“Software-defined radios let service providers reprogram basestations to reassign channels as standards change and the mix of analog vs. digital users shifts. And engineers envision handsets that someday will download from any network whatever code is needed to reprogram themselves to access a wireless service or run a mobile application.” EE Times November 15, 2000.

The hardware of a Software Defined Radio consists of Digital Signal Processors (DSPs), Field Programmable Gate Arrays (FPGAs), General Purpose Processors (GPPs) and other hardware components. The behavior and characteristic of each of these hardware building blocks is controlled by a dedicated software component. The collection of these components defines the communication and transmission characteristics of the radio and is therefore called “Waveform Application”, or just “Waveform”. Examples of waveforms include ATC (air traffic control), police, cell phone, and military communications.

The software components making up a Waveform are typically hosted by an operating system and coordinated through middleware. Existing implementations use real-time operating systems (RTOS) and CORBA. More complex systems, like future avionics systems in civil and military aircraft, or the ground station for communication satellites, are expected to be composed from multiple, collaborating hardware/software systems. These systems in their entirety will then implement the desired communication and transmission functionality. It is expected and required, that individual components of such an aggregated system will be implemented using different technologies and techniques. Nevertheless, seamless and reliable interoperation is absolutely required.

Current specifications in the Software Defined Radio Domain lack clearly defined standard interfaces between functional components of waveform applications, device services and radio services, which seriously impacts their portability. Also, the current specifications are typically bound to particular implementation techniques and are therefore unable to provide the necessary freedom for the introduction of new technologies while maintaining the interoperation guarantee at the same time.

The Software Defined Radio Forum, SDR Forum in short, has established a functional model for software-defined radios whose building blocks are those figured below:

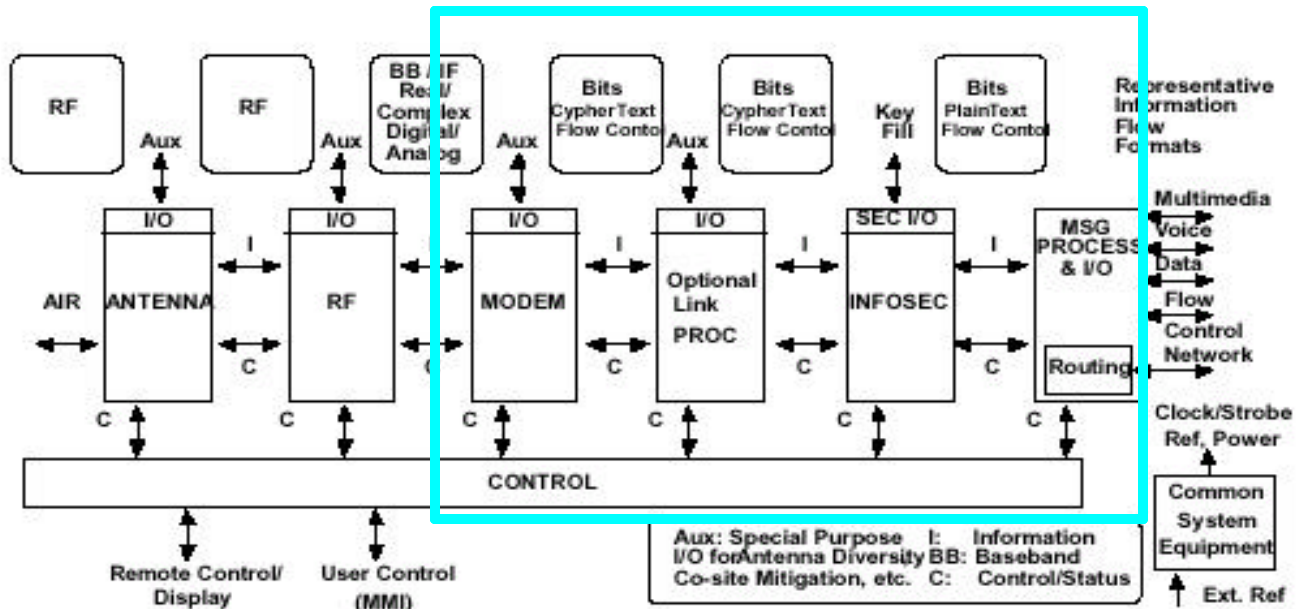


Figure 3: SDRF Model

In respect with the model above, the software architecture addressed by this submission is embedded and runs within the digitized part of the radio whose border is figured by the blue line figured above.

But, in the same time, this submission define upon an information model that abstracts the whole SDRF model including the RF and Antenna.

3 Conformance

TBD

4 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Document Number	Document Title
MSRC-5000SCA V2.2 17 Nov 2001	Software Communication Architecture Specification
	UML 2.0 Superstructure

5 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

4.1

Bluetooth

TBD

4.2

Common Object Request Broker Architecture (CORBA)

An OMG distributed computing platform specification that is independent of implementation languages.

4.3

Component

A component can always be considered an autonomous unit within a system or subsystem. It has one or more ports, and its internals are hidden and inaccessible other than as provided by its interfaces. A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component exposes a set of ports that define the component specification in terms of provided and required interfaces. As such, a component serves as a type, whose conformance is defined by these provided and required interfaces (encompassing both their static as well as dynamic semantics).

4.4

CORBA Component Model (CCM)

An OMG specification for an implementation language independent distributed component model.

4.5

Facility

4.6

Interface Definition Language (IDL)

An OMG and ISO standard language for specifying interfaces and associated data structures.

4.6

Logical Device

A software component that is an abstraction of a hardware device it represents.

4.6

Mapping

The Specification of a mechanism for transforming the elements of a model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel.

4.6

Metadata

The Data that represents models. For example, a UML model; a CORBA object model expressed in IDL; and a relational database schema expressed using CWM.

4.6

Metamodel

A model of models.

4.6

Meta Object Facility (MOF)

An OMG standard, closely related to UML, that enables metadata management and language definition.

4.6

Model

A formal specification of the function, structure and/or behavior of an application or system.

4.6

Model Driven Architecture (MDA)

An approach to IT system specification that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform.

4.6

Platform

A set of subsystems/technologies that provide a coherent set of functionality through interfaces and specified usage patterns that any subsystem that depends on the platform can use without concern for the details of how the functionality provided by the platform is implemented.

4.6

Platform Independent Model (PIM)

A model of a subsystem that contains no information specific to the platform, or the technology that is used to realize it.

4.6

Platform Specific Model (PSM)

A model of a subsystem that includes information about the specific technology that is used in the realization of it on a specific platform, and hence possibly contains elements that are specific to the platform.

4.6

Radio Platform

The Radio Platform is made of a Hardware Platform and a Software Platform.

4.6

Radio Set

A single radio set unit that can be ground fixed, mounted on a mobile platform or held by hand.

4.6

Radio System

A networked set of radio sets that provide wireless communication facilities between callers and callees.

4.6

Request for Proposal (RFP)

A document requesting OMG members to submit proposals to the OMG's Technology Committee. Such proposals must be received by a certain deadline and are evaluated by the issuing task force.

4.6

Service

A component that is shared by applications and that supports basic functions for using and implementing objects.

4.6

Unified Modeling Language (UML)

An OMG standard language for specifying the structure and behaviour of systems. The standard defines an abstract syntax and a graphical concrete syntax.

4.6

UML Profile

A standardized set of extensions and constraints that tailors UML to particular use.

6 Symbols (and abbreviated terms)

Abbreviation	Definition
API	Application Program Interface
ASIC	Application Specific Integrated Circuit
BIT	Built-In Test
BSP	Burst Schedule Packets
COMSEC	Communication Security
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off the Shelf
CPU	Central Processing Unit
DLPI	Data Link Protocol Interface
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
GIOP	General Inter-ORB Protocol
GPP	General Purpose Processor
GPRS	General Packet radio Services
GPS	Global Positioning System

Abbreviation	Definition
GSM	Global System for Mobiles
HCI	Human-Computer Interface
HW	Hardware
I/O	Input/Output
ID	Identification, Identifier
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
INFOSEC	Information Security
I/O	input/output
IOR	Interoperable Object Reference
IP	Internet Protocol
ISO	International Standards Organization
LAPx	Link Access Protocol x (where x represents 1 of several protocols defined by industry)
MAC	Medium Access Control, a sublayer of the OSI Data Link Layer
MIB	Management Information Base
N/A	Not Applicable
NAPI	Networking Application Programming Interface
OE	Operating Environment
OMG	Object Management Group
ORB	Object Request Broker
OS	Operating System
OSI	Open System Interconnection
PIM	Platform Independent Model
POSIX®	Portable Operating System Interface
PPP	Point-to-Point Protocol
PSE52	Real-time Controller System Profile, defined in IEEE Std 1003.13
PSM	Platform Specific Model
QoS	Quality of Service
RAM	Random Access Memory
RF	Radio Frequency
RS-232	Electronic Industries Alliance interface standard
RS-422	Electronic Industries Alliance interface standard
RS-423	Electronic Industries Alliance interface standard
RS-485	Electronic Industries Alliance interface standard
SDR	Software Defined Radio

® POSIX is a registered trademark of the Institute of Electrical and Electronics Engineers, Inc.

Abbreviation	Definition
SW	Software
TBD	To Be Determined
TCP	Transmission Control Protocol
TOD	Time Of Day
TRANSEC	Transmission Security
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
UUID	Universally Unique Identifier
XML	eXtensible Markup Language

7 UML Profile for Software Radio (SWRADIO)

This section defines the UML Profile for SWRadio. This profile is an integral part of the “PIM and PSM for SWRADIO Components” specification. The set of stereotypes defined in this profile constitutes the core language for the definition of the SWRadio PIM and PSM. The current UML Profile for SWRadio extends the UML 2.0 meta-language, with emphasis on extensions to. It mainly extends the Components package and Deployment package of UML 2.0.

The goal of the UML Profile for SWRadio is to enable the development of UML tools to support the development of SWRadio applications and systems. The objectives are not only to facilitate the modelling of SWRadio applications and systems, but also to enable the automatic generation of descriptor files (e.g. XML descriptor files) and code (or code skeletons) from UML models, to enable validation at design time, and to enable the development of simulation environment for SWRadio.

To address the issues of the different actors involved in SWRadio product developments, the current profile has been developed with three main viewpoints in mind: the viewpoint of application and device developers, the viewpoint of infrastructure/middleware providers, and the viewpoint of SWRadio platforms providers. These three viewpoints define distinct sets of concepts (and stereotypes) that are required in different contexts.

To be consistent with the three viewpoints introduced above, the UML Profile for SWRadio is partitioned in three main packages: the Applications and Devices package, the Infrastructure package, and the Communication Equipment package. Each package defines the set of concepts and UML stereotypes required to perform a specific role in the development of an SWRadio product.

The Applications and Devices package defines the set of concepts that are required to develop SWRadio applications and devices. This package mainly contains a set of stereotypes that extends the UML 2.0 meta-classes Component and Interface. This set of stereotypes includes Resource, Device, DeviceDriver, and SWAPI (SWRadio API).

One of the main objectives of this profile is to standardize interfaces and components to enable Commercial-off-the-Shell (COTS) component SWRadio application development.

For this purpose, the Applications and Devices package defines the concept (as a stereotype) of SWRAPI (SWRadio API) as an extension of UML interfaces. This stereotype is used to type the different SWRadio APIs that aim at being standardized. The set of SWRAPIs defined in this

specification can be split in two categories: SWRadio application component interfaces and services interfaces.

WHAT IS THE ROLE OF LAYER IN THE PROFILE? Link, mac, physical

SHOULD WE HAVE A LAYER STEREOTYPE IN THE PROFILE? yes

The Infrastructure package defines the concepts that are required to develop software componentdeploy services and applications (e.g., waveforms) within a radio infrastructure for SWRadio applications, and to manage the radio's domain, services, and devices. This package mainly contains a set of stereotypes that extends the UML 2.0 meta-classes Component and Interface. This set of stereotypes includes RadioManager, DeviceManager, Application, and ApplicationFactory components.

The Communication Equipment package defines the concepts that are required to model SWRadio equipment. This package defines stereotypes for the different types of hardware devices used in SWRadio. MORE NEED TO BE ADDED HERE. This package mainly contains a set of stereotypes that extends the UML 2.0 meta-class Device. This set of stereotypes includes RF Device, I/O Device, Security Device, Antenna, Amplifier, Frequency Converter, etc. For each Device stereotype specific characteristics are defined that are required by a waveform component for deployment behaviour.

1.1 Applications and Devices

The Applications and Devices provide a set of interfaces and components that are used for the development of waveform applications, logical devices, and waveform's components. Figure 4 depicts the relationships among the packages within the Applications and Devices, which are as follows:

- Base Types – defines basic types for waveform applications, logical devices, and waveform resource component definitions.
- Properties – defines configure, query, testable, service artifact (capability and capacity) and executable properties for swradio components and artifacts.
- Devices – defines the logical device components for a swradio that are used by waveform's components and for devices that waveform's components are deployed on.
- Resources and Components – defines the interfaces and components for waveform components.
- Applications – defines the elements for waveform applications.

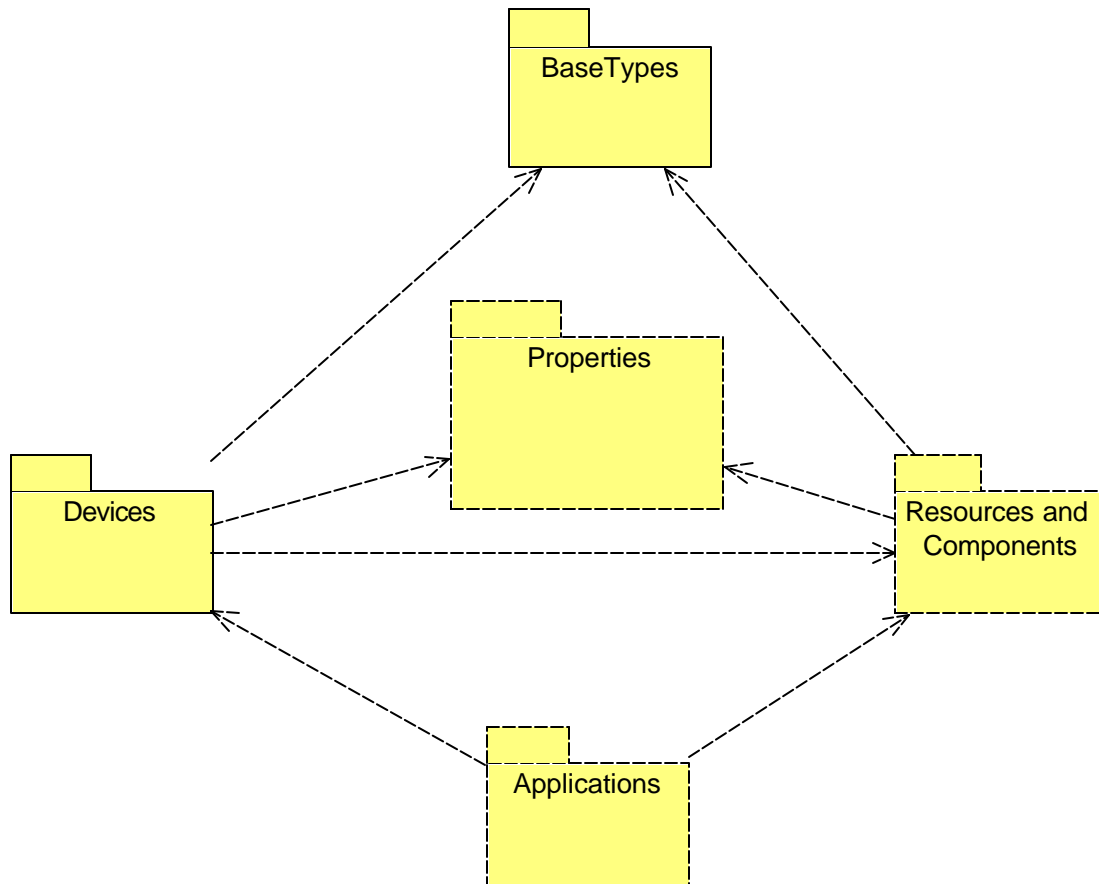


Figure 4 — Common Services Overview.

7.1.1 Base Types

The Base Types defines the basic types that are used by multiple SWRADIO components

7.1.1.1 ErrorNumberType

Description

This <<enumeration>> is used to pass error number information in various exceptions. Those enumeration names starting with “E” map to the POSIX definitions, and can be found in IEEE Std 1003.1 1996 Edition.

Attributes

Those enumeration literal names starting with “CF_E” map to the POSIX definitions that begin without “CF_”, and can be found in IEEE Std 1003.1 1996 Edition. CF_NOTSET is not defined in the POSIX specification. CF_NOTSET is a specific value that is applicable for any exception when the method specific or standard POSIX error values are not appropriate.)

Enumeration Literal values are:

CF_NOTSET, CF_E2BIG, CF_EACCES, CF_EAGAIN, CF_EBADF, CF_EBADMSG, CF_EBUSY, CF_ECANCELED, CF_ECHILD, CF_EDEADLK, CF_EDOM, CF_EEXIST, CF_EFAULT, CF_EFBIG, CF_EINPROGRESS, CF_EINTR, CF_EINVAL, CF_EIO, CF_EISDIR, CF_EMFILE, CF_EMLINK, CF_EMSGSIZE, CF_ENAMETOOLONG, CF_ENFILE, CF_ENODEV, CF_ENOENT, CF_ENOEXEC, CF_ENOLCK, CF_ENOMEM, CF_ENOSPC, CF_ENOSYS, CF_ENOTDIR, CF_ENOTEMPTY, CF_ENOTSUP, CF_ENOTTY, CF_ENXIO, CF_EPERM, CF_EPIPE, CF_ERANGE, CF_EROFS, CF_ESPIPE, CF_ESRCH, CF_ETIMEDOUT, CF_EXDEV

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

.

7.1.1.2 InvalidFileName

Description

The InvalidFileName exception, as shown in Figure 5, is a type of SystemException that indicates an invalid file name. The error number indicates an ErrorNumberType value (e.g., CF_ENAMETOOLONG). The String msg attribute is used to provide information describing why the filename was invalid.

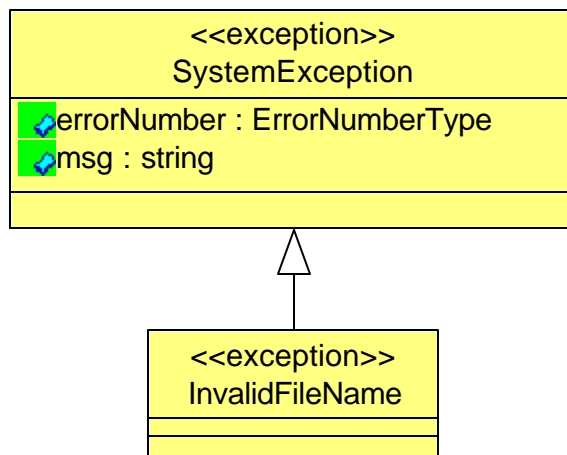


Figure 5 — InvalidFileName Definition

Attributes

No additional attributes.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.1.3 InvalidObjectReference

Description

The InvalidObjectReference exception indicates an invalid object reference error.

Attributes

- msg: String A message attribute is supplied for further information on the exception being raised.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.1.4 InvalidProfile

Description

The InvalidProfile exception indicates an invalid profile error.

Attributes

No additional attributes.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.1.5 Octet

Description

The Octet <<datatype>> is an unsigned integer with a range of values from 0 to 255.

Attributes

No additional attributes.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.1.6 OctetSequence

Description

This type is an unbounded sequence of octets as shown in Figure 6.

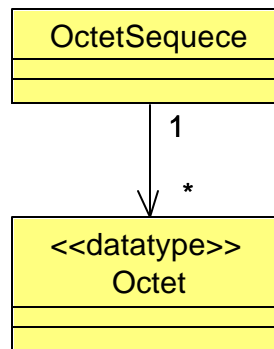


Figure 6 — OctetSequence Definition

Attributes

No additional attributes.

Associations

- octet: Octet [*] A sequence of Octets.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.1.7 Properties

Description

The Properties, as shown in Figure 7, is an unbounded sequence of PropertyValue(s), which is used in defining a sequence of id and value pairs.

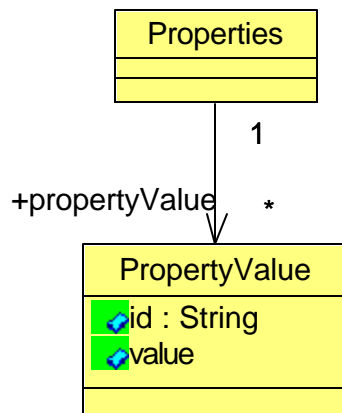


Figure 7 — Properties Definition

Attributes

No additional attributes.

Associations

- propertyValues (*):
PropertyValue A set of PropertyValues.

PropertyValue

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.1.8 PropertyValue

Description

PropertyValue is a type used to hold any «datatype». The element has two attributes, id and value. The id attribute indicates the kind of value and type (e.g., frequency, preset, etc.).

Attributes

- | | |
|--|--|
| <ul style="list-style-type: none">• id: String• value: Classifier | <p>The id attribute identifies a specific property of the component.</p> <p>The value attribute contains the property's value.</p> |
|--|--|

Associations

- | | |
|--|---|
| <ul style="list-style-type: none">• properties: Properties [1] | <p>The container for a set of PropertyValue's..</p> |
|--|---|

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

The Classifier for value is a <<datatype>> or Properties.

Semantics

No additional semantics.

7.1.1.9 StringSequence

Description

This type is an unbounded sequence of Strings as shown in Figure 8.

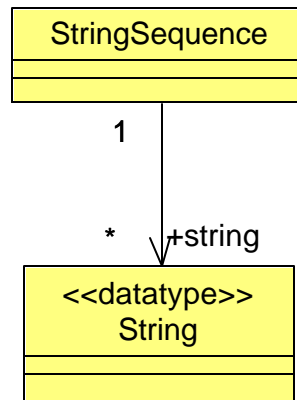


Figure 8 — StringSequence Definition

Attributes

No additional attributes.

Associations

- string: String [*] A StringSequence consists of a set of Strings.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.1.10 SystemException

Description

SystemException is an exception that denotes a type of system exception as shown in Figure 9.

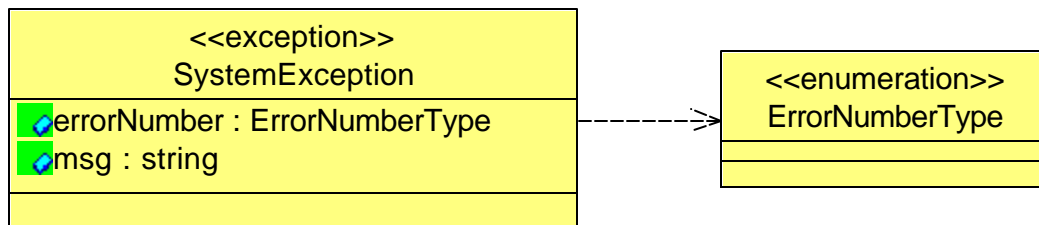


Figure 9 — SystemException Definition

Attributes

- errorNumber: ErrorNumberType The errorNumber indicates the type of system error.
- msg: String The message attribute is used to add additional information on the error that occurred.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.1.11 UnknownProperties

Description

The UnknownProperties exception indicates a set of properties unknown by the component as shown in Figure 10.

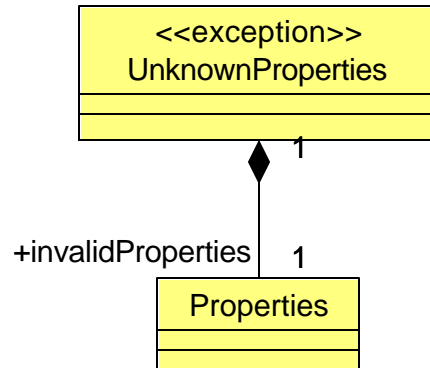


Figure 10 — UnknownProperties Definition

Attributes

No additional attributes

Associations

- invalidProperties: Properties [1] A list of invalid properties.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.2 Properties

This section defines the property types for swradio components. All properties are based upon primitive data type values (e.g., char, integer, string, etc.). The reason for this is two fold: 1) these primitive types are supported by distributed component middleware and 2) allows generic mechanisms to be built such as deployment, component interaction, and Human Computer Interface (HCI). The details of each property are described in the following subsections.

Table 1 — Properties Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
ConfigureQueryProperty		N/A	mode		A property that represents a configurable and/or queryable property.
ConfigureQuerySimpleProperty		ConfigureQueryProperty, RadioProperty			A property that represents a simple property that is configurable and/or queryable.
ExecutableProperty		SimpleProperty			A property that represents an executable parameter property.
RadioProperty		N/A	identifier, name, description		A property that represents the common attributes for all SWRADIO property types.
ServiceArtifactProperty					
SimpleProperty		RadioProperty	enumerations, type, units, value		A property that represents a RadioProperty that contains a primitive value type.
SimpleSequenceProperty		ConfigureQueryProperty, RadioProperty		Same primitive type definition is applied to all simple property's values.	A property that represents a Property that is a sequence of SimpleProperties.

Stereotype	Base Class	Parent	Tags	Constraints	Description
StructProperty	Property	ConfigureQueryProperty, RadioProperty			A property that represents a struct of SimpleProperties.
StructSequenceProperty	Property	ConfigureQueryProperty, RadioProperty			A property that represents a sequence of StructProperties.
TestProperty	Property	RadioProperty			A property that describes a test and its associated inputs and expected results.

7.1.2.1 ConfigureQueryProperty

Description

The <<abstract>> ConfigureQueryProperty as shown in Figure 11. The ConfigureQueryProperty extends the RadioProperty definition by adding configure and query mode of operation for a property. There are four types of ConfigureQueryProperty, which are: ConfigureSimpleSimpleProperty, SimpleSequenceProperty, StructProperty, and StructSequenceProperty. These properties are described in the following subsections.

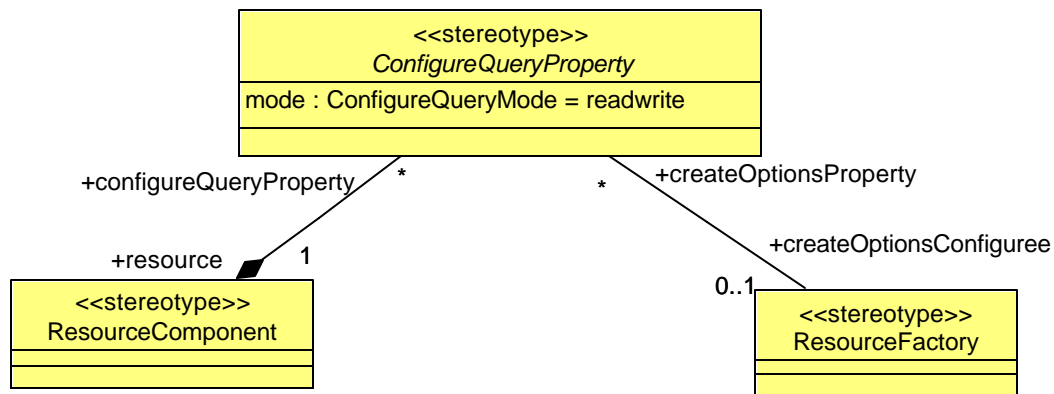


Figure 11 — ConfigureQueryProperty

Attributes

- mode: ConfigureQueryMode = readwrite The mode attribute indicates configure and query mode of operation for the property.

Associations

- createOptionsConfiguree: ResourceFactory [0..1] The createOptionsConfiguree creates a component using the createOptionsProperty.

Operations

No additional operations.

Types and Exceptions

- <<enumeration>>ConfigureQueryMode The ConfigureQueryMode type indicates the configure and query mode of operation values, which are:
 - readonly – The property can only be queried
 - readwrite – The property can be configured and queried.
 - writeonly – The property can be configured only.

Mapping

N/A.

Constraints

No additional semantics.

Semantics

The ConfigureQueryProperty is properties associated with the PropertySet interface implementations. The properties supported by a SWRADIO component are described in a component's descriptor.

7.1.2.2 ConfigureQuerySimpleProperty

Description

The ConfigureQuerySimpleProperty is a type of ConfigureQueryProperty and SimpleProperty as shown in Figure 12. The ConfigureQuerySimpleProperty provides a definition that indicates the mode of operation for a SimpleProperty.

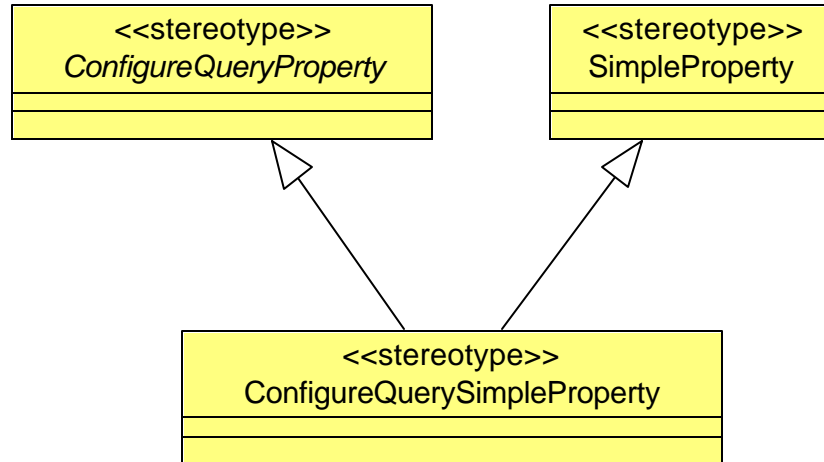


Figure 12 — ConfigureQuerySimpleProperty Definition

Attributes

No additional attributes.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

The value attribute cannot be a null String (length = 0) when the mode attribute is readwrite and writeonly.

Semantics

No additional semantics.

7.1.2.3 ExecutableProperty

Description

The ExecutableProperty is a type of SimpleProperty as shown in Figure 13 that defines executable parameters for an ExecutableCode element such as main process.

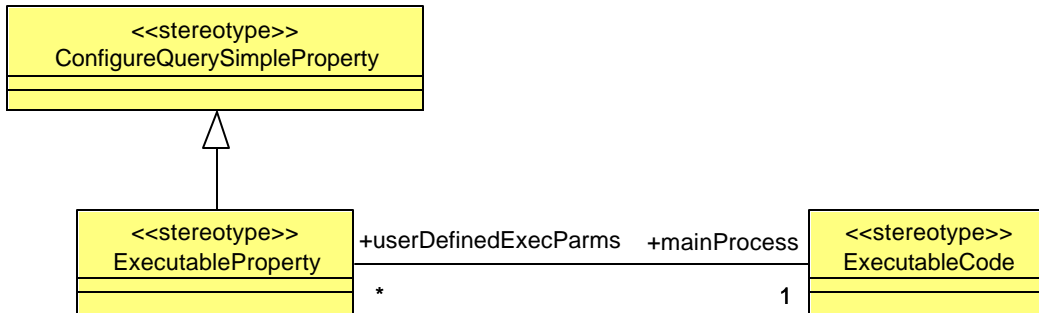


Figure 13 — ExecutableProperty Definition

Attributes

No additional operations.

Associations

- mainProcess: ExecutableCode [1] A main process may have executable parameters as part of its definition.

Operations

No additional operations.

Types and Exceptions

No additional operations.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.2.4 RadioProperty

Description

The RadioProperty is a property that provides the basic elements for all radio properties definitions.

Attributes

- description: String [0..1] The optional description attribute contains a description of a property.
- id: String The id attribute contains the property's identifier.

- name: String [0..1]

The optional name attribute contains a property's human readable label that can be used when the identifier is not human readable.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.2.5 ServiceArtifactProperty

Description

The ServiceArtifactProperty, a type of SimpleProperty, as shown in Figure 14 provides the means to define capability and/or capacity characteristics for a Service.

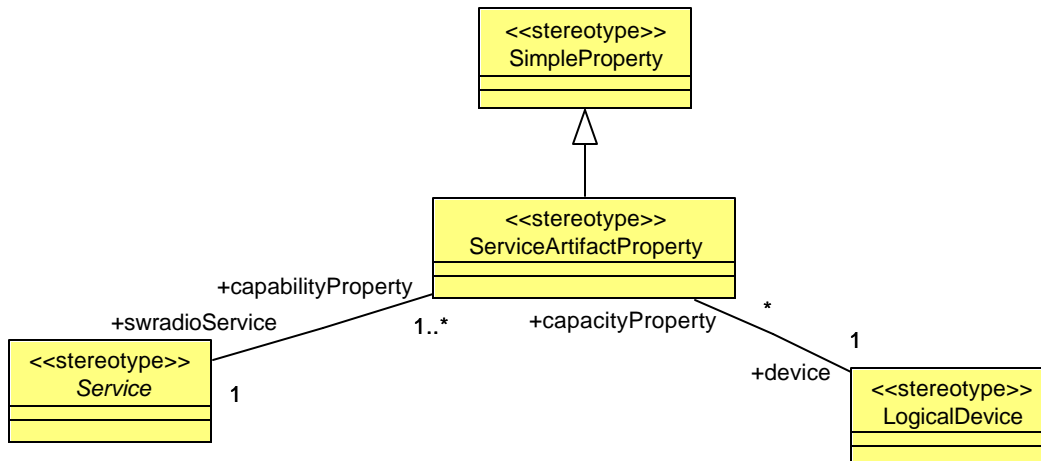


Figure 14 — ServiceArtifactProperty Definition

Attributes

- TBC TBC

Associations

- device: LogicalDevice [1] A LogicalDevice manages a set of capacities.
- SwradioService: Service [1..*] A Service manages a set of capability properties that describe the characteristics of the Service.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional semantics.

Semantics

No additional semantics.

7.1.2.6 SimpleProperty

Description

The SimpleProperty, a type of RadioProperty, is a simple primitive property type as shown in Figure 15. A SimpleProperty only contains a primitive data type value (e.g., character, integer, string, etc.).

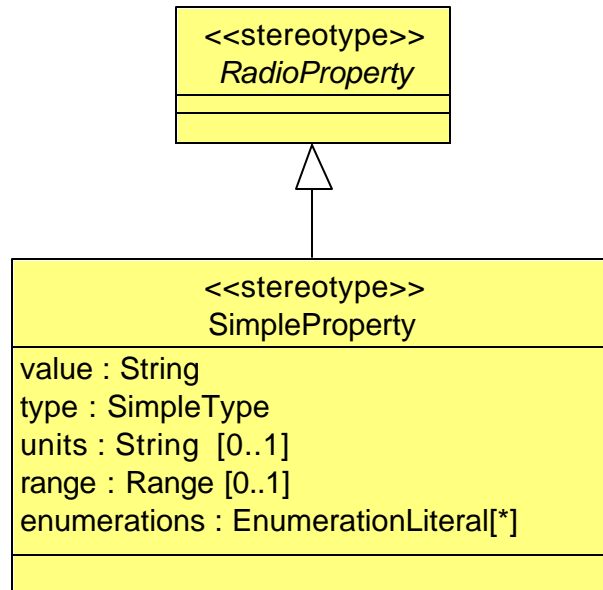


Figure 15 — SimpleProperty Definition

Attributes

- enumerations: EnumerationLiteral [*]

The enumerations attribute contains a set of enumeration literals and their associated value. This allows a SimpleProperty to be express as an enumeration.
- range: Range [0..1]

The optional range attribute is used to indicate the allowable min and max values for value attribute.
- type: SimpleType

The type attribute indicates the simple primitive data type. The type is used to convert the value string attribute to the correct type.
- units: String [0..1]

The optional units attribute is used to indicate the unit of measure for the value attribute.
- value: String

The value attribute contains the string value for a property

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- EnumerationLiteral The EnumerationLiteral type defines an enumeration value.
- Range The Range type defines the min and max allowable values for a property
- SimpleKind The SimpleKind type defines the primitive data type for the simple value.

Mapping

N/A.

Constraints

The value attribute must conform to string formats that are valid based upon the type attribute.

Semantics

No additional semantics.

7.1.2.7 SimpleSequenceProperty

Description

The SimpleSequenceProperty is a type of ConfigureQueryProperty and RadioProperty as shown in Figure 16. The SimpleSequenceProperty defines a list of simple values that are all based on the same simple type definition.

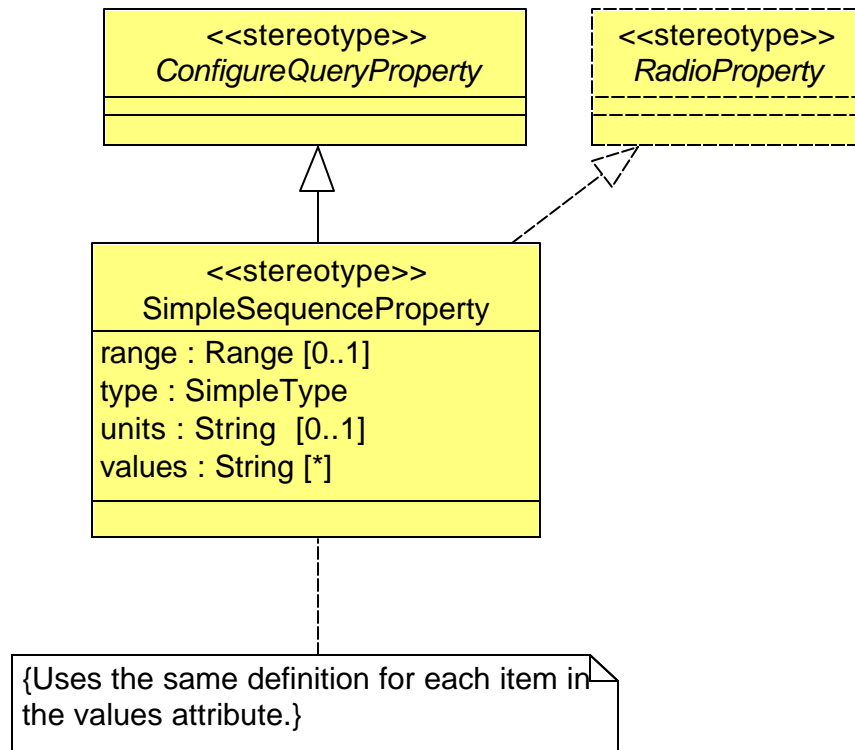


Figure 16 — SimpleSequenceProperty Definition

Attributes

- range: Range [0..1]
The optional range attribute is used to indicate the allowable min and max values for each value attribute.
- type: SimpleType
The type attribute indicates the simple primitive data type. The type is used to convert the value string attribute to the correct type.
- units: String [0..1]
The optional units attribute is used to indicate the unit of measure for the value attribute.
- values: String [*]
The values attribute contains a list of String values that conform to the type and range attributes.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

Each item in the values attribute must conform to the type attribute.

Semantics

No additional semantics.

7.1.2.8 StructProperty

Description

The StructProperty is a type of ConfigureQueryProperty and RadioProperty as shown in Figure 17. The StructProperty contains a list of SimpleProperties.

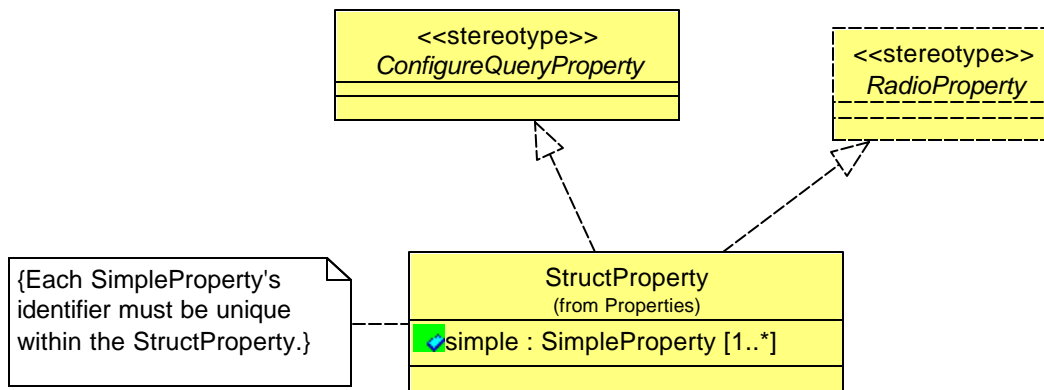


Figure 17 — StructProperty Definition

Attributes

- `simple: SimpleProperty [1..*]` A list of SimpleProperties that form the Struct definition

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

Each SimpleProperty's identifier in the simple attribute list must be unique within the StructProperty.

Semantics

No additional semantics.

7.1.2.9 StructSequenceProperty

Description

The StructSequenceProperty is a type of ConfigureQueryProperty as shown in Figure 18. The StructSequenceProperty defines

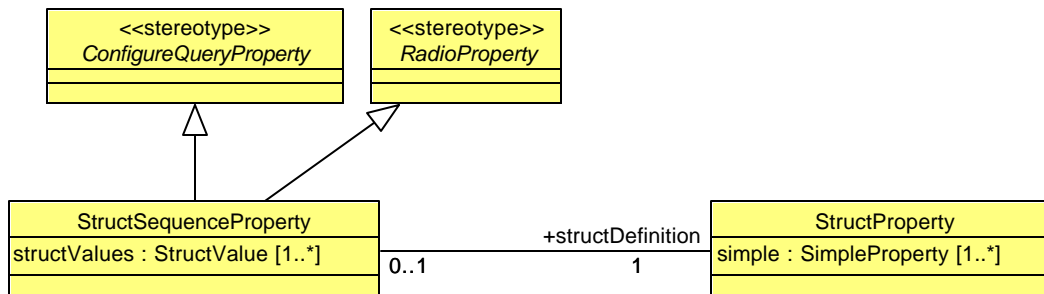


Figure 18 — StructSequenceProperty Definition

Attributes

- structValues: StructValue [1..*] The structValues attribute contains a list of StructProperty values, where each value is associated with the same StructProperty definition.

Associations

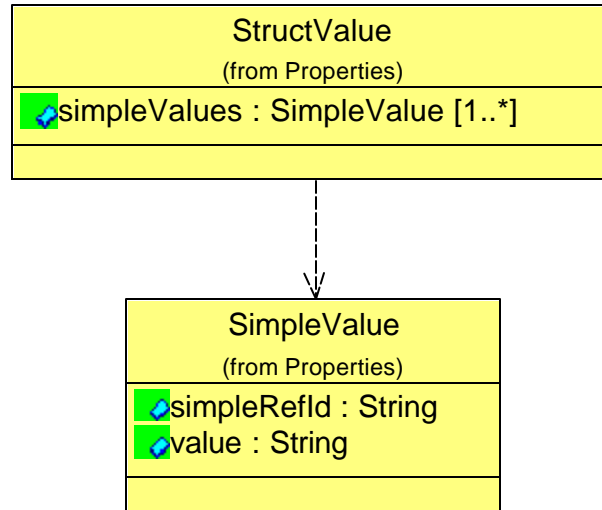
- structDefinition: StructProperty [1] A StructProperty provides the property definition for the StructSequenceProperty's structValues attribute.

Operations

No additional operations.

Types and Exceptions

- StructValue The StructValue defines a list of SimpleValues.



Mapping

N/A.

Constraints

The StructSequenceProperty uses the same StructProperty definition for each item in the structValues attribute.

Each StructValue's SimpleValue's simpleRefID attribute must match a SimpleProperty item in the associated structDefinition.

Each StructValue's SimpleValue's value attribute must match a SimpleProperty item in the associated structDefinition.

Semantics

No additional semantics.

7.1.2.10 TestProperty

Description

The TestProperty, a type of RadioProperty, as shown in Figure 19 provides the capability to define a test for a component.

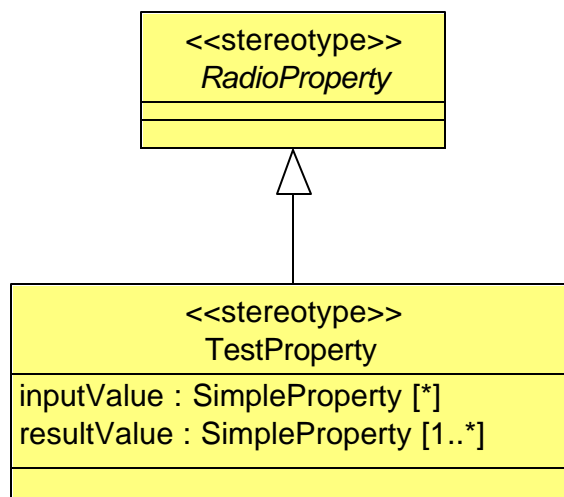


Figure 19 — TestProperty Definition

Attributes

- inputValue: SimpleProperty [*] The inputValue attribute defines the input parameters for a test.
- resultValue: SimpleProperty [1..*] The resultValue attribute defines the expected results that are returned from a test.

Associations

- applicationDeployer: ApplicationFactory [1] The associated ApplicationFactory that deployed the Application.
- assemblyController: ApplicationResource [1] The ApplicationAssembly's ApplicationResource that is the assemblyController that Application delegates operations to.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional semantics.

Semantics

No additional semantics.

7.1.3 Resources and Components

This section defines the elements for a swradio resource component. The Resource Component stereotypes are depicted in the table below, which are extensions of the UML Interface and Component. In Table 2 and in Figure 20 — Resource Component Overview depicts the base interfaces that are used in defining software radio components for waveform application, logical devices, and communication channel. The following subsections describe the details of these elements.

Table 2 — Resources Interfaces and Components Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
IControllableComponent	Interface	N/A			An Interface that represents basic operations for controlling a swradio's component.
ILifeCycle	Interface	N/A			An Interface that represents basic life cycle operations for a component.
IPortConnector	Interface	N/A			An Interface that represents basic port connections operations for a swradio's component.
IPortSupplier	Interface	N/A			An Interface that represents basic operation for getting a port provider object reference.
IPropertySet	Interface	N/A			An Interface that represents basic property operations for a swradio's component.
IResource	Interface	IControllableComponent, LifeCycle, IPropertySet, ITestableObject, IPortConnector, IPortSupplier			An Interface for a swradio's component.
ITestableObject	Interface	N/A			An Interface that represents basic test operation for a swradio's component.
ResourceComponent	Component	SWRadioComponent			A Component that represents a component of a waveform application.
ResourceFactory	Component	SWRadioComponent			A Component that manages ResourceComponents

Stereotype	Base Class	Parent	Tags	Constraints	Description
SWRadioComponent	Component	N/A			A Component that represents the base definition for all software radio Components.
SWRAPI	Interface	N/A			An interface implemented or required by a waveform or logical device component.
swrapiRealization	Association	N/A			An association that indicates a realization of a SWRAPI.
swrapiUsage	Association	N/A			An association that indicates a usage of a SWRAPI.

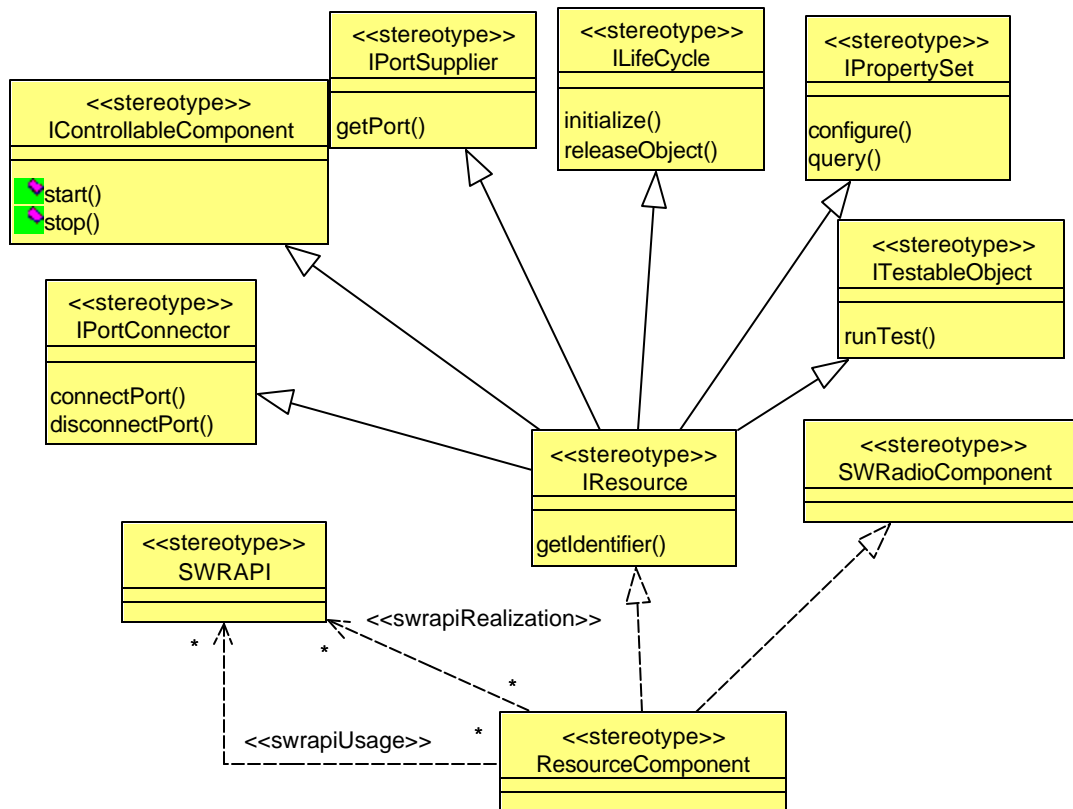


Figure 20 — Resource Component Overview

7.1.3.1 IControllableComponent

Description

The IControllableComponent interface defines the generic operations for controlling a swradio's component.

Attributes

No additional attributes.

Associations

No additional associations.

Operations

- `start(): {raises = (StartError)}`

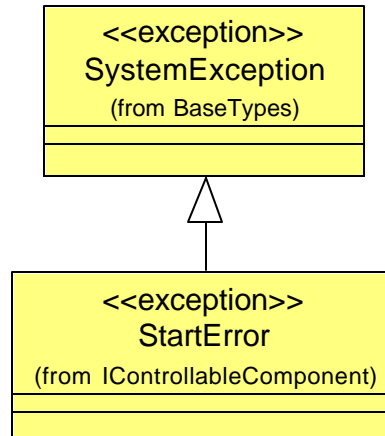
The start operation is provided to command a component implementing this interface to start internal processing. The start operation puts the component in an operating condition. This operation does not return a value. The start operation raises the StartError exception if an error occurs while starting the component.
- `stop(): {raises = (StopError)}`

The stop operation is provided to command a component implementing this interface to stop internal processing. The stop operation disables all current operations and put a component in a non-operating condition. This operation does not return a value. The stop operation raises the StopError exception if an error occurs while stopping the component.

Types and Exceptions

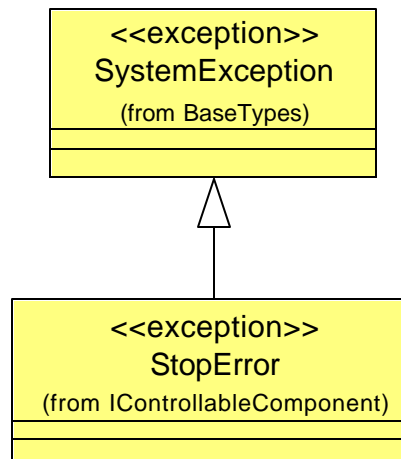
- `<<exception>>StartError`

The StartError, a type of SystemException, indicates that an error occurred during an attempt to start the Resource. The error number value (e.g., CF_EDOM, CF_EPERM, CF_ERANGE) and message is component-dependent, providing additional information describing the reason for the error.



- <<exception>>StopError

The **StopError**, a type of SystemException, indicates that an error occurred during an attempt to stop the *Resource*. The error number (e.g., CF_ECANCELED, CF_EFAULT, CF_EINPROGRESS) and message is component-dependent, providing additional information describing the reason for the error.



Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.3.2 ILifeCycle

Description

The ILifeCycle interface defines the generic operations for initializing or releasing instantiated component-specific data and/or processing elements.

Attributes

No additional attributes.

Associations

No additional associations.

Operations

- initialize(): {raises = (InitializeError)}

The purpose of the initialize operation is to provide a mechanism to set a component to a known initial state. (For example, data structures may be set to initial values, memory may be allocated, hardware devices may be configured to some state, etc.). The Initialization behavior is component implementation dependent.

This operation does not return a value. The initialize operation raises InitializeError when an initialization error occurs

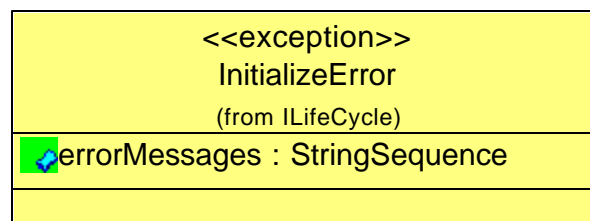
- releaseObject(): {raises = (ReleaseError)}

The purpose of releaseObject is to provide a means by which an instantiated component may be torn down. The releaseObject operation releases all internal memory allocated by the component during the life of the component. The releaseObject operation tears down the component from the Operating Environment (OE). This operation does not return a value. The releaseObject operation raises ReleaseError if an error occurs

Types and Exceptions

- <<exception>>InitializeError
(errorMessage:
StringSequence)

The InitializeError exception indicates an error occurred during component initialization. The message is component-dependent and provides additional information describing the reason the error occurred.

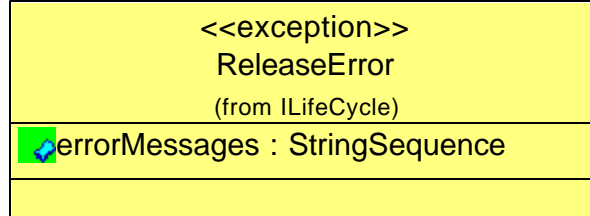


- <<exception>>ReleaseError
(errorMessage:
StringSequence)

The ReleaseError exception indicates an error occurred during component releaseObject. The message is component-dependent and provides additional information

StringSequence)

describing the reason the error occurred.



Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.3.3 IPortSupplier

Description

This interface provides the getPort operation for those components that provide provider ports.

Attributes

No additional attributes.

Associations

No additional associations.

Operations

- getPort(in name: String, return Object): {raises = (UnknownPort)}

The getPort operation provides a mechanism to obtain a specific provider Port. The getPort operation returns the object reference that is associated with the input port name. The getPort operation raises UnknownPort if the port name is invalid.

Types and Exceptions

- <<exception>>UnknownPort

The UnknownPort exception is raised if an undefined provider port is requested.

Mapping

NA.

Constraints

The getPort operation supports the entire provider named ports as stated in the component's descriptor file..

Semantics

No additional semantics.

7.1.3.4 IPortConnector

Description

The IPortConnector interface provides operations for managing associations between ports. The IPortConnector interface is used to connect a Required port to a Provider port.

Attributes

No additional attributes.

Associations

No additional associations.

Operations

- connectPort(in
requiredPortName: String, in
connection: Object, in
connectionId: String): {raises =
(InvalidPort, OccupiedPort)}

Components require the connectPort operation to establish associations between required and provider Ports. Ports provide communication mechanisms through which data and/or control pass. The connectPort operation makes a connection to a component's provider port identified by its input parameters. This operation does not return a value.

A port may support several connections. The input connectionId is a unique identifier to be used by disconnectPort when breaking this specific connection. The connectPort operation raises InvalidPort when connection is an invalid connection for this Port.

The connectPort operation raises OccupiedPort when unable to accept the connections because the Port is already fully occupied.

- disconnectPort (in
requiredPortName: String, in
connectionId: String): {raises =
(InvalidPort)}

Components require the disconnectPort operation in order to allow producercomponents to disassociate themselves from their counterparts (consumer). The disconnectPort operation breaks the connection to the component identified by connectionId. The disconnectPort operation raises InvalidPort when the requiredPortName or connectionId passed to disconnectPort is not connected or associated with the component. This operation does not return a value.

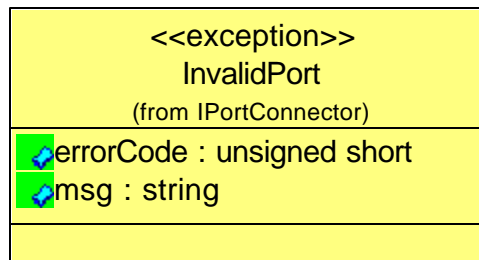
Types and Exceptions

- <<exception>>InvalidPort
The InvalidPort exception indicates one of the following errors has occurred in the specification of a Port association:

1. errorCode 1 means the Provider Port component is invalid (unable to narrow object reference) or illegal object reference,

2. errorCode 2 means the Provider Port name is not found (not used by this Port).

3. errorCode 3 means the Required Port name does exist for this component.



- <<exception>>OccupiedPort
The OccupiedPort exception indicates the Port is unable to accept any additional connections.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

A component establishes the operations for transferring data and control. The component also establishes the meaning of the data and control values. Examples of how components may use ports in different ways include: push or pull, synchronous or asynchronous, mono- or bi-directional, or whether to use flow control (e.g., pause, start, stop). The nature of IPortConnector fan-in, fan-out, or one-to-one is component dependent.

How components' ports are connected is described in a component assembly descriptor.

7.1.3.5 IPropertySet

Description

The IPropertySet interface defines configure and query operations to access component properties/attributes.

Attributes

No additional attributes.

Associations

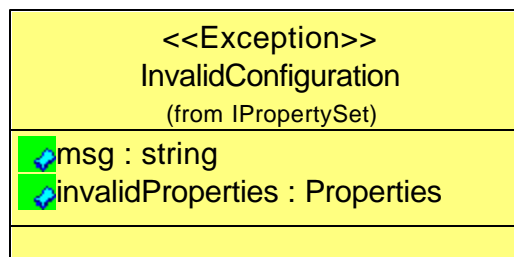
No additional associations.

Operations


- configure**(in configProperties: Properties): {raises = (InvalidConfiguration, PartialConfiguration)}
 The configure operation allows id/value pair configuration properties to be assigned to components implementing this interface. The configure operation assign values to the properties as indicated in the configProperties argument. This operation does not return a value. The configure operation raises PartialConfiguration when some configuration properties were successfully set and some configuration properties were not successfully set. The configure operation raises InvalidConfiguration when a configuration error occurs that prevents any property configuration on the component.
- query**(inout configProperties: Properties): {raises = (UnknownProperties)}
 The query operation allows a component to be queried to retrieve its properties. The query operation returns all the component queryable properties if the input configProperties are zero size. The query operation returns only those id/value pairs specified in the input configProperties if the configProperties are not zero size. The query operation raises UnknownProperties when one or more properties being requested are not known by the component.

Types and Exceptions

- <<exception>>InvalidConfiguration** (msg: String, invalidProperties: Properties)
 The InvalidConfiguration exception indicates the configuration of a component has failed (no configuration at all was done). The message is component-dependent, providing additional information describing the reason why the error occurred. The invalidProperties returned indicate the properties that were invalid.



- <<exception>>Partial Configuration** (invalidProperties: Properties)
 The PartialConfiguration exception indicates the configuration of a Component was partially successful. The invalidProperties returned indicate the properties that were invalid.

<<exception>> PartialConfiguration (from IPropertySet)	
	invalidProperties : Properties

Mapping

N/A.

Constraints

Valid properties for the *configure* operation are ConfigureQueryProperties that are readwrite and writeonly properties as stated in the component's descriptor.

Valid properties for the query operation shall at a minimum be the configure, readwrite, and readonly properties, and allocation properties that have an action value of "external" as referenced in the component's SPD.

Semantics

No additional semantics.

7.1.3.6 ITestableObject

Description

The ITestableObject interface defines a set of operations that can be used to test component implementations.

Attributes

No additional attributes.

Associations

No additional associations.

Operations

- runTest(in testId: unsigned long ,
inout testValues: Properties):
{raises = (UnknownTest,
UnknownProperties)}
- The runTest operation allows components to be "blackbox" tested. This allows Built-In Test (BIT) to be implemented and this provides a means to isolate faults (both software and hardware) within the system. The runTest operation uses the testId parameter to determine which of its predefined test implementations should be performed. The runTest operation does not execute any testing when the input testId or any of the input testValues are not known by the component or are out of range. The testValues parameter Properties (id/value pair(s)) is used to provide additional information to the implementation-specific test to be run. The runTest operation returns the result(s) of the test in the

testValues parameter.

Tests to be implemented by a component are component-dependent and may be specified in a component's Properties descriptor. The runTest operation raises UnknownTest when there is no underlying test implementation that is associated with the input testId given.

The runTest operation raises UnknownProperties when the input parameter testValues contains any input test parameters and DataTypes that are not known by the component's test implementation or any values that are out of range for the requested test. The UnknownProperties's invalidProperties attribute contains the invalid inputValues properties id(s) that are not known by the component or the value(s) that are out of range.

Types and Exceptions

- <<exception>>UnknownTest The UnknownTest exception indicates the requested testId for a test to be performed is not known by the component.

Mapping

N/A.

Constraints

Valid testId(s) and both input and output testValues (properties) for the runTest operation shall at a minimum be test properties defined in the properties test element of the component's Properties Descriptor (refer to Appendix D Domain Profile). The testId parameter corresponds to the XML attribute testId of the property element test in a propertyfile

Semantics

No additional semantics.

7.1.3.7 IResource

Description

The IResource interface, as shown in **Error! Reference source not found.**, provides a common API for the control and configuration of software radio components (waveform and device). The Resource interface inherits from the **Error! Reference source not found.**, **Error! Reference source not found.**, **Error! Reference source not found.**, **Error! Reference source not found.**, and **Error! Reference source not found.** interfaces

Attributes

No additional attributes.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.3.8 SWRADIOComponent

Description

The SWRADIOComponent, as shown in Figure 21, depicts the relationships for any software radio component.

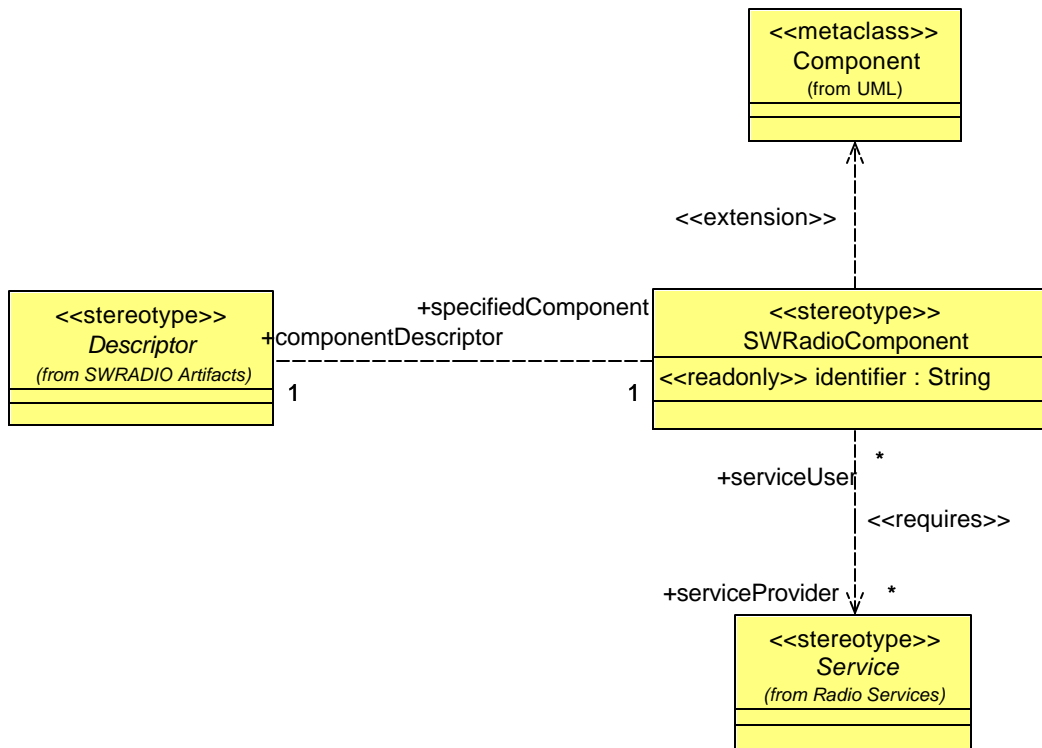


Figure 21 — SWRadioComponent Definition

Attributes

- `<<readonly>>identifier: String` The unique identifier for a component.
- `<<readonly>>softwareProfile: String` The profile descriptor (data/command uses and provides ports, configure and query properties, capacity properties, status properties, etc.)

Associations

- `componentDescriptor: Descriptor [1]` A SWRADIO component has at least one descriptor that describes the component's characteristics such as ports and properties.
- `serviceProvider: Service [*]` A SWRADIO component can be optional associated with zero to many Services.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.3.9 ResourceComponent

Description

The ResourceComponent, as shown in Figure 22, provides the component definition for software radio resource component.

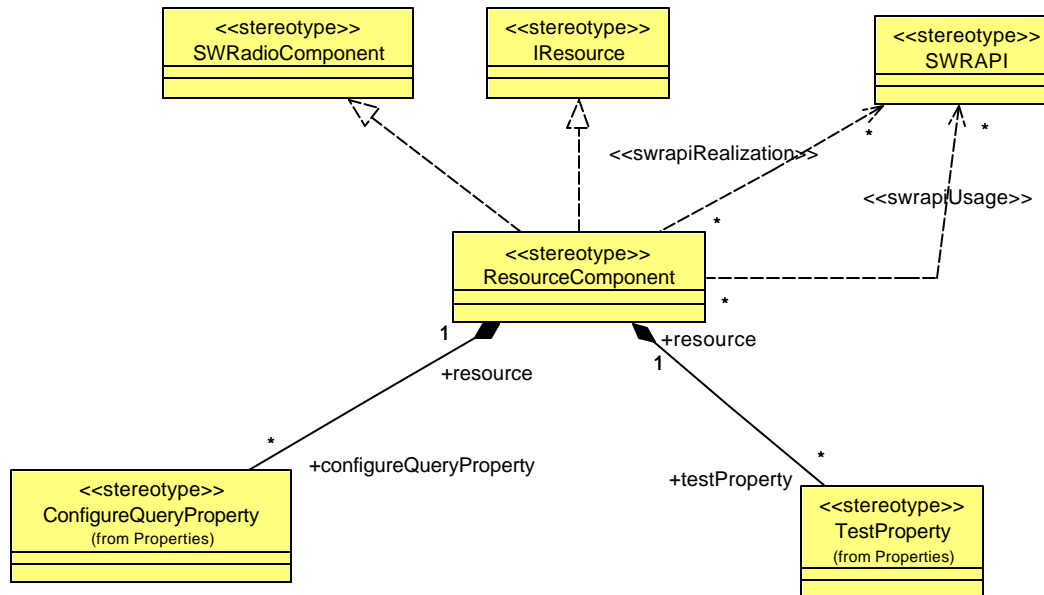


Figure 22 — ResourceComponent Definition

Attributes

No additional attributes.

Associations

- configureQueryProperty: ConfigureQueryProperty [*] A SWRADIO ResourceComponent may have zero to many configurable and queryable properties.

- testProperty: TestProperty [*] A SWRADIO ResourceComponent may have zero to many test properties.
- <<swrapiRealization>>:SWRAPI [*] A ResourceComponent may realize many SWRAPIs depending on type of ResourceComponent.
- <<swrapiUsage>>:SWRAPI [*] A ResourceComponent may require many SWRAPIs depending on type of ResourceComponent.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

The mapping to the ConfigureQueryProperty types to the ResourceComponent's configure and query's properties parameter are:

1. A ConfigureQuerySimpleProperty corresponds to a PropertyValue item in Properties list. The PropertyValue item's id matches the ConfigureQuerySimpleProperty's id attribute and the PropertyValue item's value matches the ConfigureQuerySimpleProperty's value attribute but is converted in native form that agrees with the ConfigureQuerySimpleProperty's type attribute.
2. A SimpleSequenceProperty corresponds to a PropertyValue item in Properties list. The PropertyValue item's id matches the SimpleSequenceProperty's id attribute and the PropertyValue item's value matches the SimpleSequenceProperty's values attribute that is converted to a primitive sequence type (as described in section Base Types 7.1.1), which agrees with the SimpleSequenceProperty's type attribute.
3. A StructProperty corresponds to a PropertyValue item in Properties list. The PropertyValue item's id matches the StructProperty's id attribute and the PropertyValue item's value contains a Properties type, where each StructProperty's SimpleProperty (id, value) corresponds to a Properties element in the list as described in item 1 above. The Properties element list size is based on the number of StructProperty's SimpleProperty items.
4. StructSequenceProperty corresponds to a PropertyValue item in Properties list. The PropertyValue item's id matches the StructSequenceProperty's id attribute and the PropertyValue item's value contains a Properties type. The Properties element list size is based on the number of StructSequenceProperty's structValues attribute. Each item in the Properties element also contains a Properties type that is used to contain a structValue (StructProperty) as described in item 3 above.

Semantics

No additional semantics.

7.1.3.10 ResourceFactory

Description

The ResourceFactory class, as shown in Figure 23, provides an optional mechanism for the creation of ResourceComponents by an Application.

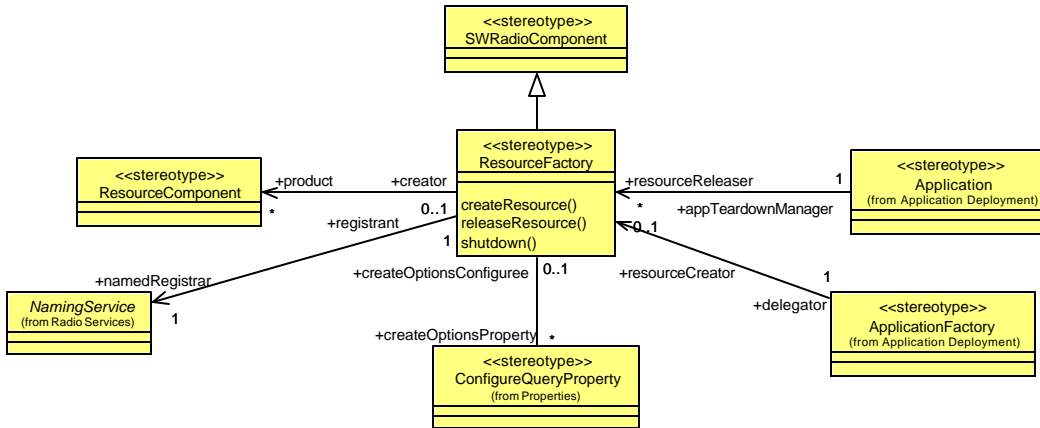


Figure 23 — ResourceFactory Definition

Attributes

No additional attributes.

Associations

- product: ResourceComponent [*] A ResourceComponent can be created from a ResourceFactory.
- appTeardownManager: Application[1] An Application may delegate the release of a ResourceComponent to a ResourceFactory.
- delegator: ApplicationFactory[1] An ApplicationFactory may delegate the ResourceComponent creation to a ResourceFactory.
- createOptionsProperty: ConfigureQueryProperty [*] These are the properties that a ResourceFactory understands and can used when creating up a ResourceComponent.
- NamedRegistrar: NamingService [1] The NamingService that contains a named ResourceFactory component reference.

Operations

- createResource(in resourceId: String, in qualifiers: Properties, Return IResource): {raises = The createResource operation provides the capability to create ResourceComponents in the same process space as the ResourceFactory or to return a

(CreateResourceFailure)}

ResourceComponent that has already been created.

The resourceId parameter is the identifier for Resource. The qualifiers are parameter values used by the ResourceFactory in creation of the ResourceComponent. The qualifiers may be used to identify, for example, specific subtypes of Resources created by a ResourceFactory. If no Resource exists for the given resourceId, the createResource operation creates a ResourceComponent. If the Resource already exists, the ResourceComponent's reference is returned. The createResource operation assigns the given resourceId to a new ResourceComponent and either set a reference count to one, when the ResourceComponent is initially created, or increment the reference count by one, when the ResourceComponent already exists. The reference count is used to indicate the number of times that a specific Resource reference has been given to requesting clients. This ensures that the ResourceFactory does not release a ResourceComponent that has a reference count greater than 0. When multiple clients have obtained a reference to the same ResourceComponent, each client will request release of the ResourceComponent when through with the ResourceComponent. However, the ResourceComponent must not be released until the release request comes from the last client in existence.

The createResource operation returns a reference to the created ResourceComponent or the existing ResourceComponent. The createResource operation returns a nil component reference when the operation is unable to create or find the ResourceComponent.

The createResource operation raises the CreateResourceFailure exception when it cannot create the ResourceComponent.

- releaseResource(in resourceId: String): {raises = (InvalidResourceId)}

The releaseResource operation provides the mechanism of releasing the ResourceComponent in the environment on the server side when all clients are through with a specific ResourceComponent. The releaseResource operation decrements the reference count for the specified ResourceComponent, as indicated by the resourceId parameter. The releaseResource operation makes the ResourceComponent no longer available (i.e., it is released from the environment) when the ResourceComponent's reference count is zero.

The releaseResource operation raises the InvalidResourceId exception if an invalid resourceId is received.

- Shutdown(): {raises = (ShutdownFailure)}

The shutdown operation provides the mechanism for releasing the ResourceFactory from the environment on the server side. The shutdown operation results in the

(ShutdownFailure)}

ResourceFactory being unavailable to any subsequent calls to its component reference (i.e. it is released from the environment). The shutdown operation raises the ShutdownFailure exception if unable to terminate the ResourceFactory.

Types and Exceptions

- <<exception>>InvalidResourceId The InvalidResourceId exception indicates the resourceId does not exist in the Factory.
- <<exception>>ShutdownFailure (msg: String) The ShutdownFailure exception indicates that the shutdown method failed to release the ResourceFactory from the CORBA environment due to the fact the Factory still contains ResourceComponents. The message is component-dependent, providing additional information describing why the shutdown failed.
- <<exception>>CreateResourceFailure The CreateResourceFailure exception, a type of System Exception, indicates that the createResource operation failed to create the Resource. The error number indicates an ErrorNumberType value (e.g., NOTSET, EBADMSG, EINVAL, EMSGSIZE, ENOMEM). The message is component-dependent, providing additional information describing the reason for the error.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

A ResourceFactory is used to create and tear down a Resource. The ResourceFactory is designed after the Factory Design Patterns. The factory mechanism provides client-server isolation among Resources (e.g., Network, Link, Modem, I/O, etc.) and provides an industry standard mechanism of obtaining a Resource without knowing its identity. An ApplicationAssembly is not required to use ResourceFactories to obtain, create, or tear down resources. An ApplicationAssembly's descriptor will determine which ResourceFactories are to be used by the ApplicationFactory.

7.1.4 Devices

The Devices sections define the set of components used to communicate and manage swradio physical devices. The Devices stereotypes are depicted in the Table 3 below, which are extensions of the UML Component. The following subsections describe the details of these elements.

Table 3 — Devices Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
DeviceDriver	Component	N/A			A Component that represents a device driver that interfaces with the hardware.
ExecutableDevice	Component	LoadableDevice			A Component that represents an executable device that manages OS processes on a device.
IAggregateDevice	Interface	N/A			An interface that provides the capability to form up composite devices.
LoadableDevice	Component	LogicalDevice			A Component that represents a loadable device that manages loading behavior on a device.
LogicalDevice	Component	ResourceComponent, Service			A Component that represents a logical device that abstracts the underlying hardware.

7.1.4.1 DeviceDriver

Description

The DeviceDriver, as shown in Figure 24, represents a component that interfaces with the SWRADIO communication equipment.

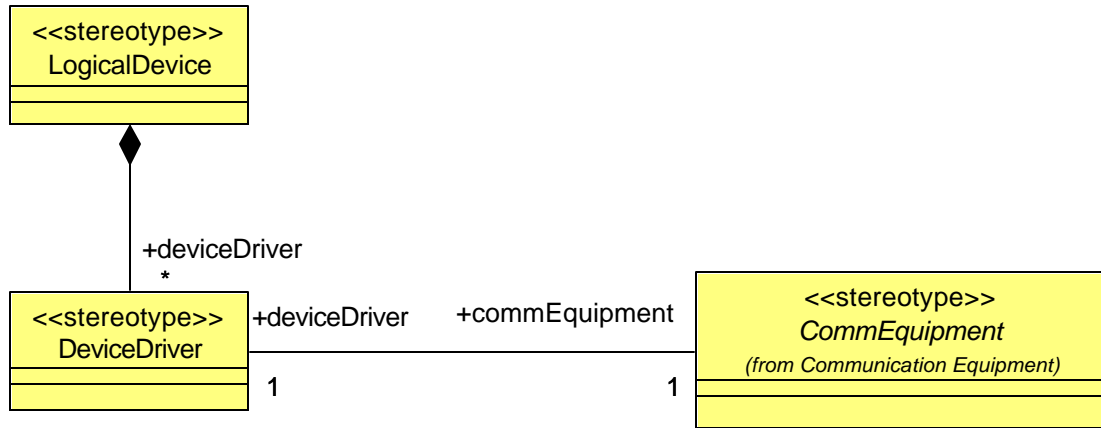


Figure 24 — DeviceDriver Definition

Attributes

No additional attributes.

Associations

- CommEquipment: CommEquipment [1] The hardware element the device driver is managing and controlling.
- logicalDevice: LogicalDevice [1] The logical device component that encapsulates the device driver.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.4.2 IAggregateDevice

Description

The IAggregateDevice, as shown in Figure 25, represents an interface that provides the capability to form up a composite device definition.

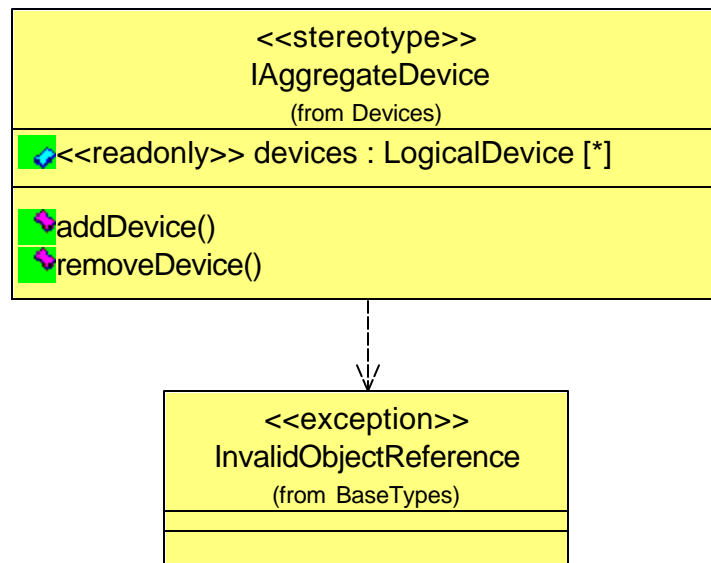


Figure 25— IAggregateDevice Definition

Attributes

- <<readonly>>devices: LogicalDevice [*]
The readonly devices attribute contains a list of devices that have been added to a composite Device or a sequence length of zero if a Device has no aggregation relationships with other Devices.

Associations

No additional associations.

Operations

- addDevice (in associatedDevice: LogicalDevice): {raises = (InvalidObjectReference)}
The addDevice operation provides the mechanism to associate a Device with another Device. The addDevice operation adds the input associatedDevice parameter to the devices attribute when the associatedDevice does not exist in the devices attribute. The associatedDevice is ignored when duplicated. This operation does not return any value. The addDevice operation raises the InvalidObjectReference when the input associatedDevice is a nil LogicalDevice reference.
- RemoveDevice (in associatedDevice: LogicalDevice): {raises = (InvalidObjectReference)}
The removeDevice operation provides the mechanism to disassociate a Device from another Device. The removeDevice operation removes the input associatedDevice parameter from the

}}

AggregateDevice's devices attribute. This operation does not return any value. The removeDevice operation raises the InvalidObjectReference when the input associatedDevice is a nil LogicalDevice reference or does not exist in the devices attribute.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

The IAggregateDevice interface provides aggregate behavior that can be used to add and remove Devices from an aggregate Device. Aggregated (child) Devices use this interface to add or remove themselves from composite (parent) Devices when being created or torn-down. When a LogicalDevice changes state or it is being torn down, its associated children LogicalDevices are affected.

7.1.4.3 LogicalDevice

Description

The LogicalDevice, as shown in Figure 26, defines the definition for a logical device component that abstracts the underlying hardware. The LogicalDevice is a type of service that would be resident within the radio. The LogicalDevice is viewed to be a type of ResourceComponent and Service with additional state and capacity behavior.

A logical Device is a functional abstraction for a set (e.g., zero or more) of hardware devices and provides the following attributes and operations:

- **State Management & Status Attributes** – This information describes the administrative, usage, and operational states of the device.
- **Capacity Operations** - In order to use a device, certain capacities (e.g., memory, performance, etc.) must be obtained from the Device. The capacity properties will vary among devices and are described in the Software Profile. A device may have multiple allocatable capacities, each having its own unique capacity model.

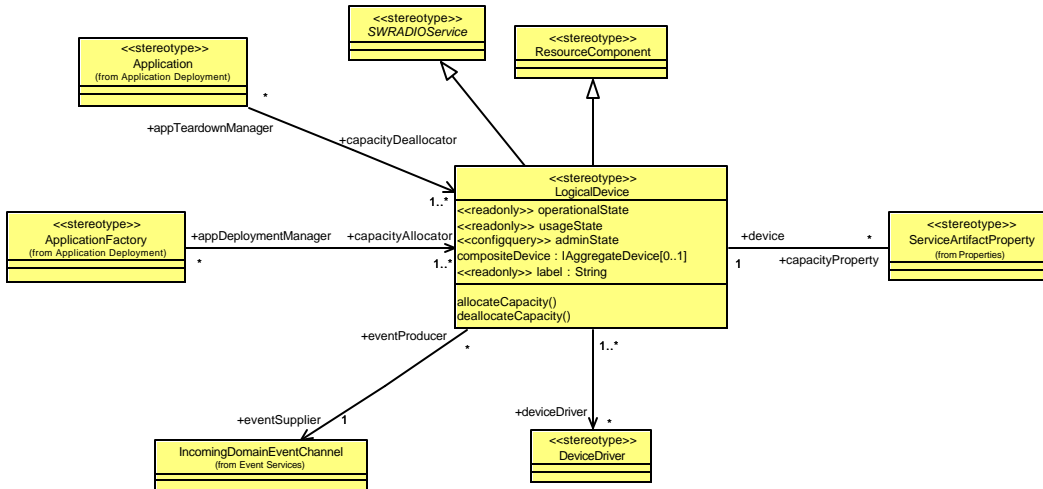


Figure 26 — LogicalDevice Definition

Attributes

- <<readwrite>>adminState:
AdminType

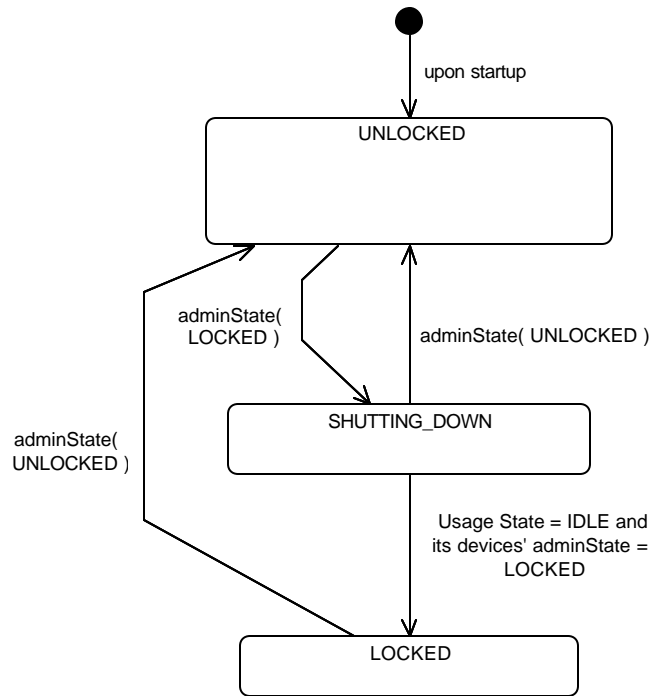
The administrative state indicates the permission to use or prohibition against using the device. The adminState attribute contains the device's admin state value. The adminState attribute only allows the setting of LOCKED and UNLOCKED values, where setting "LOCKED" is only effective when the adminState attribute value is UNLOCKED, and setting "UNLOCKED" is only effective when the adminState attribute value is LOCKED or SHUTTING_DOWN. Illegal state transitions commands are ignored.

The adminState attribute, upon being commanded to be LOCKED, transitions from the UNLOCKED to the SHUTTING_DOWN state and set the adminState to LOCKED for its entire aggregation of Devices (if it has any). The adminState then transitions to the LOCKED state when the Device's usageState is IDLE and its entire aggregation of Devices are LOCKED. Refer to TBC for an illustration of the above state behavior.

Whenever the adminState attribute changes, the Device sends an event to the Incoming Domain Management event channel with event data consisting of a StateChangeEvent type. The event data will be populated as follows:

1. The producerId field is the identifier attribute of the Device.
2. The sourceId field is the identifier attribute of the Device.
3. The stateChangeCategory field is ADMINISTRATIVE_STATE_EVENT.
4. The stateChangeFrom and stateChangeTo fields reflect the adminState attribute value before and after the state

change, respectively.



- `<<readonly>>compositeDevice: IAggregateDevice` The readonly compositeDevice attribute contains an IAggregateDevice component reference. The IAggregateDevice reference contains a list of the aggregated (e.g., children) devices associated with this composite (e.g., parent) Device or nil component/object reference if no association exists.
- `<<readonly>>label: String` The readonly label attribute contains the Device's label. The label attribute is the meaningful name given to a Device. The attribute could convey location information within the system (e.g., audio1, serial1, etc.).
- `<<readonly>>operationalState : OperationalType` The readonly operationalState attribute contains the device's operational state (ENABLED or DISABLED). The operational state indicates whether or not the device is functioning.

Whenever the operationalState attribute changes, the Device sends an event to the Incoming Domain Management event channel with event data consisting of a StateChangeEvent type. The event data is populated as follows:

1. The producerId field is the identifier attribute of the Device.
2. The sourceId field is the identifier attribute of the Device.
3. The stateChangeCategory field is OPERATIONAL_STATE_EVENT.
4. The stateChangeFrom and stateChangeTo fields reflect the

operationalState attribute value before and after the state change, respectively.

- <<readonly>>usageState:

UsageType

The readonly usageState attribute contains the Device's usage state (IDLE, ACTIVE, or BUSY, see TBC). UsageState indicates whether or not a device is actively in use at a specific instant, and if so, whether or not it has spare capacity for allocation at that instant.

Whenever the usageState attribute changes, the Device sends an event to the Incoming Domain Management event channel with event data consisting of a StateChangeEvent type. The event data is populated as follows:

1. The producerId field is the identifier attribute of the Device.
2. The sourceId field is the identifier attribute of the Device.
3. The stateChangeCategory field is USAGE_STATE_EVENT.
4. The stateChangeFrom and stateChangeTo fields reflect the usageState attribute value before and after the state change, respectively.

Associations

- appTeardownManager:
Application [*] A logical device can be associated with multiple deployed applications that have capacities allocated against devices that need to be deallocated upon Application teardown.
- appDeploymentManager:
ApplicationFactory [*] A logical device can be associated with multiple deployed applications that have capacities allocated by an ApplicationFactory.
- deviceDriver: DeviceDriver [*] The device drivers used by the logical device for communicating with the radio hardware.
- eventSupplier:
IncomingDomainEventChannel [1] The event channel used by a logical device for indicating state changes to a domain manager.
- CapacityProperty:
ServiceArtifactProperty [*] A logical device can have a set of capacities that are managed by a logical device. These capacity properties are managed by the logical device through its capacity operations and reflected by its states attributes.

Operations

- AllocateCapacity (in capacities: Properties, return Boolean):
{raises = (InvalidCapacity, The allocateCapacity operation provides the mechanism to request and allocate capacity from the Device. The allocateCapacity operation reduces the current capacities of the Device based upon the input capacities parameter, when the Device's adminState is UNLOCKED, Device's operationalState is ENABLED, and Device's

InvalidState) } usageState is not BUSY.

The allocateCapacity operation sets the Device's usageState attribute to BUSY, when the Device determines that it is not possible to allocate any further capacity. The allocateCapacity operation sets the usageState attribute to ACTIVE, when capacity is being used and any capacity is still available for allocation (refer to TBC).

The allocateCapacity operation returns "True", if the capacities have been allocated, or "False", if not allocated.

The allocateCapacity operation raises the InvalidCapacity exception, when the capacities are invalid or the capacity values are the wrong type or ID.

The allocateCapacity operation raises the InvalidState exception, when the Device's adminState is not UNLOCKED or operationalState is DISABLED.

- deallocateCapacity
(in capacities:
Properties): {raises = (
InvalidCapacity,
InvalidState)}

The deallocateCapacity operation provides the mechanism to return capacities back to the Device, making them available to other users.

The *deallocateCapacity* operation adjusts the current capacities of the *Device* based *upon* the input capacities parameter.

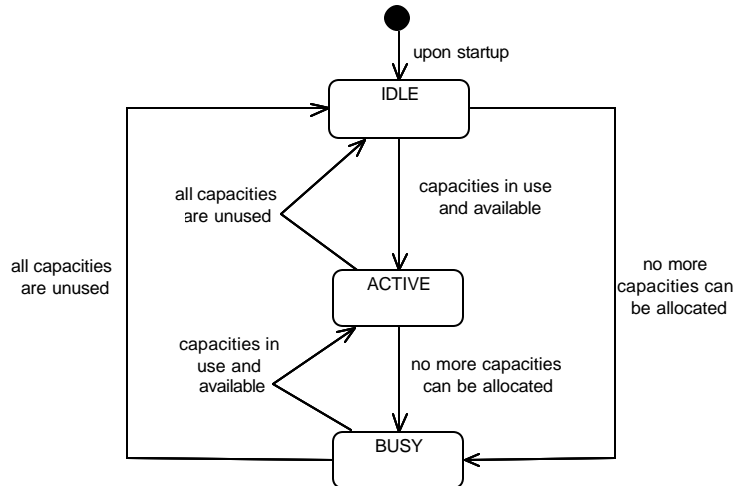
The *deallocateCapacity* operation sets the usageState attribute to ACTIVE when, after adjusting capacities, any of the *Device's* capacities are still being used.

The *deallocateCapacity* operation sets the usageState attribute to IDLE when, after adjusting capacities, none of the *Device's* capacities are still being used.

The *deallocateCapacity* operation sets the adminState attribute to LOCKED as specified in TBC.

The *deallocateCapacity* operation raises the InvalidCapacity exception, when the capacity ID is invalid or the capacity value is the wrong type. The InvalidCapacity exception states the reason for the exception.

The *deallocateCapacity* operation raises the InvalidState exception, when the *Device's* adminState is LOCKED or operationalState is DISABLED.



- `releaseResource():`
`{raises =`
`(releaseError)}`

The following behavior is in addition to the `ILifeCycle` `releaseObject` operation behavior.

The `releaseObject` operation calls the `releaseObject` operation on all of the `LogicalDevice`'s aggregated `Devices` attribute (i.e., those `Devices` that are contained within the aggregate `Device`'s `devices` attribute).

The `releaseObject` operation transitions the `Device`'s `adminState` to `SHUTTING_DOWN` state, when the `Device`'s `adminState` is `UNLOCKED`.

The `releaseObject` operation causes the `Device` to be unavailable, when the `Device`'s `adminState` transitions to `LOCKED`, meaning its aggregated `Devices` have been removed and the `Device`'s `usageState` is `IDLE`.

If the `LogicalDevice` is a composite of another `LogicalDevice` then the `releaseObject` operation causes the removal of its `LogicalDevice` from the `logicalDevice`'s `compositeDevice` (parent device).

If the `LogicalDevice` is registered with a `DeviceManager`, then the `releaseObject` operation unregisters its `LogicalDevice` from its `DeviceManager`.

Types and Exceptions

- `<<enumeration>>AdminType`

The `AdminType` defines the administrative state values for a device as follows:

- `LOCKED` – The device is reserved for administrative usage only, no capacity requests are granted by the device.
- `SHUTTING_DOWN` – a transition state from `UNLOCKED` to `LOCKED`, no further capacity

requests can be granted by the devices

- UNLOCKED – The device is available for normal usage.
- <<exception>>InvalidCapacity (msg: String, capacities: Properties) The InvalidCapacity exception returns the capacities that are not valid for this device.
- <<exception>>InvalidState (msg: String) The InvalidState exception indicates that the device is not capable of the behavior being attempted due to the state the Device is in. An example of such behavior is allocateCapacity.
- <<enumeration>>OperationalType The OperationalType defines the operational state values for a device as follows:
 - ENABLED – the device is functional
 - DISABLED – the device is non-functional
- <<enumeration>>UsageType The UsageType defines the usage state values for a device as follows:
 - IDLE – not in use
 - ACTIVE – in use, with capacity remaining for allocation, or
 - BUSY – in use, with no capacity remaining for allocation

Mapping

N/A.

Constraints

The LogicalDevice supports the capabilities and capacities properties as stated in the Device's softwareProfile attribute.

Capacities are queryable from the IPropertySet query operation.

Semantics

No additional semantics.

7.1.4.4 LoadableDevice

Description

The LoadableDevice, as shown in Figure 27, extends the LogicalDevice component by adding software loading and unloading behavior.

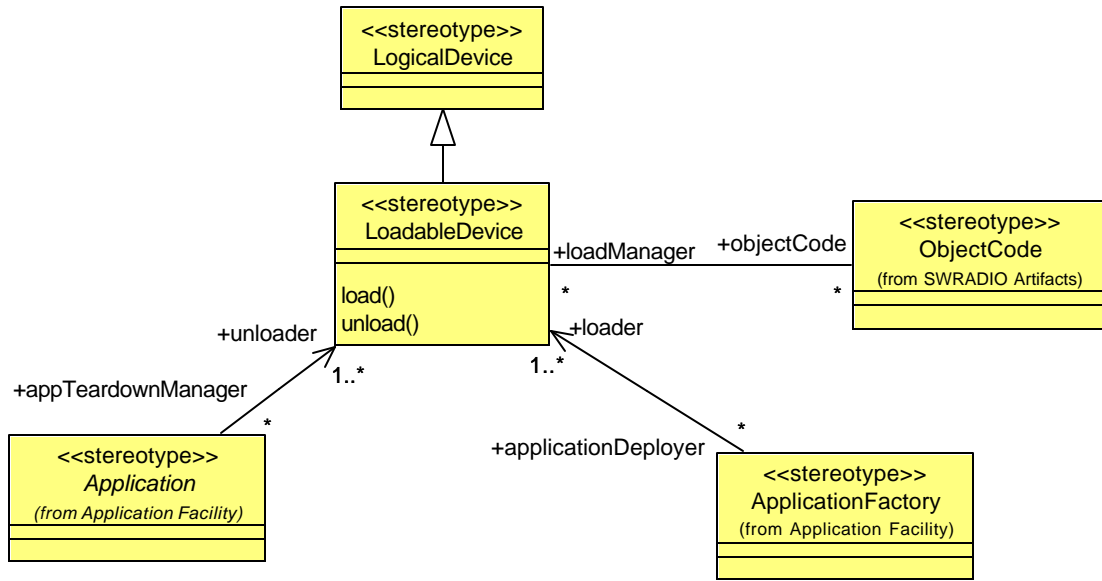


Figure 27 — Loadable Device Definition

Attributes

No additional attributes.

Associations

- applicationDeployer:**
 ApplicationFactory [*]
 A LoadableDevice may be associated with multiple ApplicationFactory components that deploys application ObjectCode artifacts.
- AppTeardownManager:**
 Application [*]
 A LoadableDevice may be associated with multiple Application components that teardown applications by unloading ObjectCode artifacts.
- ObjectCode:** ObjectCode [*]
 Zero to many ObjectCodes may be loaded on a device.

Operations

- load(in fs: FileSystem, in filename: String , in loadKind: LoadType): {raises = (InvalidState, InvalidLoadKind, InvalidFileName, LoadFail)}**
 The load operation provides the mechanism for loading software on a specific device. The loaded software may be subsequently executed on the Device, if the Device is an ExecutableDevice.

 The load operation loads a file on the specified device based upon the input loadKind and fileName parameters using the input FileSystem parameter to retrieve the file.

 The load operation keeps track of the number of times a

file has been successfully loaded.

The load operation raises the InvalidState exception when the Device's adminState is not UNLOCKED or operationalState is DISABLED.

The load operation raises the InvalidLoadKind exception when the input loadKind parameter is not supported.

The load operation raises the InvalidFileName exception when the file designated by the input filename parameter cannot be found.

The load operation raises the LoadFail exception when an attempt to load the device is unsuccessful.

- unload(in filename: String):
{raises = (InvalidState,
InvalidFileName)}

The unload operation provides the mechanism to unload software that is currently loaded. The unload operation decrements the load count for the input filename parameter by one. The unload operation unloads the application software on the device based on the input fileName parameter, when the file's load count equals zero.

The unload operation raises the InvalidState exception when the Device's adminState is LOCKED or its operationalState is DISABLED.

The unload operation raises the InvalidFileName exception when the file designated by the input filename parameter cannot be found.

Types and Exceptions

- <<enumeration>>LoadType

The LoadType defines the type of load to be performed which are:

- KERNEL_MODULE,
- DRIVER,
- SHARED_LIBRARY,
- EXECUTABLE

- <<exception>>InvalidLoadKind

The InvalidLoadKind exception indicates that the LoadableDevice is unable to load the type of file designated by the loadKind parameter.

- <<exception>>LoadFail
(ErrorNumberType errorNumber,
string msg)

The LoadFail exception indicates that the Load operation failed due to device dependent reasons. The LoadFail exception indicates that an error occurred during an attempt to load the device. The error number indicates an ErrorNumberType value (e.g. EACCES,

CF_EAGAIN, CF_EBADF, CF_EINVAL, CF_EMFILE, CF_ENAMETOOLONG, CF_ENOENT, CF_ENOMEM, CF_ENOSPC, CF_ENOTDIR). The message is component-dependent, providing additional information describing the reason for the error.

Mapping

N/A.

Constraints

The LoadableDevice supports the load types, capabilities, and capacities properties as stated in the Device's software profile.

Semantics

No additional semantics.

7.1.4.5 ExecutableDevice

Description

The ExecutableDevice, as shown in Figure 28, extends the LoadableDevice by adding execute and terminate behavior to a Device.

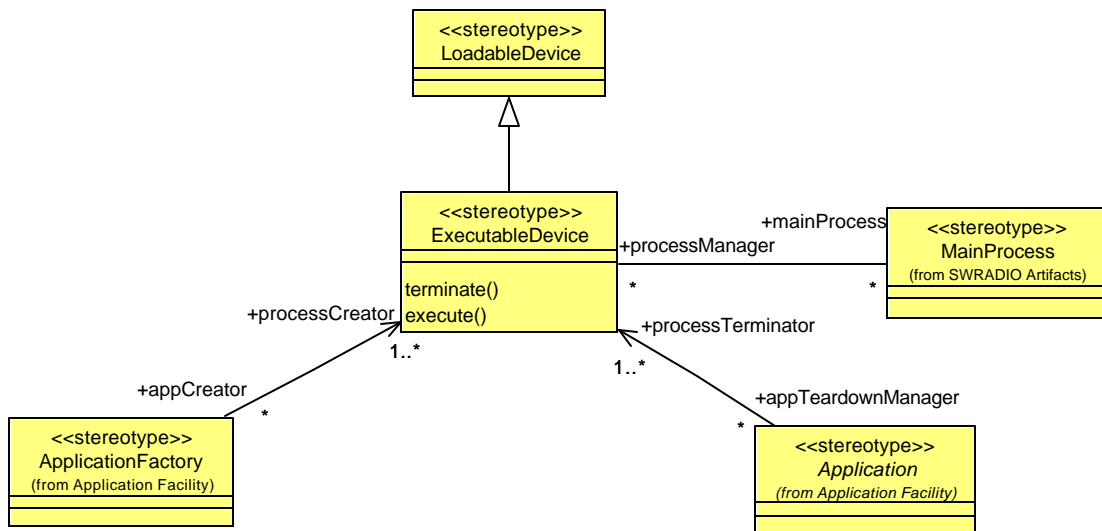


Figure 28 — ExecutableDevice Definition

Attributes

No additional attributes.

Associations

- AppCreator: ApplicationFactory [*] A ExecutableDevice may be associated with multiple ApplicationFactory components that deploys application MainProcess artifacts.
- AppTeardownManager: Application [*] A ExecutableDevice may be associated with multiple Application components that teardown applications by terminating MainProcess artifacts.
- mainProcess: MainProcess [*] Zero to many MainProcesses may be executed and terminated on a device.

Operations

- execute(in name: String, in options: Properties, in parameters: Properties, return ProcessID_Type): {raises = (InvalidState, InvalidFunction, InvalidParameters, InvalidOptions, InvalidFileName, ExecuteFail)}
- The *execute* operation provides the mechanism for starting up and executing a software process/thread on a device.
- The execute operation executes the function or file identified by the input name parameter using the input parameters and options parameters. Whether the input name parameter is a function or a file name is device-implementation-specific.
- The execute operation converts the input parameters (id/value string pairs) parameter to the standard argv of the POSIX exec family of functions, where argv(0) is the function name. The execute operation maps the input parameters parameter to argv starting at index 1 as follows, argv (1) maps to input parameters (0) id and argv (2) maps to input parameters (0) value and so forth. The execute operation passes argv through the operating system “execute” function.
- The execute operation input options parameters are STACK_SIZE_ID and PRIORITY_ID. The execute operation uses these options, when specified, to set the operating system’s process/thread stack size and priority, for the executable image of the given input name parameter.
- The execute operation returns a unique processID for the process that it created or a processID of minus 1 (-1) when a process is not created.
- The execute operation raises the InvalidState exception when the Device’s adminState is not UNLOCKED or operationalState is DISABLED.
- The execute operation raises the InvalidFunction exception when the function indicated by the input name parameter does not exist for the Device.
- The execute operation raises the InvalidFileName exception when the file name indicated by the input name

parameter does not exist for the Device.

The execute operation raises the InvalidParameters exception when the input parameters parameter item ID or value are not string types.

The execute operation raises the InvalidOptions exception when the input options parameter does not comply with sections STACK_SIZE_ID and PRIORITY_ID.

The execute operation raises the ExecuteFail exception when the operating system “execute” function for the device is not successful.

- terminate(in processId: ProcessID_Type): {raises = (InvalidProcess, InvalidState)}

The terminate operation provides the mechanism for terminating the execution of a process/thread on a specific device that was started up with the execute operation. The terminate operation terminates the execution of the process/thread designated by the processId input parameter on the Device.

The terminate operation raises the InvalidState exception when the Device’s adminState is LOCKED or operationalState is DISABLED.

The terminate operation raises the InvalidProcess exception when the processId does not exist for the Device.

Types and Exceptions

- <<exception>>ExecuteFail

The ExecuteFail exception, a type of SystemException, indicates that the Execute operation failed due to device dependent reasons. The ExecuteFail exception indicates that an error occurred during an attempt to invoke the execute function on the device. The error number indicates an ErrorNumberType value (e.g. CF_EACCES, CF_EBADF, CF_EINVAL, CF_EIO, CF_EMFILE, CF_ENAMETOOLONG, CF_ENOENT, CF_ENOMEM, CF_ENOTDIR). The message is component-dependent, providing additional information describing the reason for the error.

- <<exception>>InvalidFunction

The InvalidFunction exception indicates that a function, as identified by the input name parameter, hasn’t been loaded on this device.

- <<exception>>InvalidProcess

The InvalidProcess exception, a type of SystemException, indicates that a process, as identified by the processID parameter, does not exist on this device. The error number indicates an ErrorNumberType value (e.g., CF_ESRCH, CF_EPERM, CF_EINVAL). The message is

component-dependent, providing additional information describing the reason for the error.

- <<exception>>InvalidParameters (invalidParms: Properties)

The InvalidParameters exception indicates the input parameters are invalid on the execute operation. The InvalidParameters exception is raised when there are invalid execute parameters. Each parameter's ID and value must be a valid string type. The invalidParms is a list of invalid parameters specified in the execute operation.
- <<exception>>InvalidOptions (invalidOpts: Properties)

The InvalidOptions exception indicates the input options are invalid on the execute operation. The invalidOpts is a list of invalid options specified in the execute operation.
- ProcessID_Type: Integer

This type defines a process number within the system. Process number is unique to the Processor operating system that created the process.
- PRIORITY_ID : String = "PRIORITY"

The PRIORITY_ID is the identifier for the ExecutableDevice's execute options parameters. The value for a priority is an unsigned long.
- STACK_SIZE_ID = "STACK_SIZE"

The STACK_SIZE_ID is the identifier for the ExecutableDevice's execute options parameter. The value for a stack size is an unsigned long.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.5 Applications

The Applications sections define the set of components used to define applications and waveforms. The Applications stereotypes are depicted in Table 4 below, which are extensions of the UML Component. The following subsections describe the details of these elements.

Table 4 — Applications Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
ApplicationAssembly	Component	N/A			A Component that represents an assembly of ApplicationResou

					rces and SWRadioCompo nents.
ApplicationResource	Component	ResourceComponent			A Component that provides a common API for the control and configuration of an application resource component.
LinkLayerControlResource	Component	WaveformLayerReso urce			A Component that represents a standard Link Layer Control component of the OSI layer model.
MediumAccessControlResource	Component	WaveformLayerReso urce			A Component that represents a standard Medium Access Control component of the OSI layer model.
NetworkLayerResource	Component	WaveformLayerReso urce			A Component that represents a standard Network Layer component of the OSI layer model.
PhysicalLayerResource	Component	WaveformLayerReso urce			A Component that represents a standard Physical Layer component of the OSI layer model.
ResourceFactory	Component	ResourceComponent			A Component that represents TBC
WaveformApplication	Component	ApplicationAssembly			A Component that represents a waveform application.
WaveformLayerResource	Component	ApplicationResource			A Component that represents a standard component of the

					OSI layer model.
--	--	--	--	--	------------------

7.1.5.1 ApplicationAssembly

Description

The ApplicationAssembly, as shown in Figure 29, provides a component assembly definition for a set of ApplicationResources and SWRadioComponents.

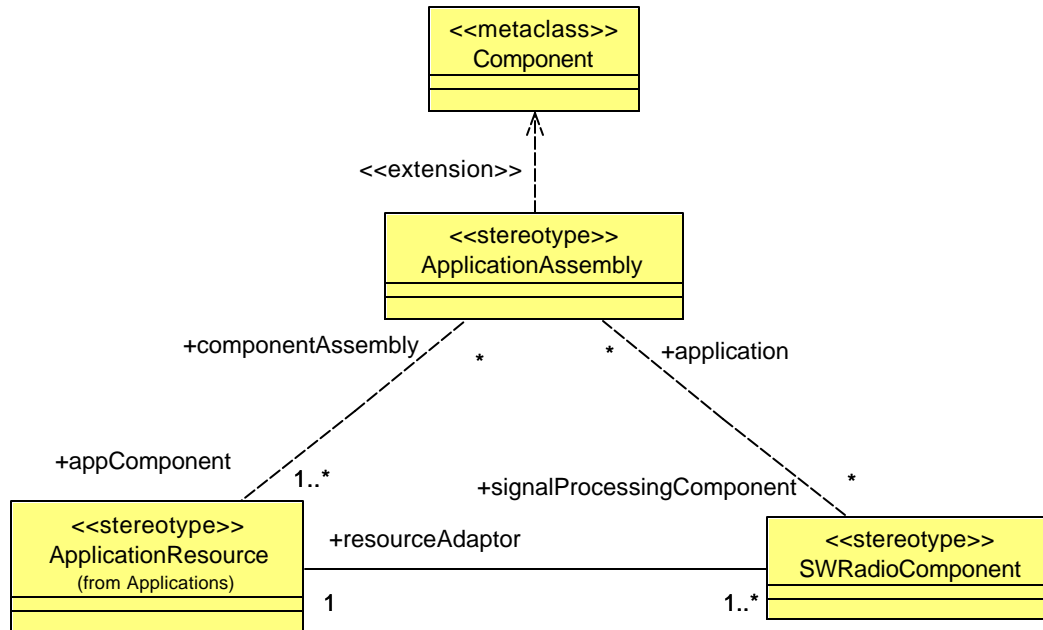


Figure 29 — ApplicationAssembly Definition

Attributes

No additional attributes.

Associations

- **appComponent:**
ApplicationResource [1..*] The set of ApplicationResources that are connected together to form the application assembly.
- **signalProcessingComponent:**
SWRadioComponent [*] The set of signal processing components that comprise the application assembly.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

TBC.

Semantics

No additional semantics.

7.1.5.2 ApplicationResource

Description

The ApplicationResource, as shown in Figure 30, provides a common API for the control and configuration of an application resource component.

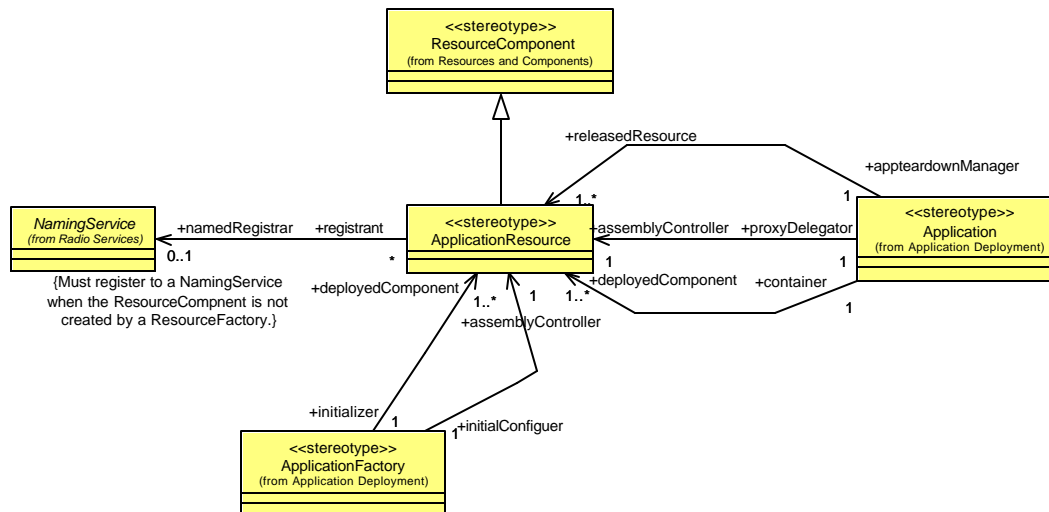


Figure 30— ApplicationResource Definition

Attributes

No additional attributes.

Associations

- appProxy: Application [1]

The Application that is the proxy for the ApplicationAssembly's AssemblyController. The Application delegates IResource operations to the

AssemblyController.

- appTeardownManager: Application [1] The Application that teardowns the deployed ApplicationAssembly's ApplicationResources.
- container: Application [1] The Application that contains the deployed ApplicationResources.
- initializer: ApplicationFactory [1] The ApplicationFactory that initializes the deployed ApplicationResources.
- initialConfigurer: ApplicationFactory [1] The ApplicationFactory that configures the deployed AssemblyController.
- namedRegistrar: NamingService [0..1] The optional NamingService that contains a named ApplicationComponent reference.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

An ApplicationResource shall be registered with a NamingService when a ResourceFactory does not create the ResourceComponent.

Semantics

ApplicationComponent references are contained in NamingService so the deployment machinery (e.g., ApplicationFactory) can obtain the component that got deployed.

7.1.5.3 WaveformLayerResource

Description

The WaveformLayerResource, as shown in Figure 31, specializes the ApplicationResource stereotype. The WaveformLayerResource stereotype provides an alternative mechanism to realize a waveform layer component, should a non-OSI layer be required. For standard OSI layers, the stereotypes that specialize the WaveformLayerResource should be used.

A WaveformLayerResource, as well as any specialization of it, can perform two types of communication. In the **horizontal communication** scenario, a waveform layer component communicates with one or more peer waveform layer components that are located in another radio set. In the **vertical communication** scenario, a waveform component communicates with other waveform components within the same radio set.

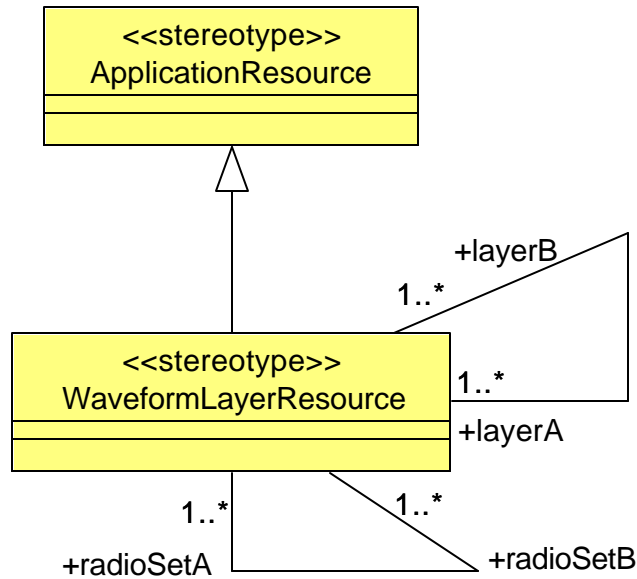


Figure 31— WaveformLayerResource Definition

Attributes

No additional attributes.

Associations

- radioSetA, radioSetB:
WaveformLayerResource [1..*]

This association shows the data transfer between two WaveformLayerResources that reside in different radio sets. This is an example of *horizontal communication*.
- layerA, layerB:
WaveformLayerResource [1..*]

This association shows the data transfer between two or more WaveformLayerResources that are in the same radio sets. This is an example of *vertical communication*.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.5.4 PhysicalLayerResource

Description

The PhysicalLayerResource, as shown in Figure 32, specializes the WaveformLayerResource stereotype. The PhysicalLayerResource stereotype provides a mechanism to realize a standard Physical Layer component of the OSI layer model. The standard facilities of a Physical Layer API are defined in the PIM facilities.

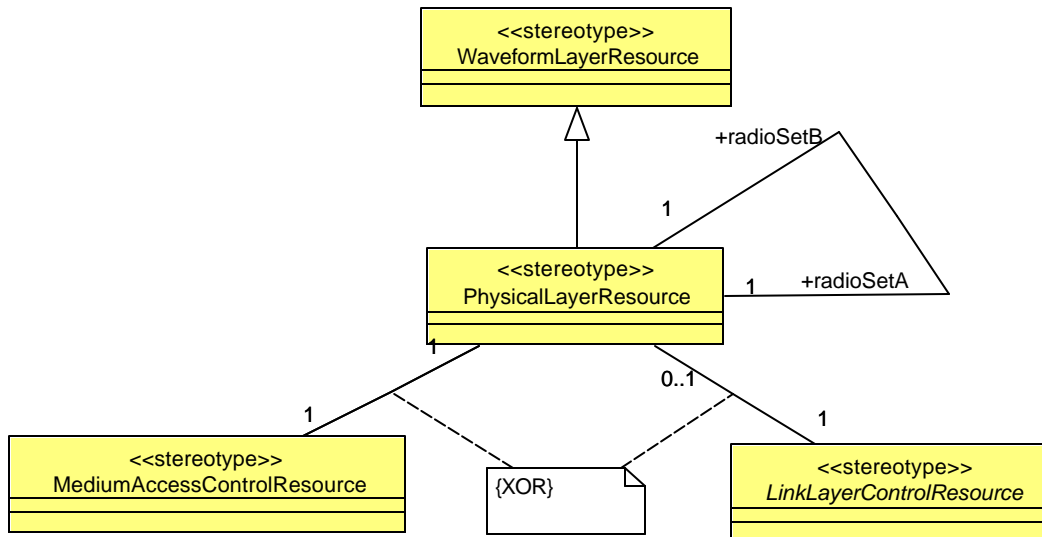


Figure 32— PhysicalLayerResource Definition

Attributes

No additional attributes.

Associations

- MediumAccessControlResource [1]

PhysicalLayerResource communicates with MediumAccessControlResource, and MediumAccessControlResource controls the transmission parameters related to the physical medium. Also, protocol data is communicated bi-directionally between those two components. (Vertical Communication)
- LinkLayerControllerResource [0..1]

PhysicalLayerResource communicates with LinkLayerControllerResource, and LinkLayerControllerResource controls the transmission parameters related to the link establishment and quality.

Also, protocol data is communicated bi-directionally between those two components. (Vertical Communication)

- radioSetA, radioSetB:
PhysicalLayerResource [1]

A PhysicalLayerResource communicates with a peer PhysicalLayerResource that resides on another radioSet. (Horizontal Communication)

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.5.5 MediumAccessControlResource

Description

The MediumAccessControlResource, as shown in Figure 33, specializes the WaveformLayerResource stereotype. The MediumAccessControlResource stereotype provides a mechanism to realize a standard Medium Access Control component of the OSI layer model. The standard facilities of a Medium Access Control API are defined in the PIM facilities.

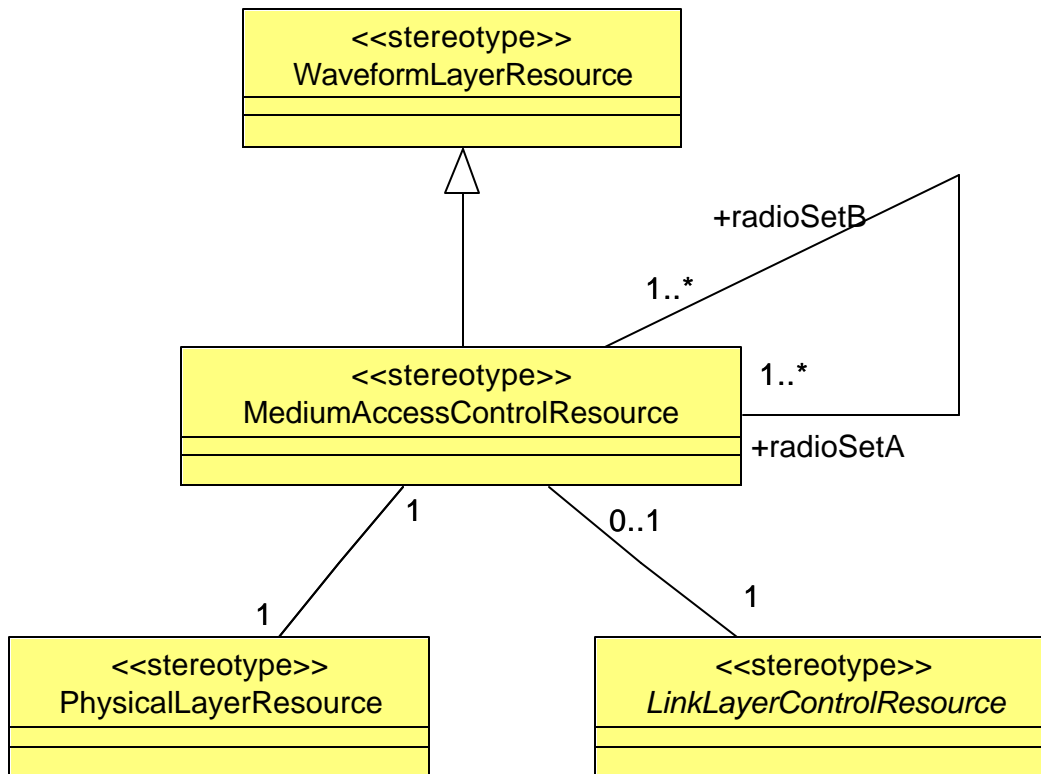


Figure 33— `MediumAccessControlResource` Definition

Attributes

No additional attributes.

Associations

- PhysicalLayerResource [1]
PhysicalLayerResource communicates with MediumAccessControlResource, and MediumAccessControlResource controls the transmission parameters related to the physical medium. Also, protocol data is communicated bi-directionally between those two components. (Vertical Communication)
- LinkLayerControllerResource [0..1]
MediumAccessControlResource communicates with LinkLayerControllerResource, and LinkLayerControllerResource controls the transmission parameters related to the link establishment and quality. MediumAccessController resource may perform some quality of service related measurements and communicate this to the Link controller. Also, protocol data is communicated bi-directionally between those two components. (Vertical Communication)
- radioSetA, radioSetB:
MediumAccessControlResource
A MediumAccessControlResource communicates with a peer MediumAccessControlResource that resides on

[1] another radioSet. (Horizontal Communication)

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.5.6 LinkLayerControlResource

Description

The LinkLayerControlResource, as shown in Figure 34, specializes the WaveformLayerResource stereotype. The LinkLayerControlResource stereotype provides a mechanism to realize a standard Link Layer Control component of the OSI layer model. The standard facilities of a Link Layer Control API are defined in the PIM facilities.

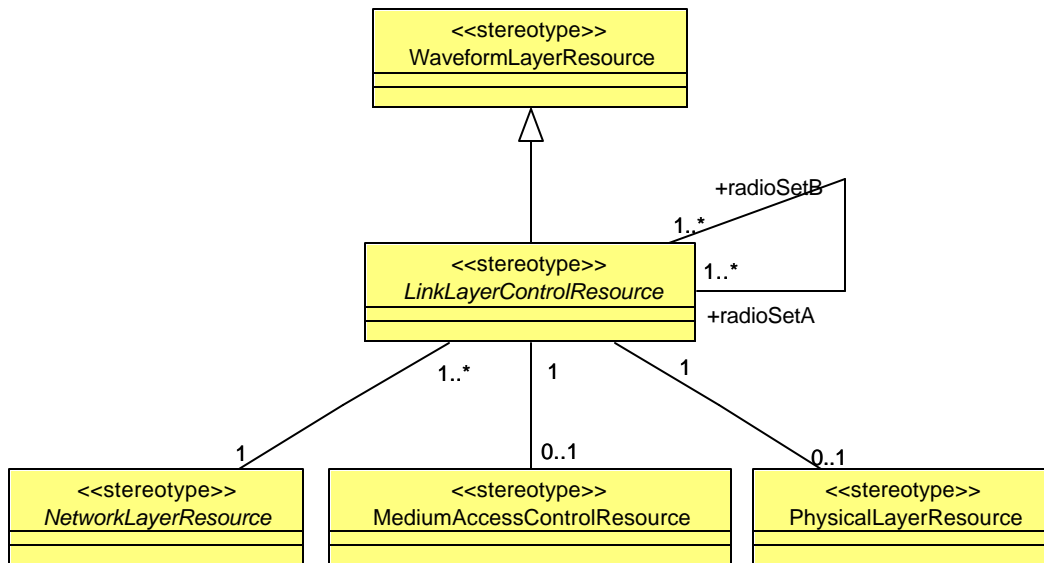


Figure 34— LinkLayerControlResource Definition

Attributes

No additional attributes.

Associations

- PhysicalLayerResource [0..1] LinkLayerControlResource may communicate directly with PhysicalLayerResource, by-passing the Medium Access layer. This communication is only in the control plane. In other scenarios where a Medium Access layer is not present in the waveform, this association encompasses both control and data plane communication between those two components. (Vertical Communication)

- MediumAccessControl [0..1] LinkLayerControlResource communicates with MediumAccessControlResource, and LinkLayerControlResource controls the transmission parameters related to the link establishment and quality. MediumAccessControlResource may perform some quality of service related measurements and communicate this to the Link controller. Also, protocol data is communicated bi-directionally between those two components. (Vertical Communication)

- radioSetA, radioSetB:
 LinkLayerControlResource [1] A LinkLayerControlResource communicates with a peer LinkLayerControlResource that resides on another radioSet. (Horizontal Communication)

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.1.5.7 NetworkLayerResource

Description

The NetworkLayerResource, as shown in Figure 34, specializes the WaveformLayerResource stereotype. The NetworkLayerResource stereotype provides a mechanism to realize a standard Network Layer component of the OSI layer model. The facilities of a network layer component is out of the scope of this submission and is not included in the PIM.

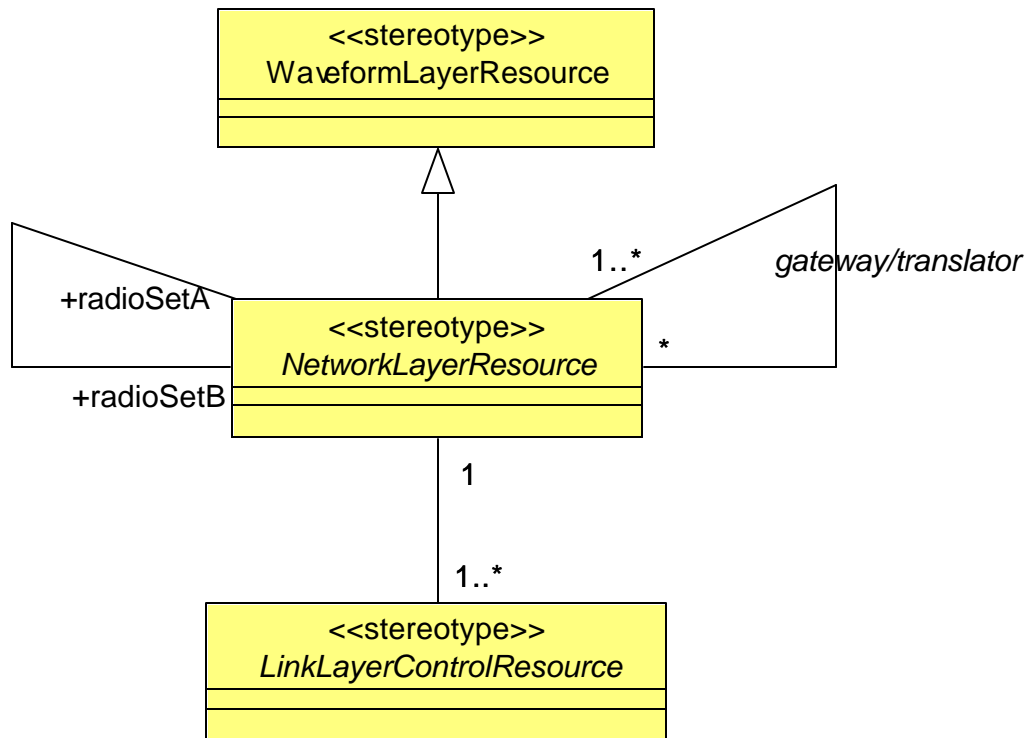


Figure 35— NetworkLayerResource Definition

Attributes

No additional attributes.

Associations

- LinkLayerControlResource [1..*] A NetworkLayerResource may communicate with one or more Link Layer components within the waveform.
- Gateway/translator:
NetworkLayerResource 1/[1..*] A radio set may be programmed to act as a waveform bridge / repeater, and in the case the network layer resource communicates with other network layer resources to provide gateway/waveform translator functionality. (Horizontal Communication)
- radioSetA, radioSetB:
NetworkLayerResource[1] A NetworkLayerResource communicates with a peer NetworkLayerResource that resides on another radioSet for data and control communication. (Horizontal Communication)

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.2 Communication Equipment

The Communication Equipment package contains device stereotypes, which are used to describe devices realizing a specific Communication Channel. The devices are chosen to represent the basic functions associated with any software radio equipment. Additional stereotypes are defined for modelling the relationships between radio devices. However, this submission does not dictate, nor restricts, the arrangement of radio devices with each other. Actual connection definitions between devices are left out to the implementer.

The purpose of the Communication Equipment package is twofold. It gives a language to describe the elements composing a specific hardware platform onto which applications execute. This description can be easily stored in XML files for automatic processing. This enables the deployment and configuration machinery to acquire knowledge about the platform capabilities. This information could be used to determine if a platform has the required capabilities to run an application before instantiating it. On the other hand, this language is also useful from a system engineering point of view. By mapping the information contained in the model to a simulation language, the operating capabilities of a radio platform can be studied off-line. This greatly eases the waveform development/porting process since the actual hardware platform is not required for determining if a specific platform can support a specific waveform.

It must be noted that this package only provides basic definition for a software radio hardware devices. Implementers can extend device definitions to match their specific needs.

Table 5 – Communication Equipment Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
CommEquipmentCommunicationPath	CommunicationPath	N/A	N/A	N/A	Indicates an association between communication equipments through which signals and messages are exchanged.
CommEquipmentConnector	Connector	N/A	N/A	A CommEquipmentConnector connects compatible hardware ports. A set of compatible ports is either one AnalogInputP	Indicates a link enabling communication between two or more instances of communication equipment ports.

Stereotype	Base Class	Parent	Tags	Constraints	Description
				ort and one AnalogOutput Port or two DigitalPort. In the case of DigitalPort	
CommEquipment	Device	N/A	equipmentInformation equipmentSize equipmentWeight powerConsumption maxOperatingTemperature minOperatingTemperature radiationCapability meanTimeBetweenFailures lastMaintenanceCheck maintenancePeriod temperatureStatus	N/A	Indicates a device that represents a radio communication device.
CryptoDevice	Device	CommEquipment	securityType deviceType processingCapability I/O_Capacity	N/A	Indicates a device that performs encryption and decryption on a set of data.
IODevice	Device	CommEquipment	dataRateCapability physicalLayer physicalLocation	N/A	Indicates a device that enables data to enter or exit from a system.
PowerSupply	Device	CommEquipment	type efficiency	N/A	Indicates a device providing electrical power to other devices.
Processor	Device	CommEquipment	maxOperatingFrequency	N/A	Indicates a programmable device that processes digital or analog data.

Stereotype	Base Class	Parent	Tags	Constraints	Description
RFDevice	Device	CommEquipment	maxPowerHandling minPowerHandling noiseFigure maxOperatingVSWR freqResponse tunedFrequency maxFrequencyResponse minFrequencyResponse maxFrequency minFrequency amplitudePhaseResponse	N/A	Indicates a non programmable device that operates on an analog signal.
ProgrammableLogicDevice	Device	Processor	deviceArchitecture logicUnitCapacity reconfigurable timeOfReconfiguration	N/A	Indicates a device that uses hardware logic to process data.
SoftwareProcessor	Device	Processor	processorArchitecture volatileMemoryCapacity nonVolatileMemoryCapacity operatingEnvironment	N/A	Indicates a device that uses software instructions to process digital data.
Antenna	Device	RFDevice	calibration radiationPattern polarization type maxRadiationPattern minRadiationPattern	N/A	Indicates a device that converts an electrical signal into an electromagnetic wave and vice versa for carrying data over an air interface.

Stereotype	Base Class	Parent	Tags	Constraints	Description
			polarizationCapability		
Amplifier	Device	RFDevice	dutyCycle gain maxGain minGain	N/A	Indicates a device that increases the energy of signals passing through it.
DigitalConverter	Device	RFDevice	sampleRate maxSampleRate minSampleRate sampleSize phaseNoise	N/A	Indicates a device that converts an analog signal into a digital signal and vice versa.
Filter	Device	RFDevice	N/A	N/A	Indicates a device that device that alters the frequency spectrum of signals passing through it.
FrequencyConverter	Device	RFDevice	nextInputFrequency nextOutputFrequency currentInputFrequency currentOutputFrequency maxInputFrequency minInputFrequency maxOutputFrequency minOutputFrequency loInputLeakagePower loOutputLeakagePower outputToInputLeakage phaseNoise loStability	N/A	Indicates a device that performs frequency translation in such a manner that the output frequencies are higher/lower in the spectrum than the input frequencies.

Stereotype	Base Class	Parent	Tags	Constraints	Description
RadiatingElement	Device	RFDevice	active radiationPattern polarization type positionInAntennaArray	N/A	Indicates the part of an antenna which actually emits and receives electromagnetic waves.
RFSwitch	Device	RFDevice	inputOutputIsolation switchSetting	N/A	Indicates a device that connects two I/O ports to each other given a specific configuration.
AnalogInputPort	Port	N/A	inputImp inputLevel maxInputLevel insertionLoss inputVSWR connector	N/A	Indicates a port used to receive an analog signal.
AnalogOutputPort	Port	N/A	maxOutputLevel outputImp outputVSWR connector	N/A	Indicates a port used to transmit an analog signal.
DigitalPort	Port	N/A	quantizationNoise connector dataFlowDirection	N/A	Indicates a port used to receive or transmit a digital signal.
CharacteristicProperty	Property	N/A	static maxLatency	N/A	Indicates a property that is not settable by a control command. The property is a characteristic of the element.
ConfigureProperty	Property	N/A	static initializable querable	N/A	Indicates a property that can be set by a control command. The property is dynamic in that it can be changed.

Stereotype	Base Class	Parent	Tags	Constraints	Description
			maxLatency stepSize		
InitializeProperty	Property	N/A	maxLatency stepSize	N/A	Indicates a property that is accessed only during initialization.

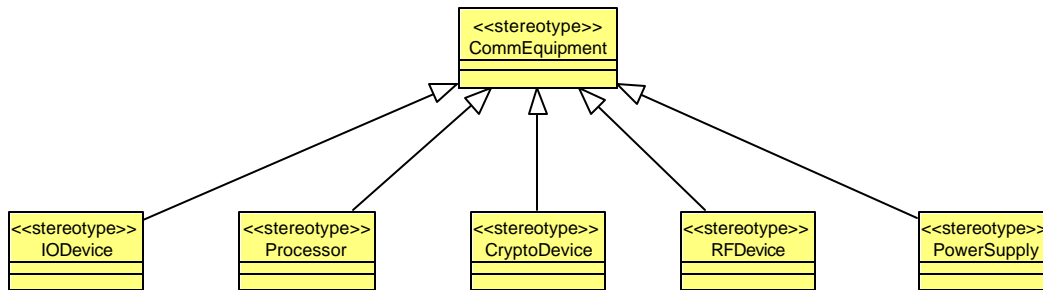


Figure 36 – Communication Equipment Overview

7.2.1 CommEquipmentCommunicationPath

Description

The CommEquipmentCommunicationPath stereotype is an extension of the UML 2.0 CommunicationPath metaclass (from UML2.0::Deployments::Nodes). A CommEquipmentCommunicationPath is an association between two communication equipments, through which communication equipments are able to exchange signals and messages.

Attributes

No additional attributes.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

To be provided.

Constraints

- The association ends of a CommEquipmentCommunicationPath are typed by CommEquipments.

Semantics

No additional semantics.

7.2.2 CommEquipementConnector

Description

The CommEquipmentConnector stereotype is an extension of the UML 2.0 Connector metaclass (from UML2.0::CompositeStructures::InternalStructures). A CommEquipmentConnector is a link enabling communication between to or more instances of communication equipments ports (see section 7.2.4).

Attributes

No additional attributes.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

To be provided.

Constraints

- The type attribute must of CommEquipmentCommunicationPath type.
- A CommEquipmentConnector connects compatible hardware ports. A set of compatible ports is either one AnalogInputPort and one AnalogOutputPort or two DigitalPorts. In the case of two DigitalPorts, one DigitalPort must be the input port and the other one must be the output port.

Semantics

No additional semantics.

7.2.3 Property

7.2.3.1 CharacteristicProperty

Description

The CharacteristicProperty stereotype is an extension of the UML 2.0 Property metaclass (from UML2.0::CompositeStructures::InternalStructures). A CharacteristicProperty is used to describe a static property of a component. This is usually associated with something that is non-configurable due to the basic nature of the component. Good examples are communication equipment's physical dimensions or the noise figure of a particular device.

Attributes

- static: Boolean = TRUE Indicates whether the property changes or not over time. Default value is true.
- maxLatency: Time [0..1] Represents the fact that some attributes, although static need time to achieve their proper operating status. The frequency of the OCXO could be a CharacteristicProperty but the maxLatency would represent the fact that the OCXO needs some time to warm up before it attains that frequency. The latency is measured from power-up.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- <<abstract>>Time Represents time duration.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.2.3.2 ConfigureProperty

Description

The ConfigureProperty stereotype is an extension of the UML 2.0 Property metaclass (from UML2.0::CompositeStructures::InternalStructures). A ConfigureProperty is used to describe a

potentially dynamic property of device. This is best illustrated by the gain of an amplifier. The gain is viewed as a ConfigureProperty because on some amplifiers, i.e. AGC, the gain can be changed during system operation.

Attributes

- static: Boolean = FALSE Indicates whether the property changes or not over time. Default value is false.
- initializable: Boolean Indicates if the property can be configured during initialization or only during normal operation. If initializable = true it means that a property can be initialized from a function call during system initialization. Otherwise, it is only initialized from the XML files.
- querable: Boolean If querable = true it means that the property can be read in addition to being written (read-write property). Otherwise it can only be written (write-only property).
- maxLatency: Time [0..1] Represents the fact that some properties need time to achieve their proper operating status. A good example is the gain of an amplifier. The maxLatency would represent the fact that the amplifier needs some time before it attains its operating gain. The latency is measured from power-up.
- stepSize: Single [0..1] Represent the fact that some properties have discrete increments. A typical example is a tuneable duplexer which uses a stepper motor to adjust the tuned frequency.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- <<abstract>>Time Represents time duration.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.2.4 Port

Communication equipments communicate with each other through hardware ports. Three extensions to the UML Port metaclass are defined: AnalogInputPort, AnalogOutputPort and DigitalPort. By using the port stereotype, the implementer can customize, or extend a device with additional ports for control, status or any other information. A good example is an amplifier. Typically, when an amplifier has two ports, it is a fixed gain amplifier, when it has three ports it can be an AGC.

A bidirectional analog port would be constructed out of both AnalogInputPort and AnalogOutputPort. A bidirectional digital port would be constructed out of two instances of a DigitalPort.

7.2.4.1 AnalogInputPort

Description

The AnalogInputPort stereotype is an extension of the UML 2.0 Port metaclass (from UML2.0::CompositeStructures::Ports). The AnalogInputPort is designed to hold the attributes of an analog input port.

Attributes

- | | |
|---|--|
| • inputImp: Impedance | Impedance of the port. |
| • inputLevel: Power | Power level currently at the input. |
| • maxInputLevel: Power | Maximum input power that the port can sustain. |
| • insertionLoss: Decibel | Loss occurring when a device is inserted in a transmission line. This value is the ratio between the signal powers on that end of the line after and before insertion of the device. |
| • inputVSWR: VSWR [0..1] | Voltage standing wave ratio of the port. |
| • connector: ConnectorFormFactor [0..1] | Connector form factor of the port. Used for physical connection of the port to an external entity. Specific connector form factors are not defined in the Profile. They are defined at the PIM level. For industry standard form factors they could be defined by an enumeration. For proprietary form factors, a more elaborate class could be created defining the dimensions, shielding, etc. The connector attribute only defines the connector form factor, not the connector entity. All ports would have the same |

connector form factor even if they are physically located in the same connector.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- <<abstract>>Impedance
Opposition that a device offers to an electric current. Impedance is composed of two components, resistance and reactance.
- <<abstract>>Power
Rate at which electrical energy is transformed to another type of energy.
- <<abstract>>Decibel
Ratio between two voltages, currents, or signal power levels.
- <<abstract>>VSWR
Ratio of the device operating impedance to a desired characteristic impedance (usually 50 ohm characteristic impedance reference).
- <<abstract>>ConnectorFormFactor
Physical connector form factor.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.2.4.2 AnalogOutputPort

Description

The AnalogOutputPort stereotype is an extension of the UML 2.0 Port metaclass (from UML2.0::CompositeStructures::Ports). The AnalogOutputPort is used to hold the attributes of an analog output port.

Attributes

- maxOutputLevel: Power
Maximum output power that the port can provide.
- outputImp: Impedance
Impedance of the port.
- outputVSWR: VSWR [0..1]
Voltage standing wave ratio of the port.
- connector: ConnectorFormFactor [0..1]
Connector form factor of the port. Used for physical connection of the port to an external entity. Specific connector form factors are not defined in the Profile. They are defined at the PIM level. For industry standard form factors they could be defined by an enumeration. For proprietary form factors, a more elaborate class could be created defining the dimensions, shielding, etc. The connector attribute only defines the connector form factor, not the connector entity. All ports would have the same connector form factor even if they are physically located in the same connector.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- <<abstract>>Power
Rate at which electrical energy is transformed to another type of energy.
- <<abstract>>Impedance
Opposition that a device offers to an electric current. Impedance is composed of two components, resistance and reactance.
- <<abstract>>VSWR
Ratio of the device operating impedance to a desired characteristic impedance (usually 50 ohm characteristic impedance reference).
- <<abstract>>ConnectorFormFactor
Physical connector form factor.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.2.4.3 DigitalPort

Description

The DigitalPort stereotype is an extension of the UML 2.0 Port metaclass (from UML2.0::CompositeStructures::Ports). The DigitalPort is used to hold the attributes of a digital port.

Attributes

- quantizationNoise: QuantizationNoiseDensity
Noise resulting from the approximation error in the quantization process. Quantization noise is related to the specific quantization process and the characteristics of the quantized signal.
- connector: ConnectorFormFactor [0..1]
Connector form factor of the port. Used for physical connection of the port to an external entity. Specific connector form factors are not defined in the Profile. They are defined at the PIM level. For industry standard form factors they could be defined by an enumeration. For proprietary form factors, a more elaborate class could be created defining the dimensions, shielding, etc. The connector attribute only defines the connector form factor, not the connector entity. All ports would have the same connector form factor even if they are physically located in the same connector.
- dataFlowDirection: Direction
Indicates whether the port is an input port or an output port.

Associations

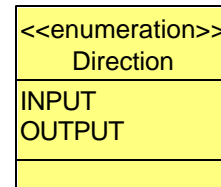
No additional associations.

Operations

No additional operations.

Types and Exceptions

- <<abstract>>QuantizationNoiseDensity Distribution function estimating the quantization noise resulting from using a specific quantization process.
- <<abstract>>ConnectorFormFactor Abstract type representing power value. This type must be bound to a concrete type before using it. The binding type would specify units (DB9, DB25, RJ-45...).
- <<enumeration>>Direction Direction of data flow either INPUT or OUTPUT.



Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.2.5 CommEquipment

Description

The CommEquipment stereotype is an extension of the UML 2.0 Device metaclass (from UML2.0::Deployment::Nodes). CommEquipment is the overall base class for describing that collection of devices, which are used to realize the Communication Channel. Note that other device classes / functional combinations are possible. The classes shown are the basic device types that typically realize the communication interfaces. Likewise, implementations may add relationships between device classes beyond those in the basic model.

Attributes

- <<characteristicproperty>>equipmentInformation: PlugAndPlayInformation Descriptive information about the physical device. This information could be used in a plug and play hardware environment.

- <<characteristicproperty>>equipmentSize: Size The size of the physical of a component.
- <<characteristicproperty>>equipmentWeight: Weight The weight of the physical component.
- <<characteristicproperty>>powerConsumption: Power The power consumed by the component.
- <<characteristicproperty>>maxOperatingTemperature: Temperature Maximum sustainable operating temperature of the physical component.
- <<characteristicproperty>>minOperatingTemperature: Temperature Minimum sustainable operating temperature of the physical component.
- <<characteristicproperty>>radiationCapability: Radiation [0..1] Sustainable radiation level of the physical component. This attribute could be used for radiation hardened components.
- <<characteristicproperty>>meanTimeBetweenFailures: Time [0..1] The length of time a user may reasonably expect a component to work properly before an incapacitating fault occurs.
- <<configureproperty>>lastMaintenanceCheck: Date [0..1] Date at which the last maintenance check was performed. Could be used for components requiring manual calibration.
- <<characteristicproperty>>maintenancePeriod: Time [0..1] Time interval between required maintenance check. Could be used for components requiring manual calibration.
- <<characteristicproperty>>temperatureStatus: Temperature [0..1] Indicates the internal temperature of the device.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- PlugAndPlayInformation Descriptive information about the device.

PlugAndPlayInformation
<<characteristicproperty>> manufacturerName : String <<characteristicproperty>> modelName : String <<characteristicproperty>> modelNumber : String <<characteristicproperty>> modelDescription : String <<characteristicproperty>> serialNumber : String <<characteristicproperty>> majorRevision : String <<characteristicproperty>> minorRevision : String

- <<abstract>>Size Represents the physical size of an object in a given unit.
- <<abstract>>Weight Represents the physical weight of an object in a given unit.
- <<abstract>>Power Rate at which electrical energy is transformed to another type of energy.
- <<abstract>>Temperature Temperature in a given unit (Celsius, Kelvin...).
- <<abstract>>Radiation Information about a specific radiation environment.
- <<abstract>>Time Represents time duration.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.2.5.1 CryptoDevice

Description

A CryptoDevice is a dedicated device that performs encryption and decryption services for Communication Channels. Typically, these devices are used in military communication systems; there are also commercial devices that perform these functions.

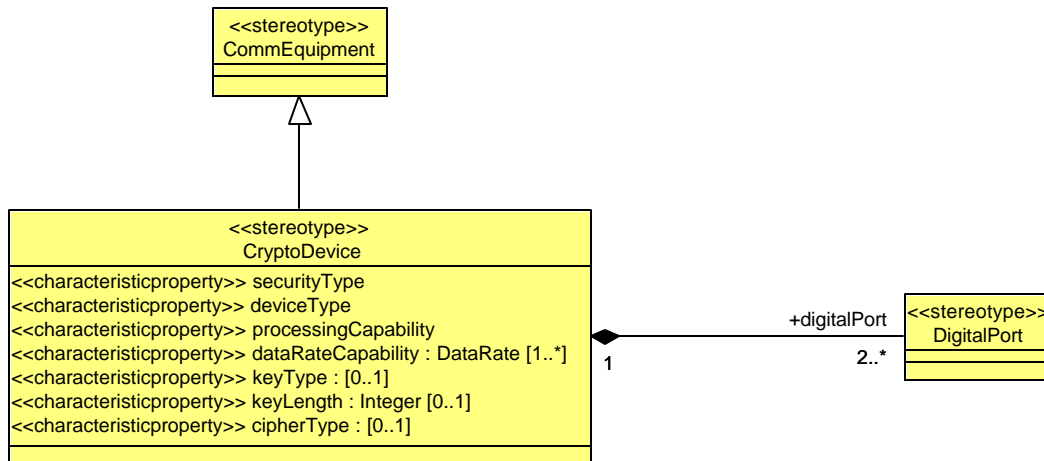


Figure 37 - CryptoDevice Definition

Attributes

- <<characteristicproperty>>securityType:
- <<characteristicproperty>>deviceType:
- <<characteristicproperty>>processingCapability:
- <<characteristicproperty>>dataRateCapability:
DataRate[1..*]
- <<characteristicproperty>>keyType:
- <<characteristicproperty>>keyLength: Integer
[0..1]
- <<characteristicproperty>>cipherType:

Associations

- digitalPort: DigitalPort [2..*]

Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports)².

Operations

No additional operations.

² UML 2.0 Superstructure Specification p.154

Types and Exceptions

No additional types and exceptions.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.2.5.2 IODevice

Description

An IODevice provides physical connections for the Communication Channel to users. This is typically performed via a baseband, or very low RF, communication system. Typical examples are Manchester encoding, Non-return to zero etc, analog voice channel.

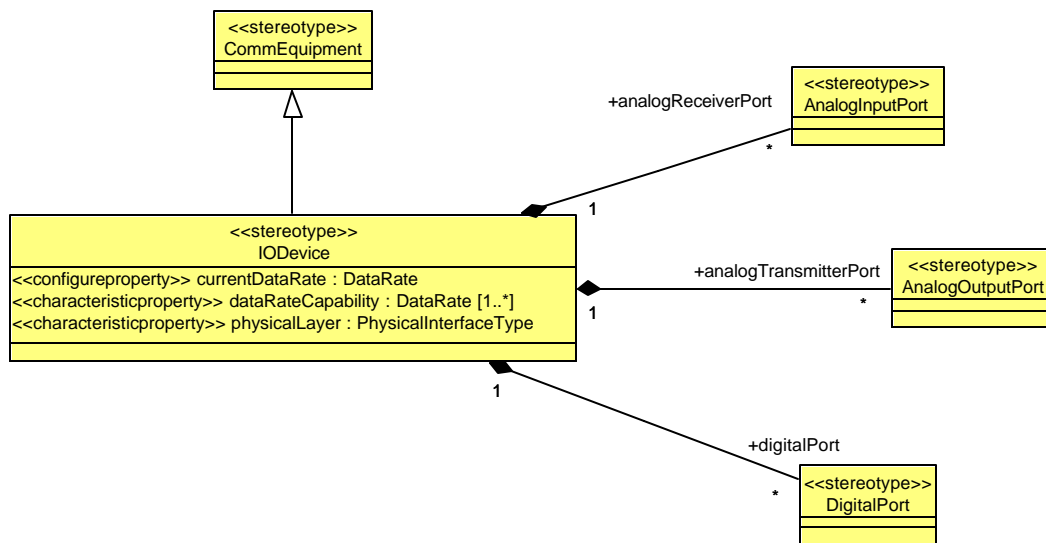


Figure 38 - I/O Device Definition

Attributes

- <<configureproperty>>currentDataRate: DataRate The current data rate of the device.
- <<characteristicproperty>>dataRateCapability: DataRate [1..*] Data rates that the device supports.
- <<characteristicproperty>>physicalLayer: PhysicalInterfaceType Identifies the specific physical layer protocol used by the device.

Associations

- analogReceiverPort:
AnalogInputPort [*] Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports)³.
- analogTransmitterPort:
AnalogOutputPort [*] Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports)⁴.
- digitalPort: DigitalPort [*] Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports)⁵.

Operations

No additional operations.

Types and Exceptions

- <<abstract>>DataRate The rate at which data is processed by the device.
- <<abstract>>PhysicalInterfaceType Physical layer protocol used by the device.

Mapping

To be provided.

Constraints

- There has to be at least two ports.

Semantics

No additional semantics.

7.2.5.3 PowerSupply

Description

A PowerSupply provides electrical power to CommEquipment components . It is therefore associated with all others CommEquipment components. It must be noted that this submission does not address the issue of power management. It is expected that power management is the responsibility of a higher level application.

³ UML 2.0 Superstructure Specification p.154

⁴ UML 2.0 Superstructure Specification p.154

⁵ UML 2.0 Superstructure Specification p.154

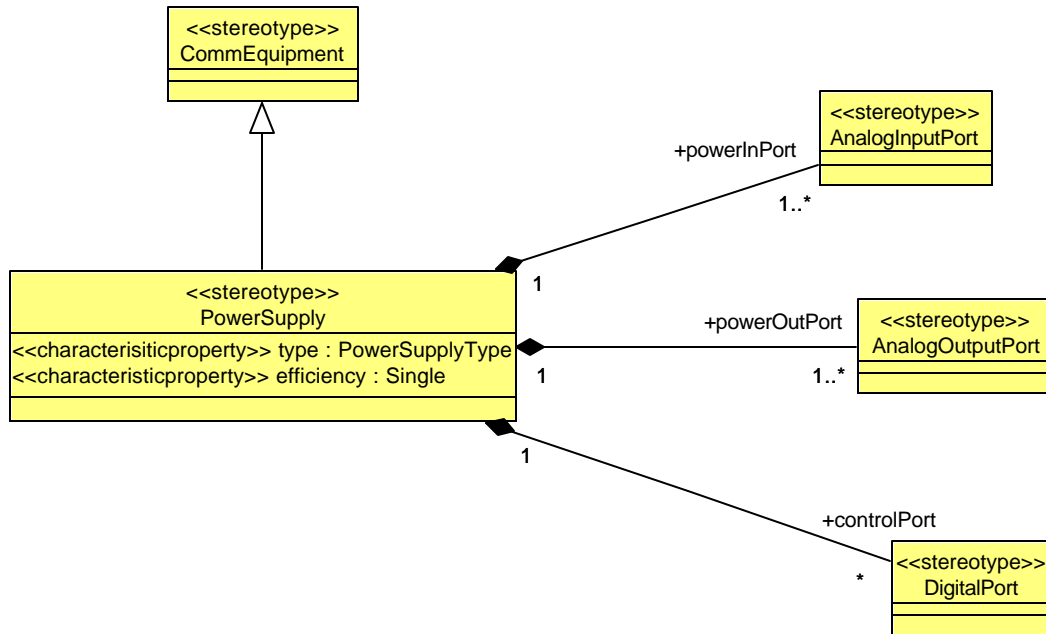


Figure 39 - PowerSupply Definition

Attributes

- **<<characteristicproperty>>type: PowerSupplyType** Indicates if the device converts AC power to DC power or DC power to DC power.
- **<<characteristicproperty>>efficiency: Single** Ratio of signal power output to total power input.

Associations

- **powerInPort: AnalogInputPort [1..*]** Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports)⁶.
- **powerOutPort: AnalogOutputPort [1..*]** Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports)⁷.
- **ControlPort: DigitalPort [*]** Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports)⁸.

⁶ UML 2.0 Superstructure Specification p.154

⁷ UML 2.0 Superstructure Specification p.154

⁸ UML 2.0 Superstructure Specification p.154

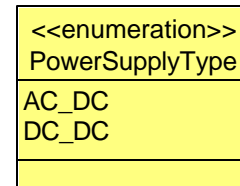
Operations

No additional operations.

Types and Exceptions

- PowerSupplyType

If a device is of AC_DC type, it converts AC power to DC power. If the device is of DC_DC type, it converts DC power to DC power.



Mapping

To be provided.

Constraints

Efficiency must always be less than 1 since it is expressed as a percentage.

Semantics

No additional semantics.

7.2.5.4 Processor

Description

A processor is a device that provides computational functions along with supporting functions such as memory and I/O. Processor types include general purpose processors (such as PowerPCs, x86s, etc.), digital signal processors, field programmable gate arrays, application-specific integrated circuits configured for computational functions, and others.

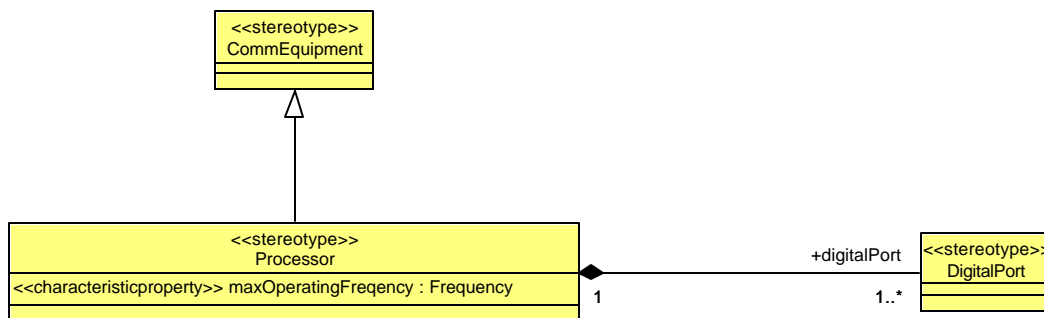


Figure 40 - Processor Definition

Attributes

- <<characteristicproperty>>maxOperatingFrequency: Maximum frequency at which the device is able to operate.
Frequency

Associations

- digitalPort: DigitalPort [1..*] Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports)⁹.

Operations

No additional operations.

Types and Exceptions

- <<abstract>>Frequency Number of complete cycles per second of a signal.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.2.5.4.1 ProgrammableLogicDevice

Description

The ProgrammableLogicDevice is a device which process digital data using hardware logic. It is a specialization of the Processor class. Examples of programmable logic device (PLD) are FPGA and CPLD. This stereotype contains attributes specific to this type of device. Basic logic blocks are use to dynamically instantiate a particular function during device initialization.

⁹ UML 2.0 Superstructure Specification p.154

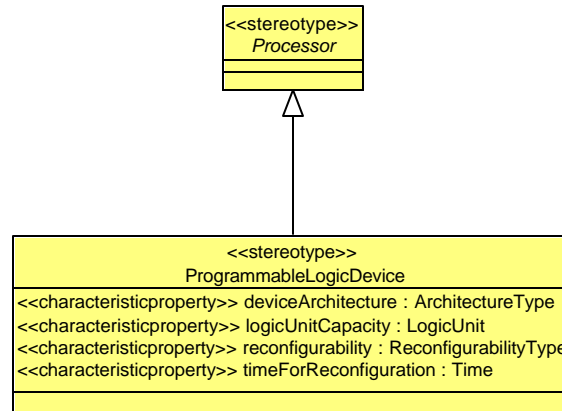


Figure 41 - ProgrammableLogicDevice Definition

Attributes

- | | |
|---|---|
| • <<characteristicproperty>>deviceArchitecture:
ArchitectureType | The specific type of programmable device. |
| • <<characteristicproperty>>logicUnitCapacity:
LogicUnit | The total amount of logic units available inside the device. |
| • <<characteristicproperty>>reconfigurability:
ReconfigurabilityType | Indicates whether the device is statically or dynamically reconfigurable. |
| • <<characteristicproperty>>timeForReconfiguration:
Time | Duration of a reconfiguration operation. |

Associations

No additional associations.

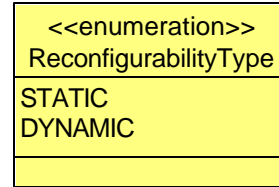
Operations

No additional operations.

Types and Exceptions

- | | |
|--------------------------------|--|
| • <<abstract>>ArchitectureType | The architecture of the device (examples could be FPGA, CPLD...). |
| • <<abstract>>LogicUnit | Description a basic logic blocks available inside the device. |
| • ReconfigurabilityType | STATIC reconfigurability means that the device is configured at the start of execution and remains unchanged for the duration of the application. DYNAMIC reconfigurability means the ability for partial reconfiguration of certain logic blocks while others are |

performing computations.



- <<abstract>>Time Represents time duration.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.2.5.4.2 SoftwareProcessor

Description

This stereotype represents a processor which executes software instructions in order to execute specific algorithms. GPP and DSP processor are good examples of device of this type.

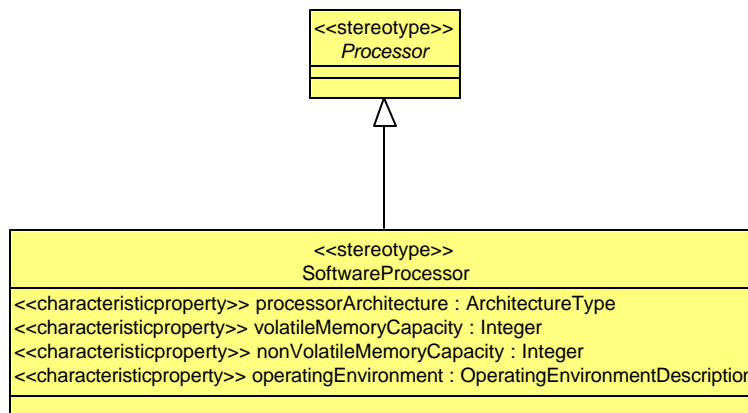


Figure 42 - SoftwareProcessor Definition

Attributes

- <<characteristicproperty>>processorArchitecture: The specific type of software

ArchitectureType	processor.
<ul style="list-style-type: none"> • <<characteristicproperty>>volatileMemoryCapacity: Integer 	Total number of bytes of volatile memory available to the processor.
<ul style="list-style-type: none"> • <<characteristicproperty>>nonVolatileMemoryCapacity: Integer 	Total number of bytes of persistent memory available to the processor.
<ul style="list-style-type: none"> • <<characteristicproperty>>operatingEnvironment: OperatingEnvironmentDescription 	Actual operating environment that the device is using.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- <<abstract>>ArchitectureType The architecture of the device (examples could be PPC, x86...).
- <<abstract>>OperatingEnvironmentDescription Description of the environment which the device is using (examples could be OS, middleware...)

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.2.5.5 RFDevice

Description

RFDevice is the base class for all devices that operate on a signal in an analog format. This is highlighted by the fact that although the number of ports of the device is variable at least one analog port must be present. This implies that a pure digital RF device is not admissible as a member of this class. Such a pure RF device would be part of the processor class.

Since RF frequency is by definition the frequency at which the signal is to be transmitted, this by no means implies a high frequency. The RFDevice class includes a “tunedFrequency” parameter which can have 0 Hz as a valid entry. This generalization permits the elements to be used in the characterization of the IF and base band components of the RF chain. It also permits the specification of the physical components present in the IO chain.

Elements inheriting the RFDevice class can be viewed as building blocks. More complex elements like receivers and excitors can be assembled by putting together specific combination of the various elements of the RFDevice class.

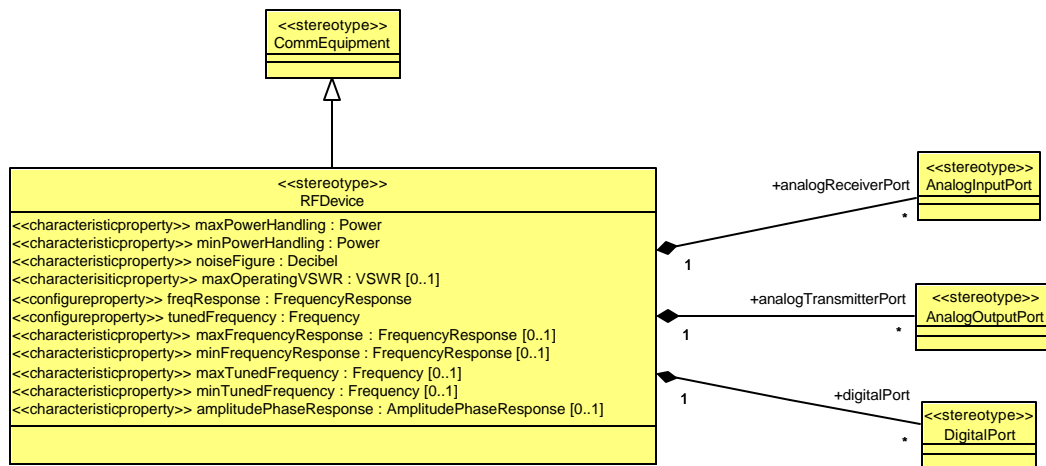


Figure 43 - RFDevice Definition

Attributes

- **<<characteristicproperty>>maxPowerHandling: Power** The maximum RF power the device can sustain.
- **<<characteristicproperty>>minPowerHandling: Power** The minimum RF power the device must be supplied in order to work.
- **<<characteristicproperty>>noiseFigure: Decibel** The maximum RF power the device can sustain.
- **<<characteristicproperty>>maxOperatingVSWR: VSWR** Ratio of the device operating impedance to a desired characteristic impedance (usually 50 ohm characteristic impedance reference)
- **<<configureproperty>>freqResponse: FrequencyResponse** The frequency response of the device.
- **<<configureproperty>>tunedFrequency: Frequency** The center frequency of the frequency response.
- **<<characteristicproperty>>maxFrequencyResponse: FrequencyResponse** The maximum frequency response the device is able to achieve. The maximum amplitude and/or phase at a given frequency.
- **<<characteristicproperty>>minFrequencyResponse: FrequencyResponse** The minimum frequency response the device is able to achieve. The minimum amplitude and/or phase at a given frequency.
- **<<characteristicproperty>>maxFrequency: Frequency** The maximum frequency of the bandwidth for which the device performance is rated.

- <<characteristicproperty>>minFrequency: Frequency The minimum frequency of the bandwidth for which the device performance is rated.
- <<characteristicproperty>>amplitudePhaseResponse: AmplitudePhaseResponse [0..1] The amplitude phase response for the device. The amplitude phase response contains two components. The first component is a representation of the output power versus the input power. The second component is a representation of the output phase versus input power. The purpose of the amplitude phase response is to describe any active element which cannot be described by an ideal relationship (non linearities) e.g.: Power Amplifier.

Associations

- analogReceiverPort: AnalogInputPort [*] Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports)¹⁰.
- analogTransmitterPort: AnalogOutputPort [*] Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports)¹¹.
- digitalPort: DigitalPort [*] Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports)¹².

Operations

No additional operations.

Types and Exceptions

- <<abstract>>Power Rate at which electrical energy is transformed to another type of energy.
- <<abstract>>Decibel Ratio between two voltages, currents, or signal power levels.
- <<abstract>>VSWR Ratio of the device operating impedance to a desired characteristic impedance (usually 50 ohm characteristic impedance reference).
- <<abstract>>FrequencyResponse A frequency response is the relation between signal amplitude or gain with frequency. A frequency

¹⁰ UML 2.0 Superstructure Specification p.154

¹¹ UML 2.0 Superstructure Specification p.154

¹² UML 2.0 Superstructure Specification p.154

response with only one point represents a single-sided 3 dB bandwidth. A frequency response with more than one point is an arbitrary frequency response with an arbitrary resolution. A given frequency response has 0 dB gain and is centered at 0 Hz (it does not have to be symmetric).

- <<abstract>>Frequency Number of complete cycles per second of a signal.
- <<abstract>>AmplitudePhaseResponse An amplitude phase response with one point represents a, 1 dB compression point. In an amplitude phase response with two points represents, the first point represents the 1 dB compression point and the second point represents the IP3 (third order intercept) point. An amplitude phase response with more than two points represents entire AM-to-AM and AM-to-PM curves. Typically, curves represent instantaneous power.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.2.5.5.1 Antenna

Description

The Antenna class represents the RF radiating elements necessary for transmission / reception of radio energy through the ether. The Antenna class consists of both a simple passive radiating element as well as an antenna array with possibly some dedicated intelligence.

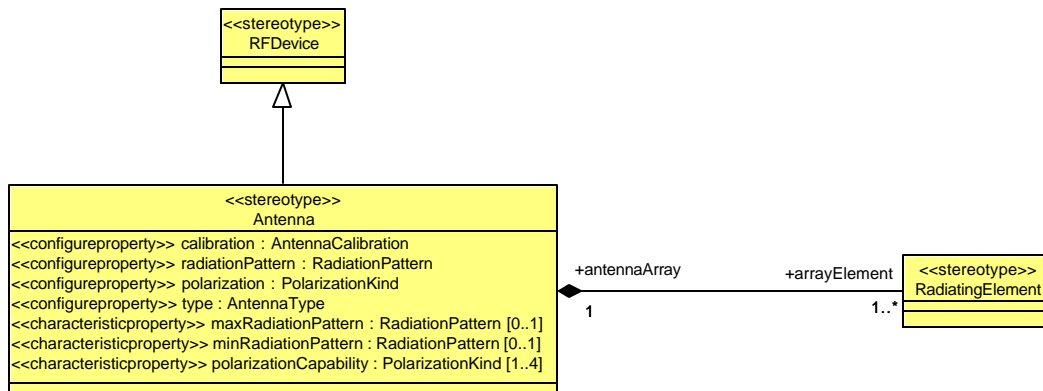


Figure 44 - Antenna Definition

Attributes

- <<configureproperty>>calibration: AntennaCalibration Calibration information of the antenna.
- <<configureproperty>>radiationPattern: RadiationPattern The current radiation pattern that is configured in the device.
- <<configureproperty>>polarization: PolarizationKind[0..1] The configurable orientation of the RF energy radiated from the antenna.
- <<configureproperty>>type: AntennaType Physical type of the antenna.
- <<characteristicproperty>>maxRadiationPattern: RadiationPattern The maximum radiation pattern that the device is able to achieve.
- <<characteristicproperty>>minRadiationPattern: RadiationPattern The minimum radiation pattern that the device is able to achieve.
- <<characteristicproperty>>polarizationCapability: PolarizationKind[1..4] The orientation capability of the RF energy radiated from the antenna.

Associations

- arrayElement: RadiatingElement [1..*] The individual radiating element objects of the antenna.

Operations

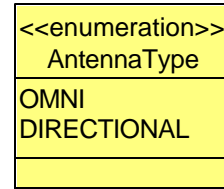
No additional operations.

Types and Exceptions

- <<abstract>>AntennaCalibration The calibration of the whole antenna.
- <<abstract>>RadiationPattern Field intensity variation of an antenna as an angular function with respect to a 3D coordinate system.
- <<enumeration>>PolarizationKind The orientation of the RF energy radiated from the device.

<<enumeration>> PolarizationKind
VERTICAL
HORIZONTAL
RIGHT_CIRCULAR_POLARIZE
LEFT_CIRCULAR_POLARIZE

- <<enumeration>>AntennaType



Mapping

To be provided.

Constraints

- self.analogReceiverPort->size() >= 1 or self.analogTransmitterPort->size() >= 1 or There has to be at least one AnalogInputPort or one AnalogOutputPort.

Semantics

No additional semantics.

7.2.5.5.2 Amplifier

Description

Amplifiers provide gain. Baseband, RF, Power and Low Noise Amplifiers are generalizations of Amplifier and are differentiated by the values of their attributes.

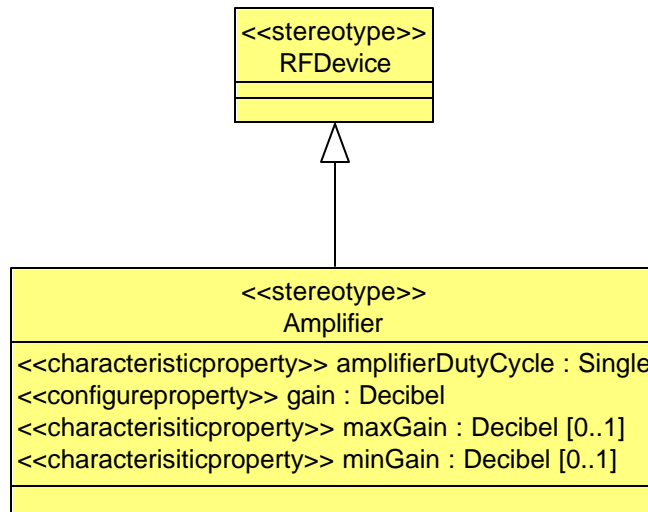


Figure 45 - Amplifier Definition

Attributes

- <<characteristicproperty>>amplifierDutyCycle: The maximum continuous duty cycle the

Single

device can operate at.

- <<configureproperty>>gain: Decibel The current power amplification factor applied to the input signal by the device.
- <<characteristicproperty>>maxGain: Decibel The maximum power amplification factor a device is able to apply to a signal.
- <<characteristicproperty>>minGain: Decibel The minimum power amplification factor a device is able to apply to a signal.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- <<abstract>>Decibel Ratio between two voltages, currents, or signal power levels.

Mapping

To be provided.

Constraints

- (self.analogReceiverPort->size() >= 1 and self.analogTransmitterPort->size() >= 1) or self.digitalPort->size() >= 2 There has to be at least (one AnalogInputPort and one AnalogOutputPort) or two DigitalPorts.

Semantics

No additional semantics.

7.2.5.5.3 DigitalConverter

Description

The Digital Converter is a device that performs analog-to-digital and / or digital-to-analog conversion of the transmit and / or receive signal.

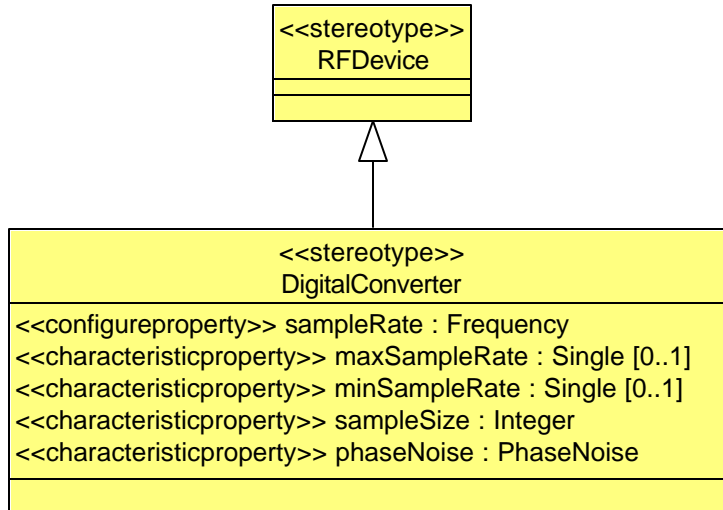


Figure 46 - DigitalConverter Definition

Attributes

- <<configureproperty>>sampleRate: Frequency The current number of samples per second converted by the device.
- <<characteristicproperty>>maxSampleRate: Single The maximum sample rate the device is able to achieve.
- <<characteristicproperty>>minSampleRate: Single The minimum sample rate the device is able to achieve.
- <<characteristicproperty>>sampleSize: Integer The size in bits of a sample.
- <<characteristicproperty>>phaseNoise: PhaseNoiseType The phase noise that the device introduces in the signal.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- <<abstract>>Frequency Number of complete cycles per second of a signal.
- <<abstract>>PhaseNoiseType Random and short duration fluctuations in the phase of a signal.

Mapping

To be provided.

Constraints

- (self.analogReceiverPort->size() >= 1 or self.analogTransmitterPort->size() >= 1) and self.digitalPort->size() >= 1 There has to be at least (one AnalogInputPort or one AnalogOutputPort) and one DigitalPort.

Semantics

No additional semantics.

7.2.5.5.4 Filter

Description

Filters provide frequency protection to the Communication Channel, limiting the signals that might cause interference. Filters also provide signal shaping in both amplitude and phase to the Communications Channel.

In a Communication Channel, filters can often be called by other names depending on their location and / or functionality (e.g. duplexers, interference cancellers, equalizers...). The filter class regroups all those devices regardless of their implementation, location and function. The filter class recognizes that the functionality of all these devices is to attenuate/enhance some frequency components of the signal. Furthermore, since the frequency response is a configure property; the filter class can represent both fixed and adaptive filters.

Due to the large number of “filters” in a Communication Channel, the filter device can be found between every other type of device. It is certainly frequent to have filters before ADC and after DAC, before and / or after amplifiers, frequency converters, antennas/radiating elements, and switches.

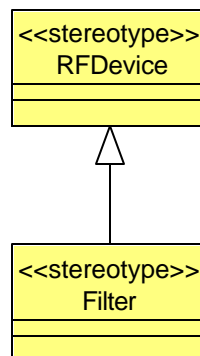


Figure 47 - Filter Definition

Attributes

No additional attributes.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

To be provided.

Constraints

- (self.analogReceiverPort->size() >= 1 and self.analogTransmitterPort->size() >= 1) or self.digitalPort->size() >= 2 There has to be at least one AnalogInputPort and one AnalogOutputPort.

Semantics

No additional semantics.

7.2.5.5.5 FrequencyConverter

Description

The FrequencyConverter translates signals between one center frequency to another center frequency. When the output center frequency is higher than the input center frequency, the device is called an up converter otherwise it is called a down converter. Like filters frequency converters can take many names or forms (e.g. Direct RF, frequency hopping, harmonic, ...).

The main particularity of the FrequencyConverter is that the local oscillator is assumed to be part of the device. Therefore the FrequencyConverter can be a two ports device. This choice was made to support elegantly harmonic converters and other devices which do not require an external local oscillator.

The FrequencyConverter device does not implement by itself the entire extender or receiver concept. It is however a key building block in the definition of those higher level of abstraction objects.

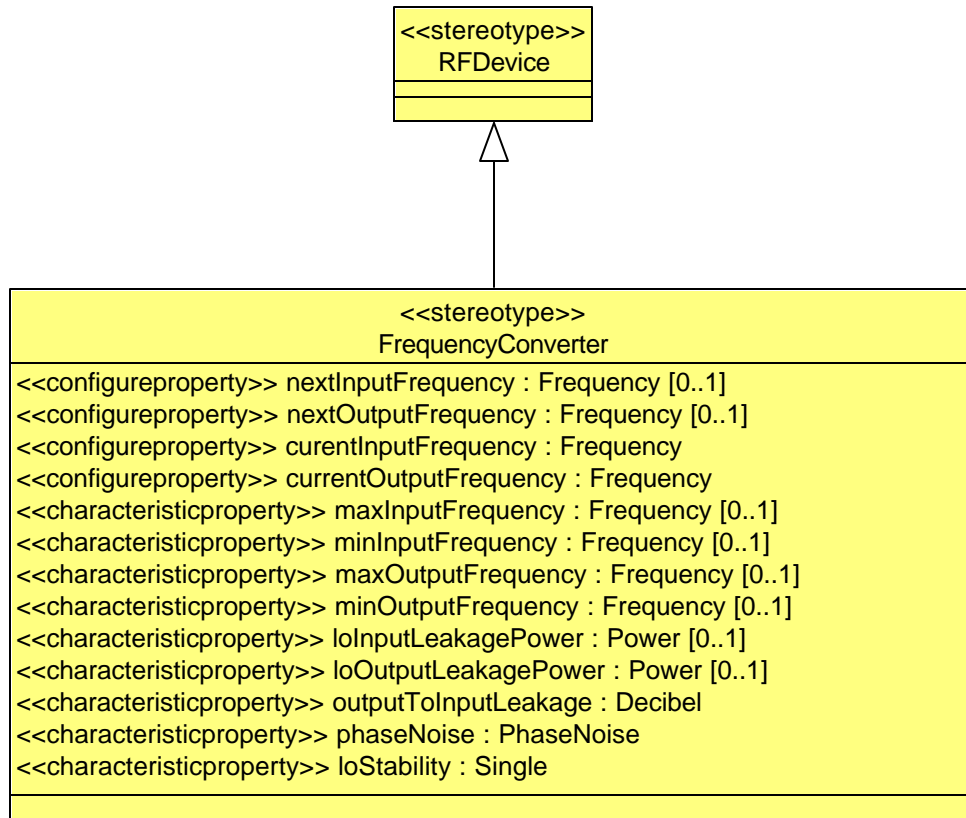


Figure 48 - FrequencyConverter Definition

Attributes

- | | |
|--|---|
| <ul style="list-style-type: none"> • <<configureproperty>>nextInputFrequency: Frequency [0..1] | <p>The input frequency that the device will select after the next triggering event. This attribute is used for instantaneous frequency changes. Typically in the context of frequency hopping and frequency scanning algorithms.</p> |
| <ul style="list-style-type: none"> • <<configureproperty>>nextOutputFrequency: Frequency [0..1] | <p>The output frequency that the device will select after the next triggering event. This attribute is used for instantaneous frequency changes. Typically in the context of frequency hopping and frequency scanning algorithms.</p> |
| <ul style="list-style-type: none"> • <<configureproperty>>currentInputFrequency: Frequency | <p>The frequency of the signal currently at the input of the device.</p> |
| <ul style="list-style-type: none"> • <<configureproperty>>currentOutputFrequency: Frequency | <p>The frequency of the signal currently at the output of the device,</p> |
| <ul style="list-style-type: none"> • <<characteristicproperty>>maxInputFrequency: | <p>The maximum input signal frequency</p> |

Frequency	the device is able to handle.
<ul style="list-style-type: none"> • <<characteristicproperty>>minInputFrequency: Frequency 	The minimum input signal frequency the device is able to handle.
<ul style="list-style-type: none"> • <<characteristicproperty>>maxOutputFrequency: Frequency 	The maximum output signal frequency the device is able to handle.
<ul style="list-style-type: none"> • <<characteristicproperty>>minOutputFrequency: Frequency 	The minimum output signal frequency the device is able to handle.
<ul style="list-style-type: none"> • <<characteristicproperty>>loInputLeakagePower: Power [0..1] 	Local oscillator input leakage power.
<ul style="list-style-type: none"> • <<characteristicproperty>>loOutputLeakagePower: Power [0..1] 	Local oscillator output leakage power.
<ul style="list-style-type: none"> • <<characteristicproperty>>outputToInputLeakage: Decibel 	Amount of the output frequency which is found at the input.
<ul style="list-style-type: none"> • <<characteristicproperty>>phaseNoise: PhaseNoiseType 	The phase noise that the device introduce in the signal.
<ul style="list-style-type: none"> • <<characteristicproperty>>loStability: Single 	Local oscillator stability expressed in PPM.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

<ul style="list-style-type: none"> • <<abstract>>Frequency 	Number of complete cycles per second of a signal.
<ul style="list-style-type: none"> • <<abstract>>Power 	Rate at which electrical energy is transformed to another type of energy.
<ul style="list-style-type: none"> • <<abstract>>Decibel 	Ratio between two voltages, currents, or signal power levels.
<ul style="list-style-type: none"> • <<abstract>>PhaseNoiseType 	Random and short duration fluctuations in the phase of a signal.

Mapping

To be provided.

Constraints

•

Semantics

No additional semantics.

7.2.5.5.6 RadiatingElement

Description

A radiating element is a passive element which translates electrical energy into an electromagnetic wave and vice-versa. The radiating elements acts as the transducer between the electrical world and the air interface. Typical examples can be cones, patches, dipoles, dishes, etc.

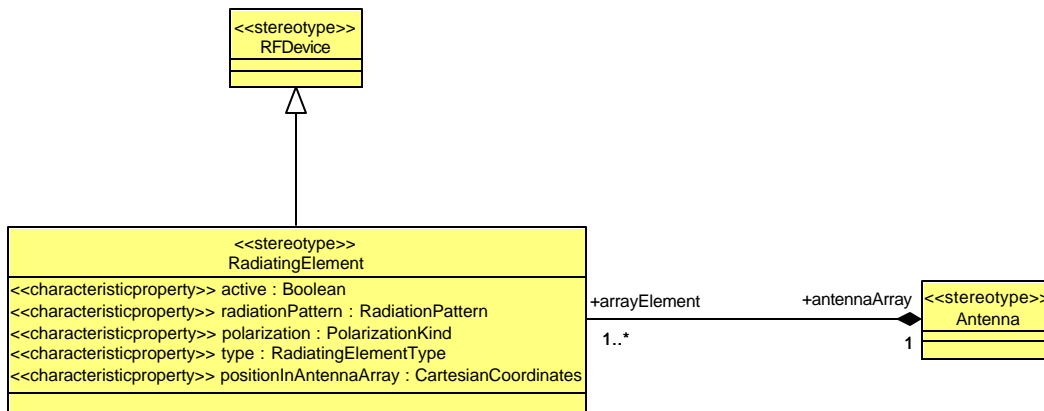


Figure 49 - RadiatingElement Definition

Attributes

- | | |
|--|--|
| • <<characteristicproperty>>active: Boolean | Indicates if the radiating element is currently active. |
| • <<characteristicproperty>>radiationPattern: RadiationPattern | The current radiation pattern for this single radiating element. |
| • <<characteristicproperty>>polarization: PolarizationKind [0..1] | The orientation of the RF energy radiated for this single radiating element. |
| • <<characteristicproperty>>type: RadiatingElementType | Type of radiating element. |
| • <<characteristicproperty>>positionInAntennaArray[0..1]: CartesianCoordinates | If the radiating element is part of an antenna array, this attribute indicates its 3D position in array with respect to the geometric center of the array. |

Associations

- antennaArray: Antenna

The antenna object which the radiating element is part of.

Operations

No additional operations.

Types and Exceptions

- <<abstract>>RadiationPattern

Field intensity variation of an antenna as an angular function with respect to a 3D coordinate system.

- <<enumeration>>PolarizationKind

The orientation of the RF energy radiated from the device.

<<enumeration>> PolarizationKind
VERTICAL
HORIZONTAL
RIGHT_CIRCULAR_POLARIZE
LEFT_CIRCULAR_POLARIZE

- <<enumeration>>RadiatingElementType

<<enumeration>> RadiatingElementType
MONOPOLE
DIPOLE
PATCH
CONE
DISH

- CartesianCoordinates

CartesianCoordinates
x : Meter
y : Meter
z : Meter

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.2.5.5.7 RFSwitch

Description

RF Switches provide for routing / rerouting of signals between various devices. They have many input ports and output ports. The switch connects the chosen input port to one or many output ports. It may also be programmed to turn off the signal transmission. (No input is connected to any output port)

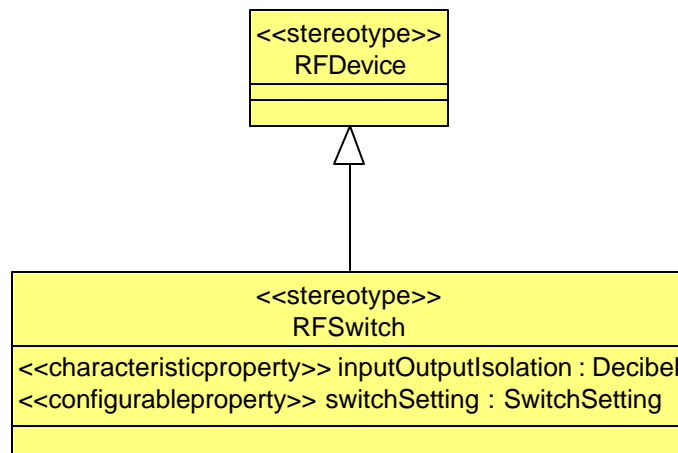


Figure 50 - RFSwitch Definition

Attributes

- <<characteristicproperty>>inputOutputIsolation: The amount of input port leakage on all unselected output ports.
Decibel
- <<characteristicproperty>>switchSetting: The current configuration matrix of the device.
SwitchSetting

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- <<abstract>>Decibel Ratio between two voltages, currents, or signal power levels.
- <<abstract>>SwitchSetting Indicates the connections between the switch's ports.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.3 Infrastructure

7.3.1 Radio Services

This section defines the stereotypes for an swradio service. The swradio service stereotypes, which are extensions of the UML Component, are depicted in the table below.

Table 6 — SWRadio Services Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
CositeService	Component	Service			A component that provides capability for managing cosite issues.
GPSService	Component	Service			A component that provides time and location from GPS.
EventService	Component	Service			A component that provides event channel capabilities for receiving and sending events amongst components.
FaultManagement Service	Component	Service			A component that provides capability to collect and report Built-In-Test (BIT) statistics
FileService	Component	Service			A component that provides

					capabilities for managing file systems and manipulating files.
InstallerService	Component	Service			A component that provides capabilities for installing applications within the radio.
LogService	Component	Service			A component that provides capabilities for writing and retrieving system log information, and managing a log.
MeasurementService	Component	Service			A component that provides capabilities for collecting and reporting statistics.
NamingService	Component	Service			A component that provides a <i>white page</i> capability for component registration and retrieval.
OSEnvironment	Component	Service			A component that provides operating system capabilities.
PresetService	Component	Service			A component that provides capability for managing waveform presets.
RetransmissionService	Component	Service			A component that provides capability to cross band two or more communication channels at different OSI layers.

ScannerService	Component	Service			A component that provides scanner capabilities for signal detection indication, resource permission and scanning management.
Service	Component	N/A			An abstract component that is viewed to be a service within the radio environment that can be used by swradio components.

7.3.1.1 Service

Description

The abstract Service component, as shown in Figure 51, defines the common service definition for any swradio service.

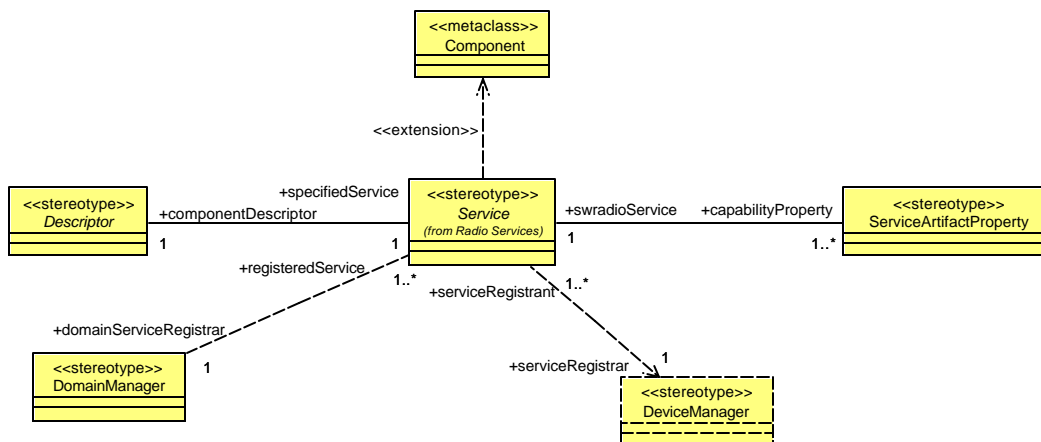


Figure 51 — Service Definition

Attributes

No additional attributes.

Associations

- componentDescriptor: Descriptor [1] An Service component is specified by a component Descriptor.
- capabilityProperty: ServiceArtifactProperty [1..*] An Service's characteristics are described by its capability properties.
- DomainServiceRegistrar: DomainManger[1] A domain consists of set of services usually registered in by its device manager.
- ServiceRegistrar: DeviceManager [1] A set of Services associated with DeviceManager.

Operations

No additional Operations.

Types and Exceptions

No additional Types and Exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.3.2 Communication Channel

Description

A software radio provides a means to enable communications between physically separated users. A software defined radio has the capability to use its devices as needed for a particular communications instance and to use them possibly in a different way in another instance. A Logical Communication Channel is the collection and interconnection of the radio's devices necessary for a particular waveform application to be able to provide communication. A radio may support one or many different logical communication channels. It may support multiple communication channels, but not all simultaneously due to the need for some device(s) by multiple waveforms. A "radio controller" needs visibility into the capabilities needed by its waveform applications and the capabilities provided by its devices in order to deploy a usable communication channel.

The Logical Communication Channel has been broken down to four "subchannels": the logical RF channel, the logical processing channel, the logical security channel, and the logical I/O

channel. **Error! Reference source not found.** shows the relationship of these channels to the communication channel, using UML.

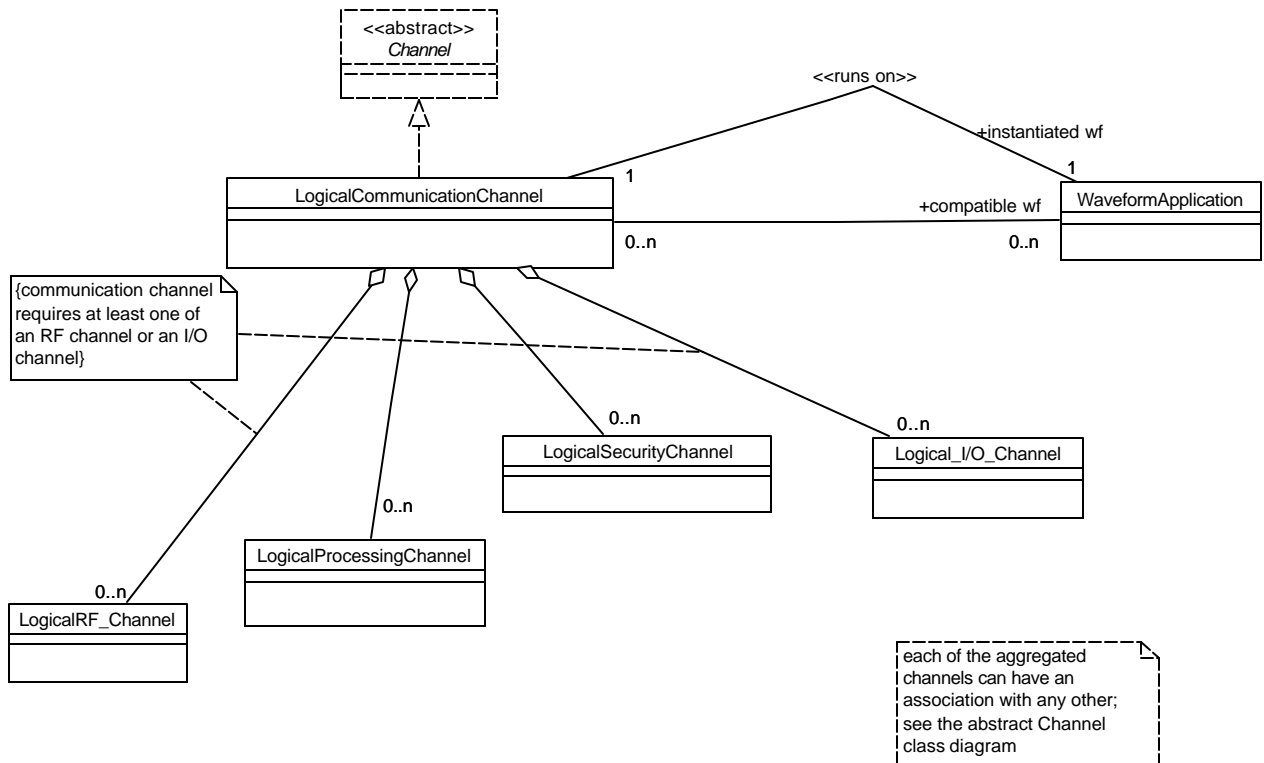


Figure 7-52. Logical Communication Channel Definition

Attributes

No additional attributes.

Associations

compatible	wf	(0..n):	A Logical Communication Channel may have all the
------------	----	---------	--

WaveformApplication	capabilities required by a WaveformApplication.
instantiated wf (1): WaveformApplication	An instantiated waveform application runs on its associated CommunicationChannel
(unnamed role)(0..n): LogicalRF_Channel	The Logical Communication Channel is composed of zero-to-n Logical RF Channels.
(unnamed role)(0..n): LogicalProcessingChannel	The Logical Communication Channel is composed of zero-to-n Logical Processing Channels.
(unnamed role)(0..n): LogicalSecurityChannel	The Logical Communication Channel is composed of zero-to-n Logical Security Channels.
(unnamed role)(0..n): LogicalI/O_Channel	The Logical Communication Channel is composed of zero-to-n Logical I/O Channels.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

This section is only intended to be used for PSM concept description.

This section will describe the mapping between PIM and PSM mappings.

Constraints

{communication channel requires at least one of an RF channel or an I/O channel}	The model allows for realizations that do not require security nor any processing (i.e. a non-software defined radio). Further, a valid channel may be an RF relay, with no local I/O, or may include a router and require no RF capability.
--	--

Semantics

To be provided.

7.3.2.1 Logical RF Channel

Description

The Logical RF Channel consists of all devices processing the analog signal after digitization, to and including the antenna(s). For convenience, A/D and D/A conversion devices, if used, are included here. The current state of the art is such that most of the operations of the interfaces realized by these devices are performed via hardware elements as opposed to software; nonetheless, the model does not force either implementation. **Error! Reference source not found.** shows the Logical RF Channel definition with attributes of its aggregated components.

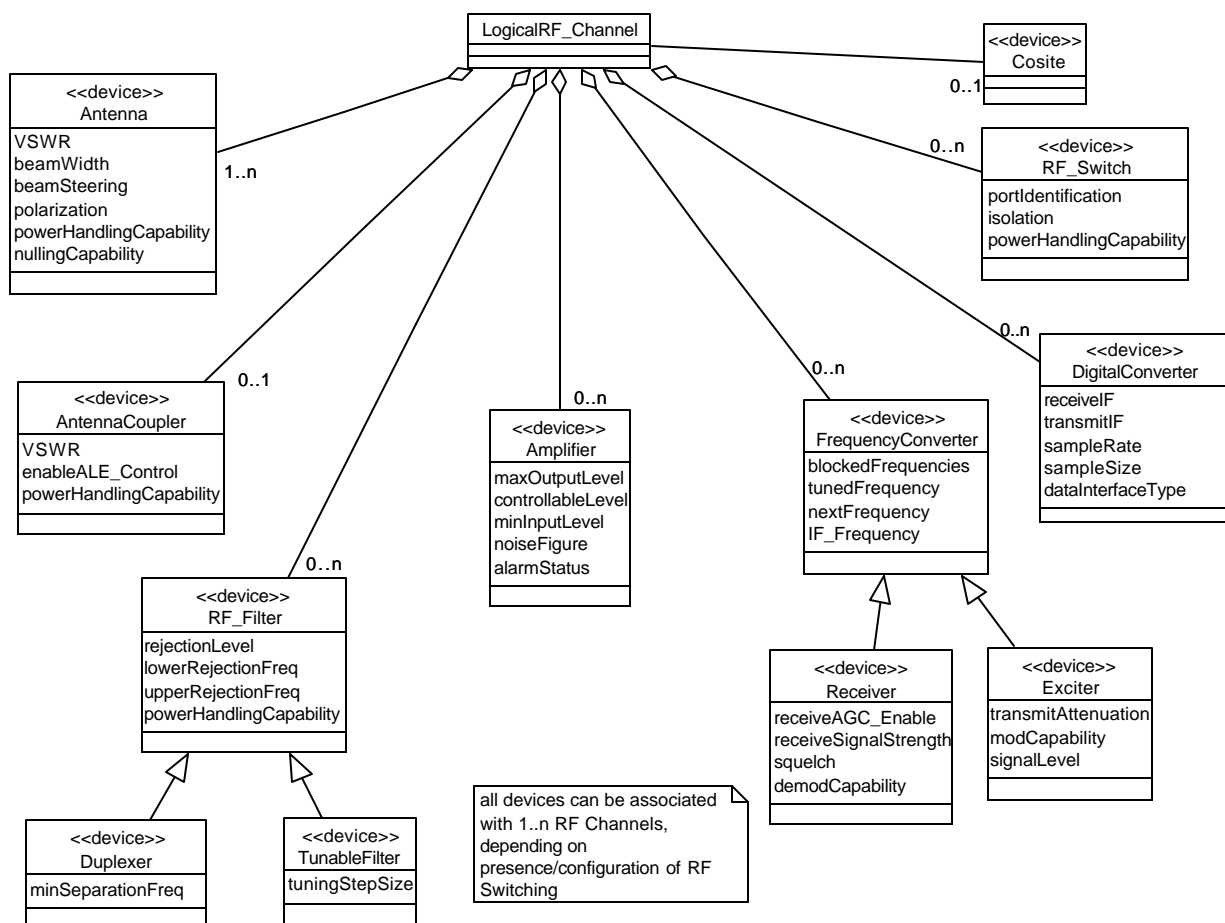


Figure 7-53. Logical RF Channel Definition

Attributes

No additional attributes.

Associations

tbid (0..1): Cosite	Description of the role.
---------------------	--------------------------

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

NA.

Constraints

No additional constraints.

Semantics

To be provided

7.3.2.2 Logical Processing Channel

Description

The Logical Processing Channel provides the processing nodes for waveform applications and radio service applications used by the waveforms and running on the processing channel's operating environment(s). An exception to this is the processing node(s) specific to supporting security functions, which are part of the Logical Security Channel. **Error! Reference source not found.** shows the Logical Processing Channel definition.

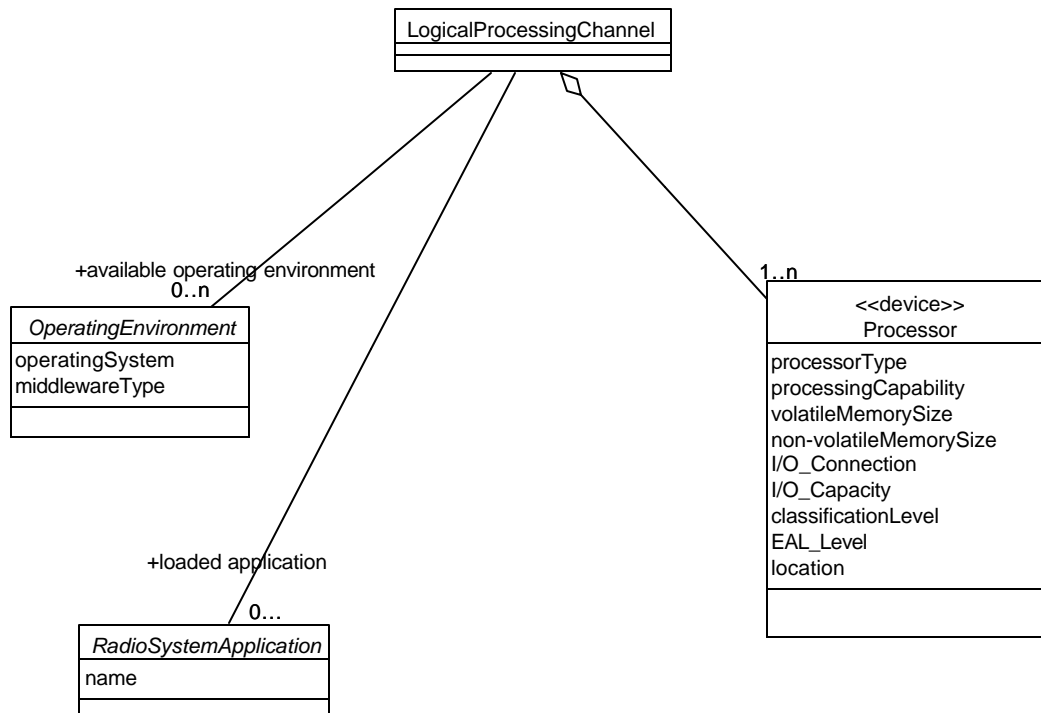


Figure 7-54. Logical Processing Channel Definition

Attributes

No additional attributes.

Associations

available operating environment(0..n): OperatingEnvironment	the Logical Processing Channel can have available zero or more operating systems that are hosted on the various processors in the system
loaded application(0..n): RadioSystemApplication	the Logical Processing Channel can have zero or more radio system applications loaded and available on the various processors in the system

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

To be provided

7.3.2.3 Logical Security Channel

Description

The Logical Security Channel provides the processing node(s) for security applications applicable to communications. Included here would be communications security processing and transmission security processing. The Logical Security Channel definition is shown in **Error! Reference source not found..**

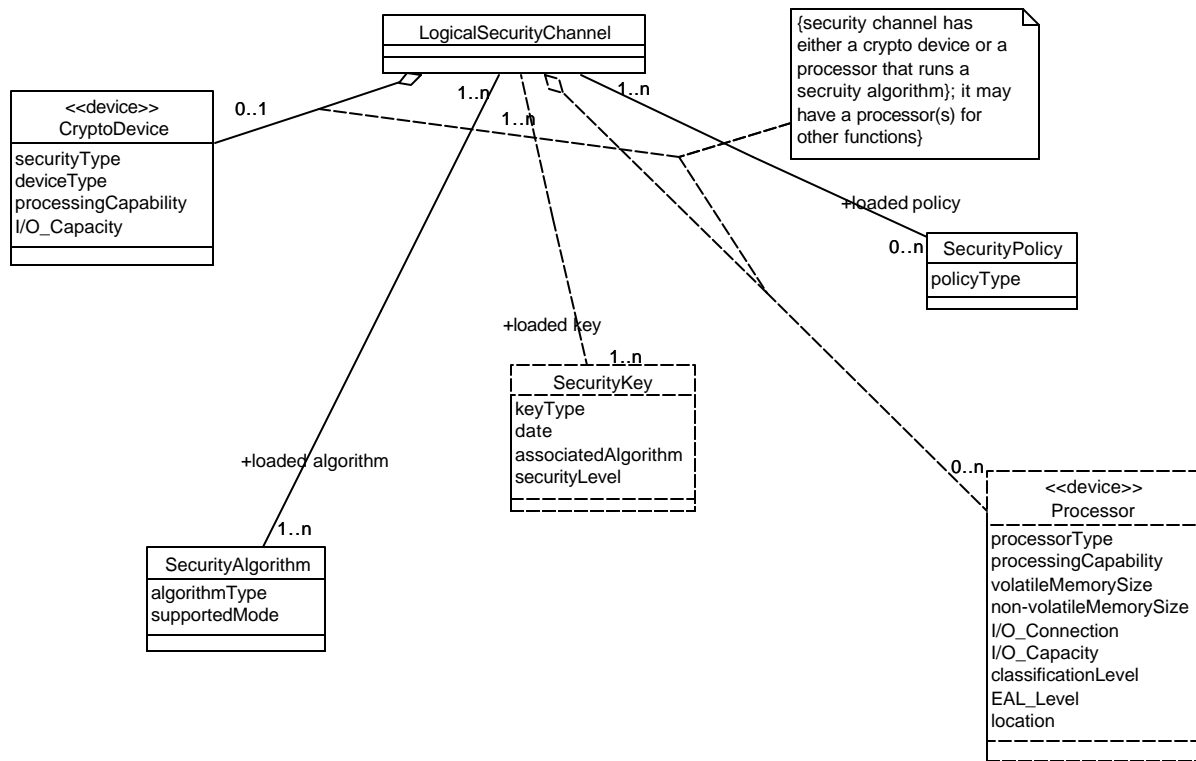


Figure 7-55. Logical Security Channel Definition

Attributes

No additional attributes.

Associations

loaded SecurityAlgorithm	algorithm(1..n):	A Logical Security Channel has one or more algorithms.
loaded key(1..n):	SecurityKey	A Logical Security Channel has one or more keys.
loaded policy(0..n):	SecurityPolicy	A Logical Security Channel may have security policy(ies) that direct its actions.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

{security channel has either a crypto device or a processor that runs a security algorithm}; it may have a processor(s) for other functions}	The Security Channel runs security algorithms on either a Processor or a dedicated Crypto Device
{SecurityAlgorithm requires either a CryptoDevice or a Processor to run on}	The Security Channel runs security algorithms on either a Processor or a dedicated Crypto Device

Semantics

To be provided

7.3.2.4 Logical I/O Channel

Description

The Logical I/O Channel provides for the baseband connection to the radio and consists of the devices that format, encode, decode, etc. the communication signals at that interface. **Error! Reference source not found.** is the UML Class diagram of the Logical I/O Channel.

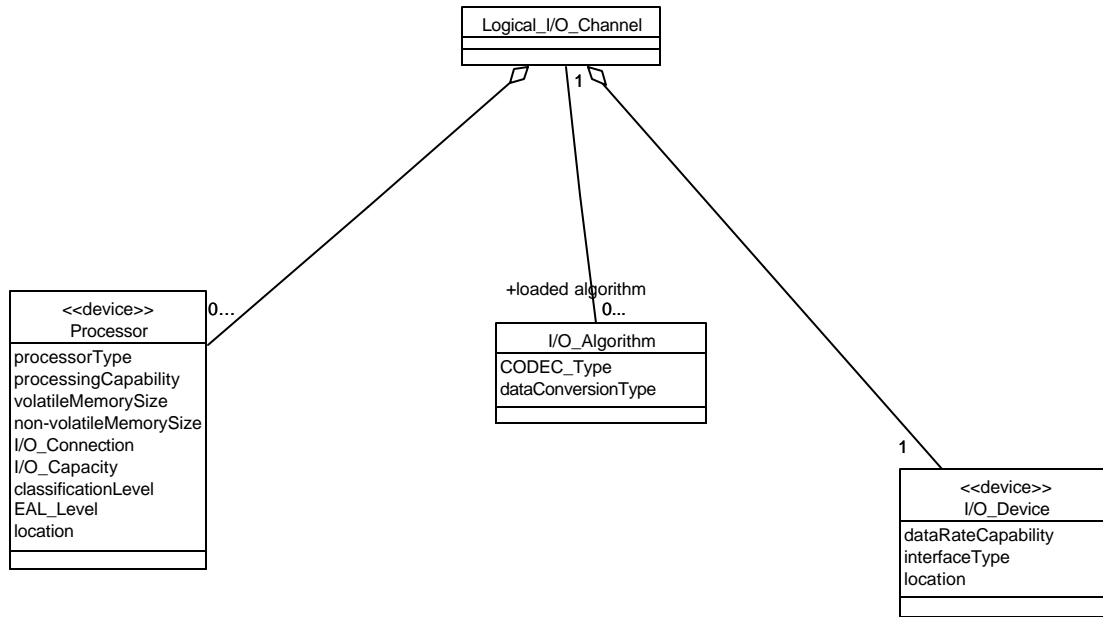


Figure 7-56. Logical I/O Channel Definition

Attributes

No additional attributes.

Associations

loaded algorithm(0..n): I/O_Algorithm	the Logical I/O Channel can have zero or more algorithms loaded in an I/O Device or Processor
---------------------------------------	---

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

To be provided

7.3.3 Radio Management

This section defines the stereotypes for radio management. Radio management involves the management of the radio and the management of devices and services within a radio. The radio management stereotypes are categorized as domain and device management. The details of each these categories are described in the following subsections.

7.3.3.1 Domain Management

This section defines the stereotypes for radio domain management. The domain management stereotypes are depicted in Table 7 below, which are extensions of the UML Interface and Component constructs. The details of each construct are described in the following subsections. The types of capabilities offered by radio domain management are categorized as follows:

1. Domain Registration Management – provides the mechanism for registering and unregistering services within a radio.
2. Domain Installation Management – provides the mechanism for installing and uninstalling applications within a radio.
3. Domain Retrieval – provides the mechanism for retrieving radio's components.
4. Domain Event Management – provides the mechanism for receiving asynchronous radio's domain event changes.

Table 7 — DomainManagement Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
IDomainEventChannels	Interface	N/A			An Interface that provides Domain Event Management capability to manage domain event channels connections.
IDomainInstallation	Interface	N/A			An Interface that provides Domain Installation Management capability to manage domain application installations.

Stereotype	Base Class	Parent	Tags	Constraints	Description
IDomainRetrieval	Interface	N/A			An Interface that provides Domain Retrieval capability to retrieve domain elements.
IDomainRegistration	Interface	N/A			An Interface that provides Domain Registration Management capability to manage domain service registration.
DomainManager	Component	SWRADIO Component			A component that manages a domain.
RadioManager	Component	DomainManager			A component that manages a RadioSet.

1.1.1.1.1 IDomainInstallation

Description

The IDomainInstallation interface, as shown in Figure 57, defines radio domain application installation capabilities. The interface provides the capabilities of adding and removing applications from a radio domain.

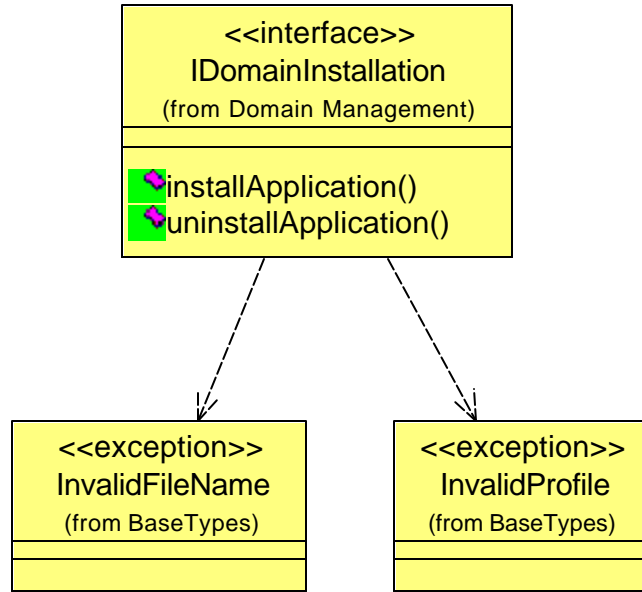


Figure 57 — IDomainInstallation Definition

Attributes

No additional attributes.

Associations

No additional Associations.

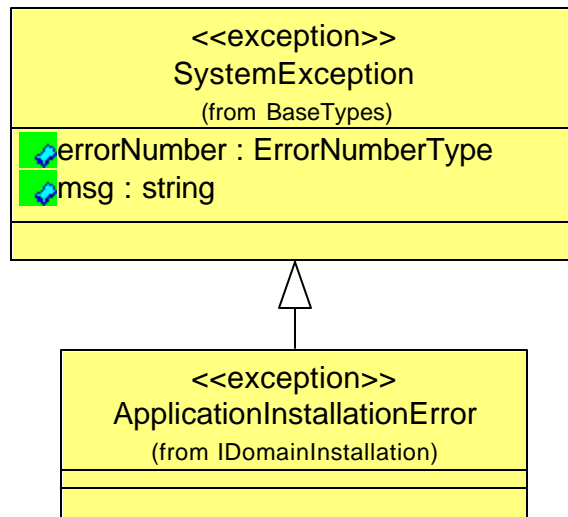
Operations

- installApplication(in profileFileName: String): {raises = (InvalidProfile, InvalidFileName, ApplicationInstallationError)}**
 The **installApplication** operation is used to install new application software in the domain.
- uninstallApplication(in applicationId: String): {raises = (InvalidIdentifier, ApplicationUninstallationError)}**
 The **uninstallApplication** operation is used to uninstall an ApplicationFactory in the domain. The **applicationId** corresponds to the identifier in the installed application profile.

Types and Exceptions

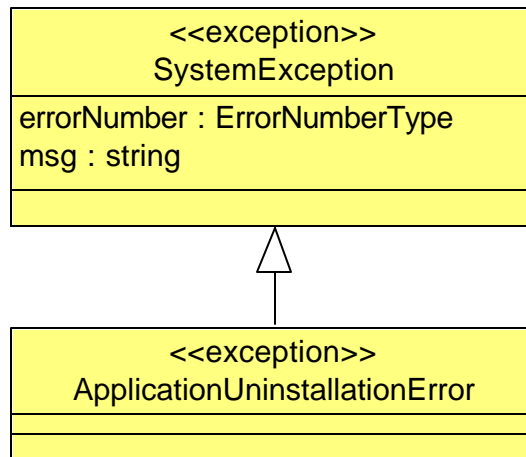
- <<exception>>ApplicationInstallationError** The **ApplicationInstallationError** exception, a type of System Exception, is raised when an Application installation has not completed correctly. The error number indicates an **ErrorNumberType** value (e.g., **EINVAL**, **ENAMETOOLONG**, **ENOENT**, **ENOMEM**,

ENOSPC, ENOTDIR, ENXIO). The message is component-dependent, providing additional information describing the reason for the error.



- `<<exception>>ApplicationUninstallationError` or

.The `ApplicationUninstallationError` exception, a type of `SystemException`, is raised when an Application uninstallation has not completed correctly. The error number value and the message is component-dependent, providing additional information describing the reason for the error.



- `<<exception>> InvalidIdentifier`

The `InvalidIdentifier` exception indicates an application identifier is invalid.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

An installer service typically invokes these operations when adding or removing an ApplicationFactory (installed ApplicationAssembly) from the domain.

1.1.1.1.2 IDomainEventChannels

Description

The IDomainRegistration interface, as shown in Figure 58, defines radio domain event channel registration capabilities. The interface provides the capabilities of adding and removing connections to event channels in a radio domain.

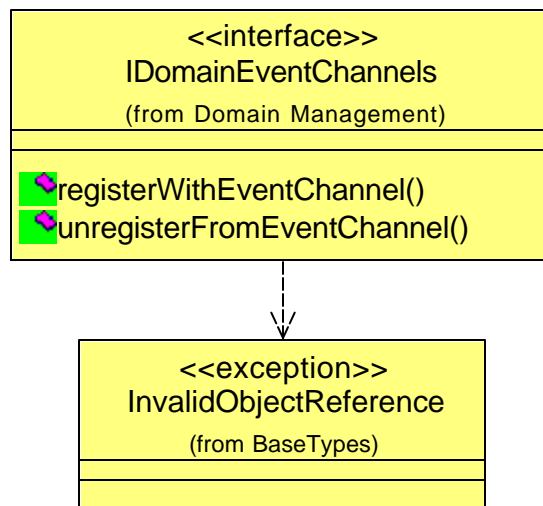


Figure 58 — IDomainEventChannels Definition

Attributes

No additional attributes.

Associations

No additional Associations.

Operations

- registerWithEventChannel(in registeringObject: Object, in registeringId: String, in eventChannelName : String): {raises = (InvalidObjectReference, InvalidEventChannelName, AlreadyConnected)}

The registerWithEventChannel operation is used to connect a consumer to a domain's event channel.

The registerWithEventChannel operation raises the InvalidObjectReference exception when the input registeringObject parameter contains an invalid reference to a event channel consumer

type interface.

The registerWithEventChannel operation raises the InvalidEventChannelName exception when the input eventChannelName parameter contains an invalid event channel name.

The registerWithEventChannel operation raises AlreadyConnected exception when the input parameter contains a connection to the event channel for the input registeringId parameter.

- unregisterFromEventChannel(in unregisteringId: String, in eventChannelName: String): { raises = (InvalidEventChannelName, NotConnected)}

The unregisterFromEventChannel operation is used to disconnect a consumer from a domain's event channel. The unregisterFromEventChannel operation raises the InvalidEventChannelName exception when the input eventChannelName parameter contains an invalid reference to an event channel.

The unregisterFromEventChannel operation raises the NotConnected exception when the input parameter unregisteringId parameter is not connected to specified input event channel.

Types and Exceptions

- <<exception>>AlreadyConnected
- <<exception>>InvalidEventChannelName
- <<exception>> NotConnected

The AlreadyConnected exception indicates that a registering consumer is already connected to the specified event channel.

The InvalidEventChannelName exception indicates that the event channel with that name does not exist within the domain.

The NotConnected exception indicates that the unregistering consumer was not connected to the specified event channel.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.1.1.3 IDomainRegistration

Description

The IDomainRegistration interface, as shown in Figure 59, defines radio domain service registration capabilities. The interface provides the capabilities of adding and removing services from a radio domain.

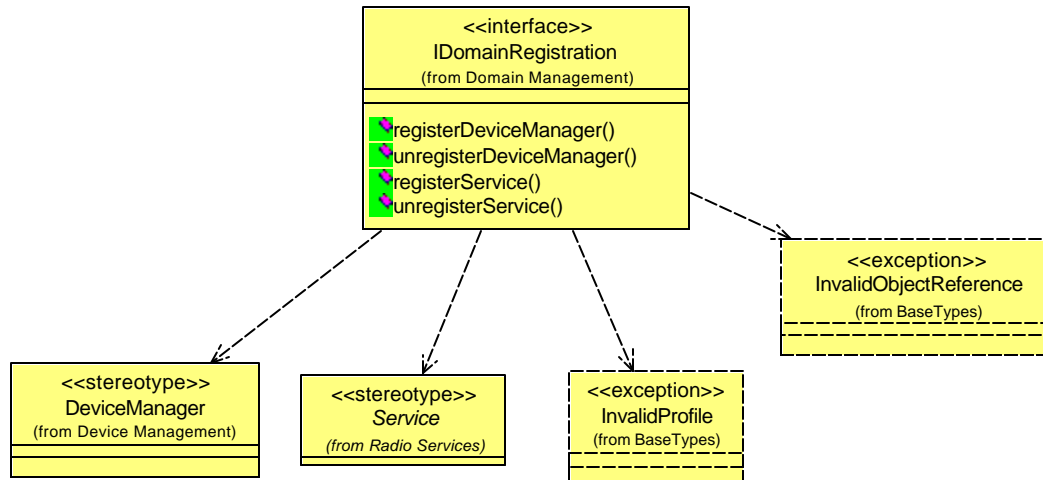


Figure 59 — IDomainRegistration Definition

Attributes

No additional attributes.

Associations

No additional Associations.

Operations

- registerDeviceManager(in deviceMgr: DeviceManager):**
 {raises = (InvalidObjectReference, InvalidProfile, RegisterError)}
 The registerDeviceManager operation is used to register a DeviceManager, its Device(s), and its Services. Software profiles can also be obtained from the DeviceManager's FileSystem.
- unregisterDeviceManager(in deviceMgr: DeviceManager):**
 {raises = (InvalidObjectReference, UnregisterError) }
 The unregisterDeviceManager operation is used to unregister a DeviceManager component from the DomainManager's Domain Profile. A DeviceManager may be unregistered during run-time for dynamic extraction or maintenance of the DeviceManager.
- registerService(in registeringService: Service, in registeredDeviceMgr: DeviceManager, in name: String):**
 {raises = (InvalidObjectReference, DeviceManagerNotRegistered,
 The registerService operation is used to register a service for a specific DeviceManager within the domain.

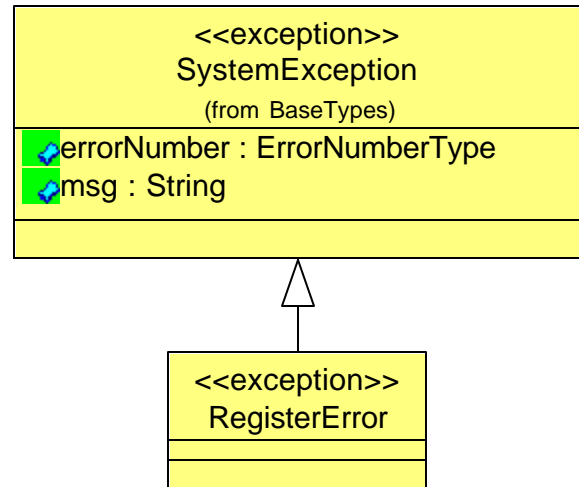
RegisterError) }

- unregisterService(in
unregisteringService: Service, in
name: String): {raises = (
InvalidObjectReference,
UnregisterError)}

The *unregisterService* operation is used to remove a service entry from the domain.

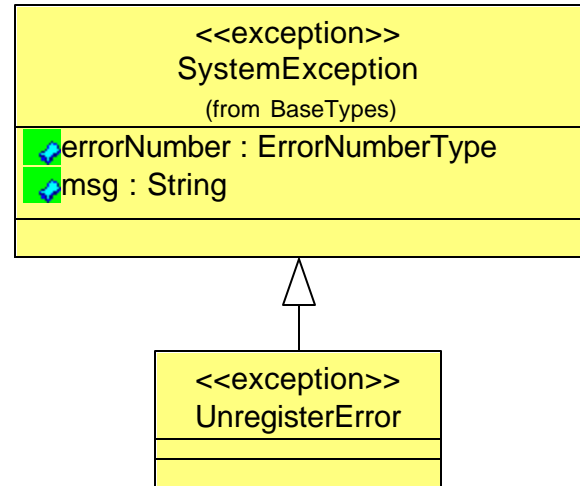
Types and Exceptions

- <<exception>>DeviceManagerNotRegistered The DeviceManagerNotRegistered exception indicates the registering service's DeviceManager is not registered in the domain. A service's DeviceManager has to be registered prior to a serviceregistration to the domain.
- <<exception>>RegisterError The RegisterError exception, a type of System Exception, indicates that an internal error has occurred to prevent domain registration operations from successful completion. The error number indicates an ErrorNumberType value. The message is component-dependent, providing additional information describing the reason for the error.



- <<exception>>UnregisterError

The *UnregisterError* exception, a type of *SystemException*, indicates that an internal error has occurred to prevent domain unregister operations from successful completion. The error number indicates an *ErrorNumberType* value. The message is component-dependent, providing additional information describing the reason for the error.



Mapping

N/A.

Constraints

No additional constraints.

Semantics

The IDomainRegistration interface provides the mechanisms for components, such as node managers, to register their services for a specific domain. As services are removed from environment, the interface provides the capability of removing them from the domain.

1.1.1.1.4 IDomainRetrieval

Description

The IDomainRetrieval interface, as shown in Figure 60, defines radio domain retrieval capabilities. The interface provides the capabilities of retrieving domain's components and profile.

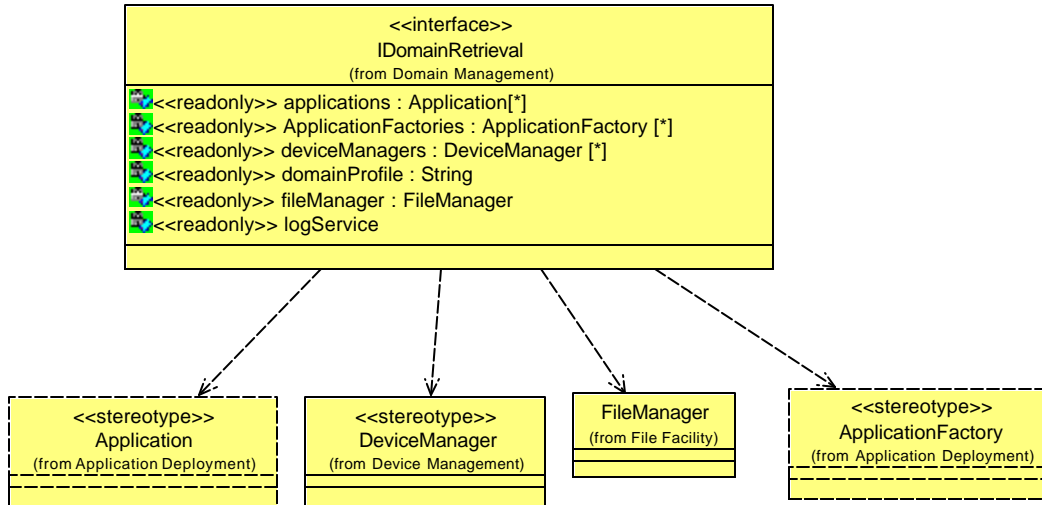


Figure 60 — IDomainRetrieval Definition

Attributes

- <<readonly>>applications :**
Applications [*]
 The readonly applications attribute contains a sequence of instantiated Applications in the domain.
- <<readonly>>applicationFactories :**
ApplicationFactory [*]
 The readonly applicationFactories attribute contains a list ApplicationFactories, with one ApplicationFactory per successfully installed (i.e. no exception raised) application (SAD file and associated files).
- <<readonly>>deviceManagers :**
DeviceManager [*]
 The readonly deviceManagers attribute contains a sequence of registered DeviceManagers in the domain.
- <<readonly>>domainProfile :**
String
 The readonly domainManagerProfile attribute contains either a profile element with a file reference to the DomainManager Configuration Descriptor (DMD) profile or the XML for the DomainManager's (DMD) profile. Files referenced within the profile will have to be obtained from the DomainManager's FileManager.
- <<readonly>>fileManager :**
FileManager
 The readonly fileMgr attribute contains a domain's FileManager.
- <<readonly>>logService :** Log
 The readonly logService attribute contains a domain's log service.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional Types and Exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.1.1.5 DomainManager

Description

The DomainManager, as shown in Figure 61, describes the definition and relationships that are common for all Domain Managers.

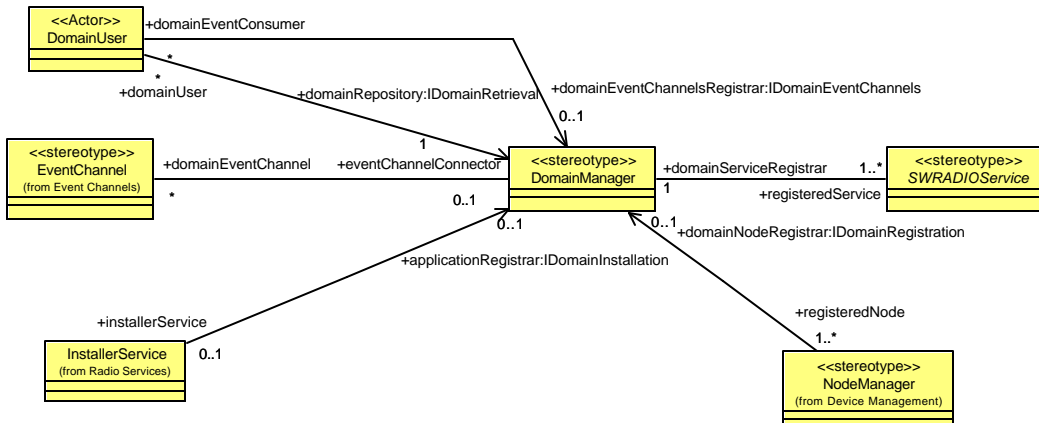


Figure 61 — DomainManager Definition

Attributes

No additional attributes.

Associations

- domainEventChannel: EventChannel [*]
A DomainManager may be associated with many event channels.
- domainEventConsumer: DomainUser [*]
A DomainManager may be associated with many users that register to consume domain events.

- domainUser: DomainUser [*] A DomainManager may be associated with many users that access the system.
- installerService: InstallerService [0..1] A DomainManager may provide the capability to install applications provided by an InstallerService.
- registeredNode: NodeManager [1..*] A DomainManager is associated with one to many nodes that make up the domain.
- registeredService: SWRADIOService [1..*] A DomainManager is associated with one to many services that up the domain.

Operations

No additional Operations.

Types and Exceptions

No additional Types and Exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

The set of interfaces realized by a DomainManager depends on the system the DomainManager is built for. As shown in Figure 61 above, the DomainManager could realize up to four interfaces.

1.1.1.1.6 RadioManager

Description

The RadioManager component, as shown in Figure 62, describes the definition and relationships that are common for radio Domain Managers.

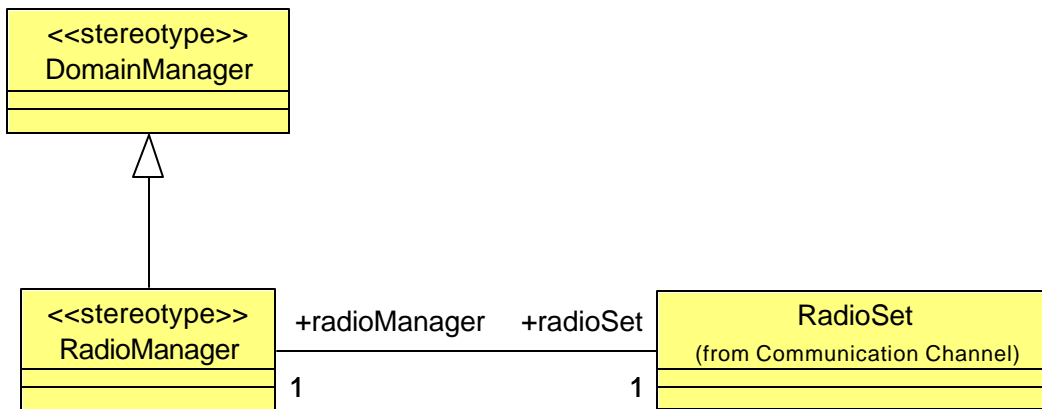


Figure 62 — RadioManager Definition

Attributes

No additional attributes.

Associations

- radioSet: RadioSet [1] A RadioManager is associated with one RadioSet

Operations

No additional Operations.

Types and Exceptions

No additional Types and Exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

7.3.3.2 Node Management

This section defines the stereotypes for radio node management. The node management stereotypes are depicted in Table 8 below, which are extensions of the UML Interface and Node constructs. The details of each construct are described in the following subsections. The types of capabilities offered by radio node management are categorized as follows:

1. Service Registration Management – provides the mechanism for registering and unregistering services within a node.
2. Node Retrieval – provides the mechanism for retrieving radio's node components.

Table 8 — Node Management Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
INodeRetrieval	Interface	N/A			An Interface that provides Node Retrieval capability to retrieve node elements.
INodeRegistration	Interface	N/A			An Interface that provides Service

Stereotype	Base Class	Parent	Tags	Constraints	Description
					Registration Management capability to manage node service registration.
NodeManager	Component	SWRADIOComponent			A component that manages a node.
DeviceManager	Component	NodeManager			A component that manages a node and its services.

7.3.3.2.1 INodeRetrieval

Description

The INodeRetrieval, as shown in Figure 63, defines radio node retrieval capabilities. The interface provides the capabilities of retrieving node's components and profile.

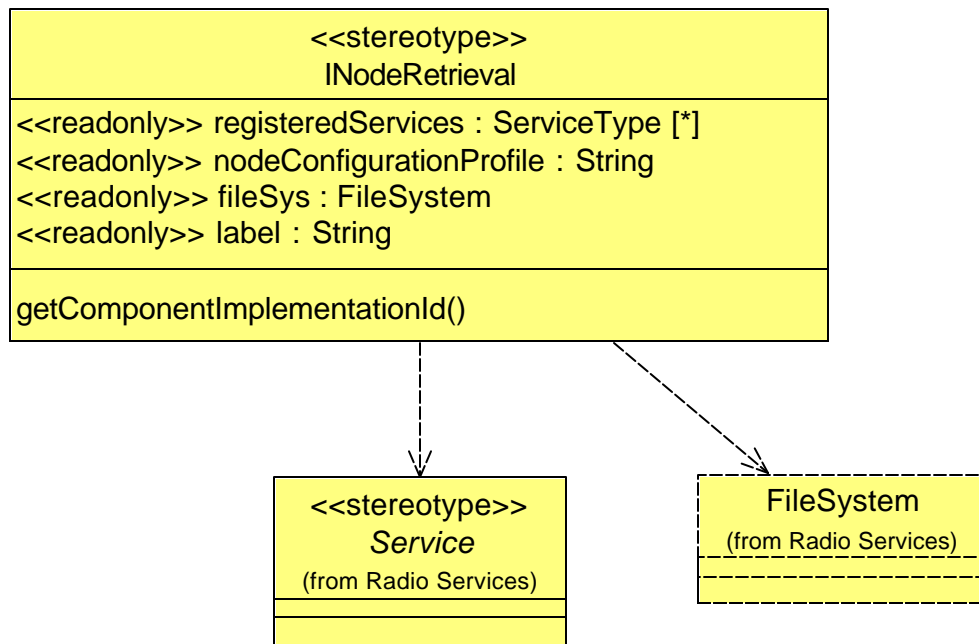


Figure 63 — INodeRetrieval Definition

Attributes

- `<<readonly>>label: String` The readonly label attribute contains a node's meaning name.

- <<readonly>>fileSys: FileSystem The readonly fileSys attribute contains the FileSystem associated with this node or a nil component reference if no FileSystem is associated with this node.
- <<readonly>>nodeConfigurationProfile : String The readonly nodeConfigurationProfile attribute contains information on the initial configuration for the node. Files referenced within the profile are obtained using the fileSys attribute.
- <<readonly>>registeredServices : ServiceSequence The readonly registeredServices attribute contains a list of Services that have registered with this node or a sequence length of zero if no Services have registered with the node.

Associations

No additional associations.

Operations

- getComponentImplementationId (in componentInstantiationId: String, return String): The getComponentImplementationId operation returns the component's (e.g., Service) implementation ID used to create the component identified by the input componentInstantiationId parameter. The implementation ID corresponds to the ServiceExecutableCode used to manifest a Service. The getComponentImplementationId operation returns an empty string when the input componentInstantiationId parameter does not match any element in the nodeConfigurationProfile. This operation does not raise any exceptions.

Types and Exceptions

- ServiceType This structure provides the Service reference and name of Service that have registered with the node.

ServiceType
serviceObject : Service serviceName : String

Mapping

N/A.

Constraints

No additional constraints.

Semantics

The set of interfaces realized by a DomainManager depends on the system the DomainManager is built for. As shown in Figure 61 above the DomainManager could realize up to four interfaces.

7.3.3.2.2 INodeRegistration

Description

The INodeRegistration interface, as shown in Figure 64, defines radio node service registration capabilities. The interface provides the capabilities of adding and removing services from a radio node.

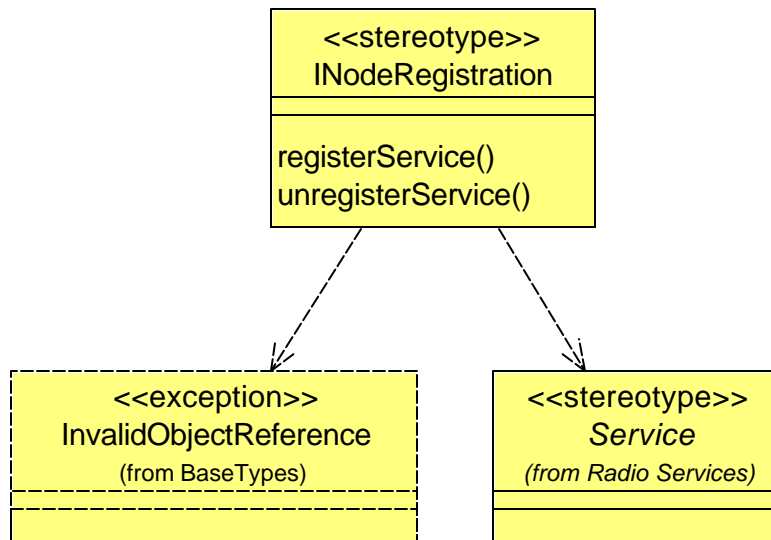


Figure 64 — INodeRegistration Definition

Attributes

No additional attributes.

Associations

No additional associations.

Operations

- registerService(in registeringService: Service, in name: String): {raises = (InvalidObjectReference) }

The registerService operation provides the mechanism to register a Service with a NodeManager and the NodeManager's associated DomainManager. The registeringService is ignored when duplicated. The registerService operation raises the InvalidObjectReference exception when the input registeringService is a nil component reference.

- unregisterService(in unregisteringService: Service, in

The unregisterService operation unregisters a Service from a NodeManager and the NodeManager's

name: String): {raises = (InvalidObjectReference)}

associated DomainManager. The unregisterService operation raises the InvalidObjectReference when the input registeredService is a nil component reference or does not exist in the node manager.

Types and Exceptions

No additional Types and Exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

The INodeRegistration interface provides the mechanisms for service components started up on a node to register them to the NodeManager that is managing a node. As services are removed from node environment, the interface provides the capability of removing them from the NodeManager.

7.3.3.2.3 DeviceManager

Description

The DeviceManager a type of NodeManager, as shown in Figure 65, defines the definition and relationships that are common for all device managers.

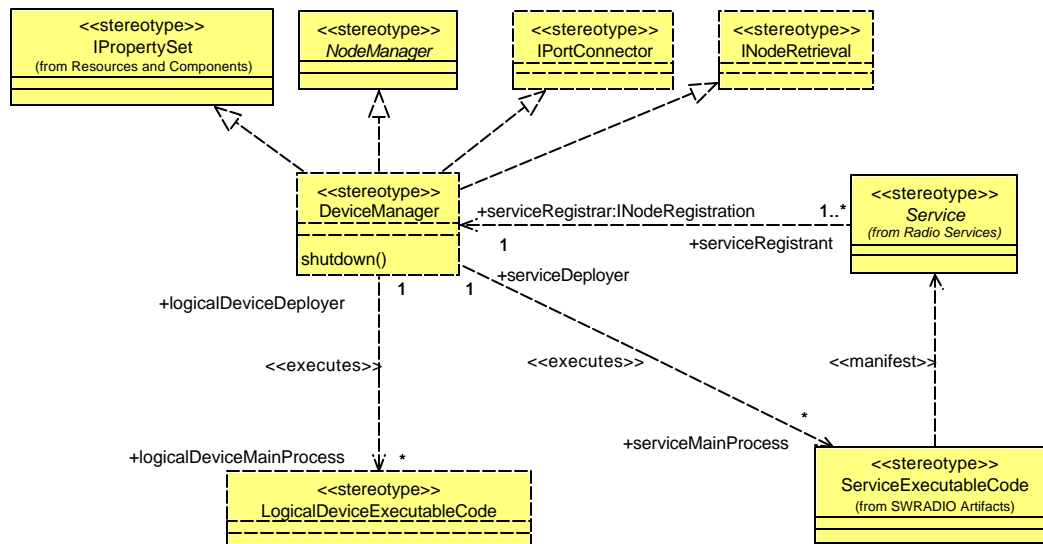


Figure 65 — DeviceManager Definition

Attributes

No additional attributes.

Associations

- serviceMainProcess: ServiceExecutableCode [*] A DeviceManager may start up LogicalServiceMainProcesses.
- ServiceRegistrant: Service [1..*] A DeviceManager must be associated with a Service.

Operations

- shutdown ()

The shutdown operation provides the mechanism to terminate a DeviceManager. The shutdown operation unregisters the DeviceManager from the DomainManager.

The shutdown operation performs releaseObject on all of the DeviceManager's registered Devices (DeviceManager's registeredServices attribute).

The shutdown operation shall cause the DeviceManager to be unavailable from the OS environment when all of the DeviceManager's registered Services are unregistered from the DeviceManager. This operation does not return any value and does not raise any exceptions.

Types and Exceptions

No additional Types and Exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

The DeviceManager provides the capability of starting up services' main processes on given node by the DeviceConfigurationDescriptor. Services started up also include LogicalDevices that are a type of managed Service. A DeviceManager could register to a DomainManager using the nodeConfigurationProfile information.

7.3.3.2.4 NodeManager

Description

The abstract NodeManager, as shown in Figure 66, defines the definition and relationships that are common for all node managers.

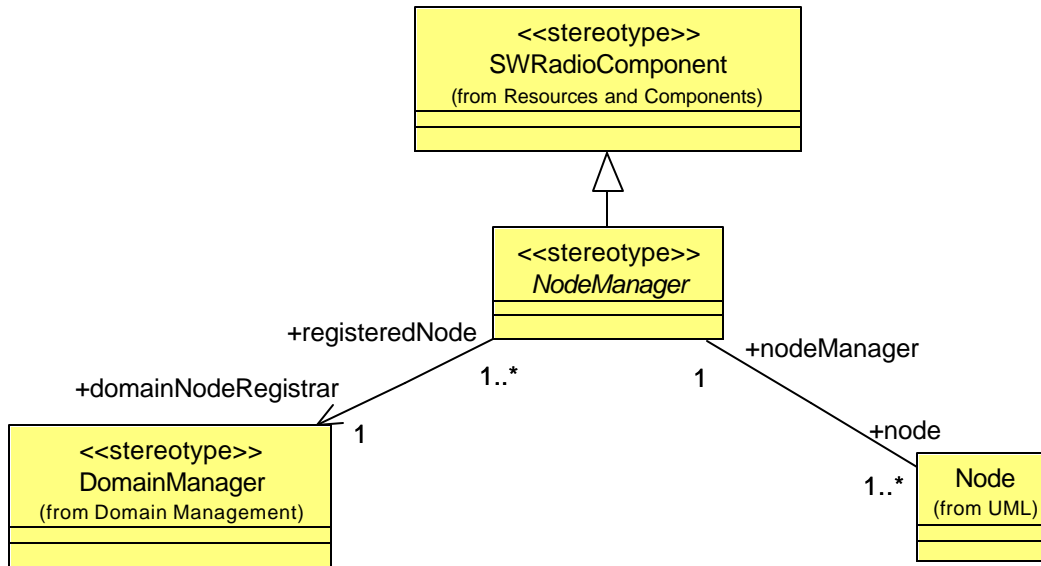


Figure 66 — NodeManager Definition

Attributes

No additional attributes.

Associations

- node: NodeManager [1..*] A NodeManager may manage one to many nodes.
- registeredNode: NodeManager [1..*] A NodeManager is associated with one DomainManager, which harnesses all nodes' services and capabilities.

Operations

No additional Operations.

Types and Exceptions

No additional Types and Exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

The type of management performed a NodeManager is not specified at this level.

7.3.3.3 Domain Event Channels

7.3.3.3.1 DomainManagementObjectEventType

7.3.3.3.2 DomainManagementObjectAddedEventType

7.3.3.3.3 DomainManagementObjectRemovedEventType

7.3.3.3.4 Event Channel

7.3.3.3.5 EventType

7.3.3.3.6 IncomingDomainEventChannel

7.3.3.3.7 OutgoingDomainEventChannel

7.3.3.3.8 StateChangeEvent

7.3.4 SWRADIO Deployment

SWRADIO deployment describes the swradio executable artifacts that are involved in the deployment of applications (e.e, waveforms), logical devices, and services within a radio environment. This section also describes the components that are involved in the deployment of and management of Applications.

8 Platform Independent Model (PIM)

The SWRADIO PIM Components are made of:

- Common Layer Facilities - This facility defines the set of interfaces that all common building blocks for components within the radio. Examples of these types of interfaces are flow control, packet, and stream interfaces.
- Common Radio Facilities - This facility defines the set of services that all components within the radio can be used. Examples of these types of services are log, naming, event service, and Files services
- Data Link Layer Facilities - These facilities define data link and Media Access Control (MAC) functionality for waveform processing.
- I/O Facilities – These facilities defines the functionality for baseband I/O such as serial and audio devices.
- Physical Layer Facilities - These facilities define the functionality to convert the digitized signal into a propagating RF wave, and conversely, to convert a propagating RF wave into a digitized signal for processing. The facilities also include frequency tuning, filters, interface cancellation, analog digital conversion, up/down conversion, gain control, synthesizer etc., functionality.

- Radio Control Facilities - These facilities define the functionality to configure, get status, and control the radio domain and channels within the radio.

1.2 Common Radio Facilities

8.1.1 File Facilities

The FileServices consist of interfaces and components that are used to manage and access a distributed file system. The File Services are used for installation and removal of application and artifact files within the system, and for loading and unloading application files on the various processors that the Devices execute upon. The File Services are identified in Table 9 — File Services Stereotypes and their relationship is graphically depicted in Figure 67.

Table 9 — File Services Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
IFileSystem	Interface	N/A			Indicates an Interface that provides remote access to a physical file system.
File	Component	SWRADIOCompone nt			Indicates a component that reads and writes a file within a FileSystem.
FileSystem	Component	SWRADIOCompone nt			Indicates a component that remotely accesses a physical file system.
FileManager	Component	SWRADIOCompone nt			Indicates a component that provides a single interface to multiple, distributed FileSystems

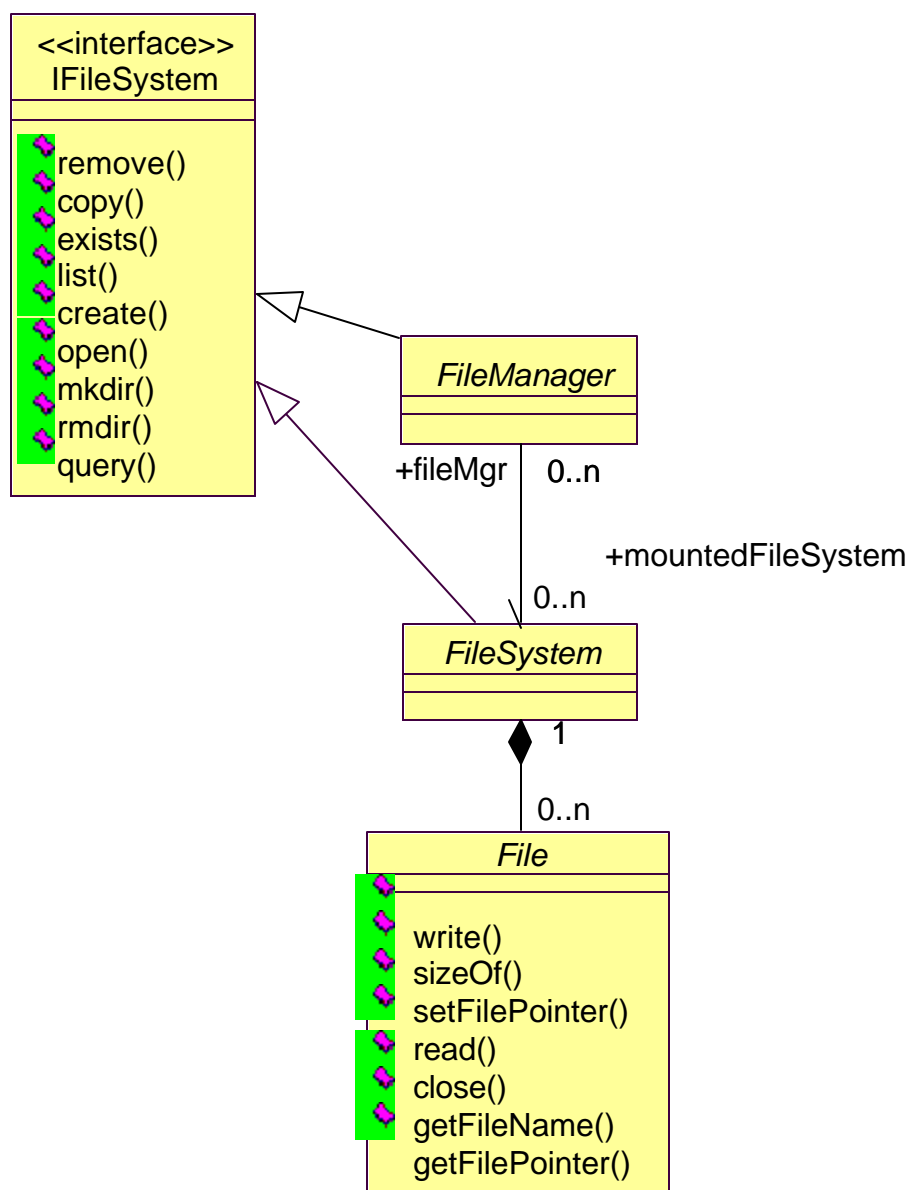


Figure 67 — File Facility Overview

8.1.1.1 IFileSystem

Description

The IFileSystem interface, as shown in Figure 68 — IFileSystem Definition, defines common operations that enable remote access to a physical file system.

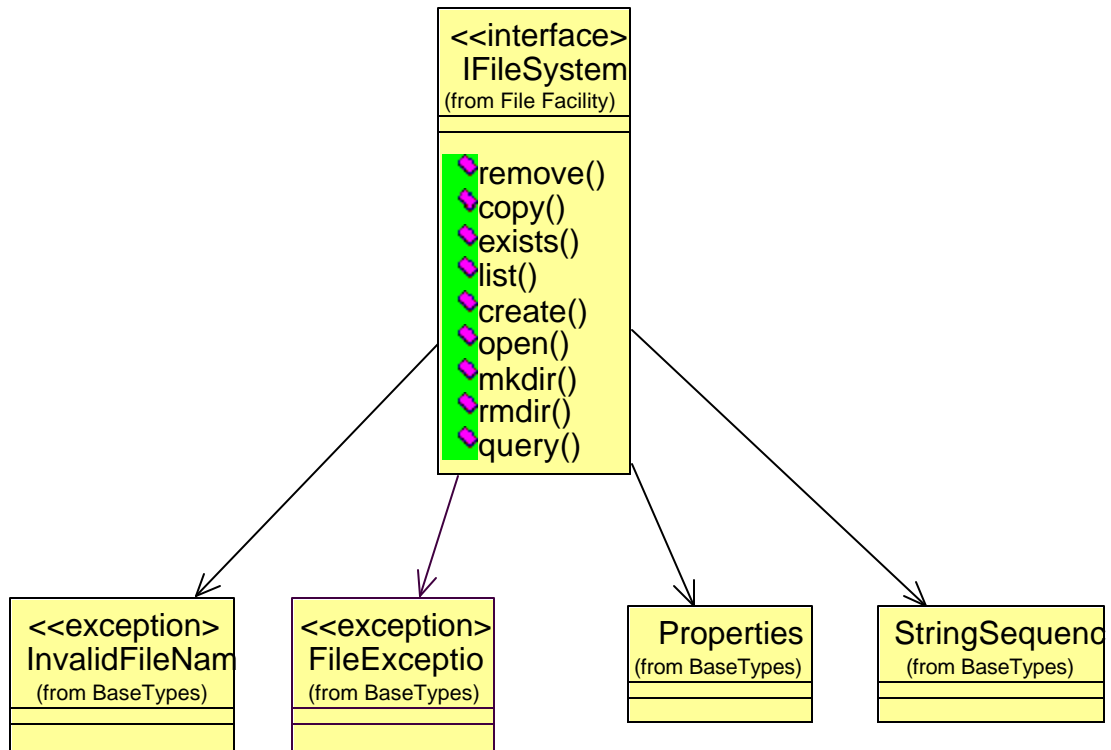


Figure 68 — IFileSystem Definition

Attributes

No additional attributes.

Associations

No additional associations.

Operations

- `remove(in fileName : String) : {raises = (FileException, InvalidFileName)}`

The remove operation provides the ability to remove a regular file (non-directory) from a file system. The remove operation removes the file with the given fileName. The remove operation raises the InvalidFileName exception when the fileName is not a

valid fileName or not an absolute pathname.

The remove operation raises the FileException when a file-related error occurs.

- copy(in sourceFileName : String, in destinationFileName : String) : {raises = (InvalidFileName, FileException)}

The copy operation provides the ability to copy a regular file (non-directory) to another regular file. The copy operation copies the source file with the specified sourceFileName to the destination file with the specified destinationFileName. The copy operation raises the FileException when a file-related error occurs.

The copy operation raises the InvalidFileName exception when the source or destination filename is not a valid file name or not an absolute pathname.

- exists(in fileName : String, return Boolean) : {raises = (InvalidFileName)}

The exists operation checks to see if a file exists based on the fileName parameter. The exists operation returns True if the file exists, or False if it does not. The exists operation raises the InvalidFileName exception when fileName is not a valid file name or not an absolute pathname.

- list(in pattern : String, return FileInformationSequence) : {raises = (FileException, InvalidFileName)}

The list operation provides the ability to obtain a list of files along with their information in the FileSystem according to a given search pattern. The list operation can be used to return information for one file or for a set of files. The list operation returns a FileInformationSequence for files that match the wildcard specification as specified in the input pattern parameter. The list operation returns a zero length sequence when no file matching occurred for the input pattern parameter. The list operation raises the InvalidFileName exception when the input pattern does not start with a slash "/" or cannot be interpreted due to unexpected characters.

The list operation raises the FileException when a file-related error occurs.

- create(in filename : String, return : File) : {raises = (InvalidFileName, FileException)}

The create operation provides the ability to create a new file on the FileSystem. The create operation creates a new File based upon the provided fileName attribute. The create operation shall return a File component reference to the opened file. The create operation returns a null file component reference if an error occurs. The create operation raises the FileException if the file already exists or another file error occurred. The create operation raises the InvalidFileName exception when a fileName is not a valid file name or not an absolute pathname or path prefix does not exist.

- open(in filename : String, in read_Only : Boolean, return : File) : {raises = (

The open operation provides the ability to open a file for read or write. The open operation opens a file

Boolean, return : File) : {raises = (InvalidFileName, FileException)}

based upon the input fileName. The read_Only parameter indicates if the file should be opened for read access only. The open operation opens the file for write access when the read_Only parameter is false. The open operation returns a File component parameter on successful completion. The returned File's file pointer is set to the beginning of the file when the read_Only parameter is true, otherwise the File's file pointer is set to the end of the file. The open operation returns a null file component reference if the open operation is unsuccessful. If the file is opened with the read_Only flag set to true, then writes to the file will be considered an error. The open operation raises the FileException if the file does not exist or another file error occurred.

The open operation raises the InvalidFileName exception when the filename is not a valid file name or not an absolute pathname.

- mkdir(in directoryName : String) : {raises(InvalidFileName, FileException)}

The mkdir operation provides the ability to create a directory on the file system. The mkdir operation creates a FileSystem directory based on the directoryName given. The mkdir operation creates all parent directories required to create the directoryName path given. The mkdir operation raises the FileException if a file-related error occurred during the operation. The mkdir operation raises the InvalidFileName exception when the directoryName is not a valid directory name.

- rmdir(in directoryName : String) : {raises(InvalidFileName, FileException)}

The rmdir operation provides the ability to remove a directory from the file system. The rmdir operation removes a FileSystem directory, based on the directoryName given, only if the directory is empty (no files exist in directory). The rmdir operation raises the FileException when the directory does not exist, if the directory is not empty, or another file-related error occurred. The rmdir operation raises the InvalidFileName exception when the directoryName is not a valid directory name.

- query(inout fileSystemProperties : Properties) : {raises(UnknownFileSystemProperties)}

The query operation provides the ability to retrieve information about a file system. The query operation returns file system information to the calling client based upon the given fileSystemProperties' ID.

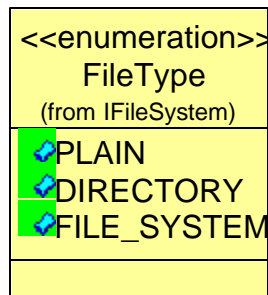
As a minimum, the FileSystem query operation supports the following fileSystemProperties:

1. SIZE – an ID value of "SIZE" causes query to return an unsigned long long containing the file system size (in octets).
2. AVAILABLE SPACE – an ID value of "AVAILABLE SPACE" causes the query operation to return an

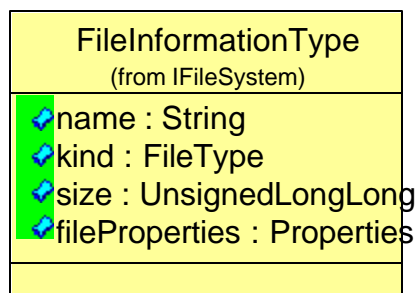
unsigned long long containing the available space on the file system (in octets). The query operation raises the UnknownFileSystemProperties exception when the given file system property is not recognized.

Types and Exceptions

- SIZE : constant String = "SIZE" Property name for file system's total size.
- AVAILABLE_SPACE : constant String := "AVAILABLE_SPACE" Property name for file system's available unused space .
- FileType : enumeration The FileType indicates the type of file entry. A file system can have PLAIN or DIRECTORY files and mounted file systems contained in a FileSystem.



- FileInformationType The FileInformationType indicates the information returned for a file. Not all the fields in the FileInformationType are applicable for all file systems. At a minimum, the FileSystem should support name, kind, and size information for a file. Examples of other file properties that can be specified for fileProperties are created time, modified time, and last access time.



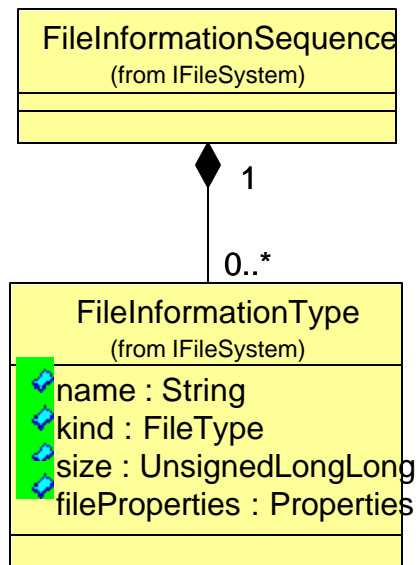
name: This field indicates the simple name of the file.

kind: This field indicates the type of the file entry.

size: This field indicates the size in octets.

- FileInformationSequence.

The FileInformationSequence type defines an unbounded sequence of FileInformationTypes.



- CREATED_TIME_ID : constant String = "CREATED_TIME".

The CREATED_TIME_ID is the identifier for the created time file property. A created time property indicates the time the file was created. The value for created time shall be unsigned long long and measured in seconds since 00:00:00 UTC, Jan. 1, 1970.

- MODIFIED_TIME_ID : constant String = "MODIFIED_TIME"

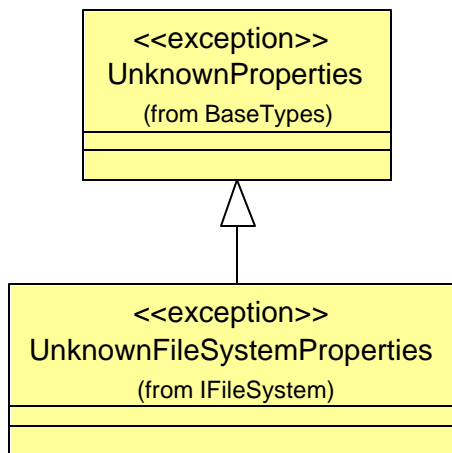
The MODIFIED_TIME_ID is the identifier for the modified time file property. The modified time property is the time the file data was last modified. The value for modified time property shall be unsigned long long and measured in seconds since 00:00:00 UTC, Jan. 1, 1970.

- LAST_ACCESS_TIME_ID : constant String = "LAST_ACCESS_TIME"

The LAST_ACCESS_TIME_ID is the identifier for the last access time file property. The last access time property is the time the file was last access (e.g. read). The value for last access time property shall be unsigned long long and measured in seconds since 00:00:00 UTC, Jan. 1, 1970.

- UnknownFileSystemProperties

The UnknownFileSystemProperties exception indicates a set of properties unknown by the component.



Mapping

N/A.

Constraints

No additional constraints

Semantics

No additional semantics

8.1.1.2 File

Description

The File class, as shown in Figure 69 — File Definition, provides the ability to read and write files residing within a distributed FileSystem. A file can be thought of conceptually as a sequence of octets with a current filePointer describing where the next read or write will occur. This filePointer points to the beginning of the file upon construction of the file object.

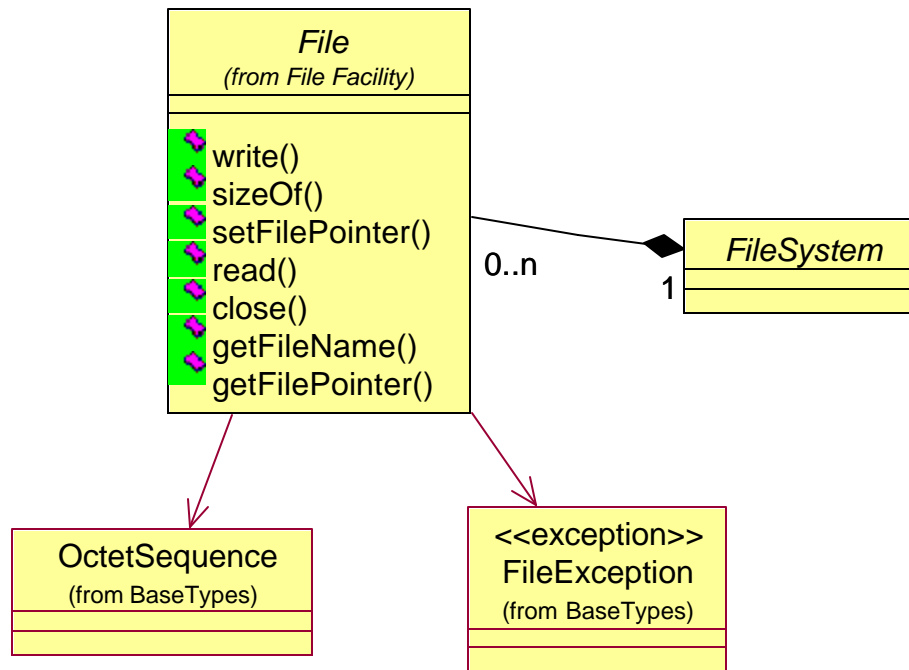


Figure 69 — File Definition

Attributes

- `<<readonly>>name: String` The file name.
- `<<readonly>>size: positive integer` The size of the file.
- `<<readonly>>data: OctetSequence` The data of the file.

Associations

- `fileSystem (1): FileSystem` A File is contained in a file system.

Operations

- `read(out data : OctetSequence, in length : UnsignedLong) : {raises= (IOException)}`
The read operation reads the number of octets specified by the input length parameter and advances the value of the `filePointer` attribute by the number of octets actually read. The read operation reads less than the number of octets specified in the input-length

parameter, when an end of file is encountered.

The read operation returns via the out Message parameter an OctetSequence that equals the number of octets actually read from the File. If the filePointer attribute value reflects the end of the File, the read operation returns a 0-length OctetSequence. The read operation raises the IOException when a read error occurs.

- write(in data : OctetSequence) : {raises (IOException)}

The write operation writes data to the file. If the write is successful, the write operation increments the filePointer attribute to reflect the number of octets written. If the write is unsuccessful, the filePointer attribute value maintains or is restored to its value prior to the write operation call. The write operation raises the IOException when a write error occurs.

- sizeOf(return UnsignedLong) : }raises (FileNotFoundException)}

The sizeOf operation returns the number of octets stored in the file. The sizeOf operation raises the FileNotFoundException when a file-related error occurs (e.g., file does not exist anymore).

- setFilePointer(in filePointer : UnsignedLong) : {raises (InvalidFilePointer, FileNotFoundException)}

The setFilePointer operation sets the filePointer attribute value to the input filePointer. The setFilePointer operation raises the FileNotFoundException when the file pointer for the referenced file cannot be set to the value of the input filePointer parameter. The setFilePointer operation raises the InvalidFilePointer exception when the value of the filePointer parameter exceeds the file size.

- getFileName(return String)

The getFilePointer operation returns the filename.

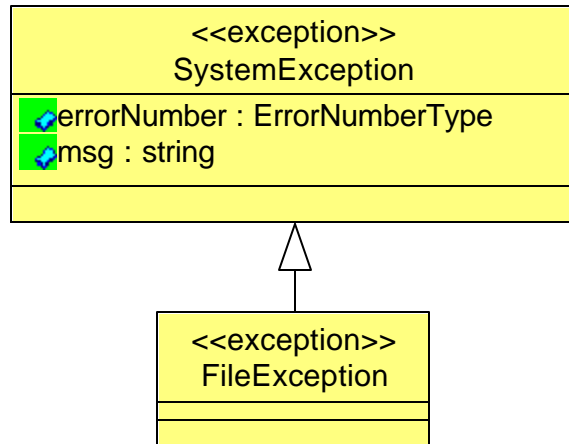
- getFilePointer(return String)

The getFilePointer operation returns the current position of the filePointer attribute.

Types and Exceptions

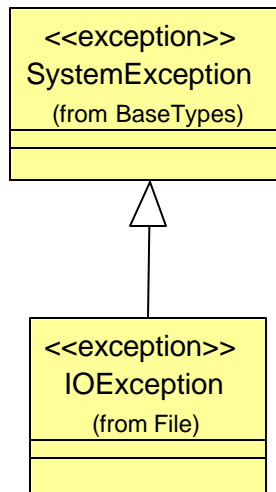
- <<exception>>FileNotFoundException

The FileNotFoundException, is a type of SystemException, that indicates a file-related error. The error number indicates an ErrorNumberType value (e.g., CF_EBADF, CF_EEXIST, CF_EISDIR, CF_EMFILE, CF_ENFILE, CF_ENOENT, CF_ENOSPC, CF_ENOTDIR, CF_ENOTEMPTY, CF_EROFS). The String msg attribute can provide information describing why the FileNotFoundException occurred.



- `IOException`

The `IOException` exception indicates an error occurred during a read or write operation to a File. The error number (e.g., `EINVAL`, `ENOSPC`, `EROFS` and message is component-dependent. The message provides additional information describing the reason for the error.



- `InvalidFilePointer`.

The `InvalidFilePointer` exception indicates the file pointer is out of range based upon the current file size

Mapping

N/A.

Constraints

Valid characters for a file name and file absolute pathname adhere to POSIX compliant file naming conventions. At a minimum a regular file name length of at least 40 characters is supported and at a minimum a combined path prefix and ending regular file name length of at least 1024 characters is supported.

The filePointer is set to the beginning of the file when a File is opened for read only or created for the first time. When a File already exists and is opened for write, the filePointer is set at the end of the File.

Semantics

No additional semantics.

8.1.1.3 FileManager

Description

The FileManager class, as shown in Figure 70 — FileManager Definition, realizes the IFileSystem interface and extends that interface by adding mount and unmount operations. This allows multiple, distributed FileSystems to be accessed through a FileManager. The FileManager appears to be a single FileSystem although the actual file storage may span multiple physical file systems. This is called a federated file system. A federated file system is created using the mount and unmounts operations.

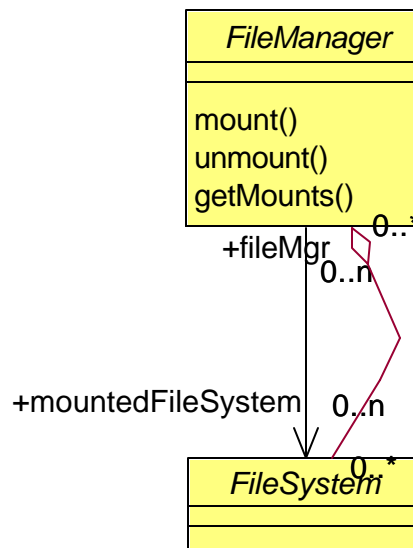


Figure 70 — FileManager Definition

Attributes

No additional attributes.

Associations

- mountedFileSystem (0..n):FileSusystem MountedFileSystems that are associated with a FileManager.

Operations

- mount(in string mountPoint, in FileSystem The FileManager supports the notion of a

file_System) : {raises(InvalidFileName,
InvalidFileSystem, MountPointAlreadyExists)}

federated file system. To create a federated file system, the mount operation associated a FileSystem with a mount point (e.g., a directory name). The mount operation associates the specified FileSystem with the given mountPoint. The mount operation raises the InvalidFileName exception when the input file name is invalid.

The mount operation raises the MountPointAlreadyExists exception when the mountPoint already exists in the file manager.

The mount operation raises the InvalidFileSystem exception when the input FileSystem is a null object reference

- unmount(in string mountPoint) : {raises(NonExistentMount)}

The unmount operation removes a mounted FileSystem from the FileManager whose mounted name matches the input mountPoint name. The unmount operation raises the NonExistentMount exception when the mountPoint does not exist.

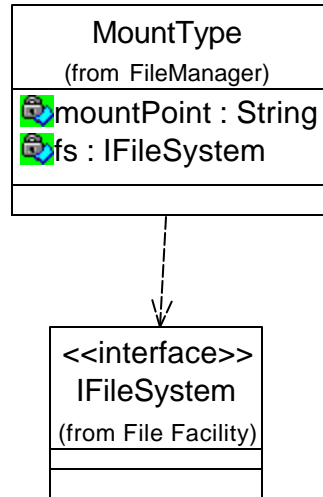
- getMounts(return MountSequence)

The getMounts operation returns a sequence of Mount structures that describe the mounted FileSystems.

Types and Exceptions

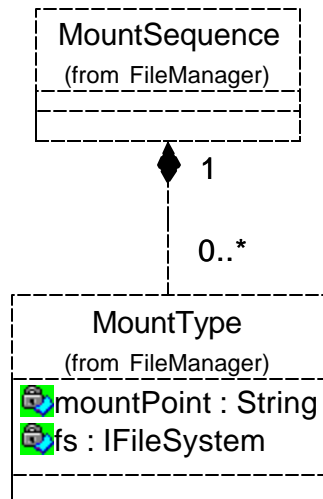
- MountType

The MountType structure shall identify the FileSystems mounted within the FileManager.



- MountSequence

The MountSequence is an unbounded sequence of MountTypes.



- NonExistentMount
- MountPointAlreadyExists

The NonExistentMount exception indicates a mount point does not exist within the FileManager.

The MountPointAlreadyExists exception indicates the mount point is already in use in the FileManager.

- InvalidFileSystem The InvalidFileSystem exception indicates the FileSystem is a null (nil) object reference.

Mapping

N/A.

Constraints

The FileManager's realized IFileSystem operations behavior implements the requirements of the FileSystem operations against the mounted file systems. The FileSystem operations ensure that the filename/directory arguments given are absolute pathnames relative to a mounted FileSystem. The FileManager's operations removes the FileSystem mounted name from the input fileName before passing the fileName to an operation on a mounted FileSystem.

The FileManager uses the mounted FileSystem for FileSystem operations based upon the mounted FileSystem name that exactly matches the input fileName to the lowest matching subdirectory.

The FileManager propagates exceptions raised by a mounted FileSystem.

Semantics

The system may support multiple FileSystem implementations. Some FileSystems will correspond directly to a physical file system within the system. The FileManager supports a federated, or distributed, file system that may span multiple FileSystem components. From the client perspective, the FileManager may be used just like any other FileSystem component since the FileManager realizes all the IFileSystem operations.

Based upon the pathname of a directory or file and the set of mounted FileSystems, the FileManager will delegate the FileSystem operations to the appropriate FileSystem. For example, if a FileSystem is mounted at /ppc2, an open operation for a file called /ppc2/profile.xml would be delegated to the mounted FileSystem. The mounted FileSystem will be given the filename relative to it. In this example the FileSystem's open operation would receive /profile.xml as the fileName argument.

Another example of this concept can be shown using the copy operation. When a client invokes the copy operation, the FileManager will delegate operations to the appropriate FileSystems (based upon supplied pathnames) thereby allowing copy of files between FileSystems.

8.1.1.4 FileSystem

Description

A FileSystem class, as shown in Figure 71 — FileSystem Definition, realizes the IFileSystem interface and be associated with a FileManager and consist of many Files.

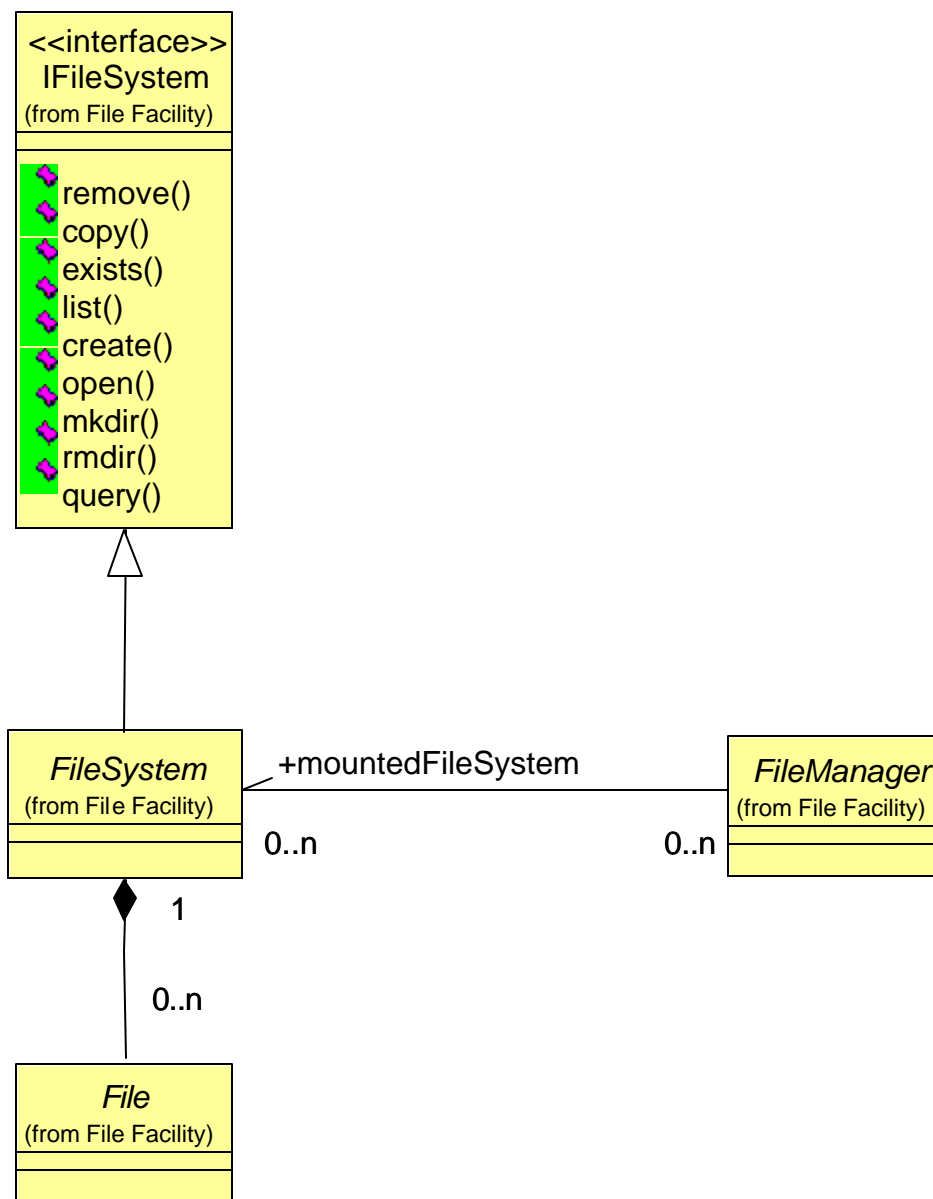


Figure 71 — FileSystem Definition

Attributes

No additional attributes.

Associations

- fileManager (0..n): FileManager A file system can be associated with many FileManagers

- file (0..n): File A file system can consist of many files.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2 Common Layer Facilities

This section defines the Common Layer Facilities, which provide the facilities that can be used by any waveform layer. Most of the packages under Common layer Facilities package are named as **Building Blocks**. Building blocks define facilities that can be used commonly across any waveform layers. Figure 72 shows the relationships among the packages contained in the Common Layer Facilities part of the PIM. These packages are given as follows:

- Quality of Service BB – defines the quality of service related facilities.
- Flow Control BB – provides means to control communication flow so that a sender does not transmit more packets than a receiver can process.
- Measurement BB – specifies facilities to set up waveform related measurement parameters and schedule the measurement.
- Error Control BB -- allows the Receiver to tell the Sender about frames damaged or lost during transmission, and coordinates the re-transmission of those frames by the Sender.
- Transmission BB – provides a simple mechanism for sending data and control information.
- PDU BB – defines the Protocol data Unit (PDU) building block concept that can be used in connectionless communication among radio sets as well as inter-component communication within a radio.
- Stream BB – defines the stream building block concept that can be used in connection oriented communication among radio sets as well as inter-component communication within a radio.

- SAP BB – defines the Service Access Point (SAP) building block concept that is a specialized port connector for a waveform component.
- Security Facilities – specifies the security related facilities such as transec, infosec, crypto keys, policies etc.

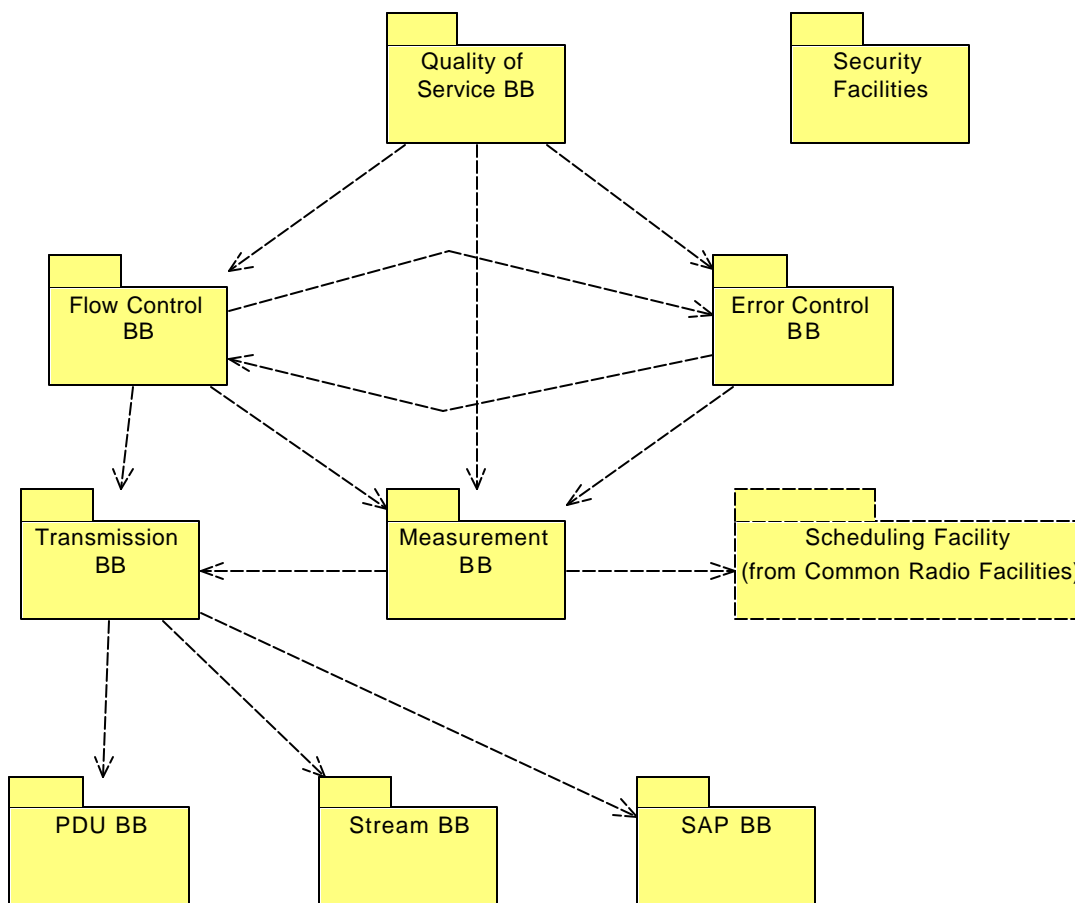


Figure 72 - Common Layer Facilities Overview

8.2.1 Quality of Service Building Block

Quality of service (QoS) building block elements define the facilities that can be used to control quality of service related parameters. The quality of service parameters that can be set up are given in the DLPI specification document (REFERENCE HERE). Figure 73 shows an overview of Quality of Service facilities.

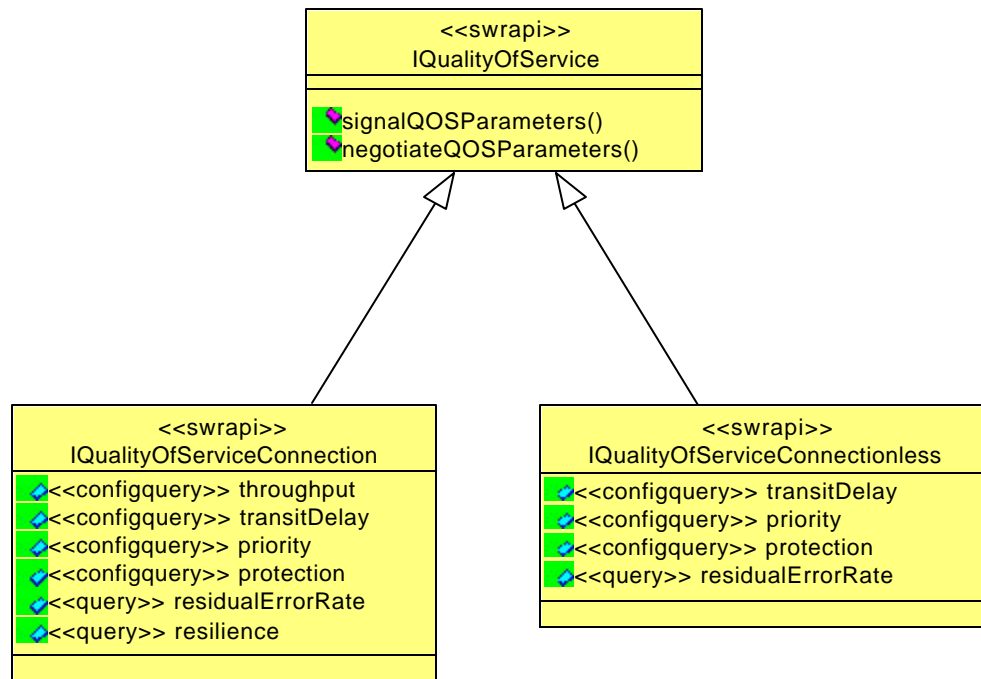


Figure 73 - Quality of Service Building Block Overview

8.2.1.1 IQualityOfService

Description

IQualityOfService, as shown in Figure 74, is the main interface that is used to control the quality of service parameters of a waveform. The parameters that are to be controlled depend on the nature of the established communication link. (connection oriented, connectionless). This interface provides the capabilities of signalling and negotiating QoS parameters with waveform components.

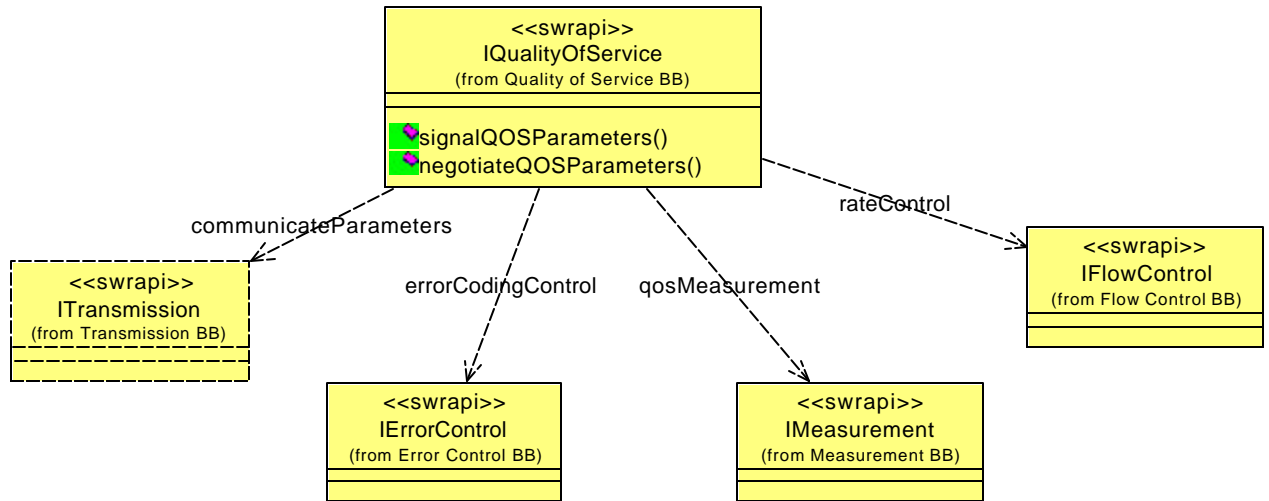


Figure 74 – IQualityOfService Definition

Attributes

No additional attributes.

Associations

No additional associations.

Operations

- signalQOSParameters ()**

This operation signals the quality of service parameters to the requester. AssemblyController, another waveform component within the same radio set, or the peer receiving radio set may acquire the QoS parameters. IQualityOfService interface depends on the ITransmission interface to communicate those parameters.
- negotiateQOSParameters()**

This operation provides a generic interface to negotiate the quality of service parameters with the peer receiver/transmitter, usually through a horizontal communication link. This also includes interfacing with error control, flow control and measurement related components within the waveform, to setup their parameters that will meet the quality of service requirements.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2.1.2 IQualityOfServiceConnection

Description

IQualityOfServiceConnection specializes the IQualityOfService interface to provide QoS attributes for connection oriented communication establishment. Figure 75 shows the definition of IQualityOfServiceConnection.

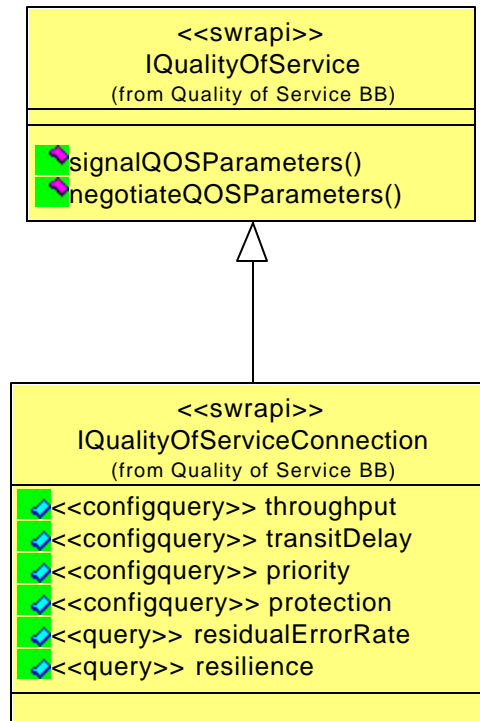


Figure 75 – IQualityOfServiceConnection Definition

Attributes

- **<<configquery>> throughput**

Throughput is a connection-mode QoS parameter that has end-to-end significance. It is defined as the total number of SDU bits successfully transferred divided by the greater of both:

- the time between the first and last data request in a sequence
- the time between the first and last data indication in the

sequence.

Throughput is only meaningful for a sequence of complete DLSUs.

Throughput is specified and negotiated for the transmit and receive directions independently at connection establishment. The throughput specification defines the target and minimum acceptable values for a connection. Each specification is an average rate.

- << configquery >> transitDelay

This attribute indicates the elapsed time between a data request and the corresponding receipt of data. The elapsed time is only computed for SDUs successfully transferred. Transit delay is negotiated on an end-to-end basis during connection establishment. For each connection, transit delay is negotiated for the transmit and receive directions separately by specifying the target value and maximum acceptable value.

The transit delay for an individual SDU may be increased if the receiving user flow controls the interface. The average and maximum transit delay values exclude any user flow control of the interface.

- << configquery >> priority

Priority is negotiated locally between each user and the provider in connection-mode service. The specification of priority is concerned with the relationship between connections.

The parameter specifies the relative importance of a connection with respect to:

- the order in which connections are to have their QOS degraded, if necessary
- the order in which connections are to be released to recover resources, if necessary.

Each user negotiates a particular priority value with the provider during connection establishment. The value is specified by a minimum and a maximum within a given range. This parameter only has meaning in the context of some management entity or structure able to judge relative importance. The priority has local significance only.

- << configquery >> protection

Protection is negotiated locally between each user and the provider in connection mode. Protection is the extent to which a provider attempts to prevent unauthorized monitoring or manipulation of user-originated information. Protection is specified by a minimum and maximum protection option within a range of possible protection options.

Provider protects against modification, replay, addition, or deletion of user data. Each user negotiates a particular value with the provider during connection establishment. The value is specified by a minimum and a maximum within a given range. Protection has local significance only.

- <<query>> residualErrorRate

Residual error rate (RER) is the ratio of total incorrect, lost and duplicated SDUs to the total SDUs transferred between radio sets during a period of time. This property cannot be

- <<query>> resilience

configured and is used for QoS monitoring purposes only. Resilience is meaningful in connection mode only, and represents the probability of either: provider-initiated disconnects or provider-initiated resets during a time interval of 10,000 seconds on a connection.

Resilience is not a negotiated QOS parameter. It is set by an administrative mechanism, which is informed of the value by network management.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2.1.3 IQualityOfServiceConnectionless

Description

IQualityOfServiceConnectionless specializes the IQualityOfService interface to provide QoS attributes for connectionless communication establishment. Figure 76 shows the IQualityOfServiceConnectionless definition.

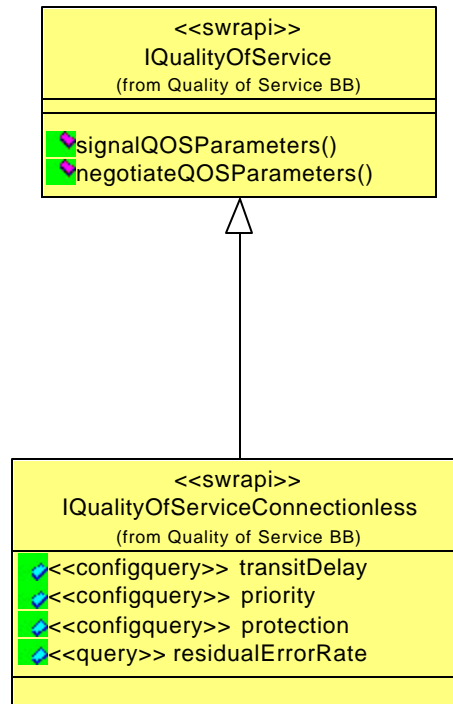


Figure 76 – IQualityOfServiceConnectionless Definition

Attributes

- << configquery >> transitDelay**

This attribute indicates the elapsed time between a data request and the corresponding receipt of data. The elapsed time is only computed for SDUs successfully transferred. The transmitting radio set selects a particular value within the supported range, and the value may be changed for each SDU submitted for connectionless transmission.

The transit delay for an individual SDU may be increased if the receiving user flow controls the interface. The average and maximum transit delay values exclude any user flow control of the interface.
- << configquery >> priority**

Priority is determined locally for each user in connectionless mode service. The specification of priority is concerned with the relationship between connectionless data transfer requests.

This attribute specifies the relative importance of data units with respect to gaining use of shared resources. The transmitting radio set selects a particular priority value within the supported range, and the value may be changed for each SDU submitted for transmission.

This parameter only has meaning in the context of some management entity or structure able to judge relative importance. The priority has local significance only.

- << configquery >> protection
Protection is the extent to which a provider attempts to prevent unauthorized monitoring or manipulation of user-originated information. Protection is specified by a minimum and maximum protection option within a range of possible protection options. The transmitting radio set selects a particular value within the supported range, and the value may be changed for each SDU submitted for transmission.

Provider protects against modification, replay, addition, or deletion of user data. Protection has local significance only.
- <<query>> residualErrorRate
Residual error rate (RER) is the ratio of total incorrect, lost and duplicated SDUs to the total SDUs transferred between radio sets during a period of time. This property cannot be configured and is used for QoS monitoring purposes only.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2.2 Flow Control Building Block

Flow Control building block elements define the facilities that relate to flow control of data transmission and reception. This BB controls packet flow so that a provider does not transmit more packets than a receiver can process. Flow control is necessary because users and providers are often unmatched in capacity and processing power. The goal of flow-control mechanisms is to prevent dropped packets that must be retransmitted. Figure 77 shows an overview of Flow control Building block facilities. Flow control can be implemented between peer layers for both connection oriented and connectionless communication modes (horizontal communication), or at the service boundary between different layers within the same SDR (vertical communication).

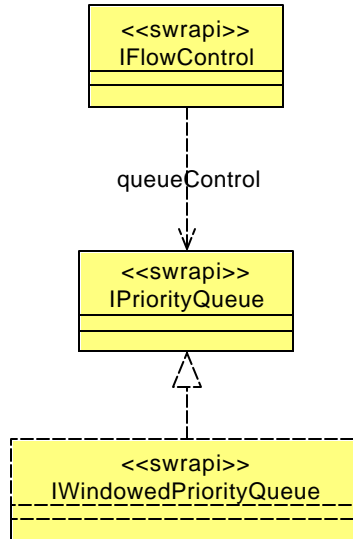


Figure 77 - Flow Control Building Block Overview

8.2.2.1 IFlowControl

Description

IFlowControl provides an interface to control flow control related arguments. The interface provides the capabilities of enabling and disabling flow control signaling, sending flow control signals, and managing priority queues. Figure 78 shows the definition of IFlowControl.

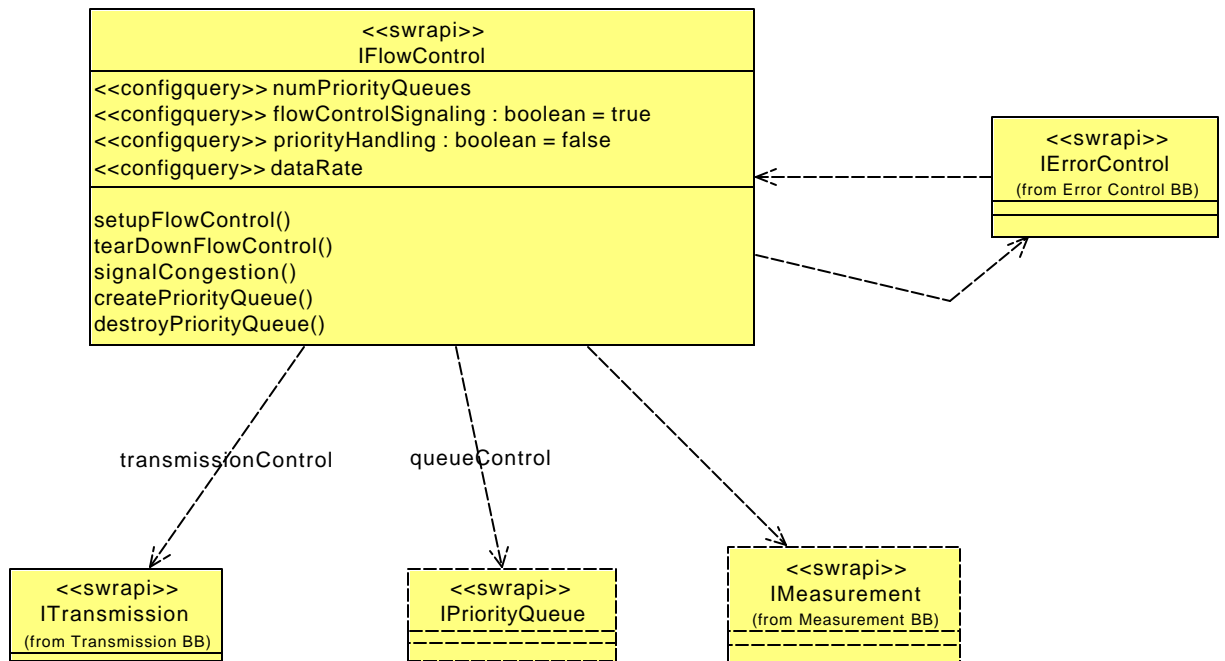


Figure 78 - IFlowControl Definition

Attributes

- <<configquery>> numPriorityQueues: Number of priority queues that the flow controller component is managing.
- <<configquery>> flowControlSignalling: Boolean: This attribute indicates whether flow control signalling is currently enabled or not.
- <<configquery>> priorityHandling: Boolean: This attribute indicates whether priority queue handling is currently enabled or not.
- <<configquery>> dataRate: Target data rate.

Associations

No additional associations.

Operations

- setupFlowControl () : This operation sends a flow control request to the remote radio set in case of a horizontal communication scenario, or to another waveform component in case of a vertical communication scenario. It also sets up flow control related parameters.

- `tearDownFlowControl ()` `tearDownFlowControl` operation terminates an existing flow control between the user and provider. Termination signal is also propagated to the remote sender/receiver.
- `signalCongestion ()` This operation signals a congestion (the user can not handle incoming packets and they are being dropped).
- `createPriorityQueue ()` This operation creates a priority queue bound to the flow control manager.
- `destroyPriorityQueue ()` This operation destroys a previously instantiated priority queue.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2.2.2 IPriorityQueue

Description

IPriorityQueue provides a mechanism to control priority queues. Priority queues are used by the flow control mechanism in order to direct incoming PDUs with different priority tags to corresponding queues. Queues with higher priority get easier access to system resources, while lower priority queue elements wait until higher priority ones are processed. The interface provides the capabilities signaling high and low watermarks, as well as acknowledging transmissions. Figure 79 shows the definition of IPriorityQueue.

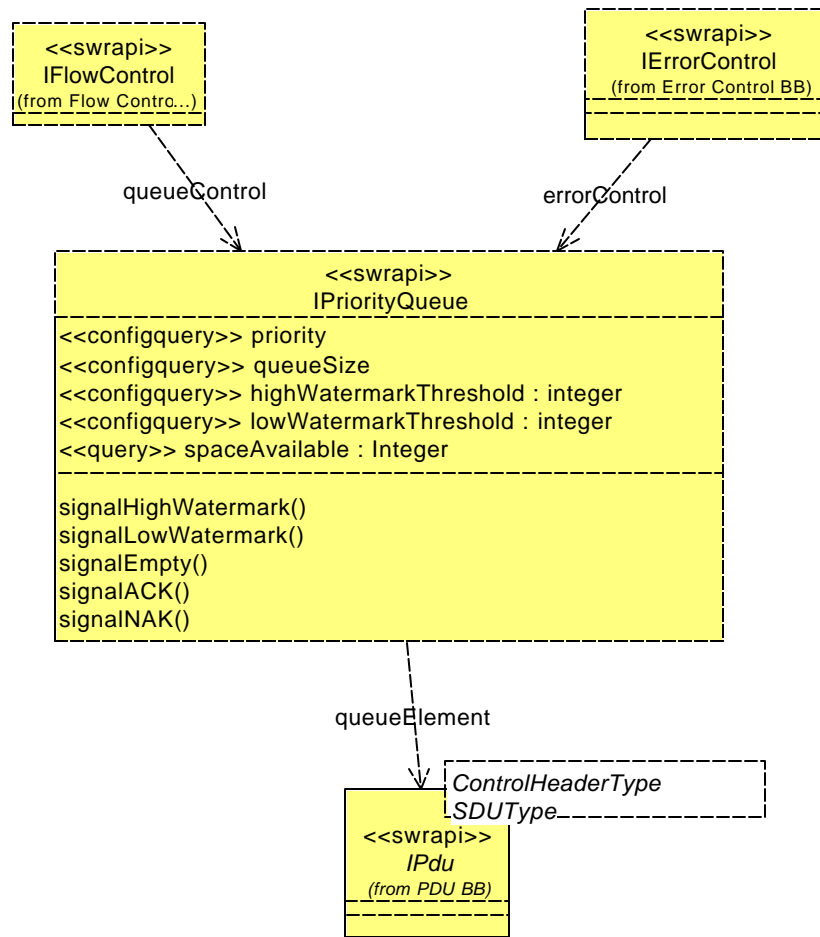


Figure 79 - IPriorityQueue Definition

Attributes

- <<configquery>> priority Priority level of the queue.
- <<configquery>> queueSize Size of the queue
- <<configquery>> highWatermarkThreshold High watermak threshold shows the point in the queue where a dangerous occupancy is reached in the buffer and probability of dropping PDUs has increased. The value of this attribute is aplication dependent and usually is determined by the flow controller of the QoS controller after negotiating with the remote entity.
- <<configquery>> lowWatermarkThreshold Low watermak threshold shows the point in the queue where a dangerous occupancy is reached in the buffer and probability of dropping PDUs has increased, although is less than the highWatermarkThreshold point. The value of this attribute is aplication dependent and

usually is determined by the flow controller of the QoS controller after negotiating with the remote entity.

- <<query>> spaceAvailable The size of available buffer space in the priority queue.

Associations

No additional associations.

Operations

- signalHighWatermark () The signalHighWaterThreshold operation is used to alert the peer entity that high watermark threshold has reached.
- signalLowWatermark () The signalLowWaterThreshold operation is used to alert the peer entity that low watermark threshold has reached.
- signalEmpty () The signalEmpty operation signals that the buffer is empty and ready to receive data.
- signalACK () The signalACK operation is used to acknowledge successful reception of a PDU sent by the provider.
- signalNAK () The signalNAK operation is used to acknowledge unsuccessful reception of a PDU sent by the provider.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2.2.3 IWindowedPriorityQueue

Description

IWindowedPriorityQueue specializes the IPriorityQueue interface in order to provide a mechanism for windowed acknowledgement in a priority queue. Figure 80 shows the definition of IWindowedPriorityQueue.

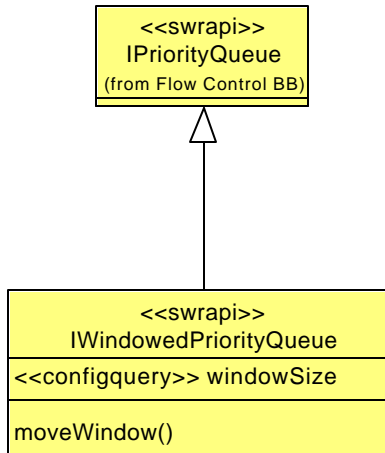


Figure 80 - IWindowedPriorityQueue Definition

Attributes

- **<<configquery>> windowSize** Siz of the window. This attribute can be changed during initilization and/or after the communication has been established.

Associations

No additional associations.

Operations

- **moveWindow ()** The moveWindow operation provides a mechanism for moving the acknowledgement window.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2.3 Measurement Building Block

Measurement building block defines the facilities that relate to performing a measurement as requested by a component that has controller functionality over the component that implements the measurement building block. Any waveform layer component can be scheduled to perform a measurement, such as traffic volume measurement, bit error rate measurement, voice silence duration measurement, link quality measurement, etc. These measurements plans are communicated to the component through a class of type MeasurementPlanType.

8.2.3.1 IMeasurement

Description

The IMeasurement interface provides a mechanism to set up a measurement plan, manually perform, send or request measurements from components. Figure 81 shows the definition of IMeasurement.

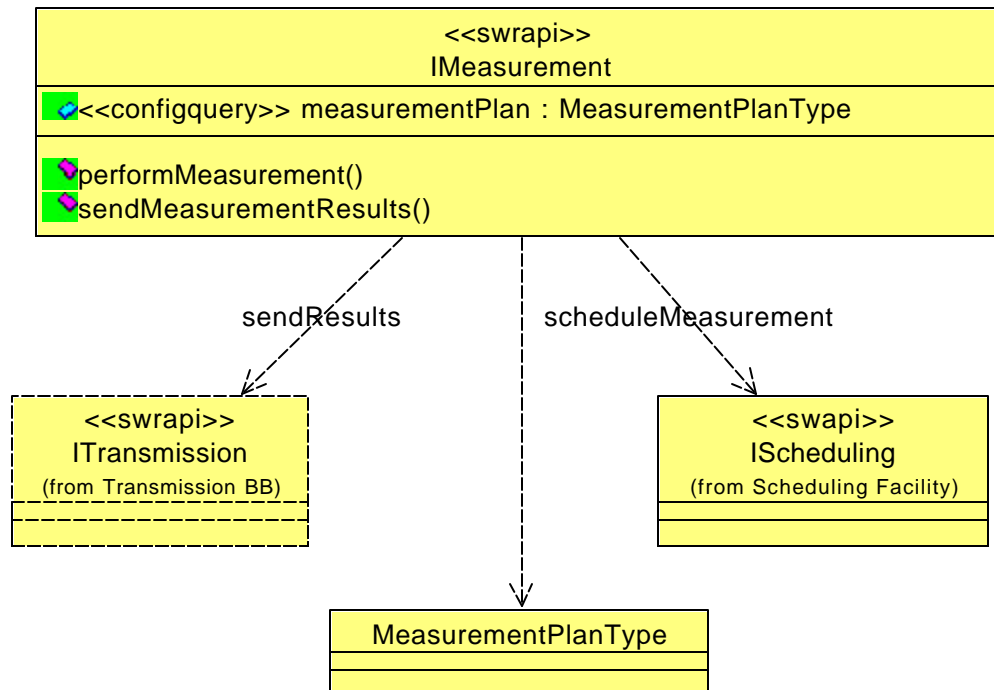


Figure 81 - IMeasurement Definition

Attributes

- <<configquery>>measurementPlan: MeasurementPlanType This attribute defines the complete measurement plan, including the parameters to be measured, scheduling information of the measurement, and the requester component for the measurement.

Associations




No additional associations.

Operations

- `performMeasurement()` The `performMeasurement` operation provides a mechanism for starting the execution of a measurement. A measurement can be executed manually by a measurement requester component using this operation, or the measurement may execute automatically if scheduling information is provided when creating the measurement plan.
- `sendMeasurementResults()` This operation provides a mechanism to send the measurement results to the measurement requester designated in the `measurementPlan` attribute.

Types and Exceptions

- `MeasurementPlanType` `MeasurementPlanType` defines the attributes that are required to define a measurement plan.
 - `measurementAttributesList` is a list of attributes that will be measured.
 - `measurementSchedule` is an optional parameter, and defines the scheduling of the measurement plan. This attribute is only defined if the measurement is to be executed automatically at a given time. For manual execution of the measurement, this attribute is not needed.
 - `measurementRequester` defines the component port to which the measurement results will be sent to.

MeasurementPlanType
 <<configquery>> measurementAttributesList
 <<configquery>> measurementSchedule [0..1]
 <<configquery>> measurementRequester

Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2.4 Error Control Building Block

Error control building block allows the User to tell the Provider about data units damaged or lost during transmission, and coordinates the re-transmission of those data units by the Provider. Since the flow control building block provides the User's acknowledgement (ACK) of correctly-received data units, it is closely linked to error control. The error control parameters are communicated to the component through a class of type ErrorControlType.

8.2.4.1 IErrorControl

Description

The IErrorControl interface provides a mechanism to establish error control related facilities at both the Provider and User sides of communication. The error control mechanism can be used to change the error control parameters that affect any layer of the waveform. Figure 82 shows the definition of IErrorControl.

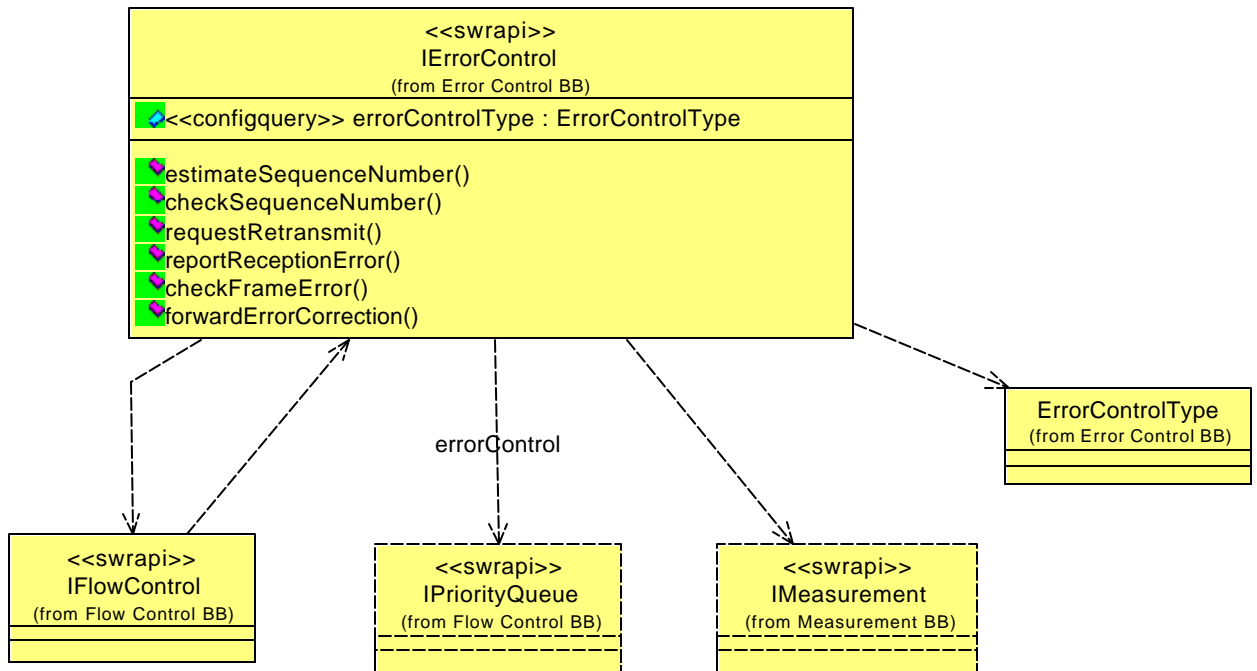


Figure 82 - IErrorControl Definition

Attributes

- <<configquery>> errorControlType This attribute defines which error control attributes are currently enabled to execute for the existing

errorControlType

communication link.

Associations





No additional associations.

Operations

- estimateSequenceNumber() The estimatePduCounter operation provides a mechanism to estimate the sequence number for the next Pdu that is expected.
- checkSequenceNumber() This operation provides a mechanism to check the sequence number of the received PDU against what has been estimated.
- requestRetransmit() The requestRetransmission operation allows the user to request a retransmission of a recently arrived PDU which contained a frame error.
- reportReceptionError() This operation provides a mechanism to report an error at the reception to the provider port. It is different from the requestRetransmit operation in the sense that it only reports the error and does not request the data to be retransmitted. This operation is more suitable for radio links that has low latency requirements. (like video stream)
- checkframeError() The checkFrameError operation provides a mechanism to check the incoming data unit against any errors. This operation may be implemented by cyclic redundancy check (CRC) algorithm, or any other algorithm that introduces some redundancy to the SDU and checks for authenticity at the receiver side.
- forwardErrorCorrection() This operation allows the user to correct some of the errors that occurred during reception. Forward error correction works without any feedback mechanism or reporting back to the original sender. The channel coding type of redundancy introduced to the SDU allows the receiver to correct some of the bit errors introduced by the physical channel.

Types and Exceptions

- ErrorControlType The ErrorControlType defines the error control attributes that can be enabled as a part of the error control facility.

ErrorControlType (from Error Control BB)
 <<configquery>> errorControl : Boolean
 <<configquery>> slidingWindowARQ : Boolean
 <<configquery>> ARQStopWait : Boolean
 <<configquery>> forwardErrorCorrection : Boolean

Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2.5 Transmission Building Block

Transmission building block elements define the facilities that relate to data transmission and reception for both horizontal and vertical communication. The transmission BB can be implemented between peer layers for both connection oriented and connectionless communication modes (horizontal communication), or at the service boundary between different layers within the same SDR (vertical communication).

8.2.5.1 ITransmission

Description

This interface provides a mechanism for transmitting data using different modes of communication depending on the communication link that has already been established. The ITransmission interface depends on several interfaces for transmitting data. For both vertical and horizontal communication with connectionless mode of operation, ITransmission interface depends on the IPdu interface for generating the PDUs. Similarly, for both vertical and horizontal communication with connection oriented mode of operation, ITransmission interface depends on the IStream interface for generating the streams. Those streams can both be realized as the stream implementation of the middleware being used (eg. CORBA streams), or as special streams that govern the link establishment across remote radio sets. This includes handshaking between radios before initializing the stream, setting up a circuit switched transmission path, in case repeaters are to be used etc. The transmission building block also depends on the IServiceAccessPoint interface for vertical data transmission across the waveform layer boundaries. Figure 83 shows the definition of ITransmission.

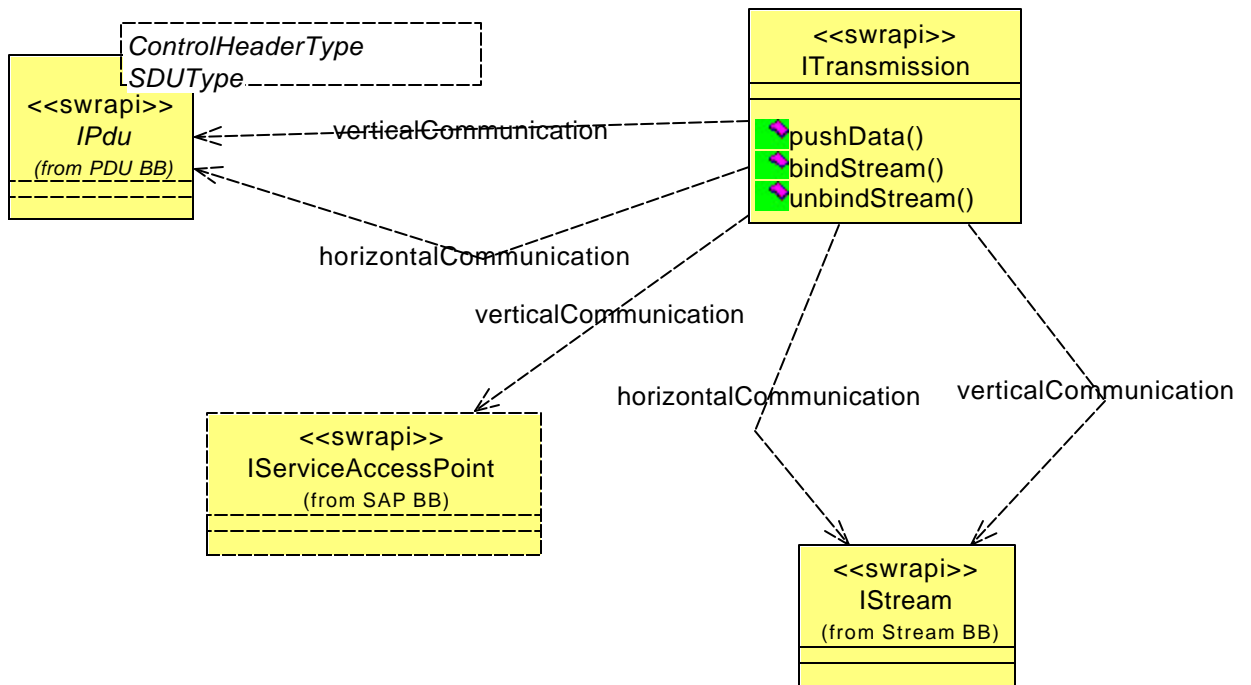


Figure 83 - ITransmission Definition

Attributes

No additional attributes

Associations

No additional associations.

Operations

- pushData()**

The pushdata operation provides a mechanism to send PDU's to the user. The user can either be another component within the same radio set, or a waveform layer component within another radio set.
- bindStream()**

This operation provides a mechanism to bind a pre-established stream to a data source, and allow data transmission through that stream.
- unbindStream()**

The unbindStream operation allows a controller component to detach the data source from the stream. This operation does not release the established stream. In order to release the stream, one has to use the IStream interface.

Types and Exceptions

No additional types or exceptions.

Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2.6 Packet Data Unit Building Block

This building block defines the Packet Data Unit (PDU) concept that can be used as the smallest data unit element in any waveform layer. PDU's are data elements that are used to store protocol data, and query certain attributes that relate to the usage of PDUs in the waveform protocol. The PDU concept has been taken from the SCA API supplement, which defines the Packet BB. Packet terminology is very specific to Logical Link Layer, so in order to make this concept applicable to any waveform layer that carries data in small units, packet has been renamed as a Protocol Data Unit.

8.2.6.1 IPdu

Description

The IPdu interface is a parametrized class that can be implemented using different header and service data unit (SDU) types. Those types are defined as dependencies of the IPdu interface. This interface can be used for both vertical and horizontal communication links. Figure 84 shows the definition of IPdu interface.

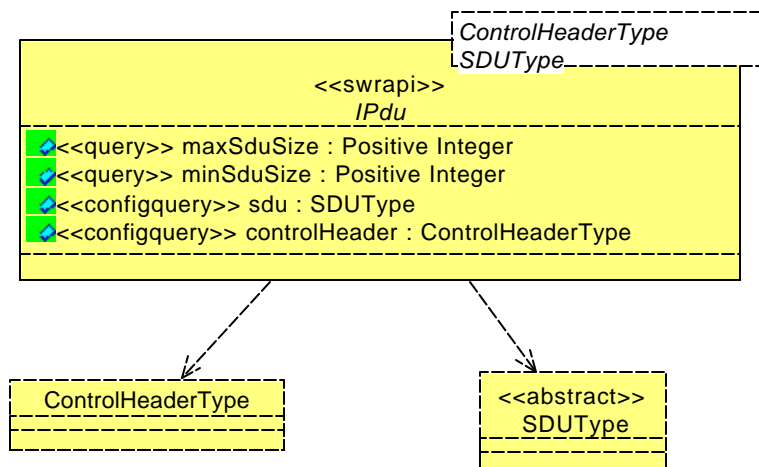


Figure 84 - IPdu Definition

Attributes

- <<query>> maxSduSize The maxSduSize attribute specifies the maximum payload size that can be stored in a single PDU.
- <<query>> minSduSize This attribute specifies the minimum payload size that can be stored in a single PDU.
- <<configquery>> sdu : SDUType This configurable attribute is the entire SDU that has been received from a higher layer in case of transmitting data, and has been decoded from the lower layer in case of receiving data.
- <<configquery>> controlHeader: ControlHeaderType The controlHeader attribute contains the header information of the PDU, which includes any in-band signalling that relates to waveform command and control, as well as flow control, error control related information that is piggy-backed to the SDU.

Associations

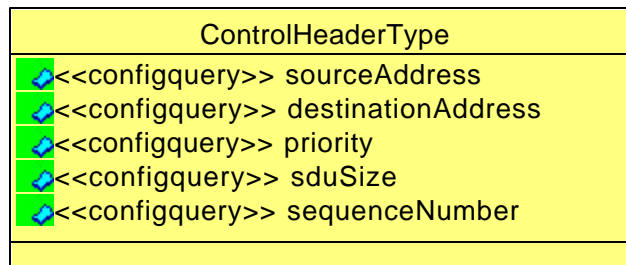
No additional associations.

Operations

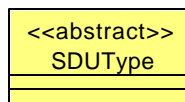
No additional operations.

Types and Exceptions

- ControlHeaderType ControlHeaderType class defines the base class of information that can be included the header of a PDU. This list can be extended to contain any in-band waveform command and control messages incorporated into it.



- <<abstract>> SDUType The SDUType class is stereotyped as an abstract class, and can have any type of information (octet sequence, ANSI encoded, bit stream etc) that relates to the upper waveform layers.



Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2.7 Stream Building Block

The stream building block defines interfaces to establish and control data streams. These data streams can be used for various purposes and may have different implementations. They can be used for in-band signalling such as transmitting data along with some waveform command and control signals embedded in the stream. They can be used for out-of-band streaming which may be implemented as non-standard CORBA streams. The IStream interface has IHorizontalStream and IVerticalStream specializations that realize horizontal and vertical communication requirements. These components are closely tied to horizontal and vertical association stereotypes as defined in the UML profile. (NEEDS CLARIFICATION).

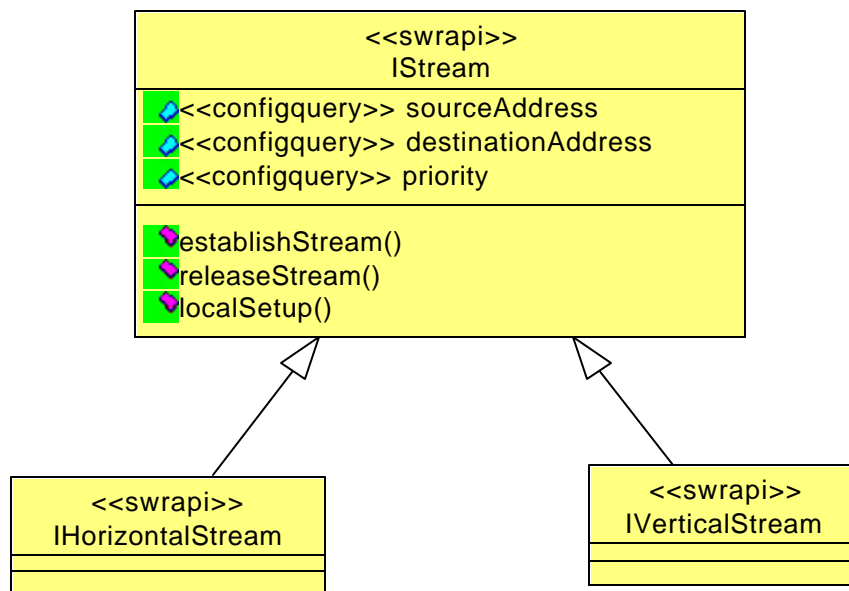


Figure 85 - Stream Building Block Overview

8.2.7.1 IStream

Description

The IStream interface, as shown in Figure 86, defines stream communication capabilities. The interface provides capabilities of establishing and releasing a stream, as well as setting up local parameters at the initialization time of stream.

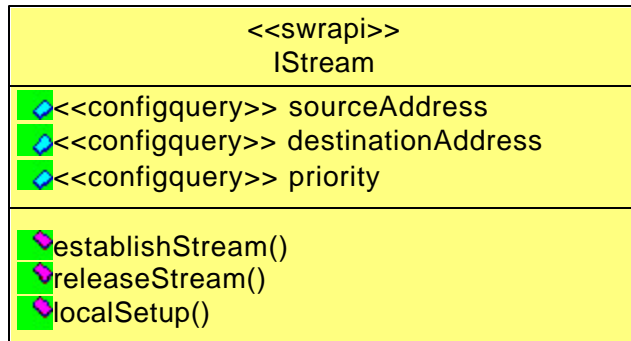


Figure 86 - IStream Definition

Attributes

- <<configquery>> sourceAddress
 - <<configquery>> destinationAddress
 - <<configquery>> priority

Source address attribute designates the stream provider. This may refer to a port definition, or in the horizontal communication scenario, an address provided by a high layer protocol such as IP.

Destination address attribute designates the stream user. This may refer to a port definition, or in the horizontal communication scenario, an address provided by a high layer protocol such as IP.

Priority attribute specifies the priority level of the established stream. High priority streams are allocated more resources, relatively less latency and high quality of service operation is expected from a high priority stream implementation.

Associations

No additional associations.

Operations

- establishStream()
 - releaseStream()
 - localSetup()

The establishStream operation is used to establish a data stream by handshaking the stream parameters with the remote component.

The releaseStream operation is used to release the currently established stream. This operation can be a simple teardown of the stream, or a connection termination with acknowledging the peer end, depending on the implementation.

This operation sets up the local parameters required for setting up a communication stream. These parameters are discussed in the quality of Service building block.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2.7.2 IHorizontalStream

Description

The IHorizontalStream interface, as shown in Figure 87, specializes the IStream interface but it does not provide any extra operations or attributes. This interface is to be implemented by a component that has to establish a horizontal communication link. The interface provides capabilities of establishing and releasing a stream, as well as setting up local parameters at the initialization time of stream.

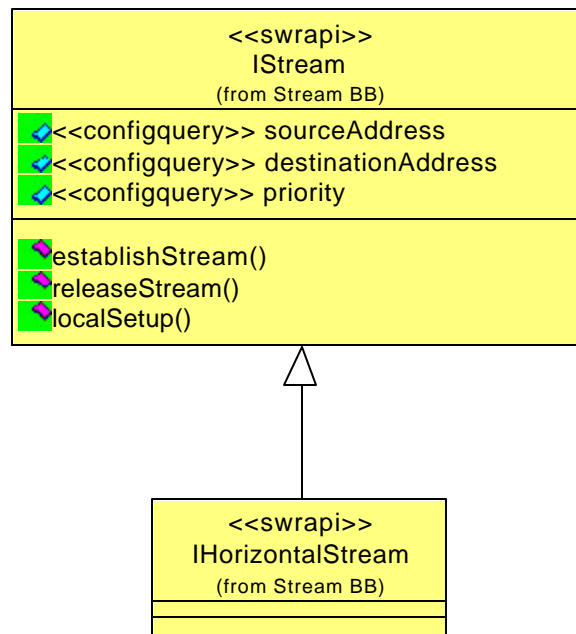


Figure 87 – IHorizontalStream Definition

Attributes

No additional attributes.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2.7.3 IVerticalStream

Description

The IVerticalStream interface, as shown in Figure 88, specializes the IStream interface but it does not provide any extra operations or attributes. This interface is to be implemented by a component that has to establish a vertical communication link (i.e. within the same radio set, among different waveform layers). The interface provides capabilities of establishing and releasing a stream, as well as setting up local parameters at the initialization time of stream.

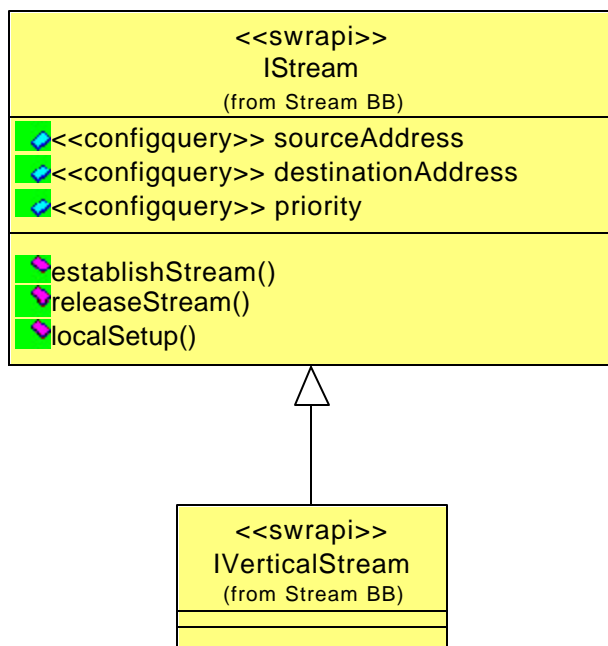


Figure 88 – IVerticalStream Definition

Attributes

No additional attributes.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.2.8 Service Access Point Building Block

Service Access Point (SAP) building block provides a mechanism to communicate data between different waveform layers, within the same radio set. Usually at the local management stage, a stream is bound to an SAP for the connection oriented mode, and frames are addressed to the SAP for the connectionless mode of communication. SAPs are used only in vertical communication scenarios.

8.2.8.1 IServiceAccessPoint

Description

The IServiceAccessPoint, as shown in Figure 89, defines service access point capabilities for all waveform layers. SAP concept is defined by the OSI as a point at which a designated service may be obtained and it provides a standard way of communicating with a waveform layer. The interface provides the capabilities of resetting and initializing a SAP, as well as changing SAP related attributes.

I PROPOSE DROPPING THE SAP CONCEPT FROM THE PIM AND PROVIDING IT AS A SPECIALIZATION OF THE CORBA PORT CONCEPT, WITH ADDED BEHAVIOR FOR ONLY INTERFACING WITH WAVEFORM LAYERS.

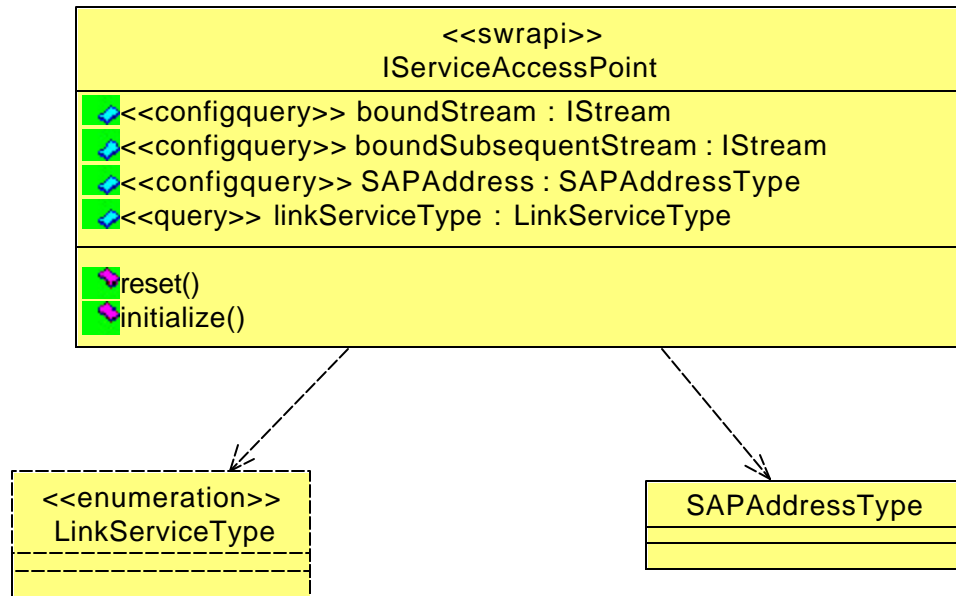


Figure 89 – IServiceAccessPoint Definition

Attributes

- <<configquery>> boundStream Description of the attribute.
- <<stereotype>> attribute2: Description of the attribute.
AttributeClass [cardinality]

For Interfaces and software components, use the following stereotypes:

query - a property that can be queried by the PropertySet interface

config - a property that can be configured by the PropertySet interface

configquery - a property that can be configured and queried by the PropertySet interface.

readonly - a property that is only retrievable only

writeonly - property that is writeable only

readwrite - property that is both readable and writeable.

These properties may generate set and get operations.

For communication equipment profile use the following stereotypes: characteristicproperty, configureproperty, and initializeproperty as appropriate.

Stereotypes for attributes are not optional for interfaces and for software components.

Associations

No additional associations. *OR*

- role1: Classifier associated with the role [cardinality] Description of the role.
- role2: Classifier associated with the role [cardinality] Description of the role.

Operations

No additional operations. *OR*

- operationName Description of the operation.
(parameterDirection
parameterName: ClassifierName,
... return ClassifierName): {raises
= (exceptionName1, ...)}

parameterDirection: in, out inout

return and raises are optional

- operation2 (list of parameters): Description of the operation.
{raises (exceptionName1, ...)}

Types and Exceptions

No additional types and exceptions. *OR*

- <<stereotype>>type1 Description of the type.
- stereotype is optional, values are: Paste in figure as needed
abstract, enumeration
- <<stereotype>>type2 Description of the type.
- <<exception>>exception1 Description of the exception.
- Paste in figure as needed.

Mapping

This section is only intended to be used for PSM concept description.

This section will describe the mapping between PIM and PSM mappings.

Constraints

No additional constraints. *OR*

- constraint1 Description of the constraint.
- constraint2 Description of the constraint.

Semantics

No additional semantics. *OR*

8.3 Data Link Layer Facilities

8.3.1 Link Layer Control Facilities

This section defines the Link Layer Control (LLC) facilities. LLC layer provides facilities to upper layers, for management of communication links between two or more radio sets. LLC layer definition is mainly based on the DLPI specification. DLPI specifies an SCA conformant API that is an instantiation of the ISO Data Link Service Definition DIS 8886 and Logical Link Control DIS 8802-2 (LLC). Where the two standards do not conform, DIS 8886 prevails.

The LLC interface supports three modes of communication: **connection**, **connectionless** and **acknowledged connectionless**. The connection mode is circuit-oriented and enables data to be

transferred over a pre-established connection in a sequenced manner. After the link parameters are negotiated and the link is established, data provider can send a data stream through the link. Data may be lost or corrupted in this service mode, however, due to provider-initiated resynchronization or connection aborts.

The connectionless mode is message-oriented and supports data transfer in self-contained units (PDUs) with no logical relationship required between units. Because there is no acknowledgement of each data unit transmission, this service mode can be unreliable in the most general case. However, a specific logical link provider can provide assurance that messages will not be lost, duplicated, or reordered.

The acknowledged connectionless mode provides the means by which a data link user can send data and request the return of data at the same time. By this way, the transmitter knows which data packets made it through, and retransmits the required packets. Although the exchange service is connectionless, in-sequence delivery is guaranteed for data sent by the initiating station. The data unit transfer is point-to-point.

For each of these communication modes, established link should be controlled locally using local link management interfaces. For this purpose, the LLC facilities are sub-packaged into four different categories as shown in

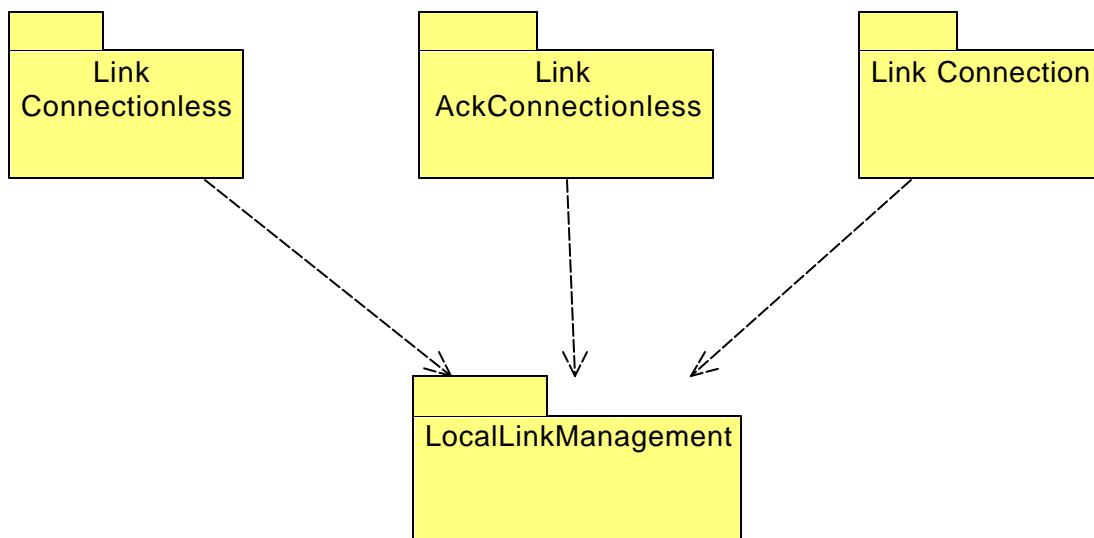


Figure 90 - Link Layer Control Package Structure

8.3.1.1 ILinkLayerControl

Description

Each of the packages defined within the LLC facilities have a main interface that a LLC resource can realize. The LLC facilities also specify an ILinkLayerControl interface that specializes the interfaces defined in those packages as shown in Figure 91. This main LLC interface can be realized by a resource of stereotype LinkLayerControlResource. The LinkLayerControlResource stereotype is defined in the UML Profile section.

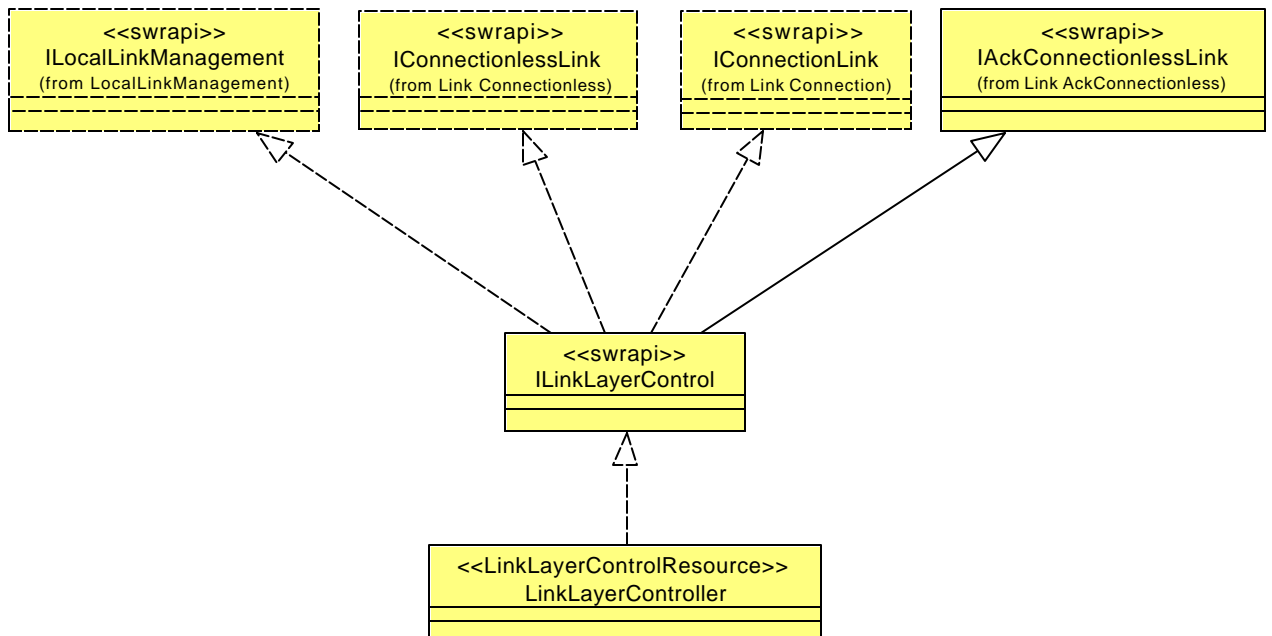


Figure 91 - ILinkLayerControl Definition

Attributes

No additional attributes

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

N/A

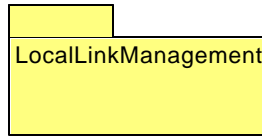
Constraints

No additional constraints.

Semantics

No additional semantics.

8.3.1.2 Local Link Management Package



This package provides a mechanism to manage the properties of communication links that are instantiated or established by the LLC. The local management services apply to all modes of service. These services, which fall outside the scope of standards specifications, define the method for initializing a stream that is connected to a logical link provider. Logical link provider information reporting services are also supported by the local management facilities. This package consists of a single interface, ILocalLinkManagement and several other type definitions that the interface depends upon.

8.3.1.2.1 ILocalLinkManagement

Description

ILocalLinkManagement interface provides functionality to control local parameters that are related to link establishment, binding, information reporting, as well as managing connection properties. ILocalLinkManagement is defined in Figure 92. Every logical link is referenced by a ConnectionID that describes the service access point(s) that the link is bound to.

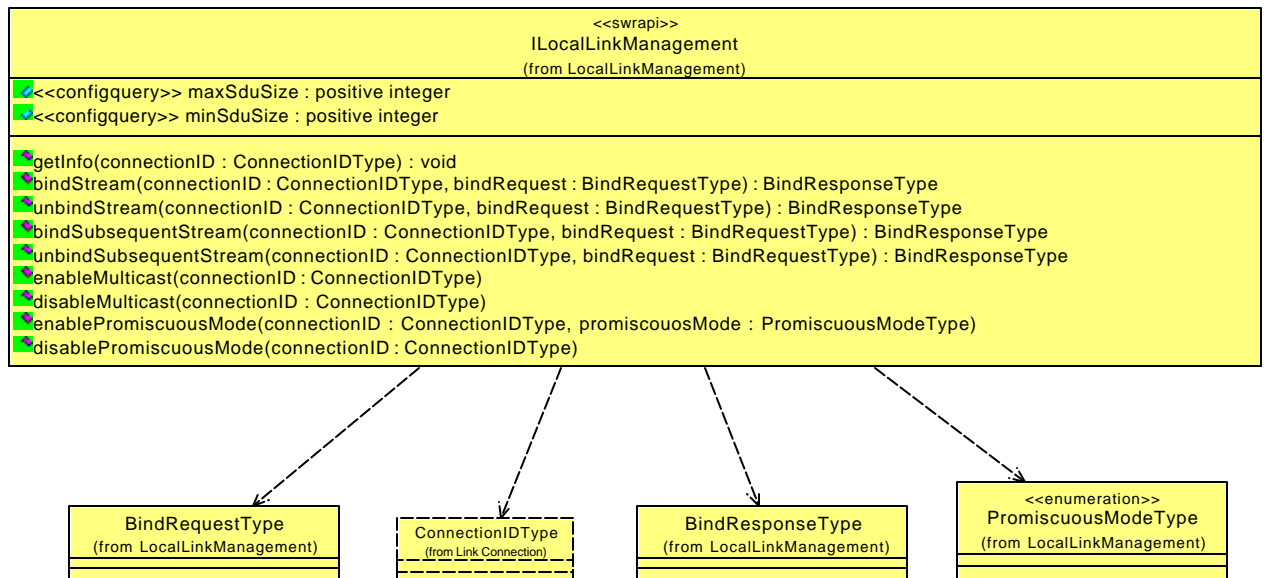


Figure 92 - ILocalLinkManagement Definition

Attributes

- <<configquery>> maxSduSize: positive integer: This attribute specifies the maximum service data unit size the LLC resource can transfer.
- <<configquery>> minSduSize: This attribute specifies the minimum service data unit

positive integer

size the LLC resource can transfer. If incoming data is less than this amount, the LLC resource waits to transmit until more data comes in. (or until timeout occurs, depending on the implementation)

Associations

No additional associations.

Operations

- `getInfo(connectionID : ConnectionIDType)`
This operation requests information of the provider about the currently established connection. The `connectionID` parameter identifies a stream (defined as a user connected to the provider). The operation may raise the `InvalidPort` exception or `SystemError` exception.
- `bindStream(connectionID : ConnectionIDType, bindRequest : BindRequestType) : BindResponseType`
The `bindStream` operation associates a service access point (SAP) with a stream. The SAP is identified by a SAP address. It requests that the logical link provider bind a SAP to a stream. It also notifies the logical link provider to make the stream active with respect to the SAP for processing connectionless and acknowledged connectionless data transfer and connection establishment requests. Protocol-specific actions taken during activation should be described in logical link provider specific addenda.
- `bindSubsequentStream(connectionID : ConnectionIDType, bindRequest : BindRequestType) : BindResponseType`
Certain logical link providers require the capability of binding a stream on multiple SAP addresses. `BindSubsequentStream` operation provides that added capability. The logical link provider returns the bound SAP address in the same primitive. The logical link provider indicates failure by raising an exception.
- `unbindStream(connectionID : ConnectionIDType, bindRequest : BindRequestType) : BindResponseType`
`unbindStream` operation requests the logical link provider to unbind all SAP(s) from a stream. This operation also unbinds all the subsequently bound SAPs that have not been unbound.
- `unbindSubsequentStream(connectionID : ConnectionIDType, bindRequest : BindRequestType) : BindResponseType`
`unbindSubsequentStream` requests the logical link provider to unbind the subsequently bound SAP.
- `enableMulticast(connectionID : ConnectionIDType)`
`enableMulticast` operation requests the logical link provider to enable specific multicast addresses on a per stream basis.
- `disableMulticast(connectionID : ConnectionIDType)`
`disableMulticast` operation requests the logical link provider to disable specific multicast addresses on a per stream basis.

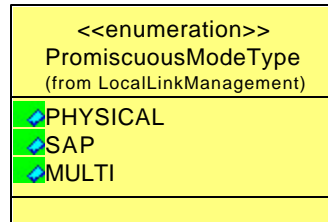
a per stream basis.

- `enablePromiscuousMode(connectionID : ConnectionIDType, promiscuousMode : PromiscuousModeType)` This operation requests the provider to enable promiscuous mode on a per Stream basis, either at the physical level or at the Service Access Point (SAP) level.
- `disablePromiscuousMode(connectionID : ConnectionIDType)` This operation requests the provider to disable promiscuous mode on a per Stream basis.

Types and Exceptions

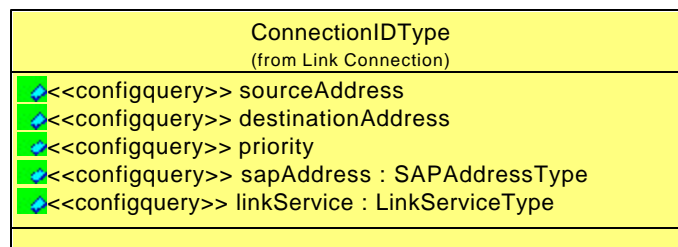
- `<<enumeration>>`
`PromiscuousModeType`

Promiscuous mode can be enabled on a per connection basis, either at the physical level (PHYSICAL) or at the Service Access Point (SAP) level, or at a multiple (MULTIPLE) level.



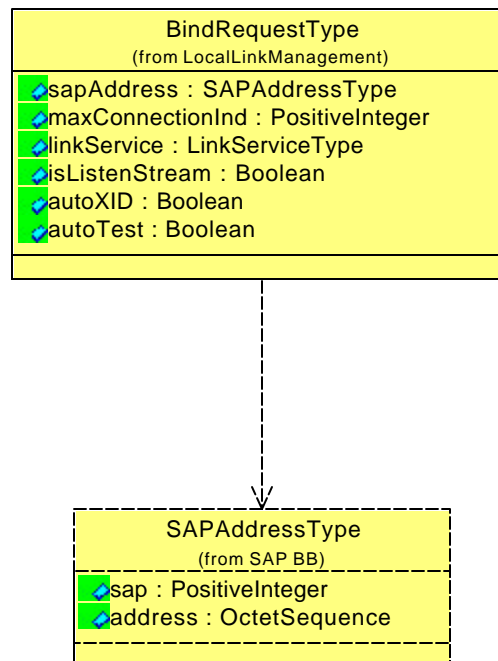
- `ConnectionIDType`

`ConnectionIDType` class completely specifies a logical link that is established at the LLC layer. It specifies the source and destination radio sets, the SAP address that the logical link is bound to within the local radio set, as well as the link service type (connection, connectionless, ack connectionless).



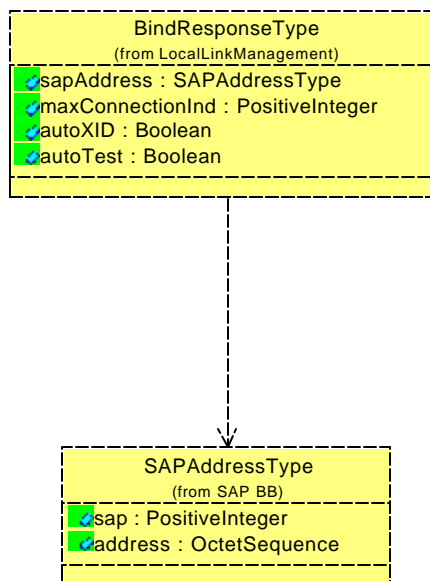
- `BindRequestType`

This class defines the `BindRequest` header attributes. This header is passed to the LLC when a connection is required to be bound to a SAP.



- BindResponseType

This class defines the BindResponse header attributes.



Mapping

N/A

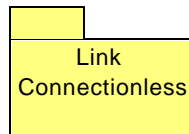
Constraints

No additional constraints.

Semantics

No additional semantics.

8.3.1.3 Connectionless Link Package



This package provides facilities to provide connectionless mode communication for an LLC layer. The connectionless mode is message-oriented and supports data transfer in self-contained units with no logical relationship required between units. This package consists of a main interface, `ICConnectionlessLink` and several other type definitions that the interface depends upon.

The connectionless mode package does not use the connection establishment and release phases of the connection-mode service. The local management phase is still required to initialize a stream. Once initialized, however, the connectionless data transfer phase is immediately entered. Because there is no established connection, however, the connectionless data transfer phase requires the LLC user to identify the destination of each protocol data unit to be transferred. The destination LLC user is identified by the address associated with that user. Since there is no acknowledgement of each PDU transmission, this service mode can be unreliable in the most general case. However, a specific link layer or MAC provider can provide flow and error control mechanisms to assure that messages will not be lost, duplicated, or reordered.

8.3.1.3.1 ICConnectionlessLink

Description

`ICConnectionlessLink` interface as shown in Figure 93, provides functionality to control parameters that are related to connectionless link establishment, and management as well as preparing and sending protocol data units. After the connection is established, data can be transferred using `ICConnectionlessLink` interface for unacknowledged connectionless communication scenario. This interface inherits the `IQualityOfServiceConnectionless` interface for quality of service related facilities, `IServiceAccessPoint` for performing vertical communication tasks, `IFlowControl` for flow control interfaces, `IPriorityQueue` for queueing incoming and outgoing PDUs for flow control purposes. Connectionless link interface can provide facilities for segmentation and reassembly of protocol data, as well as concatenation and padding of PDU's to match the protocol specification. Every logical link is referenced by a `ConnectionID` that describes the service access point(s) that the link is bound to. Local link management interface establishes the links and bounds them to vertical (internal) streams. A component realizing the `ICConnectionlessLink` API can have a user role, a provider role, or both; depending on the waveform scenario. This interface encompasses all of the possibilities.

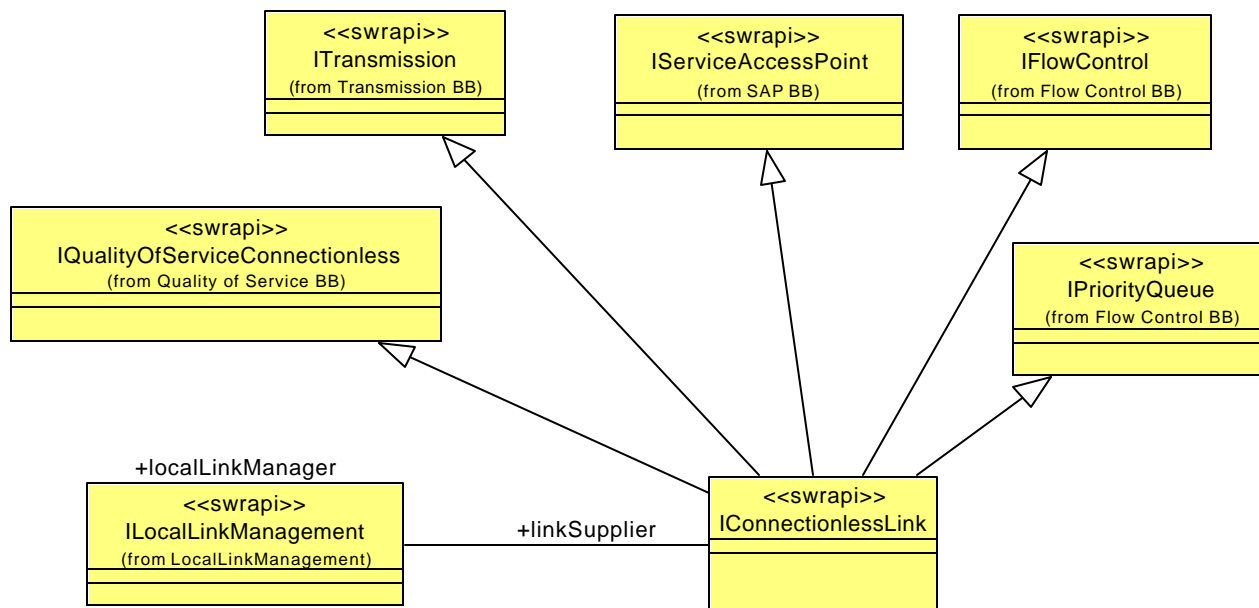


Figure 93 – IConnectionlessLink Definition

Attributes

No additional attributes.

Associations

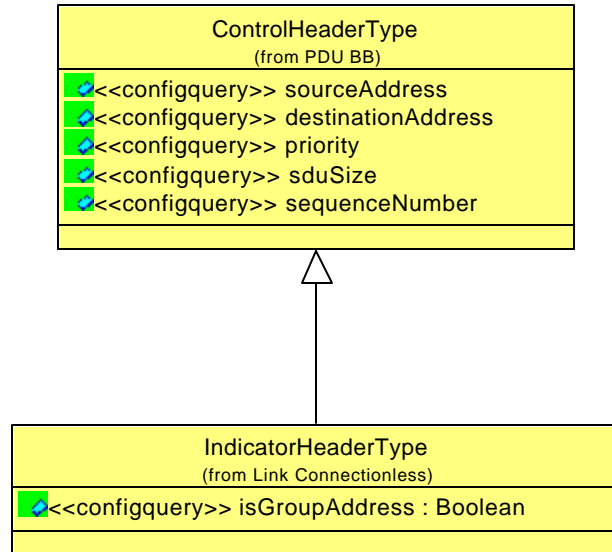
- localLinkManager:
ILocalLinkManagement
- ILocalLinkManagement interface can act as a local link manager to the links that are used by the IConnectionlessLink interface.

Operations

No additional operations.

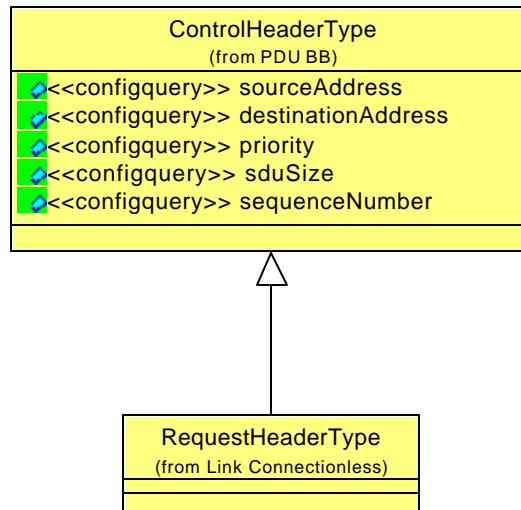
Types and Exceptions

- IndicatorHeaderType
- Indicator header is used to convey one SDU from the LLC provider to the LLC user.



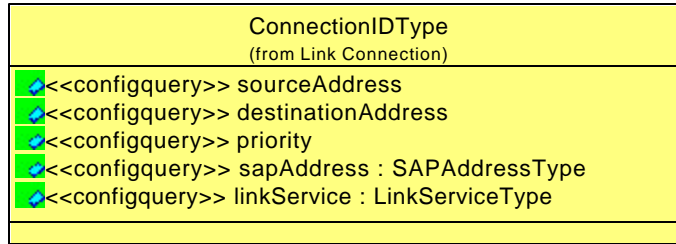
- RequestHeaderType

This header type conveys one SDU from the LLC user to the LLC provider for transmission to a peer LLC user.



- ConnectionIDType

ConnectionIDType defines the parameters related to an LLC connection. This includes the source and destination address (horizontal communication parameters) as well as the SAP address that the connection is bound to. Priority parameter sets the priority value if priority handling is used in the LLC. LinkService parameter determines the type of link (connection, connectionless, acknowledged connectionless)



Mapping

N/A

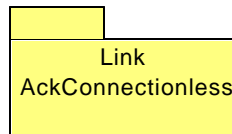
Constraints

No additional constraints.

Semantics

No additional semantics.

8.3.1.4 Acknowledged Connectionless Link Package



This package provides facilities to provide acknowledged connectionless mode communication for LLC layer. The acknowledged connectionless mode is message-oriented and supports data transfer in self-contained units with no logical relationship required between units. Although the exchange service is connectionless, in-sequence delivery is guaranteed for data sent by the initiating station. The acknowledged connectionless mode provides the means by which a data link user can send data and request the return of data at the same time. The data unit transfer is point-to-point. This package consists of a main interface, `IAckConnectionlessLink` and several other type definitions that the interface depends upon.

The acknowledged connectionless mode package also does not use the connection establishment and release phases of the connection-mode service. The local management phase is still required to initialize a stream. Once initialized, the acknowledged connectionless data transfer phase is immediately entered. Because there is no established connection, the LLC user is required to identify the destination of each protocol data unit to be transferred. The destination LLC user is identified by the address associated with that user.

Acknowledged connectionless data transfer guarantees that data units will be delivered to the destination user in the order in which they were sent. A data link user entity can send a data unit to the destination LLC user, request a previously prepared data unit from the destination LLC user, or exchange data units.

8.3.1.4.1 IAckConnectionless

Description

IAckConnectionlessLink interface as shown in Figure 94, provides functionality to control parameters that are related to acknowledged connectionless link establishment, and management as well as preparing and sending protocol data units. After the connection is established, data can be transferred using IAckConnectionlessLink interface for acknowledged connectionless communication scenario. This interface inherits the IQualityOfServiceConnectionless interface for quality of service related facilities, IServiceAccessPoint for performing vertical communication tasks, IFlowControl for flow control interfaces including acknowledgment, IPriorityQueue for queueing incoming and outgoing PDUs for flow control purposes and IErrorControl for detecting and reporting errors in the reception or transmission. Acknowledged connectionless link interface can provide facilities for segmentation and reassembly of protocol data, as well as concatenation and padding of PDU's to match the protocol specification. Every logical link is referenced by a ConnectionID that describes the service access point(s) that the link is bound to. Local link management interface establishes the links and bounds them to vertical (internal) streams. A component realizing the IAckConnectionlessLink API can have a user role, a provider role, or both; depending on the waveform scenario. This interface encompasses all of the possibilities.

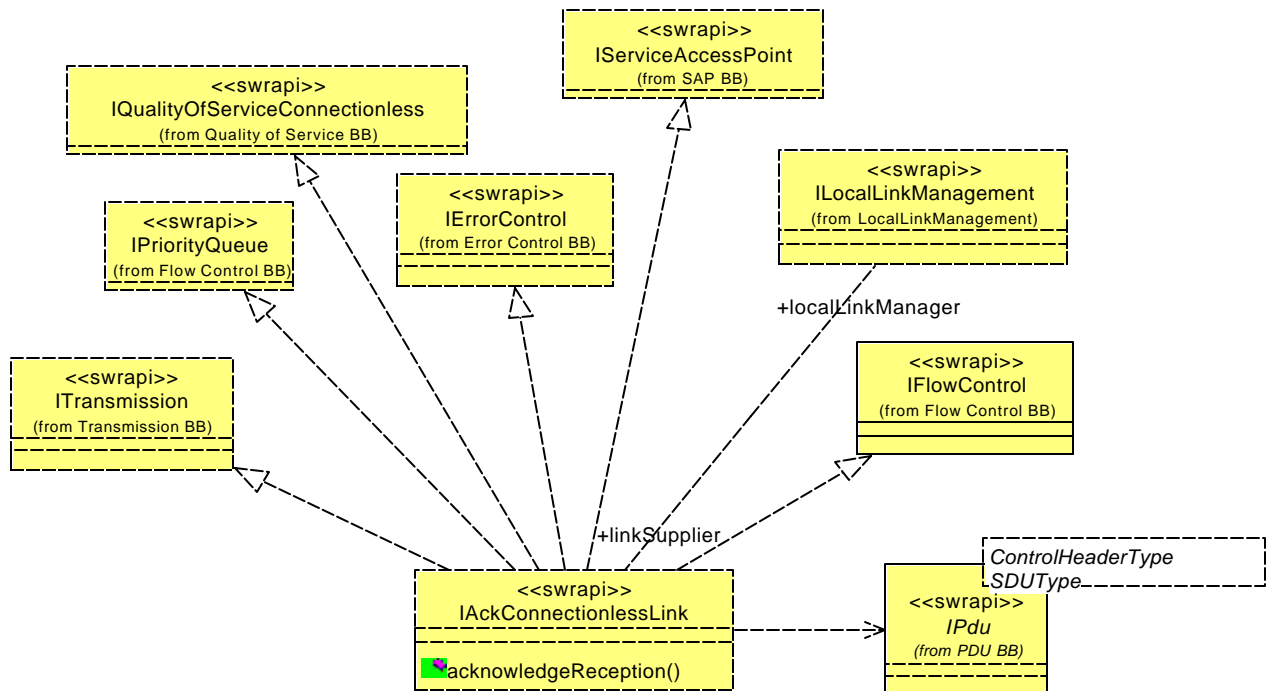


Figure 94 - IAckConnectionless Definition

Attributes

No additional attributes.

Associations

- localLinkManager: ILocalLinkManagement ILocalLinkManagement interface can act as a local link manager to the links that are used by the

ILocalLinkManagement

IAckConnectionlessLink interface.

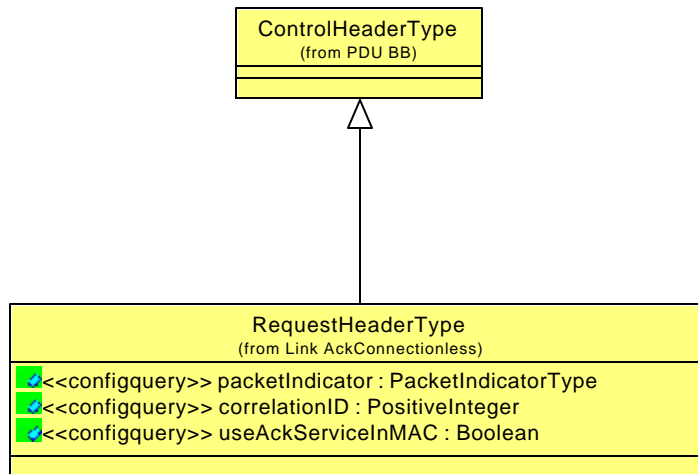
Operations

- acknowledgeReception () Acknowledgement of received data

Types and Exceptions

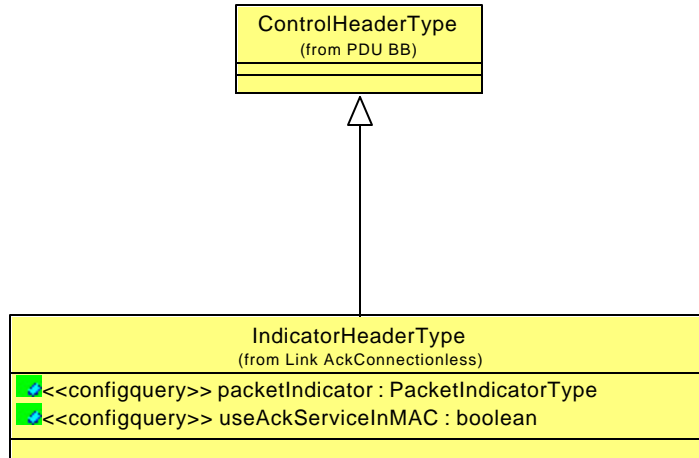
- requestHeaderType

This header type is passed to the LLC provider by the LLC user to request that a SDU be returned from a peer LLC provider or that SDUs be exchanged between stations using acknowledged connectionless mode data unit exchange procedures.



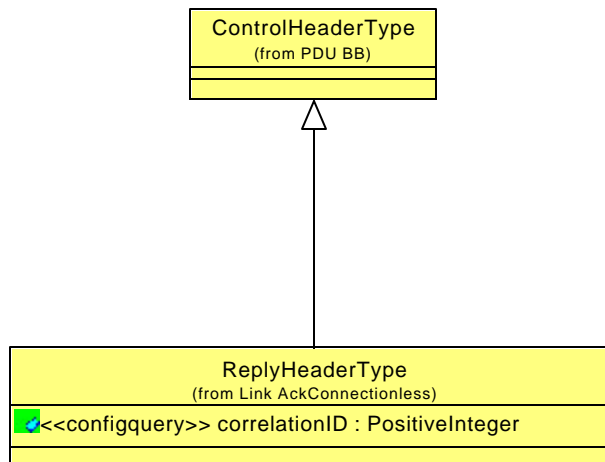
- indicatorHeaderType

This header type is passed from the LLC provider to the LLC user to indicate either a successful request of a SDU from the peer data link user entity, or exchange of SDUs with a peer data link user entity.



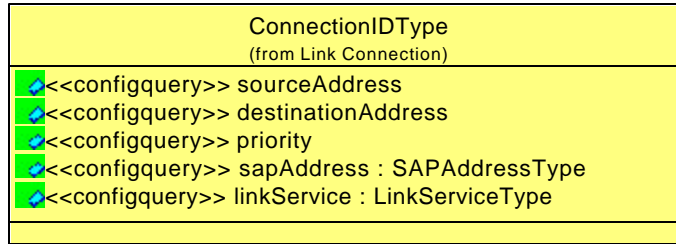
- replyHeaderType

Conveys a SDU to the LLC provider from the LLC user to be held by the LLC provider and sent out at a later time when requested to do so by the peer LLC provider.



- ConnectionIDType

ConnectionIDType defines the parameters related to an LLC connection. This includes the source and destination address (horizontal communication parameters) as well as the SAP address that the connection is bound to. Priority parameter sets the priority value if priority handling is used in the LLC. LinkService parameter determines the type of link (connection, connectionless, acknowledged connectionless)



Mapping

N/A

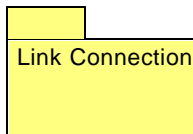
Constraints

No additional constraints.

Semantics

No additional semantics.

8.3.1.5 Connection Link Package



This package provides facilities to provide connection mode communication for LLC layer. The connection mode is circuit switched and supports data transfer in streams. The connection-mode service is characterized by four phases of communication: local management, connection establishment, data transfer, and connection release. Local management functionality is provided by the local management package defined earlier. Rest of the functionality is defined in this package.

8.3.1.5.1 IConnectionLink

Description

IConnectionLink interface as shown in Figure 95, provides functionality to control parameters that are related to connection oriented link establishment, and management as well as enabling and disabling data streams. After the connection is established, data can be transferred using IConnectionLink interface for connection oriented communication scenario. This interface inherits the IQualityOfServiceConnection interface for quality of service related facilities, IServiceAccessPoint for performing vertical communication tasks, IFlowControl for flow control interfaces, and ITransmission for controlling data and control streams. Every logical link is referenced by a ConnectionID that describes the service access point(s) that the link is bound to. Local link management interface establishes the links and bounds them to vertical (internal) streams. A component realizing the IConnectionLink API can have a user role, a provider role, or both; depending on the waveform scenario. This interface encompasses all of the possibilities

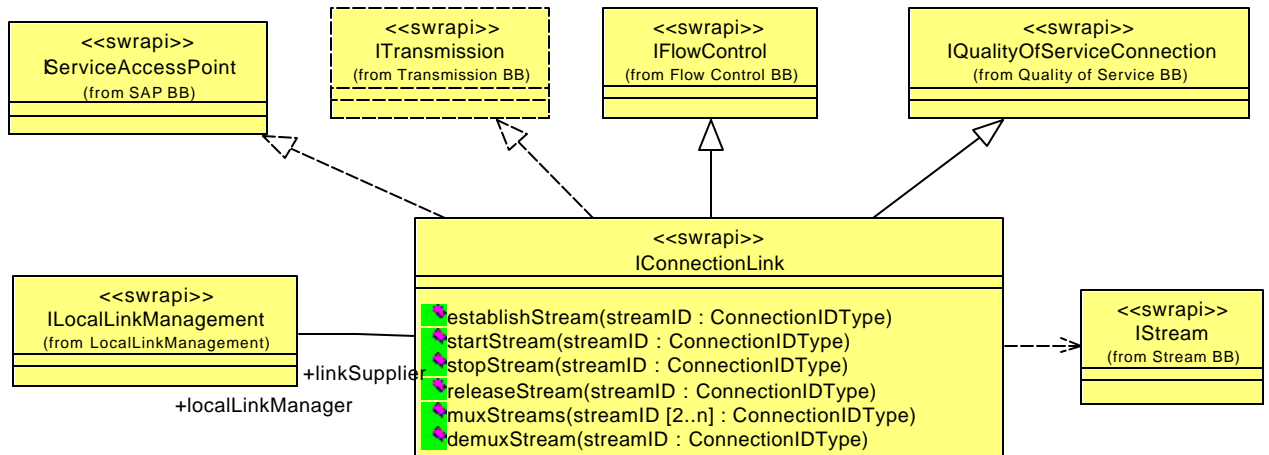


Figure 95 - IConnectionLink Definition

Attributes

No additional attributes.

configquery - a property that can be configured and queried by the

Associations

- localLinkSupplier : ILocalLinkManagement
ILocalLinkManagement interface can act as a local link manager to the links that are used by the IAckConnectionlessLink interface.






Operations

- establishStream(streamID : ConnectionIDType)
This operation allows the LLC serve user to initialize a stream.
- startStream(streamID : ConnectionIDType)
This operation starts data transfer through a previously established stream
- stopStream(streamID : ConnectionIDType)
This operation stops data transfer through the given stream
- releaseStream(streamID : ConnectionIDType)
The releaseStream operation destroys the stream and releases all of the resources associated with it.
- muxStreams(streamID [2..n] : ConnectionIDType)
This operation multiplexes multiple (two or more) streams into a single stream. This can be done by both the receiving or transmitting entity.
- demuxStream(streamID : ConnectionIDType)
This operation demultiplexes a stream that is composed of multiple data streams.

Types and Exceptions

- ConnectionIDType

ConnectionIDType defines the parameters related to an LLC connection. This includes the source and destination address (horizontal communication parameters) as well as the SAP address that the connection is bound to. Priority parameter sets the priority value if priority handling is used in the LLC. LinkService parameter determines the type of link (connection, connectionless, acknowledged connectionless)

ConnectionIDType (from Link Connection)	
	<<configquery>> sourceAddress
	<<configquery>> destinationAddress
	<<configquery>> priority
	<<configquery>> sapAddress : SAPAddressType
	<<configquery>> linkService : LinkServiceType

Mapping

N/A

Constraints

No additional constraints.

Semantics

No additional semantics.

8.3.2 MAC Facilities

The facilities are at the components providing the media access control behavior of a waveform.

The Media Access Control (MAC) Facilities define interactions between a user of the MAC layer, termed a Service User, and a MAC layer, termed a Service Provider. The MAC Facilities declare operations that can be invoked by a Service User on a Service Provider for pushing data or sending non-real-time control signals (for configuration purposes). There are also callback operations that can be invoked by a Service Provider on a Service User to report event occurrences.

There are seven MAC Facilities. Each facility defines generic concepts shared by several specific waveforms. Facilities are selected and the parameterized classes they contain are actualized in order to build a specific waveform.

8.3.2.1 MAC Common Utility Facility

The MAC Common Utility Facility defines operations for activating (or selecting) a preset channel and setting the control mode of the RF transmission power.

It is assumed that a radio as one active channel, which can be selected by an operator. It is also assumed that channels have preset numbers, such as 1, 2, guard or emergency. The control mode of the RF transmission power of the active channel can be changed. A power mode can be, for instance, auto, low, medium, high or maximum.

The MAC Common Utility Facility consists of a single concept: MACCommonUtility.

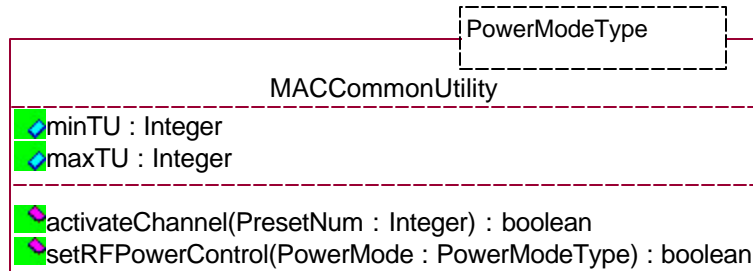


Figure X. Class diagram for the MAC Common Utility Facility.

8.3.2.1.1 MACCommonUtility

Description

This is a parameterized class with formal parameter **PowerModeType**. The latter is the control mode of the RF power. It is an enumerated type, which is waveform specific.

Attributes

- **minTU: Integer** Minimum Transmission Unit, i.e. the minimum length of a valid packet accepted by a Service Provider.
- **maxTU: Integer** Maximum Transmission Unit, i.e. the maximum length of a valid packet accepted by a Service Provider.

Associations

No associations.

Operations

- **activateChannel(PresetNum: Integer): boolean** Invoked by a Service User on a Service Provider to pass the number of a selected preset channel. The number refers to a preset channel such as the emergency, guard or primary channel. If the Service Provider knows the **PresetNum** and succeeds to set the corresponding channel it returns the value **true**, otherwise it returns the value **false**.

- `setRFPowerControl(PowerMode:PowerModeType):boolean` Invoked by a Service User on a Service Provider to pass a RF power mode. Examples of values for the power mode are auto, low, medium, high and maximum. If the Service Provider knows the RF power mode, then it sets the RF power of the currently active channel to that mode and it returns the true value. Otherwise, it returns the value false.

Constraints

No additional constraints.

Semantics

To be provided.

8.3.2.2 TRANSEC Facility

TRANSEC is an abbreviation for transmission security. The TRANSEC Facility provides non-Type 1 operations, which may be placed in either the Red security domain or Black security domain. The type of over-the-air waveform being realized is a variation point of the TRANSEC Facility. The concept is defined in a generic manner and must be actualized with a waveform-specific class.

The TRANSEC Facility generates a TRANSEC stream determined by a TRANSEC algorithm, a TRANSEC variable, and TRANSEC seed(s).

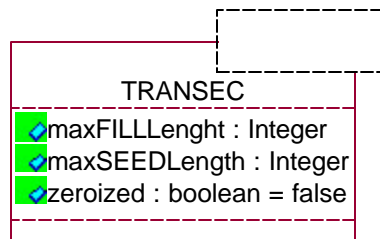


Figure X. Class diagram for the TRANSEC Facility.

8.3.2.2.1 TRANSEC

Description

Generic parameters

FillType

FillInfoType

FillIIDType

SeedType

SeedInfoType

Attributes

- maxFILLLength : Integer
- maxSEEDLength : Integer
- zeroized : boolean

Associations

No associations.

Operations

- loadFill(PresetNumber : Integer, Fill : FillType, FillInfo : FillInfoType) : Boolean
Transfers TRANSEC Fill data (inserted non information data) from a Service User to a MAC layer. PresetNumber is the number of a selected preset channel. It returns the status of the transfer. Fill types may include Hopsets, Lockouts, TRANSEC Keys, etc.
- loadFillIID(PresetNumber : Integer, Fill : FillType, FillID : FillIIDType) : boolean
Changes a Fill ID specified by a Preset Number by transferring a new ID, which replaces an existing ID. It returns the status of each ID change. FillType may include, for example, Hopsets.
- readFillID(PresetNumber : Integer, Fill : FillType, out FillID : FillIIDType) : boolean
Reads the identity of a fill element, but not the actual Fill data. For example, readFill returns the ID of a Hopset, but not the Hopset data. For a TRANSEC variable, it returns the variable ID, but nor the variable.
- loadSeed(SeedNumber : Integer, Seed : SeedType, SeedInfo : SeedInfoType) : boolean
Transfers TRANSEC Seed data from a Service User to a MAC layer. The status of the transfer is returned. Seed types includes Base Time of Day and Net TOD.
- readSeed(SeedNumber : Integer, Seed : SeedType, out SeedInfo : SeedInfoType) : boolean
Reads the Seed data. For example, it may return the Time of Day or Word of Day.
- zeroize() : boolean
Removes all Fill and Seed data from the memory owned by the TRANSEC facility. It overwrites all Fill and Seed data memory space with a variety of

data patterns to ensure that the information is not maintained. The memory is not returned to the operating system. The status of the operation is returned.

Constraints

No additional constraints

Semantics

To be provided (should explain how a channel is disabled, loaded with fill and seed data, and enabled again)

8.3.2.3 Channel Error Control Facility

8.3.2.3.1 ChannelErrorControl

Description

The Channel Error Control Facility is about maintaining the integrity of messages over a physical channel. Possible error control mechanisms are forward error correction, interleave, scramble, bit tracking including fade bridging and transmit RF power level control.

Attributes

No attributes.

Associations

No associations.

Operations

- `channelErrorControl(ErrorControl : boolean)` Enables or disables error control over the physical channel.

Constraints

No additional constraints.

Semantics

To be provided.

8.3.2.4 Channel Access Facility

The Channel Access Facility detects incoming messages and executes a channel access protocol. The channel access protocol may include sync detection, sync search, end-of-message detection and channel access algorithm including TDMA, CSMA, DASA/DAMA and WDW.

8.3.2.4.1 ChannelAccess

Description

Generic parameters

AccessParameterType

Attributes

No attributes.

Associations

No associations.

Operations

- `sertAccessParameters(Parameters AccessParameterType) : boolean` : Configures a network access algorithm with information required to establish, for example, the slot width, slot frequency, slot time, number of slots, slot priorities (e.g. flash, routine), inter-transmit delay. It returns the status of the configuration operation.

Constraints

No additional constraints.

Semantics

To be provided.

8.3.2.5 MAC Address Facility

The MAC Address Facility allows setting the address of the MAC address.

8.3.2.5.1 MACAddressing

Description

Generic parameters

AddressType

Attributes

No attributes.

Associations

No associations.

Operations

- `bindAddress(address : AddressType) : Boolean` Defines the MAC address. It returns the status of the bind operation.

status of the bind operation.

Constraints

No additional constraints.

Semantics

To be provided.

8.3.2.6 Drop Capture Facility

The Drop Capture Facility provides a facility to terminate an active reception and return to the signal search state. In more detail it terminates the current reception, flush the message, stop bit tracking, clear all variables associated with the current reception including forward-error-correction code and interleavers and return to the signal search state.

8.3.2.6.1 DropCapture

Description

Attributes

No attributes.

Associations

No associations.

Operations

- dropCapture() : boolean

Terminates an active reception and forces a transition to the signal search state.

Constraints

No additional constraints

Semantics

To be provided

8.3.2.7 Quality of Service (QOS) Facility

The Quality of Service Facility offers facilities for get channel quality parameters, when these are available within the MAC layer. Examples of parameters are residual bit-error-rate, block error rate, correctable block status, erasure count and raw bit-error-rate.

8.3.2.7.1 QualityOfService

Description

Generic parameters

QOSType

Attributes

No attributes.

Associations

No associations.

Operations

- | | |
|-------------------------------|--|
| • readQOS(QOS : QOSType) | Reads the quality of service information. |
| • setQOSParameters : QOSType) | Configures the quality of service to provide a desired type of error rate. |
| • enableQOS() | Enables the quality of service function. |
| • disableQOS() | Disables the quality of service function. |

Constraints

No additional constraints.

Semantics

To be provided.

8.4 IO Facilities

Inside a radioset, the IO subsystem has mission to establish bidirectional connections, termed IO channels, between a waveform stack and a physical radioset wired line. Those wired IOs may serve several purposes such as :

- connecting the radioset with a human operator through a microphone and a headset,
- linking a radioset with a Local Area Network (LAN) to provide a bridge between LAN stations and mobile equipments,
- connecting peripheral sensors devices to the radioset,
- clustering radiosets together to offer scalable and/or fault-tolerant capabilities,
- providing a mean to upload/download software from/to the radioset.

Currently, waveforms stacks are plugged to dedicated serial lines using single or half-duplex protocols and each additional IO physical channel require an additional physical slot (one-to-one relationship). Now, wired radiosets can be connected to multiplexed serial lines and/or buses (Ethernet, USB) and a single physical IO slot may virtually support an unlimited number of virtual channels (one-to-many relationship).

8.4.1 IO Profile

Inside any component-based architecture, facilities are supported by Components Ressources. Inside a platform-based architecture, facilities are supported by Platform Resources : for IO Facilities those ressources are termed IO Platform Resources.

IO Platform Ressources splits into 3 subtypes :

- Composite Devices that provide actual an/or virtual devices services,
- Virtual Resources that figure « pure » abstract ressources.

Composite Devices themselves splits into 2 subtypes :

- logical Hardware Devices (IODeviceHardware) that abstract actual hardware devices through a one-to-one relationship,
- logical Pseudo Devices (IODevicePseudo) that abstract sets of IO devices such as devices aggregates or devices categories.

Virtual Resources consists in :

- Services that provides device-independant services,
- Channel that provides audio and/or data processing chains.

Note that Logical Hardware Devices as well as Logical Pseudo Devices are always seen as composites (i.e : aggregated from other devices), even if they are actually not : this preserve the capability to later breakdown « big » devices into smaller ones without reworking applications operating on « big » ones.

Any Ressource Component may act both as a facilities consumer (a client) and/or as a facilities supplier (a server) :

- as a customer, a Ressource Component uses IUsedInterfaces,
- as a supplier, a Ressource Component exhibits and implements IProvidedInterfaces.

Components Interfaces themselves splits into several categories. 2 categories have been identified :

- Control Interfaces (IControllInterface) provides means to control and manage the system excluding security,
- SecurityControl Interfaces (ISecurityControllInterface) provide means to control and manage the system security,
- Stream Interfaces (IStreamInterface) provides connection points for flow processing.

The above situation is summarized and modeled on the next figure :

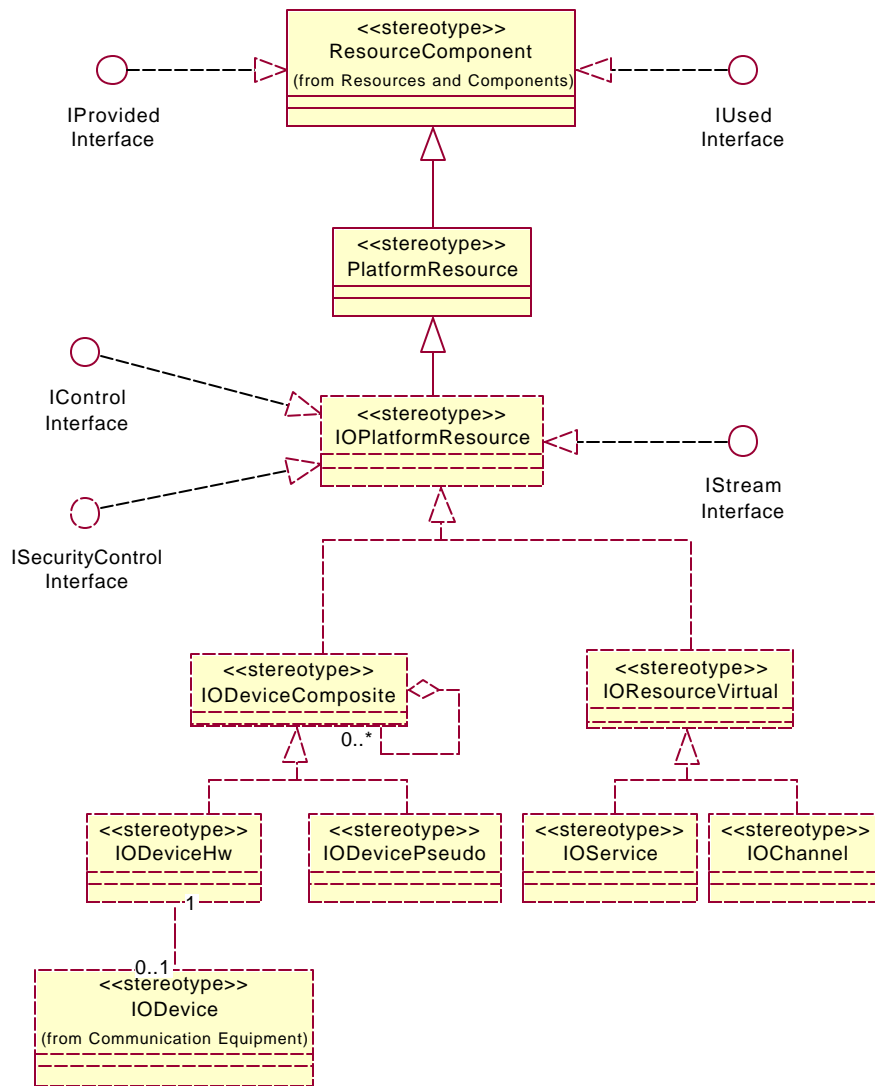


Figure 96: IO Profile

8.4.2 IO Devices

As exposed above, logical Hardware Devices (IODeviceHw) abstracts actual devices in a one-to-one relationship : inside an actual radioset configuration, a logical Hardware Device will exist for each actual Hardware Device. As such they are quite dependant of the devices specificities : so applications and especially control applications, that use them cannot be easily ported from one platform to another one.

In some cases, Hardware Devices may exhibit no attributes and no operations simply because software has no means (e.g : interfaces) to act on them. But whether or not IO Hardware Devices exhibit attributes and/or operations : in any case, they are created in order to provide handles for management operations. For example, such handles are needed to allocate the corresponding actual device inside the chain of devices that form a channel.

Pseudo-devices are logical devices that abstract one or several logical hardware devices and provide a higher level and more portable interface to applications. Due to the fact that they actually wrap IO Devices, pseudo-devices can be always considered as composite devices. Unlike Hardware devices that manages a single flow, Pseudo-devices can manage several flows.

At least, the Radio Platform supports the following default categories of logical hardware devices:

- Serial IO,
- Audio IO.

In near future, Radio Platform is expected to support additional devices such as Ethernet and USB.

The radio platform also supports the following default set of pseudo-devices :

- Generic IO pseudo-device (IODeviceGeneric) which provides a bidirectional access to payload data flow (i.e : without embedded inband control data),
- Raw IO pseudo-device (IODeviceRaw) which provides a bidirectional access to uninterpreted data as it is received on the virtual channel (i.e : with its embedded inband control data).

Default supported Device types are summarized below :

Type	Nom	Description
Pseudo	IODeviceGeneric	Provides a standardized access to payload data to/from a collection of various devices.
Pseudo	IODeviceRaw	Provides a standardized access to raw data to/from a collection of various devices.
Hardware	IODevicePTT	Abstracts a Push-To-Talk device.
Hardware	IODeviceSerial	Abstracts a serial line (rs232 ...).

Hardware	IODeviceSpeaker	Abstracts a speaker (headsets, baffles ...)
Hardware	IODeviceMicrophone	Abstracts a microphone.
Hardware	IODeviceVocoder	Abstracts a vocoder.
Hardware	IODeviceEthernet	Abstracts an ethernet bus.
Hardware	IODeviceUSB	Abstracts an USB bus.

Tableau 1: IO Devices

Other devices types are expected to be supported later as Platform standardisation will evolve.

IO Hardware Devices are modelled below with UML:

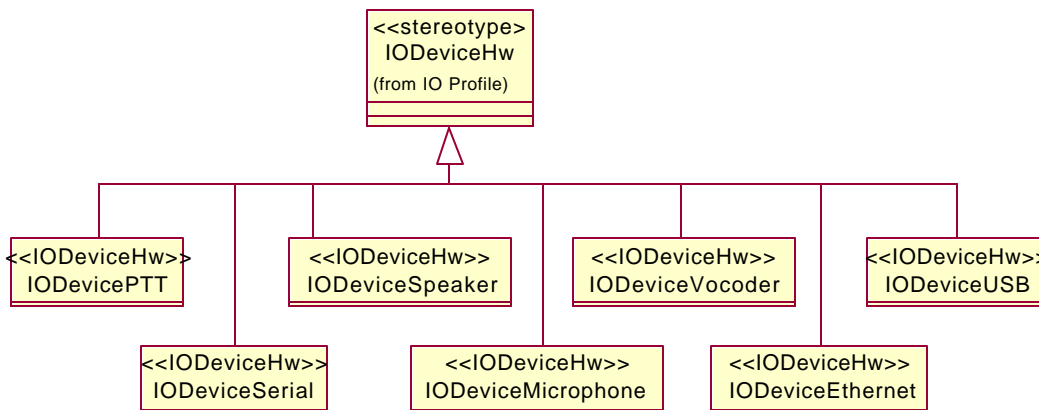


Figure 97: IO Hardware Devices

IO Pseudo Devices are modelled below with UML:

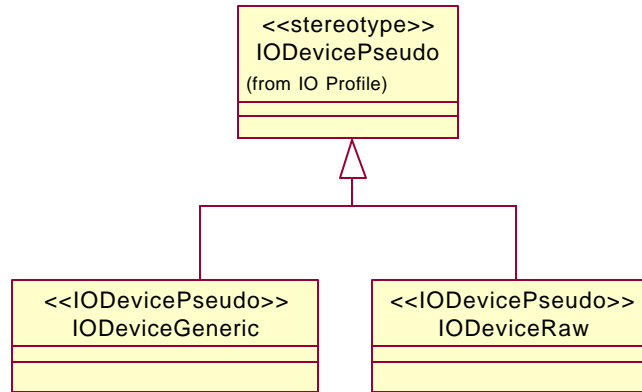


Figure 98: IO Pseudo-Devices

8.4.3 IO Interfaces

Each Platform Ressource can both provides (server side) and/or uses (client side) interfaces. Each client or server interfaces can be categorized into one of those interfaces type :

- Flow Interfaces that processes the flow of data with or without embedded inband control data,
- Control Interfaces that processes outband control data.

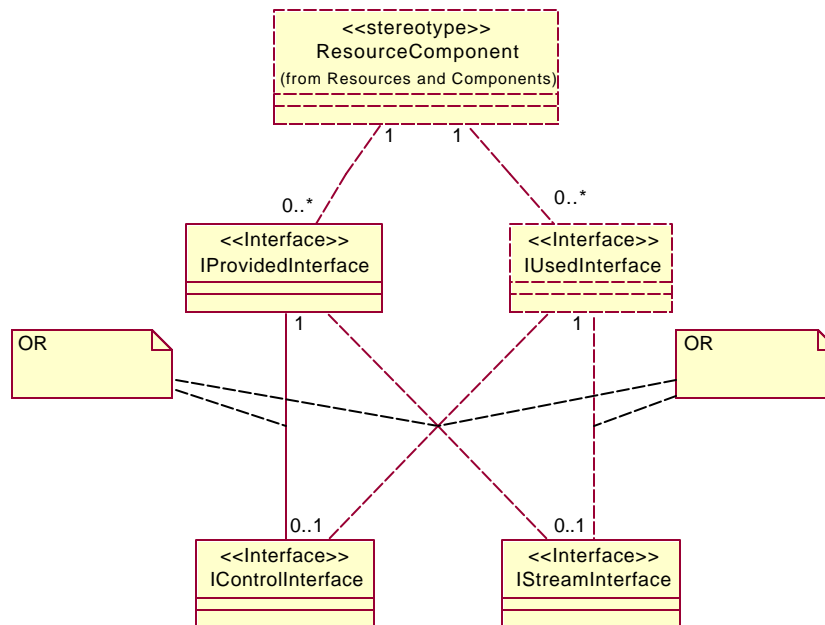


Figure 99: IO Interfaces Types

8.4.3.1 Control Interfaces

Control interfaces are designed to serve as a basis for platform and radioset management. = Control Interfaces mainly service Configuration and Deployment facilities.

8.4.3.1.1 Device Control

Device Control interfaces are designed to manage Hardware and Pseudo Devices. Device Control Interfaces mainly support global (e.g : outband) configuration and deployment.

Device Control Interfaces act upon Hardware and Pseudo Devices exposed above see (§8.4).

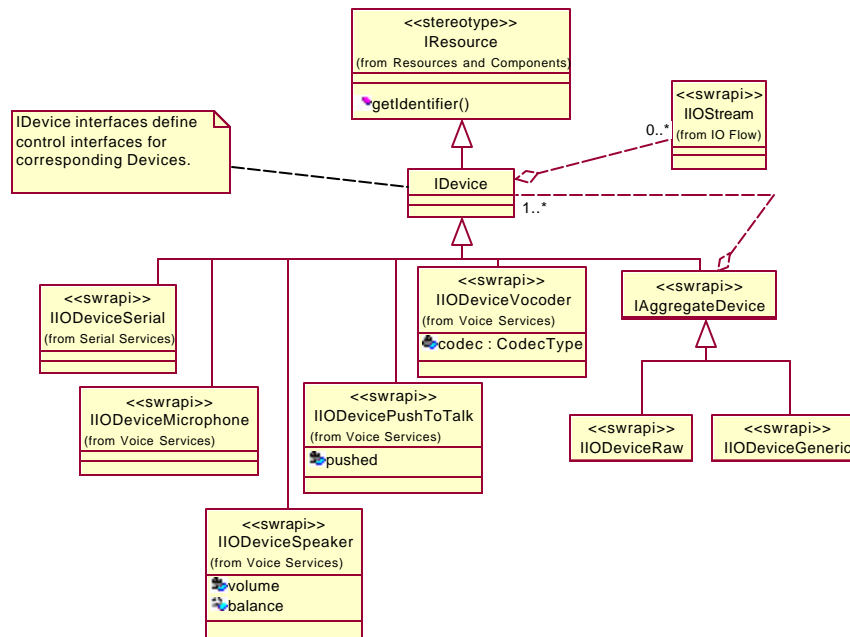


Figure 100: IO Devices Control Interfaces

8.4.3.1.2 IO Device Control

Description

An IODevice provides physical connections for the Communication Channel to users and baseband systems.

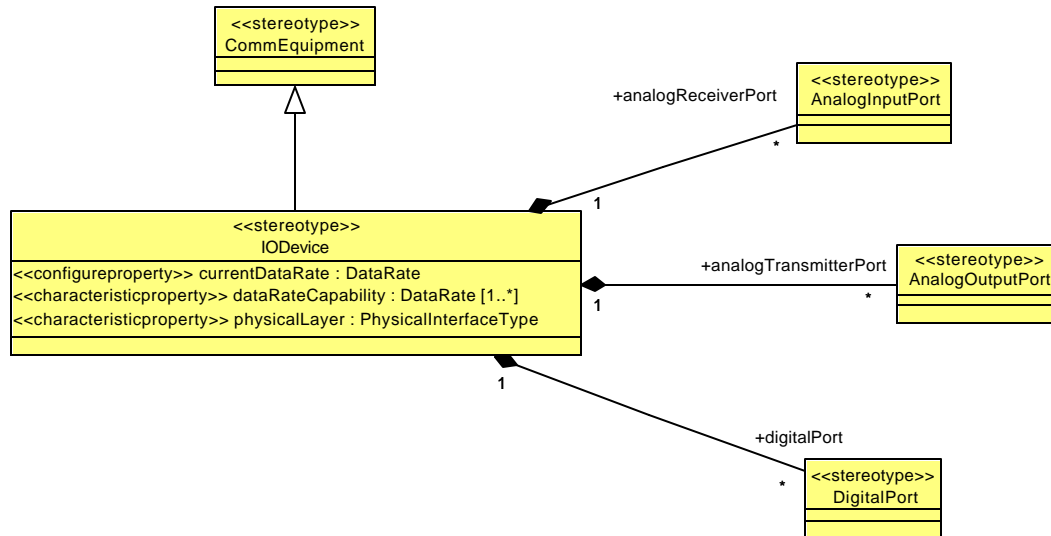


Figure 101 - I/O Device Definition

Attributes

<code><<configureproperty>>currentDataRate:</code>	The current data rate of the device.
<code>DataRate</code>	
<code><<characteristicproperty>>dataRateCapability:</code>	Data rates that the device supports.
<code>DataRate [1..*]</code>	
<code><<characteristicproperty>>physicalLayer:</code>	Identifies the specific physical layer protocol used by the device.
<code>PhysicalInterfaceType</code>	

Associations

<code>analogReceiverPort:</code> <code>AnalogInputPort [*]</code>	Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports) ¹³ .
<code>analogTransmitterPort:</code> <code>AnalogOutputPort [*]</code>	Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports) ¹⁴ .

¹³ UML 2.0 Superstructure Specification p.154

Ports) and Ports (from Ports)¹⁴.

digitalPort: DigitalPort [*]

Refines the ownedPort composition relationships defined between an EncapsulatedClassifier (from Ports) and Ports (from Ports)¹⁵.

Operations

No additional operations.

Types and Exceptions

<<abstract>>DataRate

The rate at which data is processed by the device.

<<abstract>>PhysicalInterfaceType

Physical layer protocol used by the device.

Mapping

TBD

.

Constraints

No additional constraints.

Semantics

No additional semantics.

8.4.3.2 Virtual Resource Control

Virtual Resource Control interfaces are designed to manage software resources. Virtual Resource Control Interfaces mainly support global (e.g. : outband) configuration and deployment.

¹⁴ UML 2.0 Superstructure Specification p.154

¹⁵ UML 2.0 Superstructure Specification p.154

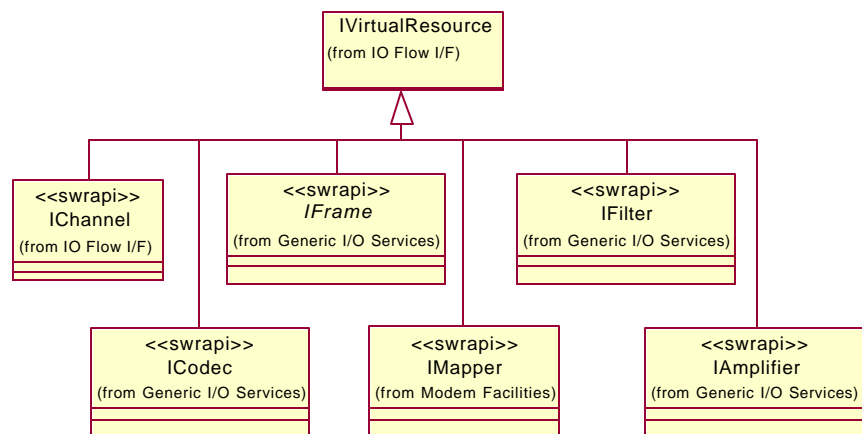


Figure 102: IO Virtual Resources Interfaces

8.4.3.2.1 IO Codec Control

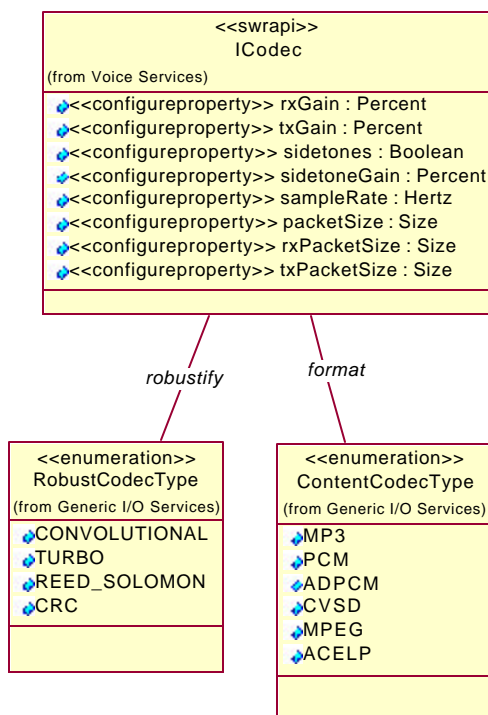


Figure 103: IO Codec Control

Attributes

- I. <<configureproperty>>rxGain: Receive gain rate.
 Percent
- II. <<characteristicproperty >>txGain: Transmit gain rate.
 Percent
- III. <<characteristicproperty Allow for sidetones if TRUE.
 >>sidetones: Boolean

Associations

- RobustCodecType:robustify Dependency with robustification code types.
- ContentCodecType: format Dependency with formatting code types.

Operations

No additional operations.

Types and Exceptions

- <<enumeration>>RobustCodecType Set of robustification code types.
- <<enumeration>>FormatCodecType Set of formatting code types.

Mapping

TBD.

Constraints

No additional constraints.

Semantics

No additional semantics.

8.4.3.2.2 IO Mapper Control

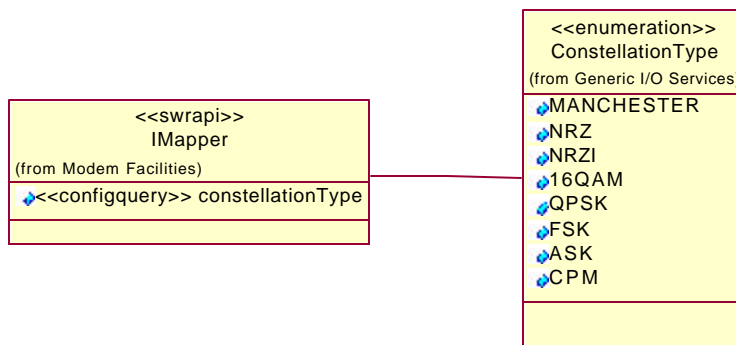


Figure 104: IO Mapper Control

Attributes

<<configurequery>>constellationType: Configured constellation type.
ConstellationType

Associations

- I. ConstellationType:robustify Dependency with constellation types.

Operations

No additional operations.

Types and Exceptions

<<enumeration>>ConstellationType Set of constellation code types.

Mapping

TBD.

Constraints

No additional constraints.

Semantics

No additional semantics.

8.4.3.2.3 IO Filter

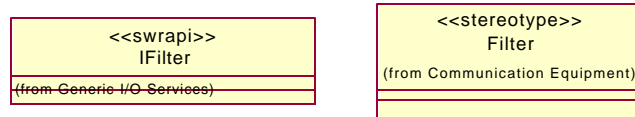


Figure 105: IO Filter

see Communication Equipment.

8.4.3.2.4 IO Amplifier

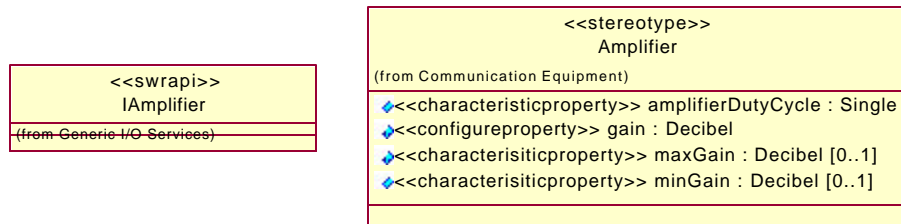


Figure 106: IO Amplifier

see Communication Equipment.

8.4.3.2.5 IO Frame Control

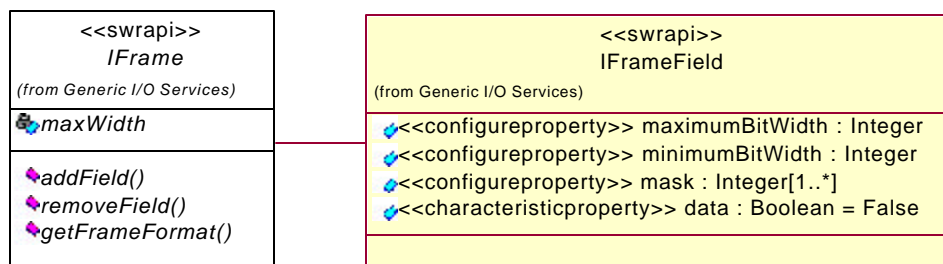


Figure 107: IO Frame Control

8.4.3.3 Security Control Interfaces

Since security is not included in the scope of this document, this paragraph only state requirements that should apply on security control interfaces.

Security Control interfaces allow to manage embedded security functions and provide a basis for security management applications. Security Control interfaces shall be specifically designed to be secure by construction (i.e: cannot be bypassed and do not allow unauthorized users to use them).

At least, Security Control interfaces shall support Multiple Levels of Security (MLS) and provide:

- Clearance control to ensure that radioset service only authorized users,
- Access right control to ensure that authorized users are only allowed to use their allocated rights,
- Confidentiality control to configure ciphering.

8.4.3.4 Flow Interfaces

Flow Interfaces figure interfaces between resources that handle both data with possible inband control.

8.4.3.4.1 Pseudo Devices Flow Interfaces

Flow Interfaces for Pseudo Devices rely upon :

- IPseudoDeviceStream : manages multiple streams,
- IStreamInterface : manages a single stream,
- IStreamEventType : define basic stream event types.

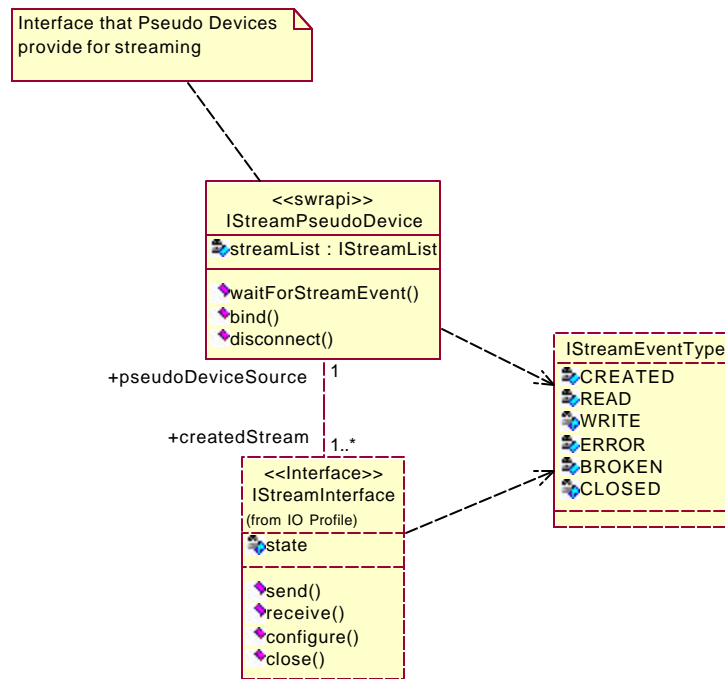


Figure 108: Pseudo Device Stream Interface

8.4.3.4.2 IStreamPseudoDevice

This interface enable to manage multiple streams. Each managed stream convey only payload data : i.e data as provided by external radioset source without any embedded control.

Attributes

<<configurequery>>state: State of current stream.
 StreamStateType

Associations

IStreamPseudoDevice:managed IStreamInterface is managed by a IStreamPseudoDevice.

Operations

I. <<configurequery>>send: Send data with possible inband control.
 IStreamEventType

ISStreamEventType

- | | | |
|------|---|--|
| II. | <<configurequery>>receive:
ISStreamEventType | Receive data with possible inband control. |
| III. | <<configurequery>>configure:
ISStreamEventType | Configure with inband control. |
| IV. | <<configurequery>>close:
ISStreamEventType | Close the stream and disconnect. |

Types and Exceptions

<<enumeration>>StreamException	Streams exceptions.
--------------------------------	---------------------

Mapping

TBD.

Constraints

No additional constraints.

Semantics

No additional semantics.

8.4.3.4.3 ISStreamInterface

Attributes

<<configurequery>>streamList: ISStreamList	List of active streams.
---	-------------------------

Associations

ISStreamInterface:manages	ISStreamPseudoDevice manages multiple ISStream.
---------------------------	---

Operations

- I. <<configurequery>>waitForStream Wait for an event on multiple streams.
 Event: IStreamEventType
- II. <<configurequery>>bind: Create a stream.
 IStreamEventType
- III. <<configurequery>>disconnect: Force a stream to disconnect.
 IStreamEventType

Types and Exceptions

<<enumeration>>StreamException Streams exceptions.

Mapping

TBD.

Constraints

No additional constraints.

Semantics

No additional semantics.

8.4.3.4.4 IStreamEventType

This type define event types that may apply to a stream. Those types are reported by some stream operations.

Attributes

<<enumeration>>CREATED	Stream has just been created.
<<enumeration>>READ	Stream is being read.
<<enumeration>>WRITE	Stream is being written.
<<enumeration>>ERROR	Error is reported on the stream, but flow is not broken (i.e: another stream operation can be invoked).

<<enumeration>>BROKEN	Stream connection is broken (I.e : no other stream operation can be invoked).
<<enumeration>>CLOSED	Stream has been normally closed.

8.4.3.5 Hardware Devices Flow Interfaces

Flow Interfaces for Hardware Devices support a single controlled flow scheme.

Flow Interfaces for Hardware Devices uses the Building Blocks exposed above.

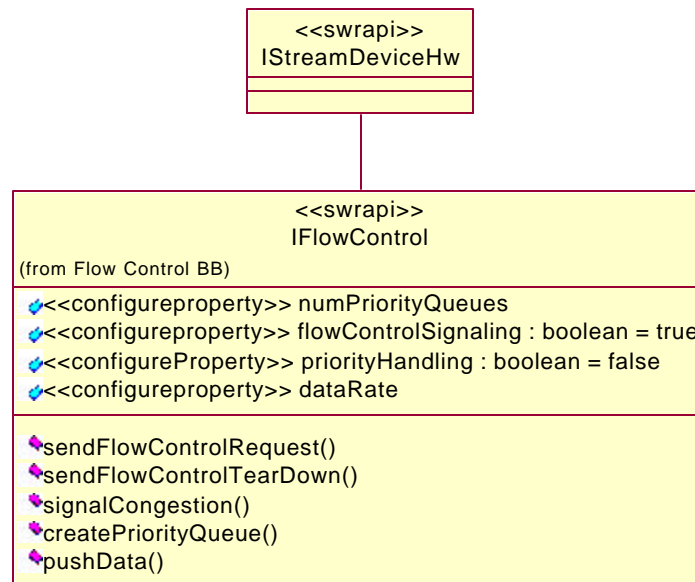


Figure 109: Hardware Device Stream Interface

1.18.5 Physical Layer Facilities

According to OSI, the physical layer's purpose is to "...provides the mechanical, electrical, functional and procedural means to activate, maintain, and de-activate physical-connections for bit transmission between data-link-entities¹⁶". It is the stated goal of the physical layer facilities to provide the necessary interfaces required to implement the functionality specified by the OSI

¹⁶ ITU-T Recommendation X.200 p.49

physical layer. Due to the proposed facilities partitioning, interfaces in the Common Layer Facilities and in the Synchronization Facilities may be required to achieve this objective. Furthermore, depending on waveform complexity, interfaces defined in this package may have to be combined with higher layer facilities such as the Data Link Layer Facilities and eventually with facilities for all other OSI model layers.

Various types of components, such as resources or devices, can implement them. Like with all other interfaces, this submission does not restrict a particular implementation. Finally although the Physical Layer Facilities was designed with the OSI model as framework, it does not impose such a layering on any waveform implementation.

The building block approach supporting this submission effectively separated the interfaces required to implement the physical layer into mainly two set of facilities. The data flow facilities which are common to many layers and the setup and control facilities that are specific to the physical layer.

1.1.48.5.1 Data Transfer

The data transfer services required by the physical layer are provided via the Common Layer Facilities. A physical layer component must realize data transfer interfaces to communicate with the upper OSI layers. Each control interface is optional and only need to be realized if the platform supports the desired functionality. The designer must pick the right interfaces for his platform.

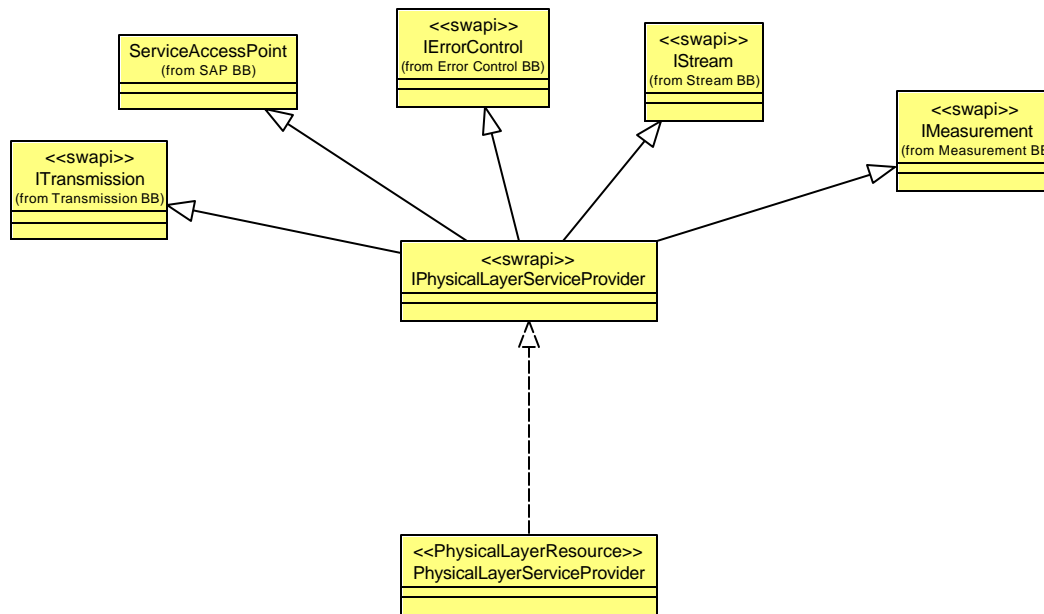


Figure 110 - PhysicalLayerServiceProvider Resource Data Transfer Relationships

1.1.4.18.5.1.1 IPhysicalLayerServiceProvider

Description

The IPhysicalLayerServiceProvider interface inherits from the data transfer interfaces of the common layer facilities. It provides a mean to transfer data to/from a physical layer resource. This interface and the Physical Real-Time APIs of the SCAS perform similar functions.

Attributes

No additional attributes.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

4.1.28.5.2 Control

The Physical Layer Facilities package contains interfaces used to configure and control components performing the physical layer functions of a waveform. Interfaces are defined using relatively high-level concepts. This degree of abstraction enables waveform developers to create waveform applications while abstracting away many of the low-level details of the supporting platform. This can greatly increase the ease and speed at which waveform applications can be developed.

Two functionally separate facilities are part of the Physical Layer Facilities. The first set of facilities is responsible for modem operation while the second set of facilities is used to control the basic devices of the RF/IF processing chain. This separation is representative of the type of technologies used for the implementation of those functionalities. In almost all systems, except those with particularly low RF and low power requirements, the RF/IF chain is not implemented using the same basic technological components as the modem. For example, it is not uncommon to have dedicated hardware in the RF/IF section while the modem section is often composed of programmable components. Technological advances and particular waveform requirements constantly change this boundary; however a boundary is expected to remain due to power, space, and cost efficiency reasons.



Figure 111 – Physical Layer Facilities Overview

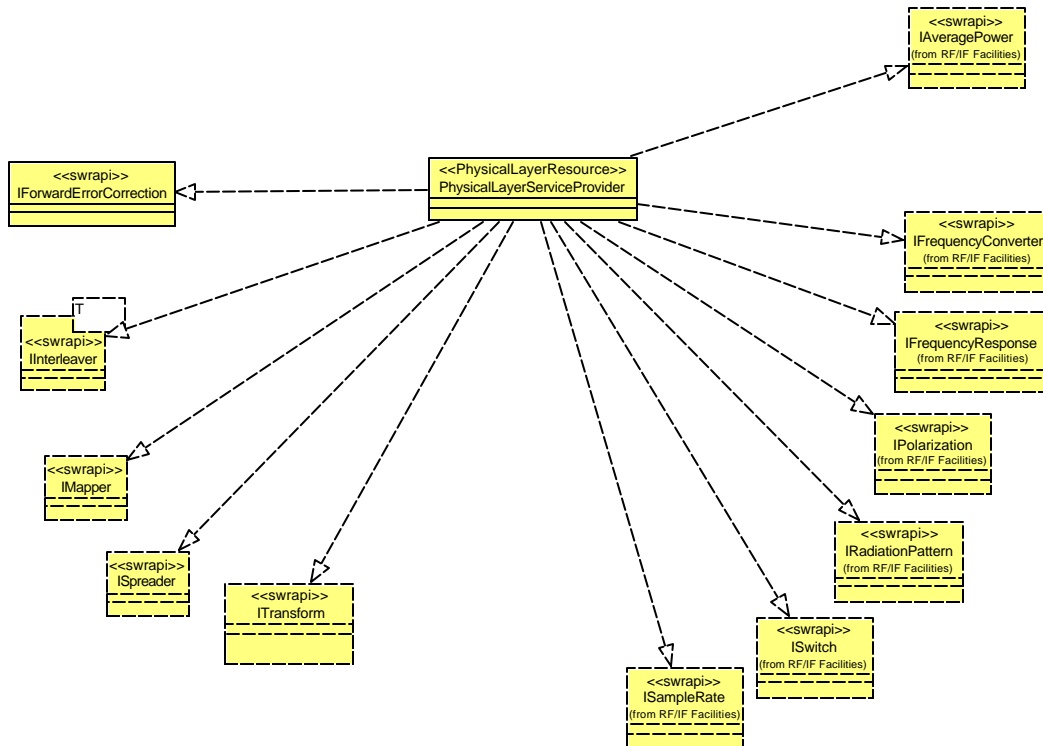


Figure 112 – PhysicalLayerServiceProvider Resource Control Relationships

1.1.2.18.5.2.1 Modem Facilities

The modem facilities include all digital signal processing elements required to convert bits into symbols and vice versa. None of these elements perform pulse shaping or any filtering required to meet the mask. In addition, they do not perform equalization or any other form of channel estimation. These functions are view as part of the RF/IF facilities described in the section 8.5.2.2. The modem is not concerned with the functionalities of the MAC. It will in some cases (CDMA) provide services (i.e. spreader) which can be used by the various MACs to achieve their objectives.

The modem should have the facilities to implement all of today's digital modulation schemes: Direct sequence spread spectrum, QAM, PSK, FSK, ASK, CPM (GMSK...), and OFDM as well as digitally represented analog modulation schemes: AM, FM, and PM.

The modem facilities include more than just simple modulation; they also provide support for forward error correction, interleaving, direct sequence spreading, and Fourier Transforms. In the implementation of a particular modulation scheme, some or even all of these interfaces may be required. Furthermore, the order is flexible. For example, coded modulation (TCM) is viewed as a special arrangement of modulation and forward error correction.

All calls are more specialized versions of the "Modulation Setup" service group of the SCA API supplement for non-real-time interface¹⁷.

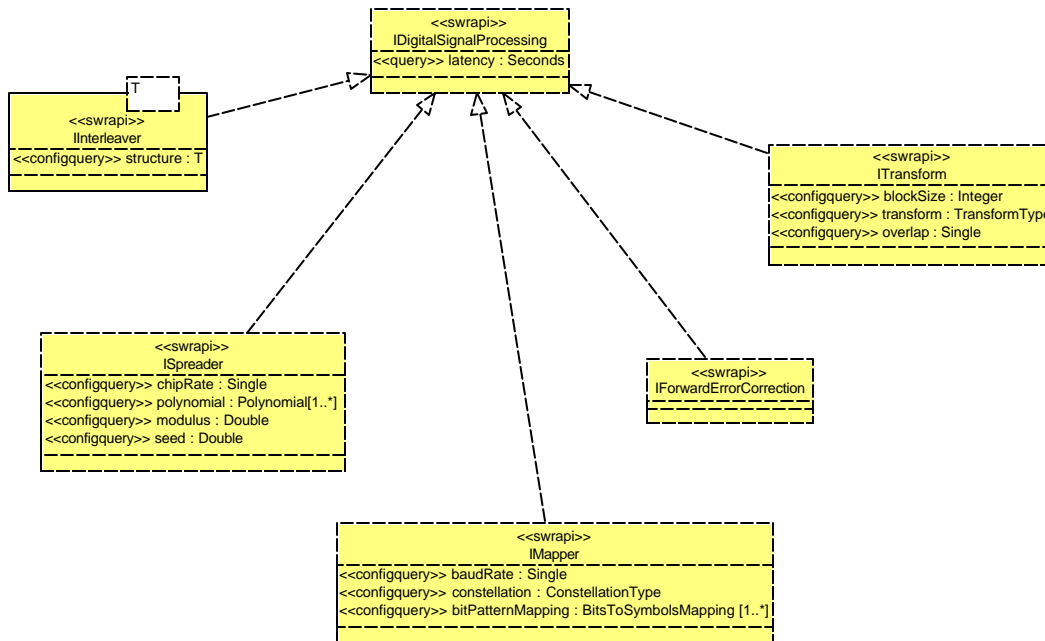


Figure 113 – Modem Facilities Overview

1.1.2.1-18.5.2.1.1 IDigitalSignalProcessing

Description

The IDigitalSignalProcessing interface is used for grouping common attributes to all digital signal processing elements.

Attributes

- <<query>>latency: Seconds The time it takes for an input data element to be carried out to the output of the component.

Associations

No additional associations.

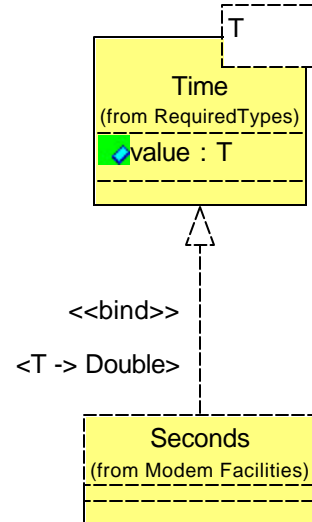
Operations

No additional operations.

Types and Exceptions

¹⁷ API Supplement to the SCAS APPENDIX E Physical Non-Real-Time Building Block Service Definition

- Seconds



Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.2.1.28.5.2.1.2 Interleaver

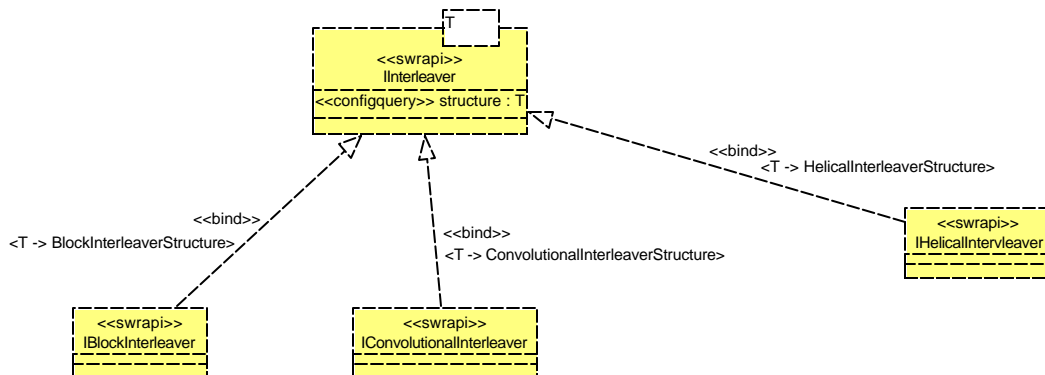


Figure 114 Interleaver Definition

Description

This interface is used to control an interleaver / deinterleaver. An interleaver permutes the incoming bit stream. It does not change the bit rate. Interleavers can be found after any

component of the modulation chain. This can be at the input, after forward error correction, spreading, mapping, and even after having applied a transformation.

Interleavers have different structures and each structure has it's own description method. For example a bloc interleaver is specified by a number of rows and columns while a convolution interleaver uses a vector of delays.

Attributes

- <<configquery>>structure: T The definition of the interleaver structure.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- BlockInterleaverStructure

BlockInterleaverStructure
rows : Integer columns : Integer

- ConvolutionalInterleaverStructure

ConvolutionalInterleaverStructure
delays : Integer [1..*]

- HelicalInterleaverStructure

HelicalInterleaverStructure
columns : Integer groupSize : Integer stepSize : Integer

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.2.1.38.5.2.1.3 IMapper

Description

This interface is used to control a mapper. The mapper operates the transformation from bits, coded or not, to symbols. This transformation can be described completely by a mathematical expression relating the bit input pattern to the corresponding output symbol. It is important to note the units representing the location of the output symbols need not always be amplitudes e.g. (X,Y) coordinates. For example, an FSK mapper would have frequencies as output.

The output of the mapper is at the baud rate.

Attributes

- <<configquery>>baudRate: Single The current baud rate.
- <<configquery>>constellation: String Constellation type used by the mapper.
- <<configquery>>bitPatternMapping: BitsToSymbolsMapping [1..*] The actual definition of the constellation. Each input bit pattern is mapped to one or more dimensional quantities.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- BitsToSymbolsMapping

bitPattern: The actual bit pattern as an integer.

dimensions: The quantity to which the bit pattern is mapped.

BitsToSymbolsMapping
bitPattern : Integer
dimensions : Single [1..*]

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.2.1.48.5.2.1.4 ISpreader

Description

This interface is used to control a spreader. A spreader converts bits, coded or not, into chips. When a spreader is involved, the mapper is considered to convert chips into symbols. A spreader involved the generation of a pseudo random sequence at a chip rate and the multiplying of this sequence with the information sequence (coded or not).

There are many techniques used to generate pseudo random sequences. Much like interleavers, each technique has its own mathematical description method. The generic formula for describing a random sequence generator is¹⁸:

$$X_{n+1} = (a_i X_{n-k}^i + a_{i+1} X_{n-k}^{i-1} \dots + a_{i+2} X_{n-k}^{i-j}) \bmod m$$

Figure 115 – Sequence number generator formula

Attributes

- <<configquery>>chipRate: Single The chip rate of the spreader.
- <<configquery>>polynomial: Polynomial [1..*] The polynomial used to generate the pseudo-random sequence. Each element of the polynomial is an element of the sequence.
- <<configquery>>modulus: Double The value by which the polynomial is divided. i.e. m in Figure 115 – Sequence number generator formula.
- <<configquery>>seed: Double The seed is the first value (X_0) used to calculate the remaining pseudo-random sequence.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- Polynomial

Polynomial
multiplier : Double
exponent : Double
previousValueIndex : Integer

¹⁸ Knuth, Donald E., The Art of Computer Programming, Volume 2

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.2.1.58.5.2.1.5 ITransform

Description

This interface is used to control the transform. The transformations included at this point are FFT and IFFT. These transformations are commonly used for the generation and reception of OFDM and COFDM waveforms.

Attributes

- <<configquery>>blockSize: Integer The block size used by the transform.
- <<configquery>>transform: TransformType The transform type.
- <<configquery>>overlap: Single The overlap of the transform in number of points.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- <<enumeration>>TransformType FFT: Fast Fourier Transform
IFFT: Inverse Fast Fourier Transform

<<enumeration>> TransformType
FFT
IFFT
OTHER

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.2.1.68.5.2.1.6 IForwardErrorCorrection

Description

This interface is used to control a channel coder or decoder. A coder introduces controlled redundancy in the transmit bit stream so the the decoder can detect and possibly correct errors that occurred during the transmission of the signal. A coder will transform an information sequence into a coded sequence. The coded sequence is normally operating at a rate superior to the information sequence. The code rate is the ratio of the input rate over the output rate.

Coders and decoders have very different structures and mathematical ways to describe them.

Attributes

- <<configquery>>codeRate: Single The ratio of the input rate over the output rate $R = N/K$.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional types and exceptions.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.2.28.5.2.2 RF/IF Facilities

The RF/IF Facility is used to configure and control the basic devices of the communication channel. The granularity at which these interfaces are implemented is not specified. For example, at the highest granularity level, the IFrequencyResponse interface can be implemented by a single component for the whole communication channel. The underlying API implementation could then break-up the frequency parameter into smaller frequency responses for configuring individual devices. The waveform application is unaware of the individual devices that make up the communication channel. The interaction point between the waveform and the platform is via

this single interface. On the other hand, at the lowest granularity, each device that makes up the communication channel could implement the interface. In this case, the waveform would have to configure each device with the correct frequency response. Like always, these design choices are left to the implementer. The same scenario could be applied to all interface defined in this package.

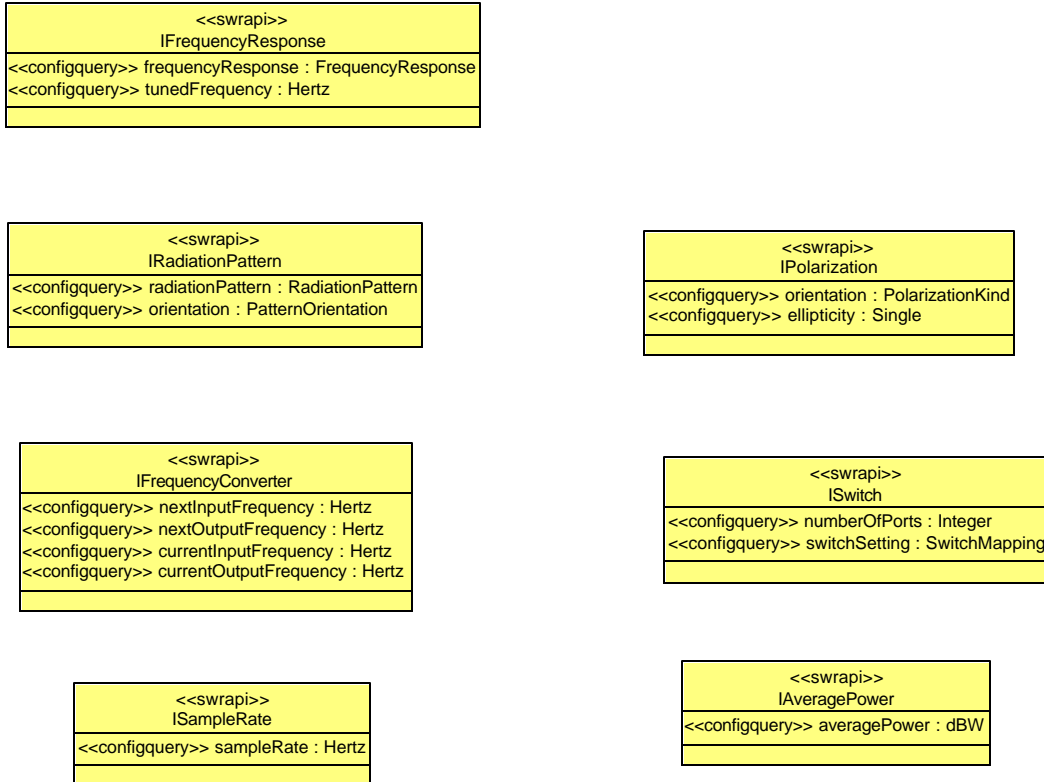


Figure 116 - RF/IF Facilities Overview

1.1.2.2.18.5.2.2.1 IFrequencyResponse

Description

This interface is used to configure the frequency response of a specific component. There are multiple ways of specifying the frequency response. For example, a 1 point frequency response could indicate the 3 dB cut-off of a symmetric spectrum. A 2 points frequency response could be the upper and lower 3 dB cut-off locations for a non symmetric spectrum. Obviously the number of points, the location of the points, and the attenuation and / or phase vary from filter to filter. In some cases, it is the pass band of the filter which is critical while in others it is the stop band. It is left to the designer to specify the key points of each filter with the degree of precision required.

Attributes

- <<configquery>>frequencyResponse: The frequency response of the device. The FrequencyResponsePoint [1..n] frequency response specified is centered at 0 Hz.

- <<configquery>>tunedFrequency: Hertz The frequency at which the frequency response is centered.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- FrequencyResponsePoint

A frequency response is the relation between signal amplitude or gain with frequency. A frequency response with only one point represents a single-sided 3 dB bandwidth. A frequency response with more than one point is an arbitrary frequency response with an arbitrary resolution. A given frequency response has 0 dB gain and is centered at 0 Hz (it does not have to be symmetric).

FrequencyResponsePoint
frequency : Hertz
amplitude : dB
phase : Degrees

- Hertz Number of cycles per second of a given signal.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.2.2.28.5.2.2.2 IRadiationPattern

Description

This interface is used to configure and/or control the radiation pattern of an antenna. The radiation pattern of an antenna is usually represented by the azimuth plane and elevation plane

plots. The radiation pattern is represented with respect to the True North (0 degree) and 0 degree elevation. The orientation of the antenna is also represented with those same measurements.

Attributes

- <<configquery>>radiationPattern: RadiationPattern The radiation pattern of the device.
- <<configquery>>patternOrientation: PatternOrientation The actual pattern orientation which is represented by an azimuth angle and an elevation angle. This attribute represent the actual orientation of the antenna. The antenna can be moved without having to change the radiation pattern.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- RadiationPattern Field intensity variation of an antenna as an angular function with respect to the azimuth and elevation axis.

RadiationPattern
azimuthPlane : RadiationPatternPoint [0..n] elevationPlane : RadiationPatternPoint [0..n]

RadiationPatternPoint
gain : dB angle : Degrees

- PatternOrientation

PatternOrientation
elevation : Degrees azimuth : Degrees

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.2.2.38.5.2.2.3 IPolarization

Description

This interface is used to configure and / or control the polarization parameters of an antenna.

Attributes

- <<configquery>>orientation: PolarizationKind The polarization of the antenna.
- <<configquery>>ellipticity: Single Ratio between the minor and major axis of the ellipse. In the case of right hand or left hand circular. If the ellipticity is 1, this means that the polarization is a perfect circle.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- PolarizationKind The orientation of the RF energy radiated from the device.

<<enumeration>> PolarizationKind
VERTICAL HORIZONTAL RIGHT_CIRCULAR_POLARIZE LEFT_CIRCULAR_POLARIZE

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.2.2.48.5.2.2.4 IFrequencyConverter

Description

This interface is used to configure and / or control a frequency converter. The frequency converter can either be an up converter or a down converter.

Attributes

- <<configquery>>nextInputFrequency:
Hertz The input frequency that the device will select after the next triggering event. This attribute is used for instantaneous frequency changes. Typically in the context of frequency hoping and frequency scanning algorithms.
- <<configquery>>nextOutputFrequency:
Hertz The output frequency that the device will select after the next triggering event. This attribute is used for instantaneous frequency changes. Typically in the context of frequency hoping and frequency scanning algorithms.
- <<configquery>>currentInputFrequency:
Hertz The frequency of the signal currently at the input of the device.
- <<configquery>>currentOutputFrequency:
Hertz The frequency of the signal currently at the output of the device,

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- Hertz Number of cycles per second of a given signal.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.2.2.58.5.2.2.5 ISampleRate

Description

This interface is used to configure and /or control the sample rate of a specific device. Typically, the device is either an analog to digital converter (ADC) or a digital to analog converter (DAC).

Attributes

- <<configquery>>sampleRate: Hertz The number of samples the device takes per second.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- Hertz Number of cycles per second of a given signal.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.2.2.68.5.2.2.6 IAveragePower

Description

This interface is used to configure and / or control the power of a specific device. Typically, the device will be either the power amplifier or a variable gain amplifier used as part of an AGC loop. Note that it is assumed that all other devices are average power neutral. That is they have a gain of 0 dB.

Attributes

- <<configquery>>averagePower: dBW The average power of the device.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- dBW dB referenced to one watt in a 50 ohms load.

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.1.2.2.78.5.2.2.7 ISwitch

Description

This interface is used to control a switch device. A switch simply connects ports together. Although the ports of the switch are considered to be bidirectional, the physical hardware used may only support unidirectional communications. The interface permits one to none, one to one, and one to many interconnection schemes. In the case of one to many, it is preferable to consider the links as unidirectional only because no rules are given for conflict resolution.

Attributes

- <<configquery>>numberOfPorts: Integer Number of ports.
- <<configquery>>switchSetting: SwitchMapping The current configuration of the switch (i.e. which ports are connected together).

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

- SwitchMapping

SwitchMapping
inputPortNumber : Integer
outputPortNumber : Integer

Mapping

To be provided.

Constraints

No additional constraints.

Semantics

No additional semantics.

8.6 Radio Control Facilities

8.6.1 Alarms and Alerts Facility

8.6.2 Audio Facility

8.6.3 Radio Management

This section defines the stereotypes for radio management. Radio management involves the management of the radio and the management of devices and services within a radio. The radio management stereotypes are categorized as domain and device management. The details of each these categories are described in the following subsections.

8.6.3.1 Domain Management

This section defines the stereotypes for radio domain management. The domain management stereotypes are depicted in Table 7 below, which are extensions of the UML Interface and Component constructs. The details of each construct are described in the following subsections. The types of capabilities offered by radio domain management are categorized as follows:

5. Domain Registration Management – provides the mechanism for registering and unregistering services within a radio.
6. Domain Installation Management – provides the mechanism for installing and uninstalling applications within a radio.
7. Domain Retrieval – provides the mechanism for retrieving radio's components.
8. Domain Event Management – provides the mechanism for receiving asynchronous radio's domain event changes.

Table 10 — DomainManagement Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
IDomainEventChannels	Interface	N/A			An Interface that provides Domain Event Management capability to manage domain event channels connections.
IDomainInstallation	Interface	N/A			An Interface that provides Domain Installation Management capability to manage domain application installations.
IDomainRetrieval	Interface	N/A			An Interface that provides Domain

Stereotype	Base Class	Parent	Tags	Constraints	Description
					Retrieval capability to retrieve domain elements.
IDomainRegistration	Interface	N/A			An Interface that provides Domain Registration Management capability to manage domain service registration.
DomainManager	Component	SWRADIO Component			A component that manages a domain.
RadioManager	Component	DomainManager			A component that manages a RadioSet.

1.2.1.1.1 IDomainInstallation

Description

The IDomainInstallation interface, as shown in Figure 57, defines radio domain application installation capabilities. The interface provides the capabilities of adding and removing applications from a radio domain.

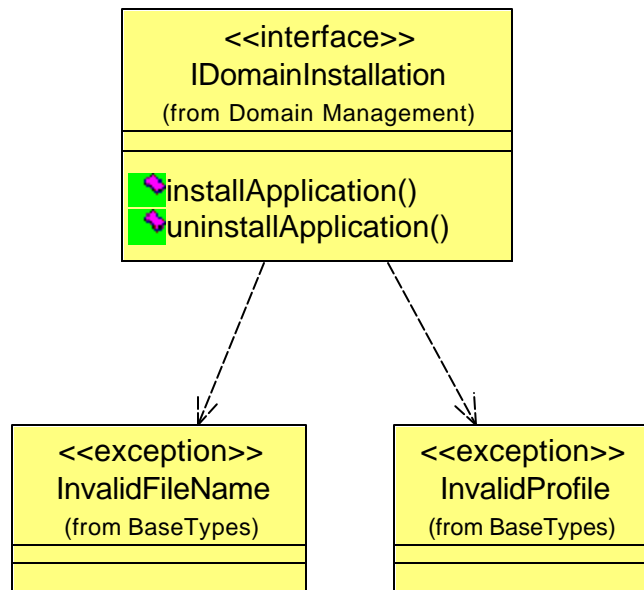


Figure 117 — IDomainInstallation Definition

Attributes

No additional attributes.

Associations

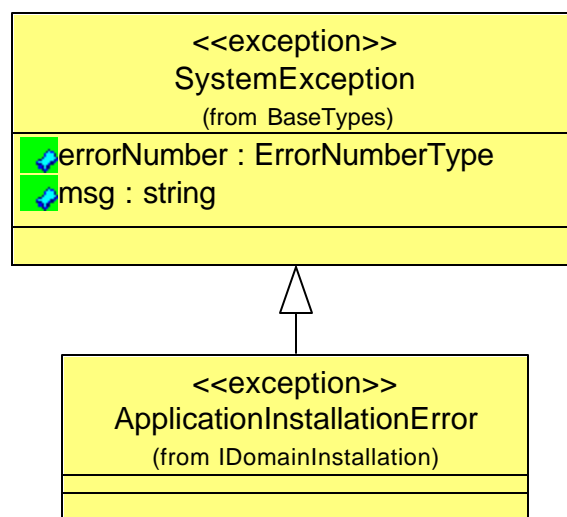
No additional Associations.

Operations

- installApplication(in profileFileName: String): {raises = (InvalidProfile, InvalidFileName, ApplicationInstallationError)}**
 The installApplication operation is used to install new application software in the domain.
- uninstallApplication(in applicationId: String): {raises = (InvalidIdentifier, ApplicationUninstallationError)}**
 The uninstallApplication operation is used to uninstall an ApplicationFactory in the domain. The applicationId corresponds to the identifier in the installed application profile.

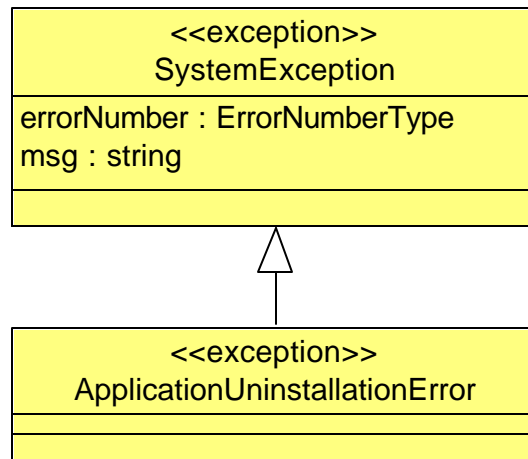
Types and Exceptions

- <<exception>>ApplicationInstallationError** The ApplicationInstallationError exception, a type of System Exception, is raised when an Application installation has not completed correctly. The error number indicates an ErrorNumberType value (e.g., EINVAL, ENAMETOOLONG, ENOENT, ENOMEM, ENOSPC, ENOTDIR, ENXIO). The message is component-dependent, providing additional information describing the reason for the error.



- <<exception>>ApplicationUninstallationError** .The ApplicationUninstallationError exception, a type of SystemException, is raised when an Application uninstallation has not completed correctly. The error number value and the message is component-dependent, providing additional information describing

the reason for the error.



- <<exception>> InvalidIdentifier

The InvalidIdentifier exception indicates an application identifier is invalid.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

An installer service typically invokes these operations when adding or removing an ApplicationFactory (installed ApplicationAssembly) from the domain.

1.2.1.1.2 IDomainEventChannels

Description

The IDomainRegistration interface, as shown in Figure 58, defines radio domain event channel registration capabilities. The interface provides the capabilities of adding and removing connections to event channels in a radio domain.

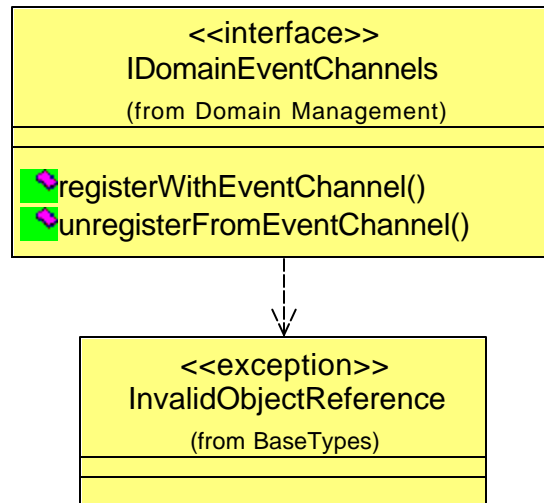


Figure 118 — IDomainEventChannels Definition

Attributes

No additional attributes.

Associations

No additional Associations.

Operations

- **registerWithEventChannel**(in registeringObject: Object, in registeringId: String, in eventChannelName : String): {raises = (InvalidObjectReference, InvalidEventChannelName, AlreadyConnected)}
- **unregisterFromEventChannel**(in unregisteringId: String, in eventChannelName: String): { raises = (InvalidEventChannelName, NotConnected)}

The **registerWithEventChannel** operation is used to connect a consumer to a domain's event channel.

The **registerWithEventChannel** operation raises the **InvalidObjectReference** exception when the input **registeringObject** parameter contains an invalid reference to a event channel consumer type interface.

The **registerWithEventChannel** operation raises the **InvalidEventChannelName** exception when the input **eventChannelName** parameter contains an invalid event channel name.

The **registerWithEventChannel** operation raises **AlreadyConnected** exception when the input parameter contains a connection to the event channel for the input **registeringId** parameter.

The **unregisterFromEventChannel** operation is used to disconnect a consumer from a domain's event channel. The **unregisterFromEventChannel** operation raises the **InvalidEventChannelName** exception when the input **eventChannelName** parameter contains an invalid reference to an event channel.

The **unregisterFromEventChannel** operation

raises the NotConnected exception when the input parameter unregisteringId parameter is not connected to specified input event channel.

Types and Exceptions

- <<exception>>AlreadyConnected
The AlreadyConnected exception indicates that a registering consumer is already connected to the specified event channel.
- <<exception>>InvalidEventChannelName
The InvalidEventChannelName exception indicates that the event channel with that name does not exist within the domain.
- <<exception>> NotConnected
The NotConnected exception indicates that the unregistering consumer was not connected to the specified event channel.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.2.1.1.3 IDomainRegistration

Description

The IDomainRegistration interface, as shown in Figure 59, defines radio domain service registration capabilities. The interface provides the capabilities of adding and removing services from a radio domain.

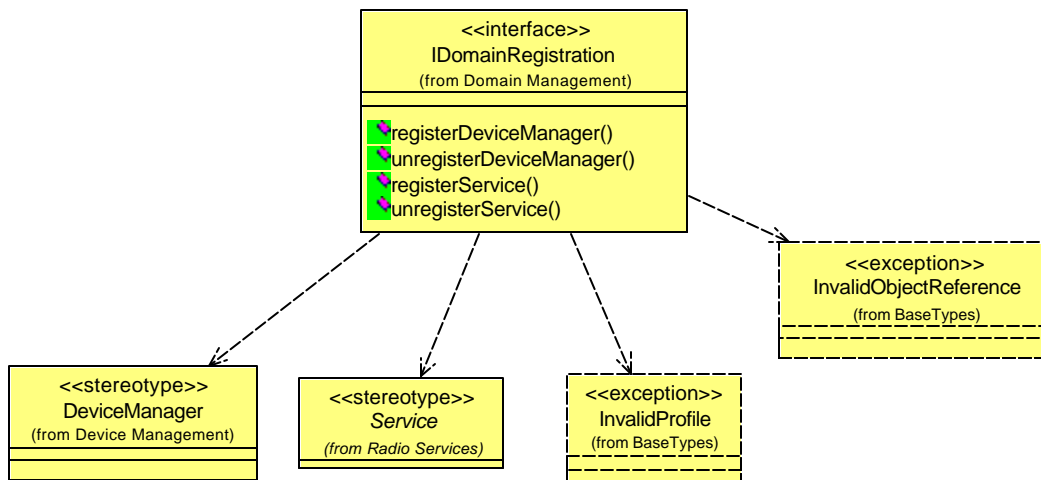


Figure 119 — IDomainRegistration Definition

Attributes

No additional attributes.

Associations

No additional Associations.

Operations

- registerDeviceManager(in deviceMgr: DeviceManager): {raises = (InvalidObjectReference, InvalidProfile, RegisterError)}

The registerDeviceManager operation is used to register a DeviceManager, its Device(s), and its Services. Software profiles can also be obtained from the DeviceManager's FileSystem.
- unregisterDeviceManager(in deviceMgr: DeviceManager): {raises = (InvalidObjectReference, UnregisterError) }

The unregisterDeviceManager operation is used to unregister a DeviceManager component from the DomainManager's Domain Profile. A DeviceManager may be unregistered during run-time for dynamic extraction or maintenance of the DeviceManager.
- registerService(in registeringService: Service, in registeredDeviceMgr: DeviceManager, in name: String): {raises = (InvalidObjectReference, DeviceManagerNotRegistered, RegisterError) }

The *registerService* operation is used to register a service for a specific *DeviceManager* within the domain.
- unregisterService(in unregisteringService: Service, in name: String): {raises = (InvalidObjectReference, UnregisterError)}

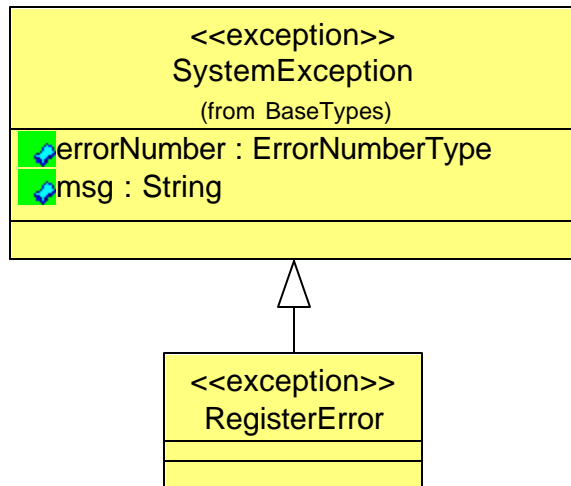
The *unregisterService* operation is used to remove a service entry from the domain.

Types and Exceptions

- <<exception>>DeviceManagerNotRegistered

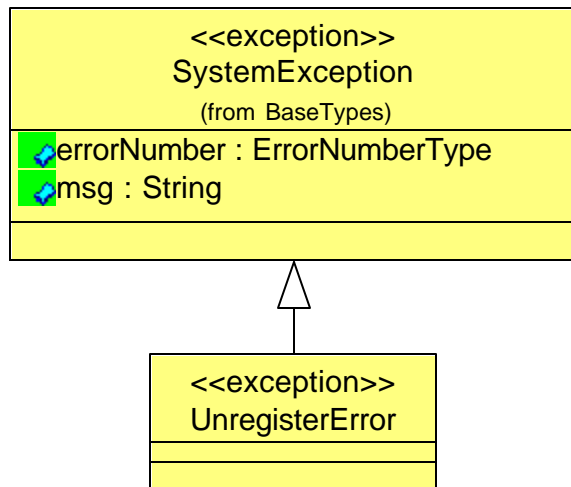
The DeviceManagerNotRegistered exception indicates the registering service's DeviceManager is not registered in the domain. A service's DeviceManager has to be registered prior to a serviceregistration to the domain.
- <<exception>>RegisterError

The RegisterError exception, a type of System Exception, indicates that an internal error has occurred to prevent domain registration operations from successful completion. The error number indicates an ErrorNumberType value. The message is component-dependent, providing additional information describing the reason for the error.



- <<exception>>UnregisterError

The UnregisterError exception, a type of SystemException, indicates that an internal error has occurred to prevent domain unregister operations from successful completion. The error number indicates an ErrorNumberType value. The message is component-dependent, providing additional information describing the reason for the error.



Mapping

N/A.

Constraints

No additional constraints.

Semantics

The IDomainRegistration interface provides the mechanisms for components, such as node managers, to register their services for a specific domain. As services are removed from environment, the interface provides the capability of removing them from the domain.

1.2.1.1.4 IDomainRetrieval

Description

The IDomainRetrieval interface, as shown in Figure 60, defines radio domain retrieval capabilities. The interface provides the capabilities of retrieving domain's components and profile.

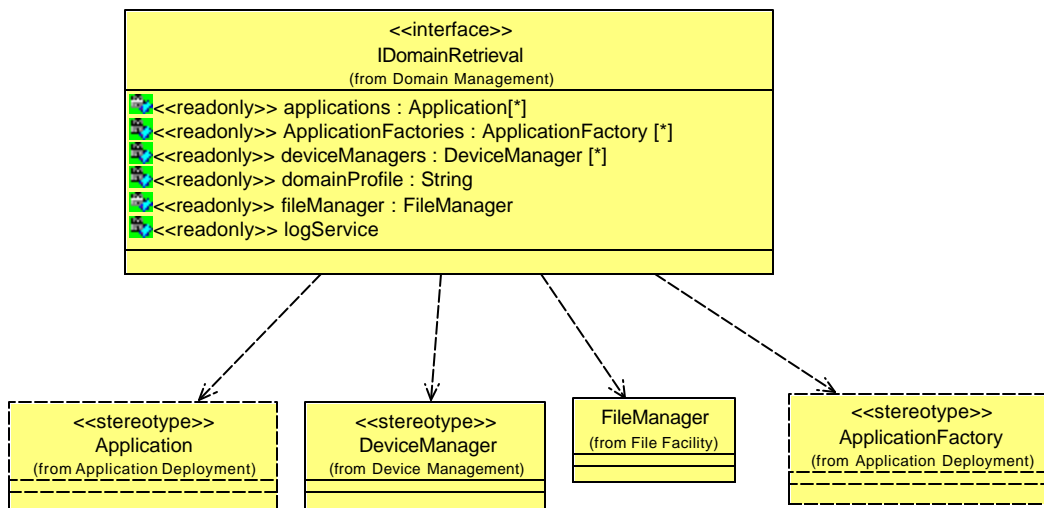


Figure 120 — IDomainRetrieval Definition

Attributes

- <<readonly>>applications : Applications [*]
The readonly applications attribute contains a sequence of instantiated Applications in the domain.
- <<readonly>>applicationFactories : ApplicationFactory [*]
The readonly applicationFactories attribute contains a list ApplicationFactories, with one ApplicationFactory per successfully installed (i.e. no exception raised) application (SAD file and associated files).
- <<readonly>>deviceManagers : DeviceManager [*]
The readonly deviceManagers attribute contains a sequence of registered DeviceManagers in the domain.
- <<readonly>>domainProfile : String
The readonly domainManagerProfile attribute contains either a profile element with a file reference to the DomainManager Configuration Descriptor (DMD) profile or the XML for the DomainManager's (DMD) profile. Files referenced within the profile will have to be obtained from the DomainManager's FileManager.
- <<readonly>>fileManager : FileManager
The readonly fileMgr attribute contains a domain's FileManager.
- <<readonly>>logService : Log
The readonly logService attribute contains a domain's log service.

Associations

No additional associations.

Operations

No additional operations.

Types and Exceptions

No additional Types and Exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

1.2.1.1.5 DomainManager

Description

The DomainManager, as shown in Figure 61, describes the definition and relationships that are common for all Domain Managers.

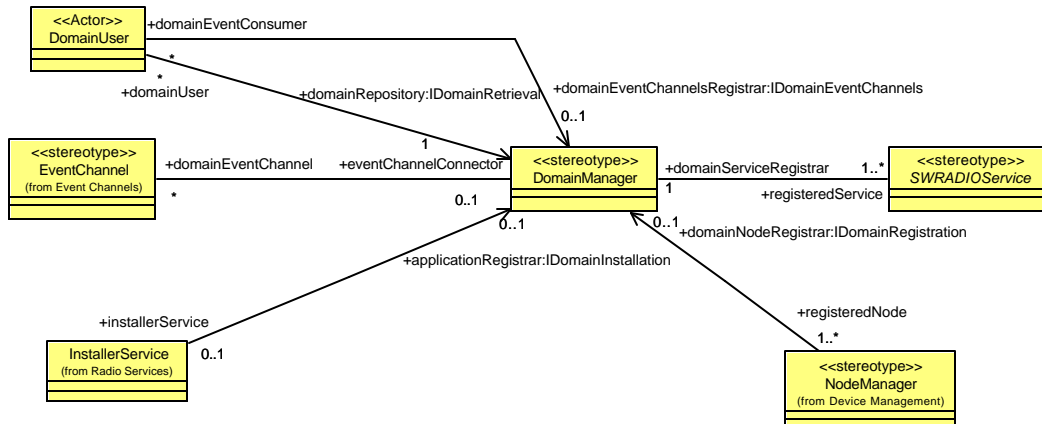


Figure 121 — DomainManager Definition

Attributes

No additional attributes.

Associations

- domainEventChannel: EventChannel [*] A DomainManager may be associated with many event channels.
- domainEventConsumer: A DomainManager may be associated with many users

DomainUser [*]

that register to consume domain events.

- domainUser: DomainUser [*] A DomainManager may be associated with many users that access the system.
- installerService: InstallerService [0..1] A DomainManager may provide the capability to install applications provided by an InstallerService.
- registeredNode: NodeManager [1..*] A DomainManager is associated with one to many nodes that make up the domain.
- registeredService: SWRADIOService [1..*] A DomainManager is associated with one to many services that up the domain.

Operations

No additional Operations.

Types and Exceptions

No additional Types and Exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

The set of interfaces realized by a DomainManager depends on the system the DomainManager is built for. As shown in Figure 61 above, the DomainManager could realize up to four interfaces.

1.2.1.1.6 RadioManager

Description

The RadioManager component, as shown in Figure 62, describes the definition and relationships that are common for radio Domain Managers.

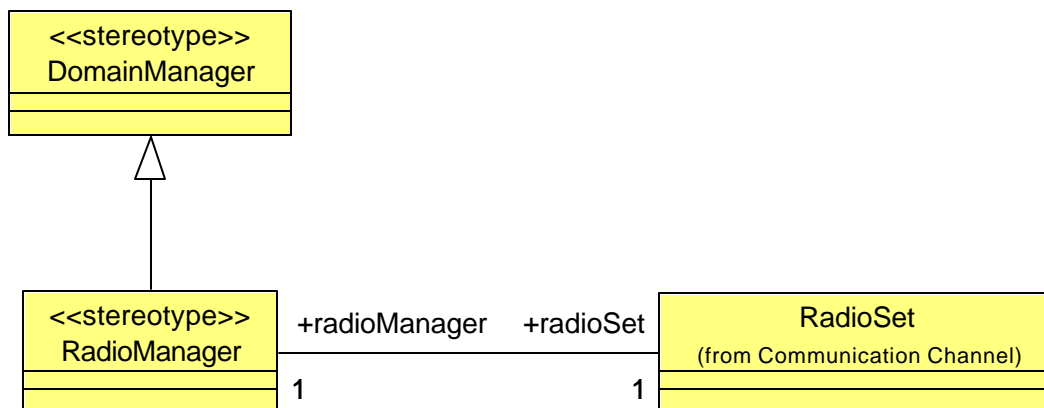


Figure 122 — RadioManager Definition

Attributes

No additional attributes.

Associations

- radioSet: RadioSet [1] A RadioManager is associated with one RadioSet

Operations

No additional Operations.

Types and Exceptions

No additional Types and Exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

No additional semantics.

8.6.3.2 Node Management

This section defines the stereotypes for radio node management. The node management stereotypes are depicted in Table 8 below, which are extensions of the UML Interface and Node constructs. The details of each construct are described in the following subsections. The types of capabilities offered by radio node management are categorized as follows:

3. Service Registration Management – provides the mechanism for registering and unregistering services within a node.
4. Node Retrieval – provides the mechanism for retrieving radio's node components.

Table 11 — Node Management Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
INodeRetrieval	Interface	N/A			An Interface that provides Node Retrieval capability to retrieve node elements.
INodeRegistration	Interface	N/A			An Interface that provides Service Registration Management capability to manage node service

Stereotype	Base Class	Parent	Tags	Constraints	Description
					registration.
NodeManager	Component	SWRADIOComponent			A component that manages a node.
DeviceManager	Component	NodeManager			A component that manages a node and its services.

8.6.3.2.1 INodeRetrieval

Description

The INodeRetrieval, as shown in Figure 63, defines radio node retrieval capabilities. The interface provides the capabilities of retrieving node's components and profile.

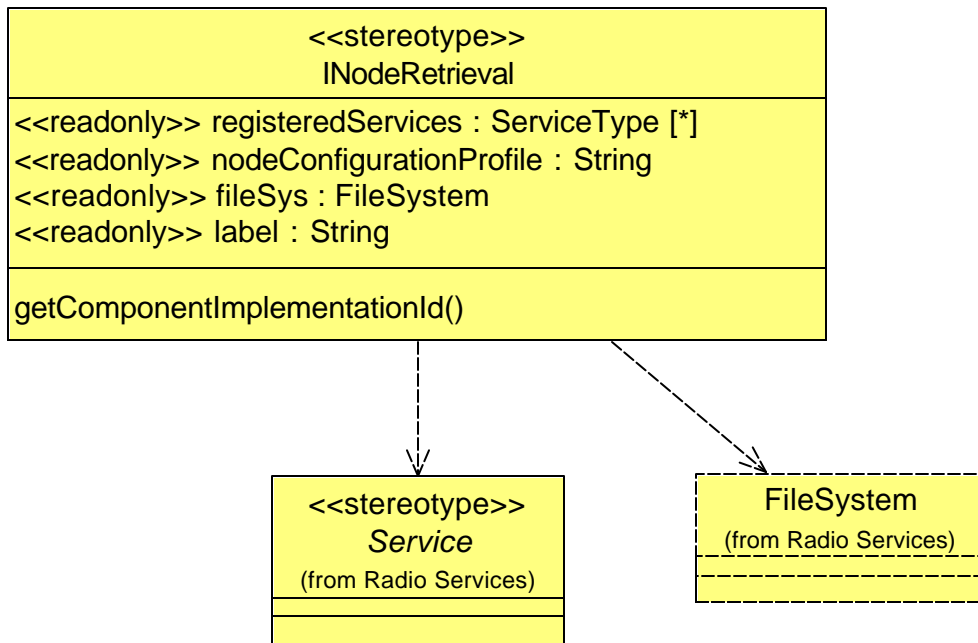


Figure 123 — INodeRetrieval Definition

Attributes

- `<<readonly>>label: String`**

The readonly label attribute contains a node's meaning name.
- `<<readonly>>fileSys: FileSystem`**

The readonly fileSys attribute contains the FileSystem associated with this node or a nil component reference if no FileSystem is associated with this node.
- `<<readonly>>nodeConfigurationProfile : String`**

The readonly nodeConfigurationProfile attribute contains information on the initial configuration for the node. Files referenced within the profile are

obtained using the fileSys attribute.

- <<readonly>>registeredServices : ServiceSequence

The readonly registeredServices attribute contains a list of Services that have registered with this node or a sequence length of zero if no Services have registered with the node.

Associations

No additional associations.

Operations

- GetComponentImplementationId (in componentInstantiationId: String, return String):

The GetComponentImplementationId operation returns the component's (e.g., Service) implementation ID used to create the component identified by the input componentInstantiationId parameter. The implementation ID corresponds to the ServiceExecutableCode used to manifest a Service. The GetComponentImplementationId operation returns an empty string when the input componentInstantiationId parameter does not match any element in the nodeConfigurationProfile. This operation does not raise any exceptions.

Types and Exceptions

- ServiceType

This structure provides the Service reference and name of Service that have registered with the node.

ServiceType
serviceObject : Service
serviceName : String

Mapping

N/A.

Constraints

No additional constraints.

Semantics

The set of interfaces realized by a DomainManager depends on the system the DomainManager is built for. As shown in Figure 61 above the DomainManager could realize up to four interfaces.

8.6.3.2.2 INodeRegistration

Description

The INodeRegistration interface, as shown in Figure 64, defines radio node service registration capabilities. The interface provides the capabilities of adding and removing services from a radio node.

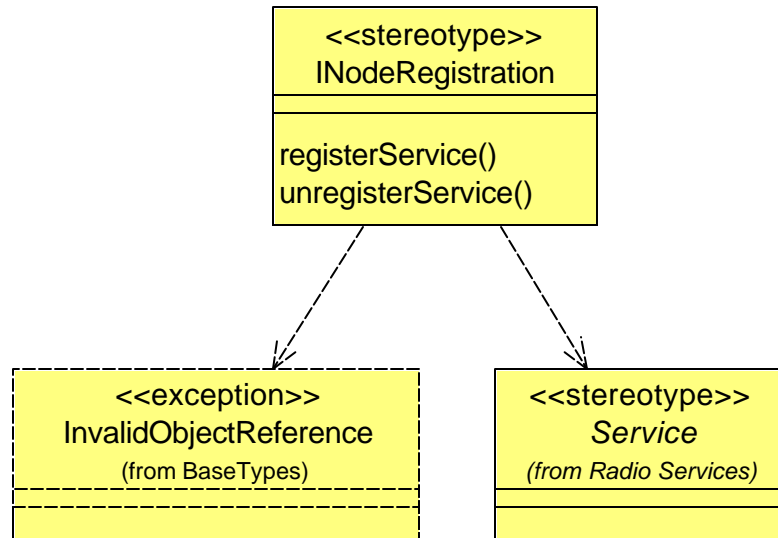


Figure 124 — INodeRegistration Definition

Attributes

No additional attributes.

Associations

No additional associations.

Operations

- registerService(in registeringService: Service, in name: String): {raises = (InvalidObjectReference) }

The registerService operation provides the mechanism to register a Service with a NodeManager and the NodeManager's associated DomainManager. The registeringService is ignored when duplicated. The registerService operation raises the InvalidObjectReference exception when the input registeringService is a nil component reference.

- unregisterService(in unregisteringService: Service, in name: String): {raises = (InvalidObjectReference)}

The unregisterService operation unregisters a Service from a NodeManager and the NodeManager's associated DomainManager. The unregisterService operation raises the InvalidObjectReference when the input registeredService is a nil component reference or does not exist in the node manager.

Types and Exceptions

No additional Types and Exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

The INodeRegistration interface provides the mechanisms for service components started up on a node to register them to the NodeManager that is managing a node. As services are removed from node environment, the interface provides the capability of removing them from the NodeManager.

8.6.3.2.3 DeviceManager

Description

The DeviceManager a type of NodeManager, as shown in Figure 65, defines the definition and relationships that are common for all device managers.

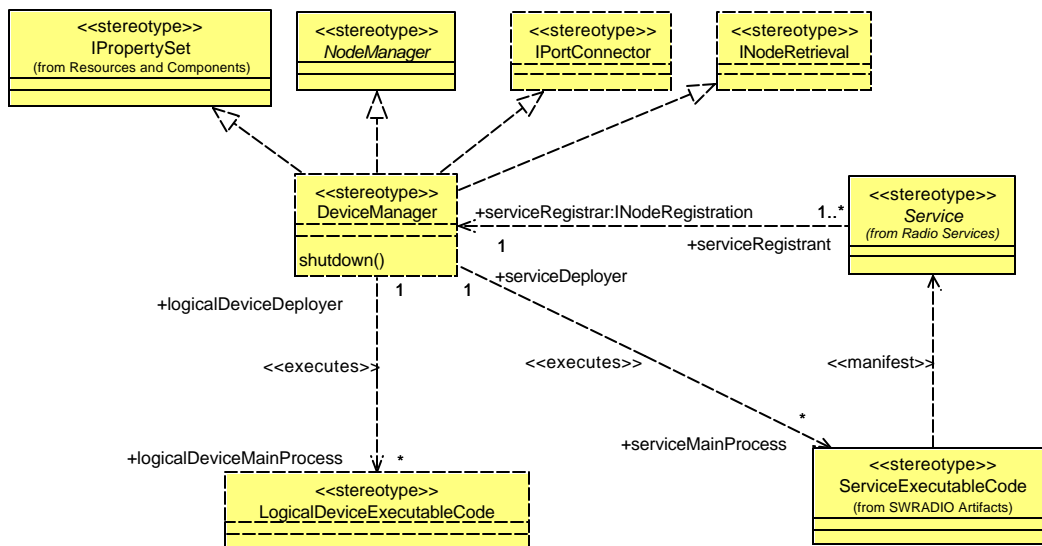


Figure 125 — DeviceManager Definition

Attributes

No additional attributes.

Associations

- serviceMainProcess: ServiceExecutableCode [*]
A DeviceManager may start up LogicalServiceMainProcesses.
- ServiceRegistrant: Service [1..*]
A DeviceManager must be associated with a Service.

Operations

- shutdown ()
The shutdown operation provides the mechanism to terminate a DeviceManager. The shutdown operation unregisters the DeviceManager from the DomainManager.

The shutdown operation performs releaseObject on all of the DeviceManager's registered Devices

(DeviceManager's registeredServices attribute).

The shutdown operation shall cause the DeviceManager to be unavailable from the OS environment when all of the DeviceManager's registered Services are unregistered from the DeviceManager. This operation does not return any value and does not raise any exceptions.

Types and Exceptions

No additional Types and Exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

The DeviceManager provides the capability of starting up services' main processes on given node by the DeviceConfigurationDescriptor. Services started up also include LogicalDevices that are a type of managed Service. A DeviceManager could register to a DomainManager using the the nodeConfigurationProfile information.

8.6.3.2.4 NodeManager

Description

The abstract NodeManager, as shown in Figure 66, defines the definition and relationships that are common for all node managers.

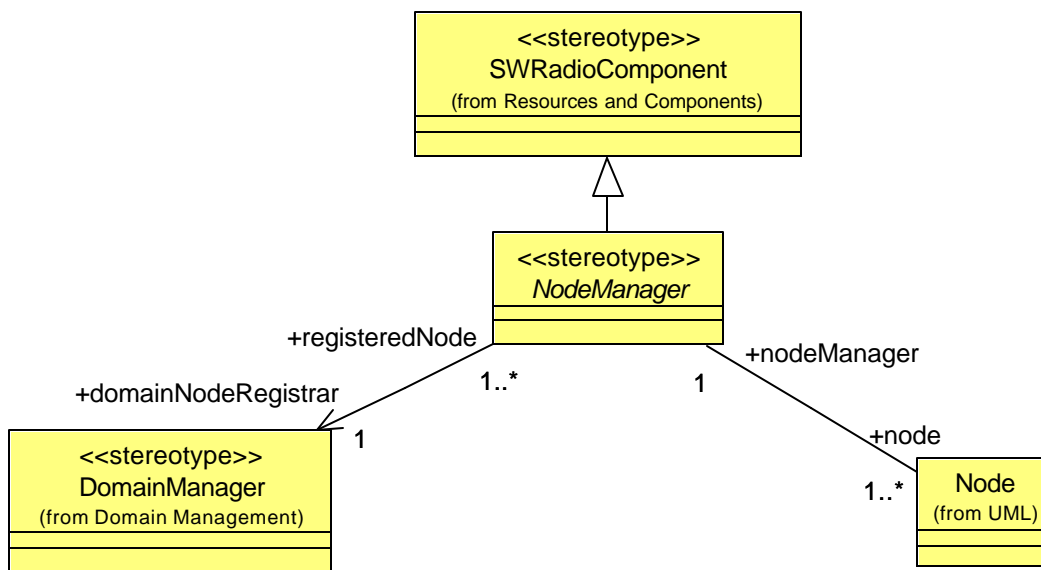


Figure 126 — NodeManager Definition

Attributes

No additional attributes.

Associations

- node: NodeManager [1..*] A NodeManager may manage one to many nodes.
- registeredNode: NodeManager [1..*] A NodeManager is associated with one DomainManager, which harnesses all nodes' services and capabilities.

Operations

No additional Operations.

Types and Exceptions

No additional Types and Exceptions.

Mapping

N/A.

Constraints

No additional constraints.

Semantics

The type of management performed a NodeManager is not specified at this level.

8.6.3.3 Domain Event Channels

8.6.3.3.1 DomainManagementObjectEventType

8.6.3.3.2 DomainManagementObjectAddedEventType

8.6.3.3.3 DomainManagementObjectRemovedEventType

8.6.3.3.4 Event Channel

8.6.3.3.5 EventType

8.6.3.3.6 IncomingDomainEventChannel

8.6.3.3.7 OutgoingDomainEventChannel

8.6.3.3.8 StateChangeEvent

9 Platform Specific Model (PSM)

The SWRADIO PSM consists of CORBA and XML that are based upon the PIM and UML Profile for SWRADIO.

The rule set for transforming UML packages, interfaces, components, types, and exceptions into CORBA constructs are as follows:

1. UML interfaces and interface extensions are map to CORBA interfaces. The CORBA interface names are without the prefix “I” in the interface name as used in the UML profile for SWRADIO and in the PIM Facilities.
2. UML components and component extensions map to CORBA interfaces when the component has operations or attributes that are stereotyped with readonly, readwrite, writeonly.
3. UML exceptions and exception extensions map to CORBA exceptions. There is no subtyping of exceptions in CORBA so TBC.
4. UML attributes that have a cardinality of many [*] map to a CORBA Typedef of sequence types.

The SWRADIO CORBA PSM corresponds to:

1. PIM Facilities

- Common Layer Facilities
- Common Radio Facilities
- Data Link Layer Facilities
- I/O Facilities
- Physical Layer Facilities
- Radio Control Facilities

2. UML Profile for SWRADIO

- Communication Equipment
- Infrastructure
 - a. Radio Management
 - i. Device Management
 - ii. Domain Management
 - b. SWRADIO Deployment
 - i. Applications Deployment
- Applications & Devices
 - a. Base Types
 - b. Devices
 - c. Resources & Components
 - d. Properties

Annex A (informative)

Core Framework CORBA IDL

A.1 Base Types Interfaces

//Source file: F:/FORUMS/OMG/SWRADIO Submission/Rose Models/code/BaseTypes IDL
Implementation Component/BaseTypes.idl

```
#ifndef __CF_BASE_DEFINED
#define __CF_BASE_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

module CF {

    struct DataType {
        string id;
        any value;
    };

    typedef sequence <DataType> Properties;

    /* . */

    exception InvalidObjectReference {
        string msg;
    };

    typedef sequence <octet> OctetSequence;

    typedef sequence <string> StringSequence;

    enum ErrorNumberType {

        CF_NOTSET,
        CF_E2BIG,
        CF_EACCES,
        CF_EAGAIN,
        CF_EBADF,
        CF_EBADMSG,
        CF_EBUSY,
        CF_ECANCELED,
        CF_ECHILD,
        CF_EDEADLK,
        CF_EDOM,
        CF_EEXIST,
        CF_EFAULT,
        CF_EFBIG,
        CF_EINPROGRESS,
        CF_EINTR,
        CF_EINVAL,
        CF_EIO,
```

```
CF_EISDIR,
CF_EMFILE,
CF_EMLINK,
CF_MSGSIZE,
CF_ENAMETOOLONG,
CF_ENFILE,
CF_ENODEV,
CF_ENOENT,
CF_ENOEXEC,
CF_ENOLCK,
CF_ENOMEM,
CF_ENOSPC,
CF_ENOSYS,
CF_ENOTDIR,
CF_NOTEMPTY,
CF_NOTSUP,
CF_NOTTY,
CF_ENXIO,
CF_EPERM,
CF_EPIPE,
CF_ERANGE,
CF_EROFS,
CF_ESPIPE,
CF_ESRCH,
CF_ETIMEDOUT,
CF_EXDEV
};

exception InvalidFileName {
    ErrorNumberType errorNumber;
    string msg;
};

};

#endif
```

A.2 Resource Interfaces

//Source file: F:/FORUMS/OMG/SWRADIO Submission/Rose Models/code/Resource IDL
Implementation Component/CF.idl

```
#ifndef __CF_RESOURCE_DEFINED
#define __CF_RESOURCE_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "BaseTypes.idl"

module CF {

    exception UnknownProperties {
        Properties invalidProperties;
    };

    interface Port {
        exception InvalidPort {
```



```
        unsigned short errorCode;
        string msg;
    };

    exception OccupiedPort {
    };

    void connectPort (
        in Object connection,
        in string connectionId
    )
        raises (InvalidPort,OccupiedPort);

    void disconnectPort (
        in string connectionId
    )
        raises (InvalidPort);

};

interface LifeCycle {
    exception InitializeError {
        StringSequence errorMessages;
    };

    exception ReleaseError {
        StringSequence errorMessages;
    };

    void initialize ()
        raises (InitializeError);

    void releaseObject ()
        raises (ReleaseError);

};

interface TestableObject {
    exception UnknownTest {
    };

    void runTest (
        in unsigned long testid,
        inout Properties testValues
    )
        raises (UnknownTest,UnknownProperties);

};

interface PropertySet {
    exception InvalidConfiguration {
        string msg;
        Properties invalidProperties;
    };

    exception PartialConfiguration {
        Properties invalidProperties;
    };

    void configure (
        in Properties configProperties
```

```
)
raises (InvalidConfiguration,PartialConfiguration);

void query (
    inout Properties configProperties
)
    raises (UnknownProperties);

};

interface PortSupplier {
    exception UnknownPort {
    };

    Object getPort (
        in string name
    )
        raises (UnknownPort);

};

interface Resource : LifeCycle, TestableObject, PropertySet, PortSupplier {
    exception StartError {
        ErrorNumberType errorNumber;
        string msg;
    };

    exception StopError {
        ErrorNumberType errorNumber;
        string msg;
    };

    readonly attribute string identifier;

    void start ()
        raises (StartError);

    void stop ()
        raises (StopError);

};

};

#endif
```

A.3 ResourceFactory Interfaces

//Source file: F:/FORUMS/OMG/SWRADIO Submission/Rose Models/code/Resource Factory
IDL Implementation Component/ResourceFactory.idl

```
#ifndef __CF_RESFACT_DEFINED
#define __CF_RESFACT_DEFINED
```

```
/* CmIdentification
   %X% %Q% %Z% %W% */
```

```
#include "Resource.idl"
```

```
module CF {

    interface ResourceFactory {

        exception InvalidResourceId {
        };

        exception ShutdownFailure {
            string msg;
        };

        exception CreateResourceFailure {
            ErrorNumberType errorNumber;
            string msg;
        };

        readonly attribute string identifier;

        Resource createResource (
            in string resourceId,
            in Properties qualifiers
        )
            raises (CreateResourceFailure);

        void releaseResource (
            in string resourceId
        )
            raises (InvalidResourceId);

        void shutdown ()
            raises (ShutdownFailure);

    };

};

#endif
```

A.4 Device Interfaces

//Source file: F:/FORUMS/OMG/SWRADIO Submission/Rose Models/code/Device IDL
Implementation Component/Device.idl

```
#ifndef __CF_DEFINED
#define __CF_DEFINED
```

```
/* CmIdentification
   %X% %Q% %Z% %W% */
```

```
#include "Resource.idl"
#include "FileServices.idl"
#include "BaseTypes.idl"
```

```
module CF {
    interface AggregateDevice;
```

```
interface Device;

typedef sequence <Device> DeviceSequence;

interface AggregateDevice {
    readonly attribute DeviceSequence devices;

    /*
    @roseuid 3A5DAE9102D6 */
    void addDevice (
        in Device associatedDevice
    )
        raises (InvalidObjectReference);

    /*
    @roseuid 3A5DAE9102D8 */
    void removeDevice (
        in Device associatedDevice
    )
        raises (InvalidObjectReference);
};

interface Device : Resource {
    exception InvalidState {
        string msg;
    };

    exception InvalidCapacity {
        string msg;
        Properties capacities;
    };

    enum AdminType {

        SHUTTING_DOWN,
        LOCKED,
        UNLOCKED
    };

    enum OperationalType {

        DISABLED,
        ENABLED
    };

    enum UsageType {

        ACTIVE,
        BUSY,
        IDLE
    };

    readonly attribute UsageType usageState;
    attribute AdminType adminState;
    readonly attribute OperationalType operationalState;
    readonly attribute string softwareProfile;
    readonly attribute string label;
    readonly attribute AggregateDevice compositeDevice;
```

```
/*
@roseuid 38B7EFD077B0 */
boolean allocateCapacity (
    in Properties capacities
)
    raises (InvalidCapacity,InvalidState);

/*
@roseuid 38B7EFFDDDD48 */
void deallocateCapacity (
    in Properties capacities
)
    raises (InvalidCapacity,InvalidState);

};

};

#endif
```

A.5 File Services Interfaces

//Source file: F:/FORUMS/OMG/SWRADIO Submission/Rose Models/code/File Services IDL Implementation Component/FileServices.idl

```
#ifndef __CF_FILE_DEFINED
#define __CF_FILE_DEFINED

/* CmIdentification
%X% %Q% %Z% %W% */

#include "BaseTypes.idl"

module CF {
    interface File;

    exception FileException {
        ErrorNumberType errorNumber;
        string msg;
    };

    interface FileSystem {

        exception UnknownFileSystemProperties {
            Properties invalidProperties;
        };

        const string SIZE = "SIZE";

        const string AVAILABLE_SIZE = "AVAILABLE_SPACE";

        enum FileType {

            PLAIN,
            DIRECTORY,
            FILE_SYSTEM
        };
    };
};
```

```
struct FileInformationType {
    string name;
    FileType kind;
    unsigned long long size;
    Properties fileProperties;
};

typedef sequence <FileInformationType> FileInformationSequence;

const string CREATED_TIME_ID = "CREATED_TIME";

const string MODIFIED_TIME_ID = "MODIFIED_TIME";

const string LAST_ACCESS_TIME_ID = "LAST_ACCESS_TIME";

void remove (
    in string fileName
)
    raises (FileException,InvalidFileName);

void copy (
    in string sourceFileName,
    in string destinationFileName
)
    raises (InvalidFileName,FileException);

boolean exists (
    in string fileName
)
    raises (InvalidFileName);

FileInformationSequence list (
    in string pattern
)
    raises (FileException,InvalidFileName);

File create (
    in string fileName
)
    raises (InvalidFileName,FileException);

File open (
    in string fileName,
    in boolean read_Only
)
    raises (InvalidFileName,FileException);

void mkdir (
    in string directoryName
)
    raises (InvalidFileName,FileException);

void rmdir (
    in string directoryName
)
    raises (InvalidFileName,FileException);

void query (
    inout Properties fileSystemProperties
```

```
)
raises (UnknownFileSystemProperties);

};

interface File {

    exception IOException {
        ErrorNumberType errorNumber;
        string msg;
    };

    exception InvalidFilePointer {
    };

    readonly attribute string fileName;
    readonly attribute unsigned long filePointer;

    void read (
        out OctetSequence data,
        in unsigned long length
    )
        raises (IOException);

    void write (
        in OctetSequence data
    )
        raises (IOException);

    unsigned long sizeof ()
        raises (FileException);

    void close ()
        raises (FileException);

    void setFilePointer (
        in unsigned long filePointer
    )
        raises (InvalidFilePointer,FileException);

};

interface FileManager : FileSystem {

    struct MountType {
        string mountPoint;
        FileSystem fs;
    };

    typedef sequence <MountType> MountSequence;

    exception NonExistentMount {
    };

    exception InvalidFileSystem {
    };

    exception MountPointAlreadyExists {
    };

    void mount (
```

```
        in string mountPoint,
        in FileSystem file_System
    )
    raises (InvalidFileName,InvalidFileSystem,MountPointAlreadyExists);

void unmount (
    in string mountPoint
)
    raises (NonExistentMount);

MountSequence getMounts ();

};

};

#endif
```

A.6 DeviceManager Interfaces

//Source file: F:/FORUMS/OMG/SWRADIO Submission/Rose Models/code/Device Manager IDL
Implementation Component/DeviceManager.idl

```
#ifndef __CF_DEVMGR_DEFINED
#define __CF_DEVMGR_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "Device.idl"
#include "FileServices.idl"
#include "Resource.idl"
#include "BaseTypes.idl"

module CF {

    interface NodeServicesRegistration {
        /*
        @roseuid 3B338F910156 */
        void registerService (
            in Object registeringService,
            in string name
        )
            raises (InvalidObjectReference);

        /*
        @roseuid 3B338F950007 */
        void unregisterService (
            in Object registeredService,
            in string name
        )
            raises (InvalidObjectReference);

    };

    interface NodeDevicesRegistration {
        /*
        @roseuid 3ACFA4F90122 */
```



```
void registerDevice (
    in Device registeringDevice
)
    raises (InvalidObjectReference);

/*
@roseuid 3ACFA5040000 */
void unregisterDevice (
    in Device registeredDevice
)
    raises (InvalidObjectReference);

};

interface NodeRetrieval {
    struct ServiceType {
        Object serviceObject;
        string serviceName;
    };

    typedef sequence <ServiceType> ServiceSequence;

    readonly attribute string deviceConfigurationProfile;
    readonly attribute FileSystem fileSys;
    readonly attribute string identifier;
    readonly attribute string label;
    readonly attribute DeviceSequence registeredDevices;
    readonly attribute ServiceSequence registeredServices;

    /*
@roseuid 3B45B6E301FB */
    string getComponentImplementationId (
        in string componentInstantiationId
    );

};

interface DeviceManager : PropertySet, PortSupplier, NodeRetrieval,
NodeServicesRegistration, NodeDevicesRegistration {
    /*
@roseuid 3ACFA50E0302 */
    void shutdown ();

};

};

#endif
```

A.7 DomainManager Interfaces

//Source file: F:/FORUMS/OMG/SWRADIO Submission/Rose Models/code/Domain Manager, App
Factory, App IDL Implementation Component/DomainManager.idl

```
#ifndef __CF_DOMMGR_DEFINED
#define __CF_DOMMGR_DEFINED
```

```
/* CmIdentification
```

```
%X% %Q% %Z% %W% */

#include "FileServices.idl"
#include "DeviceManager.idl"
#include "Resource.idl"
#include "BaseTypes.idl"

module CF {
    interface Device;
};

module CF {
    interface Application;
    interface ApplicationFactory;

    exception InvalidProfile {
    };

    struct DeviceAssignmentType {
        string componentId;
        string assignedDeviceId;
    };

    typedef sequence <DeviceAssignmentType> DeviceAssignmentSequence;

    interface Application : Resource {
        struct ComponentProcessIdType {
            string componentID;
            unsigned long processId;
        };

        typedef sequence <ComponentProcessIdType> ComponentProcessIdSequence;

        struct ComponentElementType {
            string componentId;
            string elementId;
        };

        typedef sequence <ComponentElementType> ComponentElementSequence;

        readonly attribute ComponentElementSequence componentNamingContexts;
        readonly attribute ComponentProcessIdSequence componentProcessIds;
        readonly attribute DeviceAssignmentSequence componentDevices;
        readonly attribute ComponentElementSequence componentImplementations;
        readonly attribute string profile;
        readonly attribute string name;
    };

    interface ApplicationFactory {
        exception CreateApplicationRequestError {
            DeviceAssignmentSequence invalidAssignments;
        };

        exception CreateApplicationError {
            ErrorNumberType errorNumber;
            string msg;
        };

        exception InvalidInitConfiguration {
            Properties invalidProperties;
        };
    };
};
```

```
};

readonly attribute string name;
readonly attribute string identifier;
readonly attribute string softwareProfile;

/*
@roseuid 38B7D97BCF98 */
Application create (
    in string name,
    in Properties initConfiguration,
    in DeviceAssignmentSequence deviceAssignments
)
    raises
(CreateApplicationError,CreateApplicationRequestError,InvalidInitConfiguration);

};

interface DomainRetrieval {
    typedef sequence <Application> ApplicationSequence;

    typedef sequence <ApplicationFactory> ApplicationFactorySequence;

    typedef sequence <DeviceManager> DeviceManagerSequence;

    readonly attribute string domainProfile;
    readonly attribute DeviceManagerSequence deviceManagers;
    readonly attribute ApplicationSequence applications;
    readonly attribute ApplicationFactorySequence applicationFactories;
    readonly attribute FileManager fileMgr;
    readonly attribute string logService;
    attribute string identifier;
};

interface DomainRegistration {
    exception DeviceManagerNotRegistered {
    };

    exception UnregisterError {
        ErrorNumberType errorNumber;
        string msg;
    };

    exception RegisterError {
    };

    /*
@roseuid 364B4D632938 */
    void registerDeviceManager (
        in DeviceManager deviceMgr
    )
        raises (InvalidObjectReference,InvalidProfile,RegisterError);

    /*
@roseuid 364B4EAB15E0 */
    void unregisterDeviceManager (
        in DeviceManager deviceMgr
    )
        raises (InvalidObjectReference,UnregisterError);

    /*
```

```
@roseuid 364B4CF92ED0 */
void registerDevice (
    in Device registeringDevice,
    in DeviceManager registeredDeviceMgr
)
    raises
(InvalidObjectReference,InvalidProfile,DeviceManagerNotRegistered,RegisterError);

/*
@roseuid 364B4EC8DEC0 */
void unregisterDevice (
    in Device unregisteringDevice
)
    raises (InvalidObjectReference,UnregisterError);

/*
@roseuid 3B33926D032F */
void registerService (
    in Object registeringService,
    in DeviceManager registeredDeviceMgr,
    in string name
)
    raises
(InvalidObjectReference,InvalidProfile,DeviceManagerNotRegistered,RegisterError);

/*
@roseuid 3B3392750114 */
void unregisterService (
    in Object unregisteringService,
    in string name
)
    raises (InvalidObjectReference,UnregisterError);

};

interface DomainInstallation {
    exception ApplicationInstallationError {
        ErrorNumberType errorNumber;
        string msg;
    };

    exception ApplicationUninstallationError {
        ErrorNumberType errorNumber;
        string msg;
    };

    exception InvalidIdentifier {
    };

/*
@roseuid 3896F0D83588 */
void installApplication (
    in string profileFileName
)
    raises (InvalidProfile,InvalidFileName,ApplicationInstallationError);

/*
@roseuid 3896F13747C8 */
void uninstallApplication (
    in string applicationId
)
```

```
        raises (InvalidIdentifier,ApplicationUninstallationError);

};

interface DomainEventChannels {
    exception InvalidEventChannelName {
    };

    exception AlreadyConnected {
    };

    exception NotConnected {
    };

    /*
    @roseuid 3BB49D7901AE */
    void registerWithEventChannel (
        in Object registeringObject,
        in string registeringId,
        in string eventChannelName
    )
        raises (InvalidObjectReference,InvalidEventChannelName,AlreadyConnected);

    /*
    @roseuid 3BB49DF903CF */
    void unregisterFromEventChannel (
        in string unregisteringId,
        in string eventChannelName
    )
        raises (InvalidEventChannelName,NotConnected);

};

};
#endif
```

Annex B
(normative)

Common Layer Facilities CORBA IDL

B.1 TBD

B.2 TBD

Annex C
(normative)

Common Radio Facilities CORBA IDL

C.1 File Facilities Interfaces

C.1.1 File Interface

C.1.2 FileSystem Interface

C.1.3 FileManager Interface

Annex D
(normative)

Data Link Layer Facilities CORBA IDL

D.1 Link Layer Interfaces

D.2 MAC Interfaces

Annex E
(normative)

Input/Output Facilities CORBA IDL

E.1 Serial Interfaces

E.2 Audio Interfaces

Annex F
(normative)

Physical Layer Facilities CORBA IDL

F.1 RF/IF Interfaces

Annex G
(normative)

Radio Control Facilities CORBA IDL

G.1 Radio Management Interfaces

Annex H (normative)

Operating System Profile

10 SCA Application Environment Profile

10.1 Scope.

This appendix defines an application environment profile (AEP) for the SCA, based on Standardized Application Environment Profile - POSIX[®] Realtime Application Support (AEP), IEEE Std 1003.13-1998.

10.2 Standards.

The following standards are required in whole or in part by the SCA AEP profile.

Table 10-1. Required Standards

Standard	SCA AEP
C Standard (ISO/IEC 9899:1990)	PRT
POSIX.1 (ISO/IEC 9945 -1):1997	PRT
POSIX.1b (ISO/IEC 9945 -1):1997	PRT
POSIX.1c (ISO/IEC 9945 -1):1997	PRT
POSIX.5b (IEEE 1003.5 - 1992)	OPT

NOTE:

PRT — Partial, only the subset or options or Units of Functionality called out in A.3.

MAN — Mandatory, complete with all options.

OPT — Optional, may be included in the environment.

10.3 Constraints.

The real-time profile defined in this standard requires only specific Units of Functionality of the required standards. The absence of particular elements of these standards introduces constraints on the use of some of the features of particular functions. This clause defines the constraints that an application strictly conforming to one of the profiles shall observe when using each of the functions required by that profile.

An Ada AEP has not been explicitly defined. Any Ada application shall be restricted to using the equivalent Ada functionality, as defined in POSIX.5b, designated as mandatory by this profile or may use the C interface.

[®] POSIX is a registered trademark of the Institute of Electrical and Electronics Engineers, Inc.

10.3.1 POSIX.1.**Table 10-2. POSIX.1 Option Requirements**

Option	SCA AEP
{NGROUPS_MAX}	-
{_POSIX_CHOWN_RESTRICTED}	NRQ
{_POSIX_JOB_CONTROL}	NRQ
{_POSIX_NO_TRUNC}	PRI
{_POSIX_SAVED_IDS}	NRQ
{_POSIX_VDISABLE}	NRQ

NOTE:

NRQ — Not required for this profile.

PRI — The primary file system shall generate an error for pathname components longer than NAME_MAX. The user is responsible for semantics of other file systems that may be mounted.

10.3.1.1 Single Process Function Behavior.

The functions in Table B-3 shall behave as described in the referenced clauses.

Table 10-3. POSIX_SINGLE_PROCESS Functions

Function	Reference in POSIX.1	SCA AEP
sysconf ()	4.8.1	NRQ
uname()	4.4.1	NRQ
time()	4.5.1	MAN

NRQ — Not required for this profile.

MAN — Mandatory for this profile.

10.3.1.2 Multi-Process Function Behavior.

The functions listed in Table B-4 shall behave as described in the referenced clauses.

Table B-4. - POSIX_MULTI_PROCESS Functions

Function	Reference in POSIX.1	SCA AEP
execl ()	3.1.2	NRQ
execv ()	3.1.2	NRQ
execle ()	3.1.2	NRQ
execve ()	3.1.2	NRQ
execvp ()	3.1.2	NRQ
execv ()	3.1.2	NRQ
_exit ()	3.2.2	NRQ
fork()	3.1.1	NRQ
getenv ()	4.6.1	NRQ
getpid ()	4.1.1	NRQ
getppid ()	4.1.1	NRQ
sleep ()	3.4.3	NRQ
times ()	4.5.2	NRQ
wait()	3.2.1	NRQ
waitpid ()	3.2.1	NRQ
assert ()	8.1, 8.2, 8.3	NRQ
exit ()	8.1, 8.2, 8.3	NRQ
setlocale ()	8.1, 8.2, 8.3	MAN

MAN — Mandatory for this profile.

NRQ — Not Required for this profile.

10.3.1.3 Job Control Function Behavior.

The functions listed in Table B-5 shall behave as described in the referenced clauses.

Table B-5. POSIX_JOB_CONTROL Functions

Function*	Reference in POSIX.1	SCA AEP
setpgid()	4.3.3	NRQ
tcgetpgrp()	7.2.3	NRQ
tcsetpgrp()	7.2.4	NRQ
*	7.1.1.4	NRQ

NOTE:

NRQ — Not required for this profile.

* — Further functionality is also defined here.

10.3.1.4 Signals Function Behavior.

The functions listed in Table B-6 shall behave as described in the referenced clauses, except for the following constraints:

(1) An application strictly conforming to SCA AEP shall be considered erroneous if any signal results in abnormal termination of the process because these profiles do not support multiple processes.

(2) An application strictly conforming to SCA AEP shall not call the kill() function with a negative argument because these profiles do not require process group functionality.

Table B-6. POSIX_SIGNALS Functions

Function	Reference in POSIX.1	SCA AEP
alarm(*)	3.4.1	NRQ
kill()	3.3.2	MAN
pause()	3.4.2	MAN
sigaction()	3.3.4	MAN
sigaddset()	3.3.3	MAN
sigdelset()	3.3.3	MAN
sigemptyset()	3.3.3	MAN
sigfillset()	3.2.3	MAN
sigismember()	3.3.3	MAN
sigpending()	3.3.6	MAN
sigprocmask()	3.3.5	MAN
sigsuspend()	3.3.7	MAN

abort()	8.1,8.2,8.3	MAN
siglongjmp()	8.1,8.2,8.3	NRQ
sigsetjmp()	8.1,8.2,8.3	NRQ

NOTE:

MAN — Mandatory for this profile.

NRQ — Not Required for this profile.

*Functionality provided through the POSIX timers.

10.3.1.5 User Group Function Behavior.

The functions listed in Table B-7 shall behave as described in the referenced clauses.

Table B-7. POSIX_USER_GROUPS Functions

Function	Reference in POSIX.1	SCA AEP
getegid()	4.2.1	NRQ
geteuid()	4.2.1	NRQ
getgid()	4.2.1	NRQ
getgroups()	4.2.3	NRQ
getlogin()	4.2.4	NRQ
getpgrp()	4.3.1	NRQ
getuid()	4.2.1	NRQ
setuid()	4.2.2	NRQ
setsid()	4.3.2	NRQ
setgid()	4.2.2	NRQ

NOTE:

NRQ — Not required for this profile.

10.3.1.6 File System Function Behavior.

The functions listed in Table B-8 shall behave as described in the referenced clauses.

Table B-8. POSIX_FILE_SYSTEM Functions

Function	Reference in POSIX.1	SCA AEP
access()	5.6.3	MAN
chdir()	5.2.1	MAN
closedir()	5.1.2	MAN
creat()	5.3.2	MAN
fpathconf()	5.7.1	MAN
fstat()	5.6.2	MAN
getcwd()	5.2.2	MAN
link()	5.3.4	MAN
mkdir()	5.4.1	MAN
opendir()	5.1.2	MAN
pathconf()	5.7.1	MAN
readdir()	5.1.2	MAN
rename()	5.5.3	MAN
rewinddir()	5.1.2	MAN
rmdir()	5.5.2	MAN
stat()	5.6.2	MAN
unlink()	5.5.1	MAN
utime()	5.6.6	MAN
remove()	8.1, 8.2, 8.3	MAN
rename()	8.1, 8.2, 8.3	MAN
tmpfile()	8.1, 8.2, 8.3	MAN
tmpnam()	8.1, 8.2, 8.3	MAN

NOTE:

MAN — Mandatory for this profile.

10.3.1.7 File Attributes Function Behavior.

The functions listed in Table B-9 shall behave as described in the referenced clauses, except for the following constraint:

(1) An application strictly conforming to SCA AEP shall be guaranteed that the file mode creation mask for any object created by any process is `SS--IIRRWXXUU`; that is, the object shall be fully accessible to the creator.

Table 10-9. POSIX_FILE_ATTRIBUTES Functions

Function	Reference in POSIX.1	SCA AEP
chmod()	5.6.4	NRQ
chown()	5.6.5	NRQ
umask()	5.3.3	NRQ

NOTE:

NRQ — Not required for this profile.

10.3.1.8 File and Directory Management Function Behavior.

The functions listed in Table B-10 shall behave as described in the referenced clauses.

Table B-10. POSIX_FD_MGMT Functions

Function	Reference in POSIX.1	SCA AEP
dup()	6.2.1	NRQ
dup2()	6.2.1	NRQ
fcntl()	6.5.2	NRQ
lseek()	6.5.3	MAN
fseek()	8.1, 8.2, 8.3	MAN
ftell()	8.1, 8.2, 8.3	MAN
rewind()	8.1, 8.2, 8.3	MAN

NOTE:

NRQ — Not required for this profile.

MAN — Mandatory for this profile.

10.3.1.9 Device I/O Function Behavior.

The functions listed in Table B-11 shall behave as described in the referenced clauses.

Table B-11. POSIX_DEVICE_IO Functions

Function	Reference in POSIX.1	SCA AEP
close()	6.3.1	MAN
open()	5.3.1	MAN
read()	6.4.1	MAN
write()	6.4.2	MAN
clearerr()	8.1, 8.2, 8.3	MAN
fclose()	8.1, 8.2, 8.3	MAN
fdopen()	8.1, 8.2, 8.3	MAN
feof()	8.1, 8.2, 8.3	MAN
ferror()	8.1, 8.2, 8.3	MAN
fflush()	8.1, 8.2, 8.3	MAN
fgetc()	8.1, 8.2, 8.3	MAN
fileno()	8.1, 8.2, 8.3	MAN
fgets()	8.1, 8.2, 8.3	MAN
fopen()	8.1, 8.2, 8.3	MAN
fprintf()	8.1, 8.2, 8.3	MAN
fputc()	8.1, 8.2, 8.3	MAN
fputs()	8.1, 8.2, 8.3	MAN
fread()	8.1, 8.2, 8.3	MAN
freopen()	8.1, 8.2, 8.3	MAN
fscanf()	8.1, 8.2, 8.3	MAN
fwrite()	8.1, 8.2, 8.3	MAN
getc()	8.1, 8.2, 8.3	MAN
getchar()	8.1, 8.2, 8.3	MAN

gets()	8.1, 8.2, 8.3	MAN
perror()	8.1, 8.2, 8.3	MAN
printf()	8.1, 8.2, 8.3	MAN
putc()	8.1, 8.2, 8.3	MAN
putchar()	8.1, 8.2, 8.3	MAN
puts()	8.1, 8.2, 8.3	MAN
scanf()	8.1, 8.2, 8.3	MAN
setbuf()	8.1, 8.2, 8.3	MAN
sprintf()	8.1, 8.2, 8.3	MAN
sscanf()	8.1, 8.2, 8.3	MAN
ungetc()	8.1, 8.2, 8.3	MAN

NOTE:

MAN — Mandatory for this profile.

10.3.1.10 Device-Specific Function Behavior.

The functions listed in Table G-12 shall behave as described in the referenced clauses.

Table B-12. POSIX_DEVICE_SPECIFIC Functions

Function	Reference in POSIX.1	SCA AEP
cfgetispeed()	7.1.3	NRQ
cfgetospeed()	7.1.3	NRQ
cfsetispeed()	7.1.3	NRQ
cfsetospeed()	7.1.3	NRQ
ctermid()	4.7.1	NRQ
isatty()	4.7.2	NRQ
tcdrain()	7.2.2	NRQ
tcflush()	7.2.2	NRQ
tcflow()	7.2.2	NRQ
tcgetattr()	7.2.1	NRQ
tcsendbreak()	7.2.2	NRQ
tcsetattr()	7.2.1	NRQ
ttyname()	4.7.2	NRQ

NOTE:

NRQ — Not required for this profile.

10.3.1.11 System Database Function Behavior.

The functions listed in Table B-13 shall behave as described in the referenced clauses.

Table B-13. POSIX_SYSTEM_DATABASE Functions

Function	Reference in POSIX.1	SCA AEP
getgrgid()	9.2.1	NRQ
getgrnam()	9.2.1	NRQ
getpwnam()	9.2.2	NRQ

getpwuid()	9.2.2	NRQ
------------	-------	-----

NOTE:

NRQ — Not required for this profile.

10.3.1.12 Pipe Function Behavior.

The function listed in Table 10-14 shall behave as described in the referenced clause.

Table 10-14. POSIX_PIPE_Function

Function	Reference in POSIX.1	SCA AEP
pipe()	6.1.1	NRQ

NOTE:

NRQ — Not required for this profile.

10.3.1.13 FIFO Function Behavior.

The function listed in Table B-15 shall behave as described in the referenced clause.

Table B-15. POSIX_FIFO Function

Function	Reference in POSIX.1	SCA AEP
mkfifo()	5.4.2	NRQ

NOTE:

NRQ — Not required for this profile.

10.3.1.14 C Language-Specific Services Behavior.

The functions listed in Table B-16, Table B-17, Table B-18, Table B-19, Table B-20, and Table B-21 shall behave as described in the referenced clauses.

Table B-16. POSIX_C_LANG_SUPPORT Character Handling Functions

Function	Reference in the C Standard	SCA AEP
isalnum()	4.3.1.1	MAN
isalpha()	4.3.1.2	MAN
isctrl()	4.3.1.3	MAN
isdigit()	4.3.1.4	MAN
isgraph()	4.3.1.5	MAN
islower()	4.3.1.6	MAN
isprint()	4.3.1.7	MAN
ispunct()	4.3.1.8	MAN
isspace()	4.3.1.9	MAN
isupper()	4.3.1.10	MAN
isxdigit()	4.3.1.11	MAN
tolower()	4.3.2.1	MAN
toupper()	4.3.2.2	MAN

NOTE:

MAN — Mandatory for this profile.

Table B-17. POSIX_C_LANG_SUPPORT Mathematical Functions

Function	Reference in the C Standard	SCA AEP
acos()	4.5.2.1	MAN
asin()	4.5.2.2	MAN
atan()	4.5.2.3	MAN
atan2()	4.5.2.4	MAN
ceil()	4.5.6.1	MAN
cos()	4.5.2.5	MAN
cosh()	4.5.3.1	MAN
exp()	4.5.4.1	MAN
fabs()	4.5.6.2	MAN
floor()	4.5.6.3	MAN
fmod()	4.5.6.4	MAN
frexp()	4.5.4.2	MAN
ldexp()	4.5.4.3	MAN
log()	4.5.4.4	MAN
log10()	4.5.4.5	MAN
modf()	4.5.4.6	MAN
pow()	4.5.5.1	MAN
sin()	4.5.2.6	MAN
sinh()	4.5.3.2	MAN
sqrt()	4.5.5.2	MAN
tan()	4.5.2.7	MAN
tanh()	4.5.3.3	MAN

NOTE:

MAN — Mandatory for this profile.

Table B-18. POSIX_C_LANG_SUPPORT Non-Local Jump Functions

Function	Reference in the C Standard	SCA AEP
longjmp()	4.6.2.1	MAN
setjmp()	4.6.1.1	MAN

NOTE:

MAN — Mandatory for this profile.

Table B-19. POSIX_C_LANG_SUPPORT General Functions

Function	Reference in the C Standard	SCA AEP
abs()	4.10.6.1	MAN
atof()	4.10.1.1	MAN
atoi()	4.10.1.2	MAN
atol()	4.10.1.3	MAN
bsearch()	4.10.5.1	MAN
calloc()	4.10.3.1	MAN
free()	4.10.3.2	MAN
malloc()	4.10.3.3	MAN
qsort()	4.10.5.2	MAN
rand()	4.10.2.1	MAN
realloc()	4.10.3.4	MAN
srand()	4.10.2.2	MAN

NOTE:

MAN — Mandatory for this profile.

Table B-20. POSIX_C_LANG_SUPPORT String Handling Functions

Function	Reference in the C Standard	SCA AEP
strcat()	4.11.3.1	MAN
strchr()	4.11.5.2	MAN
strcmp()	4.11.4.2	MAN
strcpy()	4.11.2.3	MAN
strcspn()	4.11.5.3	MAN
strlen()	4.11.6.3	MAN
strncpy()	4.11.2.4	MAN
strncat()	4.11.3.2	MAN

strncmp()	4.11.4.4	MAN
strpbkr()	4.11.5.4	MAN
strrchr()	4.11.5.5	MAN
strspn()	4.11.5.6	MAN
strstr()	4.11.5.7	MAN
strtok()	4.11.5.8	MAN

NOTE:

MAN — Mandatory for this profile.

Table B-21. POSIX_C_LANG_SUPPORT Data and Time Functions

Function	Reference in the C Standard	SCA AEP
asctime()	4.12.3.1	MAN
ctime()	4.12.3.2	MAN
gmtime()	4.12.3.3	MAN
localtime()	4.12.3.4	MAN
mktime()	4.12.2.3	MAN
strftime()	4.12.3.5	MAN
time()	4.12.2.4	MAN
tzset()	4.12.2.4	NRQ

NOTE:

MAN — Mandatory for this profile.

NRQ — Mandatory for this profile.

10.3.2 POSIX.1b.

Table B-22 contains the required options, limits, and any other constraints on POSIX.1b.

Table B-22. POSIX.1b Option Requirements

9	SCA AEP
{_POSIX_ASYNCHRONOUS_IO}	MAN

{_POSIX_MAPPED_FILES}	NRQ
{_POSIX_MEMLOCK}	MAN
{_POSIX_MEMLOCK_RANGE}	MAN
{_POSIX_MEMORY_PROTECTION}	NRQ
{_POSIX_MESSAGE_PASSING}	MAN
{_POSIX_PRIORITIZED_IO}	NRQ
{_POSIX_PRIORITY_SCHEDULING}	NRQ
{_POSIX_REALTIME_SIGNALS}	MAN
{_POSIX_SEMAPHORES}	MAN
{_POSIX_SHARED_MEMORY_OBJECTS}	NRQ
{_POSIX_SYNCHRONIZED_IO}	PRT*
{_POSIX_TIMERS}	MAN
{_POSIX_FSYNC}	PRT**

NOTE:

NRQ — Not required for this profile.

MAN — Mandatory for this profile.

PRT — Partial

* fdatsync not required

** fsync not required

10.3.3 POSIX.1c.

Table B-23, Table B-24, Table B-25, Table B-26, Table B-27 and Table B-28 contain the required options, limits, and any other constraints on POSIX.1c.

Table B-23. POSIX.1c Option Requirements

Option	SCA AEP
{_POSIX_THREADS}	MAN
{_POSIX_THREAD_ATTR_STACKADDR}	MAN
{_POSIX_THREAD_ATTR_STACKSIZE}	MAN
{_POSIX_THREAD_PRIO_INHERIT}	MAN

{_POSIX_THREAD_PRIO_PROTECT}	MAN
{_POSIX_THREAD_PRIORITY_SCHEDULING}	MAN
{_POSIX_THREAD_PROCESS_SHARED}	NRQ
{_POSIX_THREAD_SAFE_FUNCTIONS}	PRT

NOTE:

NRQ — Not required for this profile.

MAN — Mandatory for this profile.

PRT — Partial, only the subset of units of functionality called out in A.3.3.

B.3.3.110.3.3.1 Re-entrant User Group Function Behavior.

The function listed in Table A-25 shall behave as described in the referenced clause.

Table B-24. POSIX_USER_GROUPS_R Function

Function	Reference in POSIX.1c	SCA AEP
getlogin_r()	4.2.4	NRQ

NOTE:

NRQ — Not required for this profile.

Table B-25. POSIX_DEVICE_SPECIFIC_R Function

Function	Reference in POSIX.1c	SCA AEP
ttyname_r()	4.7.4	NRQ

NOTE:

NRQ — Not required for this profile.

B.3.3.210.3.3.2 File Locking Function Behavior.

The functions listed in Table B-26 shall behave as described in the referenced clauses.

Table B-26. POSIX_FILE_LOCKING Functions

Function	Reference in POSIX.1c	SCA AEP
getc_unlocked()	8.2.7	NRQ
getchar_unlocked()	8.2.7	NRQ
flockfile()	8.2.6	NRQ
fttrylockfile()	8.2.6	NRQ
funlockfile()	8.2.6	NRQ
putc_unlocked()	8.2.7	NRQ
putchar_unlocked()	8.2.7	NRQ

NOTE:

NRQ — Not required for this profile.

B.3.3.310.3.3.3 Reentrant C Language Support Function Behavior.

The functions listed in Table B-27 shall behave as described in the referenced clauses.

Table B-27. POSIX_C_LANG_SUPPORT_R Functions

Function	Reference in POSIX.1c	SCA AEP
asctime_r()	8.3.5	MAN
ctime_r()	8.3.6	MAN
gmtime_r()	8.3.7	MAN
localtime_r()	8.3.8	MAN
rand_r()	8.3.9	MAN
strtok_r()	8.3.4	MAN

NOTE:

MAN — Mandatory for this profile.

B.3.3.410.3.3.4 Reentrant System Database Function Behavior.

The functions listed in Table B-28 shall behave as described in the referenced clauses.

Table B-28. POSIX_SYSTEM_DATABASE_R Functions

Function	Reference in POSIX.1c	SCA AEP
getgrgid_r()	9.2.1	NRQ
getgrnam_r()	9.2.1	NRQ
getpwnam_r()	9.2.2	NRQ
getwuid_r()	9.2.2	NRQ

NOTE:

NRQ — Not required for this profile.

Annex I (normative) Modelling conventions

Bibliography

TBD