PIM and PSM for Software Radio Components

2nd FTF Convenience Document (with change bars)

This document represents the changes to the Final Adopted Specification, dtc/04-05-04, with corrections to the typographical errors and 1st and 2nd FTF issue resolutions. It is provided as a resolution to 1st FTF first ballot issues 7578, 7579, 7580, 7587, 7655, 7656, 7657, 7658, 7661 (partial), 7672, 7689, 7690, 7691, 7693, 7696, 7697, 7698, 7699, 7700, 7701, 7702, 7781, 7786; and second ballot issues: 7581, 7655, 7657, 7688, 7693, 7694, 7695, 7704, 7705, 7706, 7707, 7708, 7709. 7710, 7711, 7712, 7713, 7714, 7715, 7717, 7718, 7719, 7720, 7725, 7726, 7727, 7728, 7757, 7787, 7789, 7868, 7959, 7983, 7984, 7985; third ballot issues: 7578, 7579, 7588, 7658, 7716, 7717, 7742, 7785, 7786, 7849, 7877, 7878, 7888, 7895, 7904, 7905, 7953, 8121, 8122, 8123, 8124, 8125, 8200, 8201, 8205, 8291; the fourth ballot issue: 8697;

and 2nd FTF first ballot issues: 7582, 7583, 7586, 7703, 7729, 7845, 7894, 8296, 8830, 8831, 8832, 8833, 8834, 8835, 8836, 8837, 8838, 8839, 8840, 8841, 8842, 8857, 8858, 8868, 8869, 8872, 8873, 8931, 8934, 8948, 8949; and second ballot issues: 8980 and 8981.

Proposed resolutions are shown with a red change bar on the left, and a note preceeding the change bar stating which issue the change relates to. In case there is no note associated with the change bar, then it is a typo correction and the change is associated with issue 7781.

You may view the pending issues for this specification from the OMG revision issues web page *http://www.omg.org/issues/*.

The FTF Recommendation and Report for this specification will be published on September 23, 2005. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

OMG Adopted Specification
PIM and PSM for SWRadio Components

2nd FTF Convenience Document
dtc/2005-09-04

Accompanying documents:

SWRadio 2nd FTF Report, dtc/2005-09-03
Convenience doc, no change bars dtc/2005-09-05
IDL Files dtc/2005-09-02
XML Files dtc/2005-08-05
UML Files dtc/2005-08-06
Detailed Voting Record for the 2nd FTF dtc/2005-09-06

Date: September, 14th 2005

PIM and PSM for Software Radio Components

Final Adopted Specification
2nd FTF Convenience Document

dtc/2005-09-04

ISSUE REPORTING

Contents

1 Scope

This specification responds to the requirements set by “Request for Proposals for a Platform Independent Model (PIM) and CORBA Platform Specific Model (PSM)” (swradio/02-06-02) of radio infrastructure facilities that can be utilized in developing waveforms, which promotes the portability of waveforms across Software Defined Radios (SDR). The terms Software Radio and Software Defined Radio are used to describe radios that are implemented with strong emphasis on software. This type of radio, which is called SWRadio in this specification.

The SWRadio specification is physically partitioned into three major chapters: UML Profile for SWRadio, PIM and PSM for CORBA IDL. UML Profile for SWRadio defines a language for modeling SWRadio elements by extending the UML language with radio domain specific definitions. PIM provides a model of SWRadio system behavior and standardized application program interfaces (APIs) as well as example component definitions that realize the provided interfaces. PIM is specified independently from the underlying middleware technology. UML and its extensions provided by the UML Profile for SWRadio were used for modeling a software radio system in the PIM.

This specification also provides a mechanism for transforming the elements of the PIM model into the platform specific model for CORBA IDL. This mapping definition is given in the PSM (Chapter 10).

Finally, the SWRadio specification provides different compliance points depending on the role the implementer of this specification plays. Those different roles and respective partitioning of this document is given in the Conformance (Chapter 2).

2 Conformance

Note – Issue 7785 - Waveform vs Application changes throughout Chapter 2

2.1 Conformance Criteria

Conformance with the OMG Software Radio specification can be partial. Therefore, several separate conformance points are defined below. The conformance language references several parts of the specification:

- The Application PIM is defined in Chapter 9.
- The Application PSM is defined in Annexes A to J
- The Software Radio Infrastructure PIM is defined in Chapter 9
- The Software Radio Infrastructure PSM is defined in Annexes A to J
- The Profile-to-Waveform PIM Mapping is defined in Chapter 9.
- The PIM-to-PSM Mapping is defined in Chapter 10.
- The UML Profile for Software Radio Waveform Applications is defined in Chapter 8.

The Design Rationale in Section 6.4 explains the role that each of these items play in the overall specification.

2.2 Conformance on the Part of a Waveform Application

An application is considered to be a *conformant waveform application for the CORBA/XML platform* if it does all of the following:

- Implements the CORBA interfaces that the Applications PSM defines
- Implements the XML serialization formats that the Applications PSM defines.
- Implements the semantics that the Application PIM defines.

Note that the Applications PIM essentially defines the semantics for the CORBA interfaces and XML serialization formats. The semantics for a CORBA interface defined in the Applications PSM are defined by the semantics of the corresponding element(s) in the Applications PIM. It is possible to deduce the corresponding elements in the PIM for such a CORBA interface by reversing the PIM-PSM Mapping.

An application for a platform other than CORBA/XML can legitimately claim a degree of conformance to this specification if it implements the semantics that the Application PIM defines. For a platform “X,” such an application is considered to be a *conformant waveform application for the X platform*. In practice this would require the definition of an alternate Application PSM, and would require that it is possible to unambiguously trace back from elements of the alternate PSM to elements of the Waveform PIM.

2.3 Conformance on the Part of a Software Radio Infrastructure

A software radio infrastructure is considered to be a *conformant software radio infrastructure for the CORBA/XML platform* if it does all of the following:

- Implements the CORBA interfaces that the Software Radio Infrastructure PSM defines
- Implements the XML serialization formats that the Software Radio PSM defines.
- Implements the semantics that the Software Radio Infrastructure PIM defines.

Note that Software Radio Infrastructure PIM essentially defines the semantics for the CORBA interfaces and XML serialization formats. The semantics for a CORBA interface defined in the Software Radio Infrastructure PSM are defined by the semantics of the corresponding element(s) in the Software Radio Infrastructure PIM. It is possible to deduce the corresponding elements in the PIM for such a CORBA interface by reversing the PIM-PSM Mapping.

A software radio infrastructure for a platform other than CORBA/XML can legitimately claim a degree of conformance to this specification if it implements the semantics that the Software Radio Infrastructure PIM defines. For a platform "X," such an application is considered to be a *conformant software radio infrastructure for the X platform*. In practice this would require the definition of an alternate Software Radio Infrastructure PSM, and would require that it is possible to unambiguously trace back from elements of the alternate PSM to elements of the Software Radio Infrastructure PIM.

2.4 Conformance with the UML Profile for Software Radio Applications

There are two kinds of conformance with respect to the profile: conformance on the part of a model of a specific software radio application, and conformance on the part of a software radio tool.

2.4.1 Conformance by a Model of a Specific Application

A UML model of a specific application either conforms to the profile or it does not. There are no categories of this kind of conformance. Such a UML model conforms to the profile if it satisfies all constraints imposed by the profile package.

2.4.2 Conformance by a Software Radio Tool

2.4.2.1 Definition of Terms for Discussion of Tool Conformance

To support the discussion of conformance by a software radio tool, we define two terms: "identified subset of UML 2.0" and "all constructs defined by the profile."

The *identified subset of UML 2.0* for the profile is the set of packages contained in the UML 2.0 Superstructure specification Part 1 (Structure). Part 1 includes the following packages and the transitive closure of all packages contained by these packages and of all packages upon which these packages depend:

- Classes
- Composite Structures
- Components
- Deployments

Hereafter we sometimes use the abbreviated term identified subset to refer to the identified subset of UML 2.0

The term *all constructs defined by the profile* is defined to mean all constructs that are part of the package's identified subset of UML 2.0, plus all extensions to that subset that the profile defines. Thus this term includes UML constructs that are part of the identified subset but that are not extended by the profile.

2.4.2.2 Categories of Tool Conformance

A tool is considered to be a conformant *simple modeling tool* for the profile if it does both of the following:

- Supports expression of all constructs defined by the profile, via UML 2.0 notation.

- Supports the UML 2.0 XMI exchange mechanism for the identified subset and for UML 2.0 profiles.

A tool is considered to be a conformant *CORBA/XML-based forward engineering tool* for the profile if it does both of the following:

- Supports the Profile-to-Waveform PIM Mapping defined in Chapter 9 of this specification
- Supports the PIM-to-PSM Mapping defined in Chapter 10.
- Produces applications that are conformant waveform applications, based on the definition of such conformance in the "Conformance on the Part of a Waveform Application" section above. Alternately, if a tool only produces an application skeleton, the skeleton must not make it impossible for a full application based on the skeleton to qualify as a conformant waveform application; in other words, the skeleton must be able to form the basis of a conformant waveform application.

A forward engineering tool that targets a platform technology other than CORBA/XML can legitimately claim a degree of conformance to the profile if it conforms to the Profile-to-Waveform PIM Mapping and produces applications that are conformant waveform applications, or produces application skeletons that can form the basis of conformant waveform applications. In practice this requires the definition of an alternate PIM-PSM mapping. A forward engineering tool of this nature for the platform "X" is considered to be a conformant *X-Based forward engineering tool* for the profile.

2.5 Sample Conformance Statements

"XXX is a conformant waveform application for the CORBA/XML platform, in accordance with the OMG Software Radio specification."

"XXX is a conformant waveform application for the J2EE/XML platform, in accordance with the OMG Software Radio specification."

"XXX is a conformant software radio infrastructure for the CORBA/XML platform, in accordance with the OMG Software Radio specification."

"XXX is a conformant software radio infrastructure for the J2EE/XML platform, in accordance with the OMG Software Radio specification."

"XXX is a model of a specific waveform application. The model conforms to the UML Profile for Software Radio Waveform Applications, in accordance with the OMG Software Radio specification."

"XXX is a conformant simple modeling tool for the UML Profile for Software Radio Waveform Applications, in accordance with the OMG Software Radio specification."

"XXX is a conformant CORBA/XML-based forward engineering tool for the UML Profile for Software Radio Waveform Applications, in accordance with the OMG Software Radio specification."

"XXX is a conformant J2EE/XML-based forward engineering tool for the UML Profile for Software Radio Waveform Applications, in accordance with the OMG Software Radio specification."

Note – As pointed out in the Design Rationale chapter's Known Issues section, there are some issues with the specification that the submitters plan to resolve as part of the activity of the Finalization Task Force (FTF). These conformance criteria are written on the assumption that the issues are resolved as indicated in that section.

---

3 References

Note – Issue 8697: `1) Rename chapter 3 to "References"`

```
2) Create a subheader called "Normative References" and pull ev-
erything currently in Chapter 3 under Normative subsection.
3) Create a subheader called "Non-normative References" and put
the following in there:
"UML Model for PIM and PSM for SWRadio Components"
OMG Document number: dtc/2005-03-06
[http://www.omg.org]
```

3.1 Normative References

3.1.1 UML Specifications

3.1.1.1 UML Language Specification

Unified Modeling Language (UML) Superstructure Specification Version 2.0
Formal OMG Specification, document number: formal/2005-07-04
The Object Management Group, July 2005
[http://www.omg.org]

3.1.1.2 OCL Language Specification

Object Constraint Language (OCL) Specification Version 2.0
Final Adopted OMG Specification, document number: ptc/2005-06-06
The Object Management Group, June 2005
[http://www.omg.org]

3.1.1.3 Deployment and Configuration

Deployment and Configuration of Component-based Applications
Final Adopted OMG Specification, document number: ptc/2003-07-08
The Object Management Group, April 2002
[http://www.omg.org]

Note – Unless this document becomes a formal OMG specification, its reference is *not normative.*

3.1.1.4 UML Profile for CORBA Specification

UML Profile for CORBA Specification V1.0
Formal OMG Specification, document number: formal/2002-04-01
The Object Management Group, April 2002
[http://www.omg.org]

3.1.2 CORBA Core Specifications

3.1.2.1 CORBA Specification

Common Object Request Broker (CORBA/IIOP), version 3.0.2
Formal OMG Specification, document number: formal/2002-12-06
The Object Management Group, December 2002
[http://www.omg.org]

3.1.2.2 Minimum CORBA Specification

Minimum CORBA, V1.0
Formal OMG Specification, document number: formal/2002-08-01
The Object Management Group, August 2002
[http://www.omg.org]

3.1.3 CORBA Services Specifications

Note – Issue 8201 Resolution (Replacement of Naming and Event Service references with references to the Lightweight Services and the Lightweight Log Service)

3.1.3.1 Lightweight Services Specification

Lightweight Services, version 1.0
Formal OMG Specification, document number: formal/2004-10-01
The Object Management Group, October 2004
[http://www.omg.org]

3.1.3.2 Lightweight Log Service Specification

Lightweight Log Service, version 1.0
Formal OMG Specification, document number: formal/2003-11-03
The Object Management Group, November 2003
[http://www.omg.org]

Note – Issue 8201 Resolution (misspelling fix in 3.3.3 title)

3.1.4 Enhanced View of Time Specification

Enhanced View of Time Service, version 1.1
Formal OMG Specification, document number: formal/2002-05-07
The Object Management Group, May 2002
[http://www.omg.org]

3.1.5 Property Service Specification

Property Service, version1.0
Formal OMG Specification, document number: formal/2000-06-22
The Object Management Group, June 2000
[http://www.omg.org]

Note – Issue 8697: Provide a non-normative reference to SWRadio UML Model

3.2 Non-Normative References

3.2.1 UML Model for PIM and PSM for Software Radio Components

UML Model for PIM and PSM for Software Radio Components
OMG document number: dtc/2005-03-06

The Object Management Group, March 2005
[http://www.omg.org]

4 Terms and Definitions

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

Common Object Request Broker Architecture (CORBA)

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

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).

CORBA Component Model (CCM)

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

Facility

An environment providing a realization of certain functionality through set of well defined interfaces.

Interface Definition Language (IDL)

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

Logical Device

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

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.

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.

Metamodel

A model of models.

Meta Object Facility (MOF)

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

Model

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

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.

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.

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.

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.

Radio Platform

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

Radio Set

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

Radio System

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

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.

Service

A set of functionality with common characteristics.

Unified Modeling Language (UML)

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

UML Profile

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

5 Symbols (and abbreviated terms)

Note – Issue 7656 Typos & Acronyms

Table 5-1 – 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
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
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

Table 5-1 – Symbols (and abbreviated terms)

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

6 Additional Information

Note – Issue 7785 - Waveform vs Application changes throughout Chapter 6

6.1 Changes to Adopted OMG Specifications

The specifications contained in this document require no changes to adopted OMG specifications.

6.2 Guide to this Specification

This specification consists of four major parts, contained in the following chapters 7 to 10.

- Chapter 7 provides an introduction into the field of software defined communication and provides the architectural overview of the material contained in this specification
- Chapter 8 defines the modeling language used in this specification in form of a UML profile.
- Chapter 9 contains the Platform Independent Model (PIM). The UML language extended by the profile defined in Chapter 8 is used to specify this PIM.
- Chapter 10 contains a description of the mapping process from the Platform Independent Model (PIM) to a Platform Specific Model (PSM).
- A mapping of the SWRADIO PIM to the CORBA Component Model (CCM) is contained in the Annexes

Note – Issue 8697: Add reference to the UML Model

The UML model referenced in Section 3.2.1 is used to generate the class diagrams shown throughout this specification. This UML model is non-normative, and provided for informational purposes only. The intent of the authors is to provide a normative set of XMI files that would contain the UML Profile for SWRadio and PIM facilities, when a tool that meets the requirements of this specification becomes available.

6.3 Credits

The following organizations (listed in alphabetical order) contributed to this specification:

- BAE Systems
- BOEING
- Blue Collar Objects
- Carleton University
- Communications Research Center Canada
- École de Technologie Supérieure
- General Dynamics Decision Systems
- Harris
- ITT Aerospace
- ISR Technologies

- L-3 Communications Corporation
- Mercury Computer Systems
- The MITRE Corporation
- Mobile Smarts
- Raytheon Corporation
- Rockwell Collins
- SCA Technica
- Space Coast Communication Systems
- Spectrum Signal Processing
- THALES
- Virginia Tech University
- Zeligsoft
- 88solutions

6.4 Design Rationale

6.4.1 Supporting a Product Line of SWRadio Applications

Product Line Practices (PLP), as defined by the Carnegie-Mellon Software Engineering Institute[1], involves developing a set of core assets for a domain of software products. Application engineers reuse the core assets to build specific products. Originally PLP envisioned runtime components as the primary core assets for a product line. A number of industry practitioners have extended PLP by including specification languages and model components among the core assets for a product line[2].

PLP draws a distinction between core asset engineering on the one hand, and application engineering on the other. Application engineering reuses assets produced by the core asset engineering process.

This specification supports a product line for the domain of Software Radio Applications.

6.4.2 The UML Profile for Software Radio Applications: A Domain-Specific Language

A specification language for a product line makes it possible to specify individual products that are members of the product line. Such a language allows specification at a level of abstraction that is specific to the product line and thus is at a higher level of abstraction than that offered by a general-purpose modeling or programming language. For this reason, specification languages for product lines are called domain-specific languages (DSLs). A DSL's higher level of abstraction hides complexity from the application engineer.

1.Carnegie Mellon Software Engineering Institute, “The Product Line Practices Initiative,” Web page, http://www.sei.cmu.edu/plp/.
2.See, in particular, Krzysztof Czarnecki and Ulrich W. Eisenecker, Generative Programming: Methods, Tools, and Applications, Addison Wesley, 2000.

This specification defines a DSL for a product line of software radio applications. The DSL is the UML Profile for Software Radio Applications. Although UML is a general-purpose modeling language, this profile defines a variant of UML that is specific to the software radio product line. This DSL is an important core asset for the product line.

6.4.3 Model Components: Two PIMs and Two PSMs

As mentioned above, extensions to PLP also include model components among the core assets for a product line. Model components are bits of models as opposed to bits of executable code. Tools reuse model components in various ways.

For example, based on the application engineer's specifications expressed via the DSL, tools select model components, configure them in particular ways, and compile them. The compiled model components can (but don't have to) execute on top of runtime components that also are part of the product line.

For another example, fixed CORBA IDL is a model component that might be supplied as a core asset for a product line.

This specification defines model components for the software radio applications product line. They are embodied in two platform-independent[3] models (PIMs)-the Waveform Applications PIM and the Software Radio Infrastructure PIM-and in two platform-specific models (PSMs)-the Waveform Applications PSM and the Software Radio Infrastructure PSM.

6.4.3.1 Applications PIM

This PIM defines types that software radio applications must implement. The intent is that the application engineer never sees this PIM. Tools that process specifications expressed via the UML profile use the PIM to generate application skeletons or possibly to generate complete applications in some cases.

For example, if an application engineer marks an element of a application model with the DeviceComponent stereotype that the profile defines, then a tool that processes the model produces an element that has all the features that the application engineer explicitly defined for the stereotyped element and that also has the features defined by the DeviceComponent type in the PIM.

Note that, for the device component concept, there is both a stereotype in the profile and a type in the PIM. The semantics of the DeviceComponent stereotype are that the stereotyped element must implement the DeviceComponent type defined in the PIM.

6.4.3.2 Software Radio Infrastructure PIM

This PIM defines the types a software radio infrastructure must implement. It includes an abstraction of the Software Communications Architecture (SCA) specification, defined by the Joint Tactical Radio System (JTRS) Joint Program Office.

3.Platform-independence is a relative, not an independent concept. It is not meaningful to assert that a model is independent without a specifying what kinds of platforms the model is independent of. For the purposes of this specification, platform-independent means independent of information formatting technologies such as XML; 3GLs and 4GLs such as Java, C++, C#, and Visual Basic; distributed component middleware such as J2EE, CORBA, and .NET; and Messaging middleware such as WebSphere MQ Integrator (MQSeries) and MSMQ

Application developers do not see this PIM, and tools that implement application specifications expressed via the profile do not use this PIM. However, applications require the presence of an implementation of this PIM (see the Section 6.4.6 below How the Pieces Fit Together). Thus we can appropriately characterize this PIM as a core asset for the software radio applications product line.

6.4.3.3 Applications PSM

The Applications PSM defines CORBA IDL and XML that applications must implement in order to run in a CORBA/XML platform environment. The IDL and XML is derived from the Applications PIM. Later sections of this design rationale explain the derivations.

6.4.3.4 Software Radio Infrastructure PSM

The Software Radio Infrastructure PSM defines CORBA IDL and XML that software radio infrastructures must implement in order to run in a CORBA/XML platform environment. The IDL and XML is derived from the Software Radio Infrastructure PIM. Later sections of the design rationale explain the derivations.

6.4.4 Mappings

Among the core assets for the product line are mappings that transform application specifications at one level of abstraction to specifications at a lower level of abstraction. Application development tools implement the mappings.

The specification defines two mappings-the Profile-to-WaveformPIM Mapping and the PIM-to-PSM Mapping. In combination, the mappings define how to generate a PSM for a application from a model expressed in terms of the profile. For this specification, an application PSM consists of a combination of CORBA IDL and XML.

The specification could have defined one mapping instead of these two mappings. The one mapping would be a Profile-to-PSM mapping. However, in order to promote flexibility in choice of middleware, the mapping is broken down into the two mappings, which a tool can apply transitively. If an implementer wishes to base PSMs on some technology other than CORBA and XML, it is necessary only to replace the PIM-to-PSM Mapping.

The intention is that these mappings are complexity that tools hide from application engineers to the greatest degree possible.

6.4.4.1 Profile-to-Waveform PIM Mapping

This mapping relates stereotypes in the UML profile to types in the PIM. The mapping of the profile's DeviceComponent stereotype to the Waveform Applications PIM's DeviceComponent type, cited earlier in this design rationale as an example, is an element of this mapping.

Thus, given an element in an application model expressed via the UML profile, the mapping defines how to refine the application model into a more detailed platform-independent model of the application. The more detailed model of the application leverages the types defined in the Waveform Applications PIM.

Note that the application model expressed via the profile is itself platform independent, based on our definition of platform independence for this specification. Thus there are two platform-independent levels of abstraction that are relevant here, although one is at a higher level of abstraction than the other; that is, the model that the application engineer creates via the profile is at a higher level of abstraction than the refined application PIM that a tool produces using the Profile-to-WaveformPIM mapping.

This specification provides this mapping as part of the definition of the profile's semantics.

6.4.4.2 PIM-to-PSM Mapping

This mapping defines how to transform a refined application PIM-that is, a PIM produced by the Profile-to-WaveformPIM mapping-into a PSM for the application. In other words, it defines how to further refine an application model to the CORBA/XML level of abstraction.

The specification also applies the PIM-PSM mapping directly to the Applications PIM to derive the Waveform Applications PSM, which consists of fixed IDL and XML that applications must implement. The IDL and XML generated by applying this mapping to a refined application model includes this fixed IDL and XML.

Furthermore, the specification also applies the PIM-PSM mapping to the Software Radio Infrastructure PIM to derive the Software Radio Infrastructure PSM, which consists of fixed IDL and XML that software radio infrastructures must implement. There is one exception to this rule: The specification does not apply the mapping to the SCA PIM that is part of the Software Radio Infrastructure PIM, because the SCA IDL and XML are predefined by the SCA specification.

The fact that the specification applies this mapping directly to the fixed Applications and Software Radio Infrastructure PIMs is another reason for separating the mapping from the Profile-to-WaveformPIM mapping.

6.4.5 Runtime Components

Although runtime components are often critical core assets for a product line, this specification does not provide runtime components to support the software radio applications product line. The OMG standardizes languages, transformations, and types, not implementations of such. Tools that implement the language, transformations, and types defined in this specification will probably provide runtime components that support the implementation.

6.4.6 How the Pieces Fit Together

Figure 6-1 below illustrates some of the interconnections among a software radio infrastructure, applications, and external application clients. External clients invoke infrastructure APIs and the infrastructure invokes application APIs. Standardized interfaces for software radio infrastructures make it possible to port external clients from one software radio infrastructure to another. Standardized interfaces for applications make it possible for a software radio infrastructure to support multiple waveform applications and to port applications from one software radio infrastructure to another.

Figure 6-2 below illustrates a point explained earlier, namely that the fixed types defined in the Applications PIM and the Software Radio Infrastructure PIM map to fixed, CORBA interfaces (and associated, fixed XML descriptors) for applications and software radio infrastructures, respectively. This specification derives the fixed IDL and XML via the PIM-to-PSM mapping. Conceptually, this application of the PIM-to-PSM mapping is performed once.

Applications support additional CORBA interfaces and XML descriptors that correspond to the features that the application engineer defines in the application model (that is, the model that the application engineer expresses in terms of the UML profile). Figure 6-3 below illustrates that tools derive the IDL and XML by applying the two mappings transitively. The IDL and XML derived in this fashion also support the fixed Application IDL and XML contained in the Application PSM. Conceptually, this transitive application of the Profile-to-WaveformPIM and PIM-to-PSM mappings is performed once for each application.

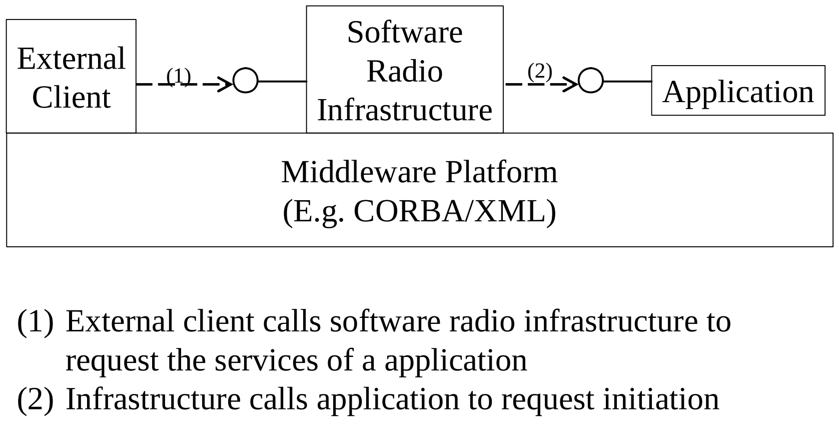

External Client
(1)
Software Radio Infrastructure
(2)
Application
Middleware Platform (E.g. CORBA/XML)
(1) External client calls software radio infrastructure to request the services of a application
(2) Infrastructure calls application to request initiation

Figure 6-1 – Interactions of External Client, Radio Infrastructure, and Application

.

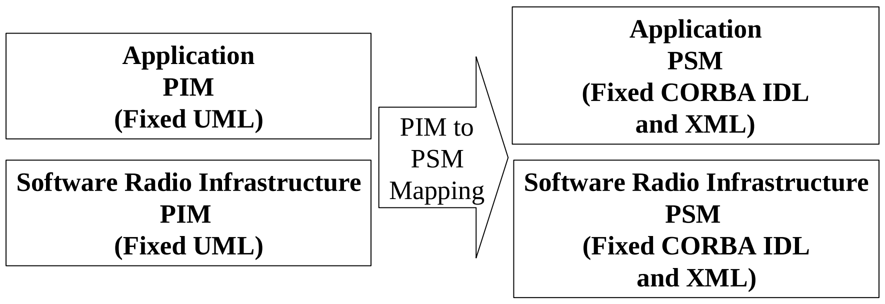


Figure 6-2 – Deriving the Fixed CORBA IDL and Fixed XML from the Fixed PIMs

.

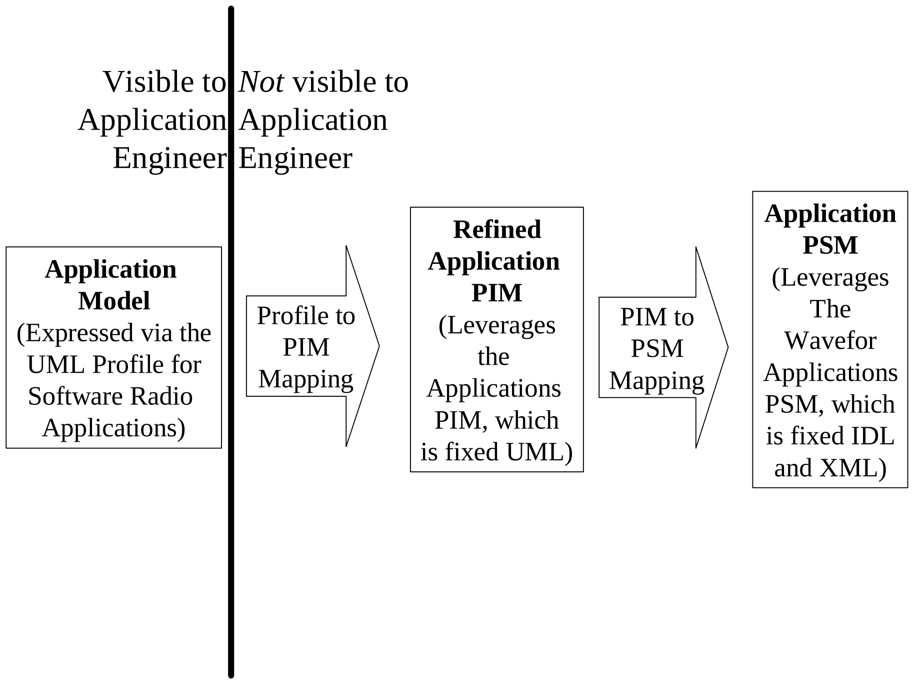


Figure 6-3 – Applying the Mappings Transitively

7 Introduction to SWRadio

Note – Issue 7785 - Waveform vs Application changes throughout Chapter 7, Issue 7845 - Add non-normative at the end of first paragraph

7.1 Introduction

The terms Software Radio and Software Defined Radio (SDR) are used to describe radios that are implemented with strong emphasis on software. This type of radio, which is called SWRadio in this specification, offers important technical and commercial advantages. This non-normative section gives an overview of the rationale and architecture of software radios.

A Software Defined Radio 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.

SWRadio technology is changing every facet of communication system design and usage. SDR is not just another way to build radios with the same functions, but SDR designs support many new critical needs. SDR supports several waveforms inside the same box, eases bug fixing, enables reconfigurability, allows for digitized, IP-based, data transmissions and improves security. It also enables an open market where waveform providers can be independent of platform providers. These improvements are essential for the radio military market but also meet the current and emerging needs of the civil radio market.

Enabling cost-effective technology insertion is a strong motivation for manufacturers suddenly faced with a more volatile market than in the past and where tomorrow's standards are unclear. The life cycle of new radio sets has become so short that the return-on investments cannot be ensured. Enabling cost-effective technology insertion is also a concern for many operators and for customers faced with exploding costs. Reconfigurability is a key feature for next generation radio systems. It involves adding, removing and modifying radio functionality.

SWRadio supports multiple concurrent waveforms inside a single radio set is critical for the military market where numerous waveforms are used by warfighters from various services and countries. Civil market changes, and future civil radios, are likely to support both a cellular waveform and a high bandwidth local waveform on the same hardware. Moreover, future radio node equipment may have to concurrently support multiple cellular waveforms such as Global System for Mobiles (GSM), General Packet Radio Services (GPRS), Universal Mobile Telecommunications System (UMTS) and high throughput waveforms such as Bluetooth and WiFi.

SWRadio facilitates repair of system defects through over-the-air or over-the-wire reprogramming of software features. Repairing system defects is an important issue for radio manufacturers since this may involve returning thousands of radio sets to factories. Software downloads to fix bugs is a key need for radios. This need is being addressed on a waveform-by-waveform basis (3GPP), but this does not solve the problem for future multi-waveform radios. SWRadio technology addresses this need through defining standard component interfaces and defining a plug & play like deployment mechanism.

For the last century, radio was mainly used to transmit human voice in the commercial sector. In the future, these radios will transmit digitized data as well as analog voice. This is a major shift. For example, a radio that was mainly focused on point-to-point links will have to support new services such as networking and be used to manage the seamless qualities of radio function challenges. Radio is sometimes viewed as being the last hop inside a network, but it is likely to also 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 norm.

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. Software Defined Radio will allow a waveform designer to provide the application that can run on a platform designed by a different vendor. In order to do this effectively the interface between the platform and the application must be well defined and published. This specification is intended to provide that definition.

Hereafter, we discuss four main elements that impacted the development of this specification: Software Communications Architecture, Model Driven Architecture, Platform and Waveform definitions and the SWRadio Architecture.

7.2 Software Communication Architecture

The Software Communication Architecture (SCA) is the software architecture developed by the US Military's Joint Tactical Radio System (JTRS) Joint Program Office (JPO) for the next generation of radio systems. SDR companies are currently developing radio systems based on this architecture. It is considered as the de-facto standard in the SDR industry. The SCA forms the basis for the development of this specification.

SCA has been published to meet the requirements for radios that will operate in multiple domains and frequency bands. SCA compliant radios shall be able to communicate with legacy systems to minimize the impact of platform integration. The architecture enables technology insertion, so that new technologies can be incorporated to improve performance, and future-proof radios can be built.

Like most other software architectures, the SCA allows for the maximum possible reuse of software components. The components support plug-and-play behavior with applications being portable from one radio platform to the next.JTRS radios support legacy network protocols, for the purpose of seamless integration. The architecture supports wideband networking capabilities for voice, data and video.

The SCA defines an Operating Environment (OE) that will be used by JTRS radios. It also specifies the services and interfaces that the applications use from the environment. The interfaces are defined in CORBA IDL, and graphically represented in UML. The OE consists of a Core Framework (CF), a CORBA middleware and a POSIX-based Operating System (OS). The OS running the SCA must provide services and interfaces that are defined as mandatory in the Application Environment Profile (AEP) of the SCA. The CF describes the interfaces, their purposes and their operations. It provides an abstraction of the underlying software and hardware layers for software application developers. An SCA compatible system must implement these interfaces. The interfaces are grouped as Base Application Interfaces, Framework Control Interfaces and Framework Services Interfaces.

The CF uses a Domain Profile to describe the component metadata in the system. The Domain Profile is a set of XML files that describe the identity, capabilities, properties, inter-dependencies, and location of the hardware devices and software components that make up the system.

7.3 Model Driven Architecture

The OMG Model Driven Architecture (MDA) defines a model-based development approach to software development. The main objective of MDA is to enable the portability/reuse of models across different technology platforms. MDA focuses on the definition of Platform Independent Model (PIM), Platform Specific Model (PSM), and Model Mappings that allows moving from one model to another in a systematic manner. One goal of MDA is to define a set of Model Mappings between standard technologies that can be reused in different contexts. One of the main benefits of MDA is that models can be defined independently of specific implementation platforms and mapped to different platforms using predefined mappings.

The current SWRadio specification fully endorses the MDA approach. It defines a UML Profile for SWRadio, a PIM, and a CORBA/XML PSM for SWRadio components. The UML Profile for SWRadio, which is defined as an extension of the UML 2.0 specification, defines a set of standard stereotypes that can be used for the development of SWRadio applications, infrastructure, and deployment platforms. This profile is used in the current specification to define the PIM and PSM. The Platform Independent Model (PIM) formally defines a set of standard facilities for SWRadio without the technical details of any specific implementation. The Platform Specific Model (PSM) defines a version of the specification that is based on CORBA and XML specific technologies. The mapping between the PIM and the CORBA/XML PSM is captured in PIM-to-PSM mappings (given in Chapter 10). This PIM-to-PSM mapping can be automated, which would allow automatically updating the PSM to reflect changes made in the PIM to maintain a complete consistency between the two models.

7.4 SWRadio Platform and Applications

This specification supports a SWRadio Platform/Application approach where:

- the SWRadio 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. This specification defines a set of Platform-Independent Interfaces and does not make any assumptions on how those interfaces are supported.
- Applications can be developed and cross ported onto various Platforms implementations,

Such a SWRadio Platform/Application approach opens the way to an open market where applications providers can be independent of platform providers.

The SWRadio Platform concept used here refers to a composite infrastructure that is intended to support a set of applications to build various dedicated configurations such as radio nodes, radio terminals and/or other radio gateways. As a matter of fact, the SWRadio Platform define a basis for a product line approach.

The SWRadio Platform concept extends the Platform concept used within MDA in the way that SWRadio Platform not only refer to a software API but also include a set of hardware and software components.

A Radioset based on a SWRadio Platform can be seen as made of several layers. From bottom to top layers are:

- Hardware layer: set of heterogeneous hardware resources including general purpose devices as well as specialized ones,
- Operating Environment layer: basically provides operating system and (distributed) middleware services,
- Facilities layer: provides sets of services to the application developer,
- Application layer: figures a standalone capability provided by the radioset.

Those layers are figured below:

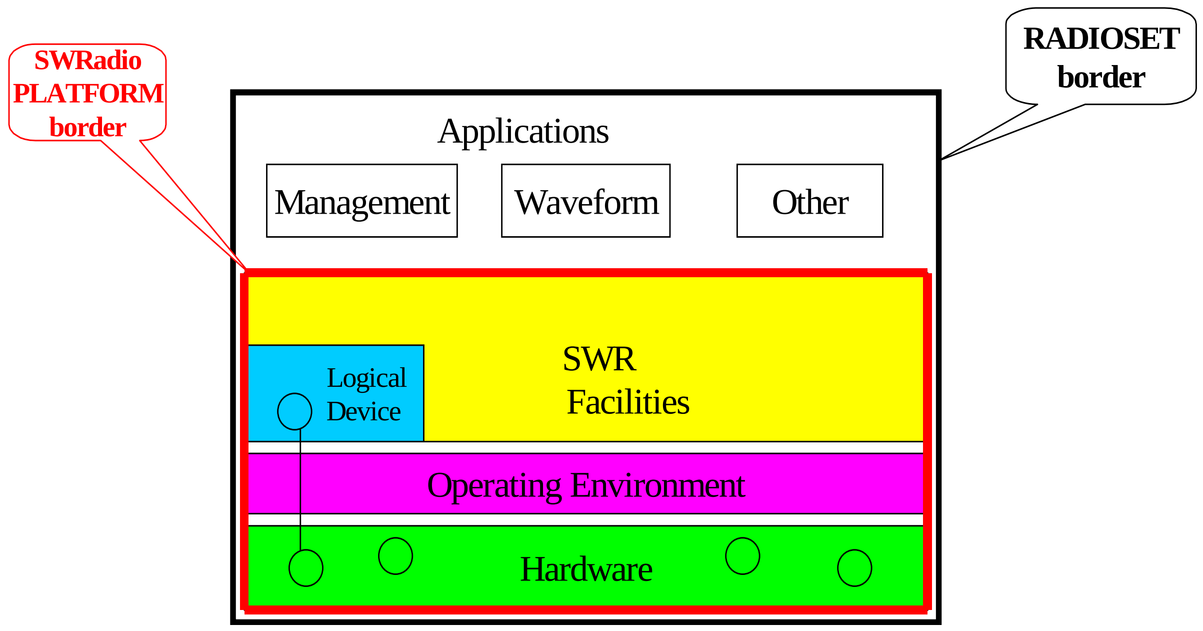


Figure 7-4 – SWRadio Layered View

Applications supported by SWRadio Platform can be dispatched into 3 categories:

- Waveform Applications that are the main focus of SWRadio and figure the waveform-specific application functions that noticeably coordinate the underlying SWRadio Platform functions to achieve the end-to-end waveform processing. These Applications also support general purpose Management Applications with waveform-specific management functions.
- Management Applications that figure general purpose, waveform-independent applications that enable to manage and control the Radioset and its embedded applications. Management Applications act as managers as defined in the OSI management framework (see IS 7498) and see Waveform Applications as agents to relay their requests. SWRadio Platform provides the management services excluding the presentation HMI.
- Other Applications figure all other kind of applications that can be provided inside a radioset such as:
    - Network applications that mainly support routing, security, directory or QoS functions),
    - End user applications such as Situation Awareness and/or other 3rd party applications.

Applications are supported by SWRadio Facilities inside which Logical Devices abstract some of the actual hardware devices of the SWRadio Platform. Logical Devices are defined for management purpose and their properties are designed to support management functions such as (re)configuration, performance or fault management.

Logical Devices should not be used directly by applications. Instead, applications should use higher level SWRadio Facilities.

7.5 SWRadio Architecture

The SWRadio architecture consists of two main concepts: services as well as applications and layering. Services concept depends on the interfaces provided and the usage of those interfaces. Application layering provides a logical grouping of functionality based on current commercial practice.

Through realization relationships in the PIM, a component can offer one or more services. A SWRadio vendor may choose to provide certain services that are required for their platform, and likewise acquire extra services from third party vendors. The services that can be provided by different actors that use this specification is detailed in the Chapter 2 Conformance.

- For a logical grouping of functionality, this specification follows the Open System Interconnection (OSI) Model elaborated and promoted by the International Standard Organization (ISO)

A full description of the OSI model can be found inside the ISO IS 7498The 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 radio set,
- the communication between peer layers is ruled by a Protocol which exchanges Protocol Data Units (PDU),
- a layer supplies Services to upper layers through Service Access Points (SAP),
- a layer acts as a PDU consumer and/or provider for its upper and lower layers.

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

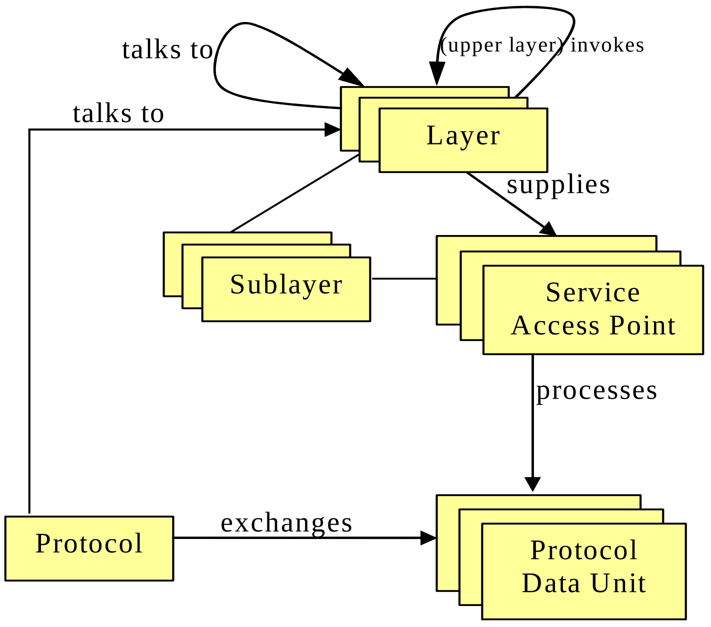


Figure 7-5 – Abstract OSI Model Core

Ironically, and despite its importance, the OSI model has primarily been a conceptual model that was implemented in hybrid manners in practice for performance reasons. Moreover, with the worldwide adoption of IP, the “old” layered model on which IP is built superseded the OSI model.Within the SDR context, the OSI model still proves to be a good design technique since it allows separation of concerns by making use of layering. This approach promotes the usage of interoperable and reconfigurable components through standard interfaces and well-defined packaging.

This specification acknowledges that the OSI communication model is a good reference design for any communication system, but conformance with this model is not mandatory for any radio set due to design and performance constraints. The proposed architecture supports not only the OSI model, but also other in-use or next-to-come models.

The OSI concepts described in this specification apply to the Extended OSI model which allows Management and QoS interfaces to cut through the waveform layer stack and communicate with any layer. Furthermore, this specification only focuses on physical and link (link layer control and medium access control) layers of the OSI stack.

8 UML Profile for Software Radio

Note – Issue 7845

This normative section defines the UML Profile for SWRadio. This profile is an integral part of the "PIM and PSM for SWRADIO Components. 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 SWRAPI (SWRadio API).

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

For this purpose, the Applications and Devices package defines the concept (as a stereotype) of SWRAPI (SWRadio API) as en 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.

The Infrastructure package defines the concepts that are required to develop software components deploy 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. 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 behavior.

Note – Issue 7694 - Explanation of different port types (2nd Ballot)

The UML Profile for SWRadio uses the concept of a Port as defined in the UML 2.0 specification by extending the Port definition for two different purposes. In Section 8.1.4, port stereotypes such as ServicePort, StreamPort are defined as software component ports which enable their users to access the associated software interfaces. In Section 8.2.4, the concept of a hardware port in a RadioSet environment is introduced, and Port specializations such as AnalogInputPort and DigitalPort are specified.

8.1 Application and Device Components

Note – Issue 7785 - Waveform vs Application changes throughout Chapter 8

The Applications and Devices package provides a set of component and interface stereotype definitions that are used for the development of SWRadio applications, logical devices, and SWRadio components. For application developers all sections are applicable except for Device Components section and for device developers all sections are applicable except for Application Components section. Figure 8-6 depicts the relationships among the packages within the Application and Device Components, which are as follows:

- Base Types - defines basic types for SWRadio applications, logical devices, and component definitions.
- Properties - defines stereotypes for configure, query, testable, service artifact (capability and capacity) and executable properties for SWRadio components and executable code artifacts.
- Interface & Port Stereotypes - defines stereotypes for SWRadio interfaces and components.
- Resource Components - defines stereotypes for the interfaces and components for the ResourceComponent, which is the basic component type for SWRadio application and device components.
- Device Components - defines stereotypes for the logical device components that represent devices that SWRadio component are deployed on or use within a SWRadio.

- Application Components - defines stereotypes for the ResourceComponents for SWRadio applications.

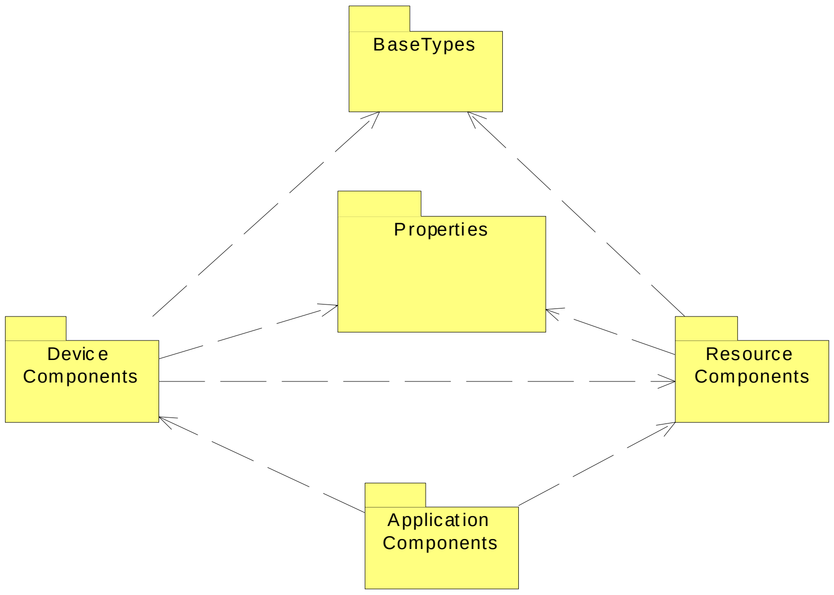
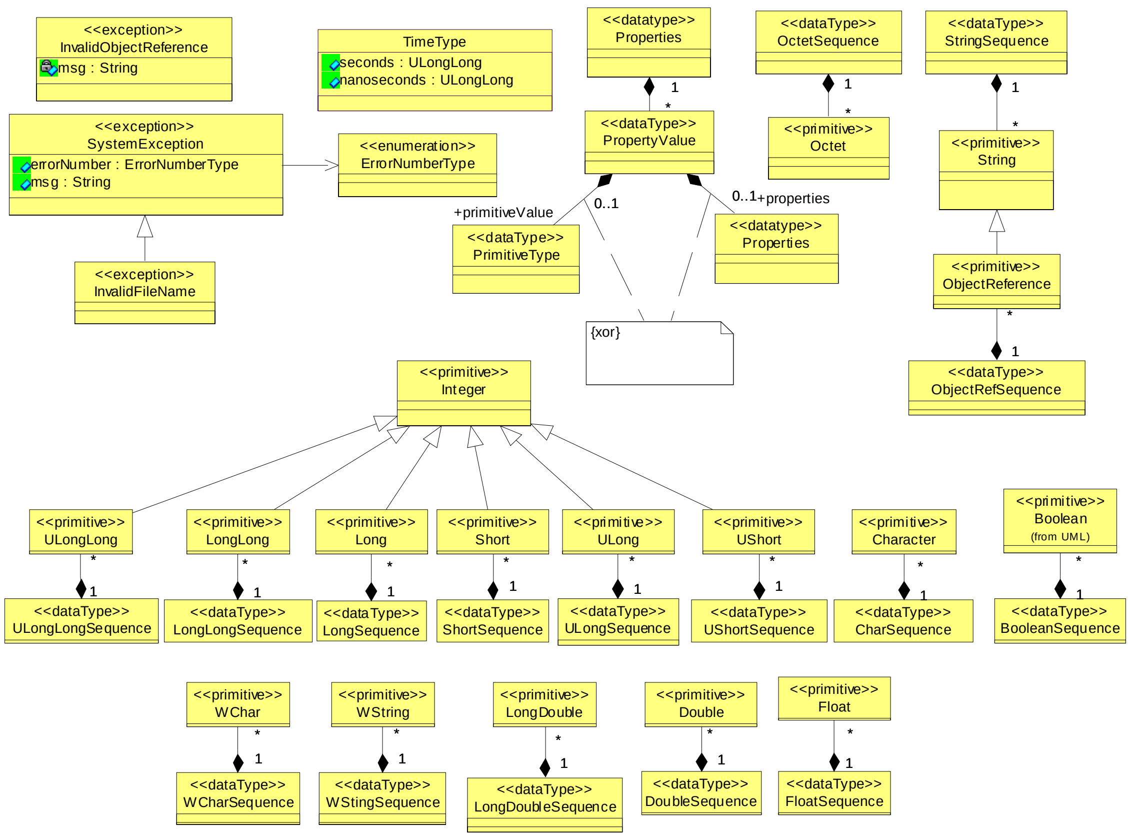


Figure 8-6 – Applications and Devices Overview

8.1.1 Base Types

The Base Types defines the basic types and exceptions used by multiple SWRadio components as shown in Figure 8-7 below.

Note – Issue 7895, added TimeType



<<exception>> InvalidObjectReference
msg : String
TimeType
seconds : ULongLong
nanoseconds : ULongLong
<<datatype>> Properties
<<dataType>> OctetSequence
<<dataType>> StringSequence
<<exception>> SystemException
errorNumber : ErrorNumberType
msg : String
<<enumeration>> ErrorNumberType
<<dataType>> PropertyValue
<<primitive>> Octet
<<primitive>> String
+primitiveValue
0..1
0..1+properties
<<dataType>> PrimitiveType
<<datatype>> Properties
<<exception>> InvalidFileName
<<primitive>> ObjectReference
{xor}
<<dataType>> ObjectRefSequence
<<primitive>> Integer
<<primitive>> ULongLong
<<primitive>> LongLong
<<primitive>> Long
<<primitive>> Short
<<primitive>> ULong
<<primitive>> UShort
<<primitive>> Character
<<primitive>> Boolean (from UML)
<<dataType>> ULongLongSequence
<<dataType>> LongLongSequence
<<dataType>> LongSequence
<<dataType>> ShortSequence
<<dataType>> ULongSequence
<<dataType>> UShortSequence
<<dataType>> CharSequence
<<dataType>> BooleanSequence
<<primitive>> WChar
<<primitive>> WString
<<primitive>> LongDouble
<<primitive>> Double
<<primitive>> Float
<<dataType>> WCharSequence
<<dataType>> WStingSequence
<<dataType>> LongDoubleSequence
<<dataType>> DoubleSequence
<<dataType>> FloatSequence
1
*

Figure 8-7 – Base Types Overview

Note – Issue 7655 (figure above), Issue 7586 changed constraint to be “xor”

Note – Issue 7655

8.1.1.1 BooleanSequence

Description

The BooleanSequence data type is an unbounded sequence of Boolean(s).

8.1.1.2 Character

Description

The Character primitive type is an 8-bit quantity that encodes a single-byte character from any byte-oriented code set.

Constraints

The Character value is a LiteralCharacter.

8.1.1.3 CharSequence

Description

The CharSequence data type is an unbounded sequence of Character(s).

8.1.1.4 Double

Description

The Double primitive type is an IEEE double-precision floating point number. See IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985, for a detailed specification.

Constraints

The Double value is a LiteralDouble.

8.1.1.5 DoubleSequence

Description

The DoubleSequence data type is an unbounded sequence of Double(s).

8.1.1.6 ErrorNumberType

Description

The ErrorNumberType enumeration defines error number information used in various exceptions.

Attributes

Those enumeration literal names starting with “CF_E” map to the POSIX definitions (starting with “E”) that 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 names 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

Note – Issue 7655, Issue 7586

8.1.1.7 Float

Description

The Float primitive type is an IEEE single-precision floating point number. See IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985, for a detailed specification.

Constraints

The Float value shall be a LiteralFloat that is an IEEE single-precision floating point number.

8.1.1.8 FloatSequence

Description

The FloatSequence data type is an unbounded sequence of Float(s).

8.1.1.9 InvalidFileName

Description

The InvalidFileName <<exception>>, type of SystemException, indicates an invalid file name. The errorNumber attribute indicates the type of error (e.g., CF_ENAMETOOLONG). The String msg attribute provides information describing why the filename was invalid.

8.1.1.10 InvalidObjectReference

Description

The InvalidObjectReference <<exception>> indicates an invalid object reference error.

Attributes

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

Note – Issue 7655, Issue 7586

8.1.1.11 Long

Description

The Long primitive type, a specialization of Integer primitive type, is a signed integer range $–2^{31}.. 2^{31} – 1$.

Constraints

The Long value shall be a LiteralInteger with a range of $–2^{31}.. 2^{31} – 1$.

8.1.1.12 LongDouble

Description

The LongDouble primitive type is an IEEE double-extended floating-point number. See IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985, for a detailed specification.

Constraints

The LongDouble value shall be a LiteralLongDouble that is an IEEE double-extended floating-point number.

8.1.1.13 LongDoubleSequence

Description

The LongDoubleSequence data type is an unbounded sequence of LongDouble(s).

8.1.1.14 LongLong

Description

The LongLong primitive type, a specialization of Integer primitive type, is a signed integer range -2^{63} .. $2^{63} - 1$.

Constraints

The LongLong value shall be a LiteralInteger with a range of -2^{63} .. $2^{63} - 1$.

8.1.1.15 LongLongSequence

Description

The LongLongSequence data type is an unbounded sequence of LongLong(s).

8.1.1.16 LongSequence

Description

The LongSequence data type is an unbounded sequence of Long(s).

8.1.1.17 Octet

Description

The Octet primitive type, a specialization of Integer primitive type, is an unsigned integer within range 0..255.

Constraints

The Octet value shall be a LiteralInteger with a range of 0..255.

8.1.1.18 OctetSequence

Description

This type is an unbounded sequence of octets as shown in Figure 8-7

Note – Issue 7655

8.1.1.19 ObjectReference

Description

The ObjectReference primitive type, a specialization of String primitive type, is a stringified object reference of an object.

Constraints

The ObjectReference value is LiteralString.

8.1.1.20 ObjectRefSequence

Description

The ObjectRefSequence data type is an unbounded sequence of ObjectReference(s).

8.1.1.21 Properties

Description

The Properties, as shown in Figure 8-7, is an unbounded sequence of PropertyValue(s), which is used in defining a sequence of id and value pairs.

8.1.1.22 PropertyValue

Description

The PropertyValue is used to hold a property's value.

Attributes

- `id: String`
  The id attribute identifies a specific property of the component.
- `value: primitive datatype or Properties`
  The value attribute contains the property's value.

Constraints

Note – Issue 7895, fix type, Issue 7586 added “either” in text

---

The value attribute shall be either a primitive type (e.g., String, ULong, etc.) or Properties.

Note – Issue 7655, Issue 7586

8.1.1.23 Short

Description

The Short primitive type, a specialization of Integer primitive type, is a signed integer range $–2^{15}.. 2^{15} – 1$.

Constraints

The Short value shall be a LiteralInteger with a range of $-2^{15} .. 2^{15} - 1$.

8.1.1.24 ShortSequence

Description

The ShortSequence data type is an unbounded sequence of Short(s).

8.1.1.25 StringSequence

Description

This type is an unbounded sequence of Strings as shown in Figure 8-7.

8.1.1.26 SystemException

Description

The SystemException exception, as shown in Figure 8-7, denotes a type that is used when an exception is raised by an operation.

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.

Note – Issue 7655, Issue 7586

8.1.1.27 ULong

Description

The ULong primitive type, a specialization of Integer primitive type, is an unsigned integer range $0.. 2^{32} - 1$.

Constraints

The ULong value shall be a LiteralInteger with a range of $0.. 2^{32} - 1$.

8.1.1.28 ULongLong

Description

The ULongLong primitive type, a specialization of Integer primitive type, is an unsigned integer range $0.. (2^{64} - 1)$.

Constraints

The ULongLong value shall be a LiteralInteger with a range of $0 .. 2^{64} - 1$.

8.1.1.29 ULongLongSequence

Description

The ULongLongSequence data type is an unbounded sequence of ULongLong(s).

8.1.1.30 ULongSequence

Description

The ULongSequence data type is an unbounded sequence of ULong(s).

8.1.1.31 UShort

Description

The UShort primitive type, a specialization of Integer primitive type, is an unsigned integer range 0.. $2^{16} – 1$.

Constraints

The UShort value shall be a LiteralInteger with a range of 0.. $2^{16} – 1$.

8.1.1.32 UShortSequence

Description

The UShortSequence data type is an unbounded sequence of UShort(s).

8.1.1.33 WChar

Description

The WChar primitive type represents a wide character that can be used for any character set.

Constraints

The WChar value shall be a LiteralWChar.

8.1.1.34 WCharSequence

Description

The WCharSequence data type is an unbounded sequence of WChar(s).

8.1.1.35 WString

Description

The WString primitive type represents a wide character sting that can be used for any character set.

Constraints

The WString value is a LiteralWString.

8.1.1.36 WStringSequence

Description

The WStringSequence data type is an unbounded sequence of WString(s).

Note – Issue 7895, Added TimeType

8.1.1.37 TimeType

Description

The TimeType, as shown in Figure 8-7, denotes a type that represents time.

Attributes

- `seconds: ULongLong`

  Seconds.
- `nanoseconds: ULongLong`

  Nanoseconds.

Note – Issue TBD added semantics

Semantics

The TimeType attribute is used to define time. Seconds in the future when an event should occur or in the past when an event has occurred. Seconds field is the seconds that have occurred since last synchronization epoch. (Epoch method will be defined by instantiating API.) In most cases, the epoch will occur one time when the box is initialized. Nanosecond offset from the seconds field (described above) when a future event will occur or past event has occurred

Note – Issue 7655

8.1.2 Literal Specifications

Note – Issue 8872, extensions not specializations

A literal specification is an abstract specialization of ValueSpecification that identifies a literal constant being modeled. The literal specifications identified in this section are extensions of the UML LiteralSpecification metaclass.

8.1.2.1 LiteralCharacter

Description

A literal character, an extension of LiteralSpecification, contains a Character-valued attribute.

Attributes

•value : Character

Semantics

A LiteralCharacter specifies a constant Character value.

8.1.2.2 LiteralDouble

Description

A literal double, an extension of LiteralSpecification, contains a Double-valued attribute.

Attributes

•value : Double

Semantics

A LiteralDouble specifies a constant Double value.

8.1.2.3 LiteralFloat

Description

A literal float, an extension of LiteralSpecification, contains a Float-valued attribute.

Attributes

•value : Float

Semantics

A LiteralFloat specifies a constant Float value.

8.1.2.4 LiteralLongDouble

Description

A literal long double, an extension of LiteralSpecification, contains a LongDouble-valued attribute.

Attributes

•value : LongDouble

Semantics

A LiteralLongDouble specifies a constant LongDouble value.

8.1.2.5 LiteralWChar

Description

A literal wide character, an extension of LiteralSpecification, contains a WChar-valued attribute.

Attributes

•value : WChar

Semantics

A LiteralWChar specifies a constant wide character value.

8.1.2.6 LiteralWString

Description

A literal wide string, an extension of LiteralSpecification, contains a WString-valued attribute.

Attributes

•value : WString

Semantics

A LiteralWString specifies a constant wide string value.

8.1.3 Properties

Note – Issues 7688, 7983, 7984, 7985

This section defines the property stereotypes for SWRadio components. A property is a named value denoting an attribute of a class. The property types contained in this package are configure, query, simple, test, structure, and

---

Note – Issue 7895, fix type

---

service. All properties are based upon primitive data type values (e.g., char, ULong, string, etc.). The reason for this is two fold: 1) these primitive types are supported by distributed component middleware and 2) the primitive types allow for a generic mechanism to be built such as deployment, component interaction, and Human Computer Interface (HCI). All the values of a sequence type (simple or structure sequence) are based upon the same property definition, in order to simplify processing within an embedded environment. Simple, Structure, and Sequence properties can be used for configuring and/or querying a component's properties. Test properties are testable properties for a component. There are four subclasses of a SimpleProperty which are CapacityProperty, CharacteristicProperty, CharacteristicSelectionProperty and ExecutableProperty. The ServiceProperty is used to describe the characteristic capabilities and capacities of a Service. The ExecutableProperty is used to describe the executable parameters for an executable implementation (e.g., process, thread). The details of each property type are described in the following subsections.

Table 8-2 – Properties Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
CapacityProperty	Property	ServiceProperty, SimpleProperty		See constraints in section below	Represents capacity property.
CharacteristicProperty	Property	ServiceProperty, SimpleProperty		See constraints in section below	Represents characteristic property.
CharacteristicSelectionProperty	Property	ServiceProperty, SimpleProperty		See constraints in section below	Represents characteristic property that is a simple list.
CharacteristicSetProperty	Property	ServiceProperty, RadioProperty		See constraints in section below	Represents a set of characteristic properties that are of the same classification.
ConfigureProperty	Property	N/A	stepSize	See constraints in section below	Represents a configurable and queryable property.
EnumerationProperty	Class	N/A		See constraints in section below	Represents a enumeration property type where the enumeration literals have an integer value.
ExecutableProperty	Property	SimpleProperty	queryable	See constraints in section below	Represents an executable parameter property.
InputValueProperty	Property	SimpleProperty			Represents an input value parameter property for a test.
QueryProperty	Property	SimpleProperty		See constraints in section below	Represents a queryable property.
<<abstract>>RadioProperty	Property	N/A	integerId, label, maxLatency, range, units	See constraints in section below	Represents the common attributes for all SWRadio property types.
ResultValueProperty	Property	SimpleProperty			Represents a result value for a test.
<<abstract>>ServiceProperty	Property	N/A	capabilityModel, locallyManaged	See constraints in section below	Represents characteristic or capacity property for a service.

Table 8-2 – Properties Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
SimpleProperty	Property	RadioProperty		See constraints in section below	Represents a RadioProperty that contains a primitive value type.
StructProperty	Class	N/A		See constraints in section below.	Represents a struct of SimpleProperties.
TestDefProperty	Class	N/A		See Constraints in section below.	Describes a test definition type and its associated inputs and expected results.
TestProperty	Property	RadioProperty		See constraints in section below.	Describes a test property.

Note – Issue8200 added EnumerationProperty before ExecutableProperty

8.1.3.1 CapacityProperty

Description

The CapacityProperty as shown in Figure 8-8 is a type of ServiceProperty and SimpleProperty that defines a managed or unmanaged dynamic capacity for a ServiceComponent.

Note – Issues 7983, 7586

Constraints

- The CapacityProperty's locallyManaged attribute default value shall be True. The corresponding OCL is as follows:
  context CapacityProperty::locallyManaged:Boolean
  init: true
- Valid values for the CapacityProperty's capabilityModel attribute value shall be: “counter” and “quantity”. The corresponding OCL is as follows:
  context CapacityProperty
  inv validcapabilitymodel: self.capabilityModel = 'counter' or self.capabilityModel = 'quantity'
- CapacityProperty shall have an initial value of ‘quantity’. The corresponding OCL is as follows:
  context CapacityProperty::capabilityModel:String
  init: 'quantity'

Semantics

Note – Issue 7895, fix type

CapacityProperty's type is usually a numeric type (e.g., ULong, float, etc.).

The meanings of the CapacityProperty's capabilityModel attribute values shall be:

- "counter" - This CapacityModel has a capacity of an counter. The capacity value is decremented by one until zero during allocation and incremented by one during deallocation. Allocation fails if the counter is at zero. Example: a sound card with 4 output channels.
- "quantity" - This CapacityModel has a certain capacity that can be consumed. The value of the deployment requirement value is subtracted from the capacity during allocation and added to the capacity during deallocation. The allocation is successful if the capacity has a value that equals or exceeds the value of the deployment requirement. Example: memory size.

Other capabilityModel attribute values can be specified. These other specified capabilityModel attribute values may be managed at the ManagedServiceComponent level or at an ApplicationFactory level.

8.1.3.2 CharacteristicProperty

Description

The CharacteristicProperty is a type a ServiceProperty and SimpleProperty that defines a static characteristic for a ServiceComponent.

Constraints

Note – Issues 7586, 7983

- The CharacteristicProperty's locallyManaged attribute default value shall be False. The corresponding OCL is as follows:
  context CharacteristicProperty::locallyManaged:Boolean
  init: false
- Valid values for the CharacteristicProperty's capabilityModel attribute value shall be: "eq", "ne", "le", "ge", "lt", "gt", "maximum" or "minimum". The corresponding OCL is as follows:
  context CharacteristicProperty
  inv validcapabilitymodel: self.capabilityModel in Set { 'eq', 'ne', 'le', 'ge', 'lt', 'gt', 'maximum', 'minimum' }
- CharacteristicProperty shall have an initial value of 'eq'. The corresponding OCL is as follows:
  context CharacteristicProperty::capabilityModel:String
  init: 'eq'

Semantics

The meanings of the CharacteristicProperty's capabilityModel attribute values shall be:

- "eq" - is an equality comparison between the CharacteristicProperty's value attribute and a deployment requirement (Infrastructure::SWRadio Deployment::SWRadio Artifacts::BasicDeploymentRequirement). They are equal if the CharacteristicProperty's value equals the deployment requirement. If they are equal then the CharacteristicProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement.
- "ne" - is a not equal comparison between the CharacteristicProperty's value attribute and a deployment requirement. They are not equal if CharacteristicProperty's value does not equal the deployment requirement. If they are not equal then the CharacteristicProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement.

- "le" - is a less than or equal comparison between the CharacteristicProperty's value attribute and a deployment requirement. If the deployment requirement is less than or equal to the CharacteristicProperty then the CharacteristicProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement.
- "ge" - is a greater than or equal comparison between the CharacteristicProperty's value attribute and a deployment requirement. If the deployment requirement is greater than or equal to the CharacteristicProperty then the CharacteristicProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement deployment requirement otherwise it cannot satisfy the deployment requirement.
- "lt" - is a less than comparison between the CharacteristicProperty's value attribute and a deployment requirement. If the deployment requirement is less than the CharacteristicProperty then the CharacteristicProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement.
- "gt" - is a greater than comparison between the CharacteristicProperty's value attribute and a deployment requirement. If the deployment requirement is greater than the CharacteristicProperty then the CharacteristicProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement deployment requirement otherwise it cannot satisfy the deployment requirement.
- "Minimum" - behaves as "ge"
- "Maximum" - behaves as "le"

Other capabilityModel attribute values can be specified. These other specified capabilityModel attribute values may be managed at the ServiceComponent level or at an ApplicationFactory level.

8.1.3.3 CharacteristicSelectionProperty

Description

Note – Issue 7985

The CharacterisiticSelectionProperty is a type a ServiceProperty and SimpleProperty that defines a static characteristic for a ServiceComponent. The property contains a list of characteristic values of the same primitive type.

Constraints

Note – Issues 7586, 7983

- The CharacteristicSelectionProperty's locallyManaged attribute default value shall be False. The corresponding OCL is as follows:
  context CharacteristicSelectionProperty::locallyManaged:Boolean
  init: false
- Valid values for the CharacteristicSelectionProperty's capabilityModel attribute value shall be: "selection". The corresponding OCL is as follows:
  context CharacteristicSelectionProperty
  inv validcapabilitymodel: self.capabilityModel = 'selection'
- CharacteristicSelectionProperty shall have an initial value of 'selection'. The corresponding OCL is as follows:
  context CharacteristicSelectionProperty::capabilityModel:String
  init: 'selection'

Semantics

The meanings of the CharacteristicSelectionProperty's capabilityModel attribute values are:

- “selection” - is an equality/match comparison between any CharacteristicSelectionProperty's value attribute and a BasicDeploymentRequirement (Infrastructure::SWRadio Deployment::SWRadio Artifacts). They shall be equal/match if any SelectionCharacteristicProperty's value equals the deployment requirement. If they are equal/match then the CharacteristicSelectionProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement.

Other capabilityModel attribute values can be specified. These other specified capabilityModel attribute values may be managed at the ServiceComponent level or at an ApplicationFactory level.

8.1.3.4 CharacteristicSetProperty

Description

The CharacteristicSetProperty is a type a ServiceProperty and RadioProperty that defines a characteristic that defines the same set of characteristics for a ServiceComponent. Each item in the set is of the same characteristic classification such as library or runtime. All the items in the set represent characteristics that are supported by the Service. Each supported characteristic has a set of qualifiers (e.g., name, version, etc.) that describe the characteristic.

Note – Issue 7985, removed attributes and types noheader sections

Constraints

Note – Issues 7586, 7983, 7985

- The CharacteristicSetProperty's locallyManaged attribute default value shall be False. The corresponding OCL is as follows:
  context CharacterisiticSetProperty::locallyManaged:Boolean
  init: false
- Valid values for the CharacteristicSetProperty's capabilityModel attribute value shall be: “selection”. The corresponding OCL is as follows:
  context CharacteristicSetProperty
  inv validcapabilitymodel: self.capabilityModel = 'selection'
- CharacteristicSetProperty shall have an initial value of ‘selection’. The corresponding OCL is as follows:
  context CharacteristicSetProperty::capabilityModel:String
  init: 'selection'
- Each CharacteristicSetProperty’s attribute type shall be a stereotyped as StructProperty. The corresponding OCL is as follows:
  context CharacteristicSetProperty
  inv: self.allAttributes()->collect(a | a.type.stereotype = ‘StructProperty’)
- Each CharacteristicSetProperty’s attribute type shall be the same type definition. The corresponding OCL is as follows:
  context CharacteristicSetProperty
  inv: self.allAttributes()->collect(a | a.type)->size() = 1

Semantics

The meanings of the CharacteristicSetProperty's capabilityModel attribute values are:

- “selection” - is an equality/match comparison between any CharacteristicSetProperty's value attribute and a DeploymentRequirementQualifier (Infrastructure::SWRadio Deployment::SWRadio Artifacts). They shall be equal/match if any CharacteristicSetProperty's characteristic equals the deployment requirement. If they are equal/match then the CharacteristicSetProperty can satisfy a deployment requirement otherwise it cannot satisfy the deployment requirement.

Other capabilityModel attribute values can be specified. These other specified capabilityModel attribute values may be managed at the ServiceComponent level or at an ApplicationFactory level.

Note – Issue 7689 - Name Problem in 8.1.2.5, Issue 7985 changed definition and name of ConfigQuery Property

8.1.3.5 ConfigureProperty

Description

The ConfigureProperty indicates a configurable and queryable property. There are four types of ConfigureProperty, which are: primitive types, primitive sequence types, StructProperty and StructProperty sequences.

Attributes

Note – Issue 7895, fix types.

- `stepSize: ULong [0..1]`

  The stepSize attribute represents the fact that some properties have discrete increments. An example is a tunable duplexer, which uses a stepper motor to adjust the tuned frequency.

Constraints

- ConfigureProperty isReadOnly attribute shall be false. The corresponding OCL is as folows:
  context ConfigureProperty
  inv validisreadonly: self.isReadOnly = false

Semantics

The ConfigureProperty defines properties associated with the PropertySet interface implementations. The properties supported by a SWRadio component are described in a component's descriptor.

Note – Issue 8200 added EnumerationProperty

8.1.3.6 EnumerationProperty

Description

The EnumerationProperty, an extension of Class, as shown in Table 8-2 defines an enumeration type where the enumeration literals have an integer value.

Constraints

Note – Issue 7586 added constraints

- The EnumerationProperty type shall be integer (Short, UShort, Long, ULong, ULongLong, LongLong).

- Each EnumerationProperty's attribute name shall be unique within the EnumeraationProperty. The corresponding OCL is as follows:
  context EnumerationProperty inv: self.allAttributes()->isUnique(a | a.name)
- Each EnumerationProperty's attribute values shall be unique within the EnumeraationProperty. The corresponding OCL is as follows:
  context EnumerationProperty inv: self.allAttributes()->isUnique(a | a.value)
- Each EnumerationProperty's attribute value shall be in the range of the EnumerationProperty Type.

Semantics

The EnumerationProperty forms an enumeration type definition. EnumerationProperty is legal for integer type properties elements. The EnumerationProperty attributes are enumeration literals, which have an integer value. EnumerationProperty attribute values are implied; if not specified, the initial value is 0 and subsequent values are incremented by 1. This allows a configure, query, or characteristic property to be expressed as an enumeration with integer values.

Note – Issue 7690 - Name problem in Section 8.1.2.6, Removed ConfigureQuerySimpleProperty and ConfigureQuerySimpleSeqProperty, Issue 7742 - removed figure since associations cannot be shown between stereotypes.

8.1.3.7 ExecutableProperty

Description

The ExecutableProperty a type of SimpleProperty as shown in Table 8-2 that defines executable parameters for an ExecutableCode element such as a main process.

Attributes

- `Queryable : Boolean = True`
  The queryable attribute indicates whether or not the ExecutableProperty can be queried for its value. True means the property is queryable.

Constraints

- The ExecutableProperty's value shall be specified.

Note – Issue 7984

8.1.3.8 InputValueProperty

Description

The InputValueProperty, a type of SimpleProperty as shown in Table 8-2, provides the capability to define a input property for a TestDefProperty.

Note – Issue 7985

8.1.3.9 QueryProperty

Description

The QueryProperty, a type of RadioProperty as shown in Table 8-2, provides the capability to define a read-only property that can be queried at run-time by a control command. It cannot be modified by a control command.

Constraints

- The type for QueryProperty shall be constrained to be SWRadio primitive types, primitive sequence types, StructProperty.
- The QueryProperty isReadOnly attribute value shall be always set to true.
  The corresponding OCL is as follows:
  context QueryProperty inv validisreadonly: self.isReadOnly = true

8.1.3.10 RadioProperty

Description

The abstract RadioProperty is an extension of the UML Property that provides the basic attributes for all SWRadio properties definitions.

Attributes

Note – Issue 7985 moved attributes to here from SimpleProperty and ConfigureProperty, Issue 8869 IntegerID type

- `integerId : Long [0..1]`
  The optional integerId attribute is an integer string that represents the identifier for the radio property.
- `label: String [0..1]`
  The optional label attribute contains a property's human readable name that can be used when the name attribute is not human readable.
- `maxLatency: TimeType [0..1]`
  The maxLatency attribute represents the time needed by an attribute to achieve its proper operating status. An example is the gain of an amplifier. The maxLatency indicates the time that the amplifier needs before it attains its operating gain. The latency is measured from power-up.
- `range: Range [0..1]`
  The optional range attribute indicates the allowable min and max values for value attribute.
- `units: String [0..1]`
  The optional units attribute indicates the unit of measure for the value attribute.

Types and Exceptions

```
Range (min: String, max: String)
```

The Range type defines the min and max allowable values for a property. The String value represents an numerical value.

Constraints

Note – Issue 7586

- The integerID attribute when specified shall have precedence over the name attribute for the identification of the RadioProperty.
- The value for the name attribute shall be case sensitive.

Semantics

Note – Issue 7983, Issue 7586

The name or integerId attribute is used for radio property identification. The label attribute is to be used for display purposes when specified instead of the property name. This is useful when the name is not human readable such as a Universal Unique IDentifier (UUID) value.

Note – Issue 7984

8.1.3.11 ResultValueProperty

Description

The ResultValueProperty, a type of SimpleProperty as shown in Table 8-2, provides the capability to define a result property for a TestDefProperty.

8.1.3.12 ServiceProperty

Description

Note – Issue7985 Updated Figure

The abstract ServiceProperty as shown in Figure 8-8, defines capability and/or capacity characteristics for a ServiceComponent.

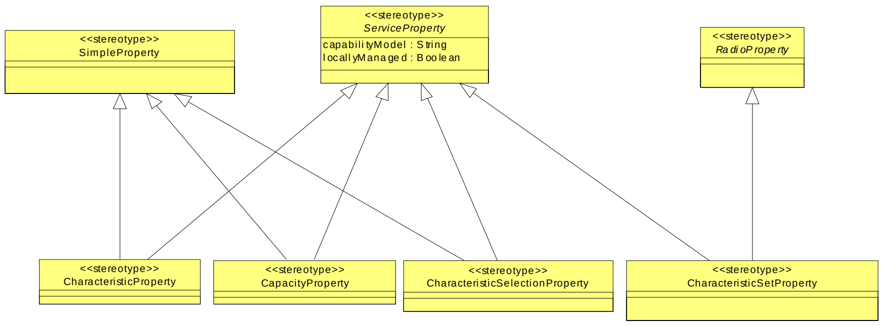


Figure 8-8 – ServiceProperty Definition

Attributes

Note – Issue7985, moved maxLatency to RadioProperty

- `capabilityModel: Boolean`

  The capabilityModel attribute identifies the CapabilityModel to be used for this ServiceProperty

- `locallyManaged: Boolean`

  The locallyManaged attribute indicates whether a Service manages this capability or capacity. A value of True means a Service manages the capacity otherwise it does not.

Constraints

Note – Issue 7983

- ServiceProperty shall have a value (not null) when the locallyManaged attribute value is false. The corresponding OCL is as follows:
  context ServiceProperty
  inv requiredValue: self.locallyManaged and self.value.nonEmpty()

Semantics

The ServiceProperty's capabilityModel attribute indicates the type of capabilityModel to be used to determine if the ServiceProperty can satisfy the deployment requirement.

8.1.3.13 SimpleProperty

Description

Note – Issue 7895, fix type

8.1.3 Properties

The SimpleProperty is a type of RadioProperty that defines a primitive data type (e.g., character, ULong, string, etc.).

Note – Issue 7985 moved range and attributes to RadioProperty, Issue 8200 remove enumerations attribute and Types, Updated Constraints text for valid type. OCL for valid needs to be done to agree with text (TBD). This will be done as part of Issue 7586

Note – Issue 7895 Removed Range typedef from here, defined in 8.2.1

Constraints

- The value shall comply with the property type.
- The type shall be limited to the SWRadio primitive types (Boolean, Character, Float, Double, Long, LongDouble, LongLong, ObjectReference, Octet, Short, String, ULong, ULongLong, UShort) and EnumerationProperty. The corresponding OCL is as follows:
  def: primTypes : Set = {Boolean, Character, Float, Double, Long, LongDouble, LongLong, ObjectReference, Octet, Short, String, ULong, ULongLong, UShort}
  context SimpleProperty
  inv validtype: primTypes->exists(t : self.oclIsTypeOf(t))
- The range attribute min and max values shall be compatible with the type and max is greater than or equal to min. The corresponding OCL is as follows:
  context simpleProperty
  inv validrange: self.range.max >= self.range.min
  inv validmax: (self.oclIsTypeOf(Boolean) or self.oclIsTypeOf(Character) or self.oclIsTypeOf(ObjectReference) or self.oclIsTypeOf(String) or self.oclIsTypeOf(WString) or self.oclIsTypeOf(WChar)) or ((self.oclIsTypeOf(Float) and (self.range.max >= 0 or self.range.max <= 255)) or (self.oclIsTypeOf(Double) and (self.range.max >= 0 or self.range.max <= 255)) or (self.oclIsTypeOf (Long) and (self.range.max >= -2^{31} or self.range.max <= 2^{31} - 1)) or (self.oclIsTypeOf (LongDouble) and (self.range.max >= 0 or self.range.max <= 255)) or (self.oclIsTypeOf(LongLong) and (self.range.max >= -2^{63} or self.range.max <= 2^{63} - 1)) or (self.oclIsTypeOf(Octet) and (self.range.max >= 0 or self.range.max <= 255)) or (self.oclIsTypeOf(Short) and (self.range.max >= -2^{15} or self.range.max <= 2^{15} - 1)) or (self.oclIsTypeOf(ULong) and (self.range.max >= 0 or self.range.max <= 2^{32} - 1)) or (self.oclIsTypeOf(ULongLong) and (self.range.max >= 0 or self.range.max <= 2^{64} - 1)) or (self.oclIsTypeOf(UShort) and (self.range.max >= 0 or self.range.max <= 2^{16} - 1)))
  inv validmin: (self.oclIsTypeOf(Boolean) or self.oclIsTypeOf(Character) or self.oclIsTypeOf(ObjectReference) or self.oclIsTypeOf(String) or self.oclIsTypeOf(WString) or self.oclIsTypeOf(WChar)) or ((self.oclIsTypeOf(Float) and (self.range.min >= 0 or self.range.min <= 255)) or (self.oclIsTypeOf(Double) and (self.range.min >= 0 or self.range.min <= 255)) or (self.oclIsTypeOf (Long) and (self.range.min >= -2^{31} or self.range.min <= 2^{31} - 1)) or (self.oclIsTypeOf (LongDouble) and (self.range.min >= 0 or self.range.min <= 255)) or (self.oclIsTypeOf(LongLong) and (self.range.min >= -2^{63} or self.range.min <= 2^{63} - 1)) or (self.oclIsTypeOf(Octet) and (self.range.min >= 0 or self.range.min <= 255)) or (self.oclIsTypeOf(Short) and (self.range.min >= -2^{15} or self.range.min <= 2^{15} - 1)) or

(self.oclIsTypeOf(ULong) and (self.range.min >= 0 or self.range.min <= 2^{32} - 1)) or (self.oclIsTypeOf(ULongLong) and (self.range.min >= 0 or self.range.min <= 2^{64} - 1)) or (self.oclIsTypeOf(UShort) and (self.range.min >= 0 or self.range.min <= 2^{16} - 1)))

- The value shall comply with the range when the type is a numeric. The corresponding OCL is as follows:
  context SimpleProperty
  inv validvaluerange: (self.oclIsTypeOf(Boolean) or self.oclIsTypeOf(Character) or self.oclIsTypeOf(ObjectReference) or self.oclIsTypeOf(String) or self.oclIsTypeOf(WString) or self.oclIsTypeOf(WChar)) or ((self.oclIsTypeOf(Float) or self.oclIsTypeOf(Double) or self.oclIsTypeOf (Long) or self.oclIsTypeOf (LongDouble) or self.oclIsTypeOf(LongLong) or self.oclIsTypeOf(Octet) or self.oclIsTypeOf(Short) or self.oclIsTypeOf(ULong) or self.oclIsTypeOf(ULongLong) or self.oclIsTypeOf(UShort)) and (self.value <= self.range.max and self.value >= self.range.min))

Note – Issue 7985, removed SimpleSequenceProperty

Note – Issue 7985, changed description and removed attributes and modified constraints

8.1.3.14 StructProperty

Description

The StructProperty is a type that contains a list of SimpleProperties.

Constraints

- Each StructProperty's attribute name must be unique within the StructProperty. The corresponding OCL is as follows:
  context StructProperty
  inv: self.allAttributes()->isUnique(a | a.name)
- Each StructProperty's attribute shall be a SimpleProperty or primitive type. The corresponding OCL is as follows:
  context StructProperty
  inv: self.allAttributes()->forAll(a | a.stereotype.name = 'SimpleProperty' or primTypes->exists(t : a.oclIsTypeOf(t)))
- The multiplicity for each StructProperty's attribute shall be one. The corresponding OCL is as follows: context StructProperty
  inv: self.allAttributes()->forAll(a | a.size = 1)

Note – Issue 7985, removed StructSequenceProperty

Note – Issue 7984

8.1.3.15 TestDefProperty

Description

The TestDefProperty, a type of Class as shown in Table 8-2, provides the capability to define the input parameters for the test and the results that can be returned for a test.

Constraints

- The attribute shall be InputValueProperty or ResultValuePoperty stereotypes. The corresponding OCL is as follows:
  context TestDefProperty
  inv: self.all Attributes()->forAll(a | a.stereotype.name = 'InputValueProperty' or a.stereotype.name = 'ResultValueProperty')
- There shall be at least one ResultValueProperty attribute defined. The corresponding OCL is as follows:
  context TestDefProperty
  inv: self.allAttributes()->exists(a | a.stereotype.name = 'ResultValueProperty')
- Each Attribute name shall be unique. The corresponding OCL is as follows:
  context TestDefProperty
  inv: self.allAttributes()->isUnique(a | a.name)
- Each Attribute value shall be specified (not null). The corresponding OCL is as foll-lows:
  context TestDefProperty
  inv: self.allAttributes()->forAll(a | a.value->notEmpty())

8.1.3.16 TestProperty

Description

The TestProperty, a type of RadioProperty as shown in Table 8-2, provides the capability to define the input parameters for the test and the results that can be returned for a test.

Constraints

Note – Issue 7984

- The type for a TestProperty shall be stereotype as TestDefProperty.

8.1.4 Interface and Port Stereotypes

This section defines the port, property, and interface stereotypes for SWRadio interfaces and components as depicted in Table 8-3. Port stereotypes categorize the function of the various ports within a SWRadio component and the type of interfaces associated with these ports. Interface stereotypes categorize the type of interface provided by or used by SWRadio components. Property stereotypes are defined for SWRadio interface and component attributes to indicate the type of property visually and how the property is going to be managed. These stereotypes are used to define elements in the UML Profile for SWRadio, PIM Facilities, and by application and device component developers.

Note – Issue 7693 - Added IStreamControl to Table 8-3. Issue 7985 removed configquery &

Table 8-3 – Interface & Port Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
ControlPort	Port	SWRadioPort		Only associated with IControl interfaces.	Represents a port for control management.
DataPort	Port	SWRadioPort		Only associated with IData interfaces.	Represents a port the sends or receives data using an IDATA interface
DataControlPort	Port	SWRadioPort		Only associated with IDataControl interfaces.	Represents a port the sends or receives data with control using an IDataControl interface
IData	Interface	SWRAPI			Provides the mechanism to send data.
IDataControl	Interface	SWRAPI			Provides the mechanism to send data with control.
IControl	Interface	SWRAPI			Provides the mechanism for sending or receiving control
IStream	Interface	SWRAPI			Provides the mechanism to manage streams
IStreamControl	Interface	SWRAPI			Provides the mechanism to manage streams that contain control information in addition to user data
ReadOnly	Property	N/A		IsReadOnly = True	Represents a queryable property for a SWRadio interface or component that has an associated get operation.
ReadWrite	Property	N/A		IsReadOnly = False	Represents a configurable and queryable property for a SWRadio interface or component that has associated set and get operations.
ServicePort	Port	SWRadioPort			Represents a port that provides or uses a SWRadio Service
StreamControlPort	Port	SWRadioPort	SAP, Address	Only associated with IStreamControl interfaces.	Represents a port that sends or receives a continuous data stream, with occasional control information
StreamPort	Port	SWRadioPort	SAP, Address	Only associated with IStream interfaces.	Represents a port that sends or receives continuous streaming of data
SWRadioPort	Port	N/A		Only associated with SWRAPI interfaces.	Represents a SWRadio port that is associated with SWRAPIs
SWRAPI	Interface	N/A			Represents an implemented or required application or logical device component interface.

Query. Issue 7785 replace waveform application with application

8.1.4.1 StreamPort

Description

The StreamPort defines a streaming port that receives or sends continuous data.

Tags

Note – Issue 7895, fix types.

● `sap: ULong`	The sap (Service Access Point) attribute contains a SAP identifier
● `address: OctetSequence`	The address attribute contains address information.

8.1.5 Resource Components

Note – Issue 7672 Resolution (SWRadio Component to ResourceComponent), Issue 7742 Broke the section into 2 subsections, one for model library interfaces defined in the profile and the second for component stereotypes defined in the profile. Issue 7785 replace waveform an/or waveform application with application

This section defines the interfaces for a SWRadio ResourceComponent along with component stereotypes. The Resource Component stereotypes are extensions of the UML 2.0 Component (UML2.0::Components::BasicComponents) classifiers as described in section 8.1.5.2. Figure 8-9, depict the base interfaces used in defining software radio components for applications, logical devices, and communication channels. The following subsection describe the details of these base interfaces (8.1.5.1), which are management interfaces for SWRadio components along with the SWRadio component stereotypes (8.1.5.2).

Note – Issue 7742 added new subsection for modelLibrary M1 interfaces. SubSections in this were renumered to be a header 5.

8.1.5.1 Resource Components Interfaces

This section defines the resource component modelLibrary interfaces contained in the profile definition as shown in Figure 8-9, which are: ComponentIdentifier, ControllableComponent, LifeCycle, PortConnector, PortSupplier, PropertySet, Resource, ResourceFactory, and TestableObject that are described in the following subsections. These interfaces provide basic management interfaces for SWRadio component developers.

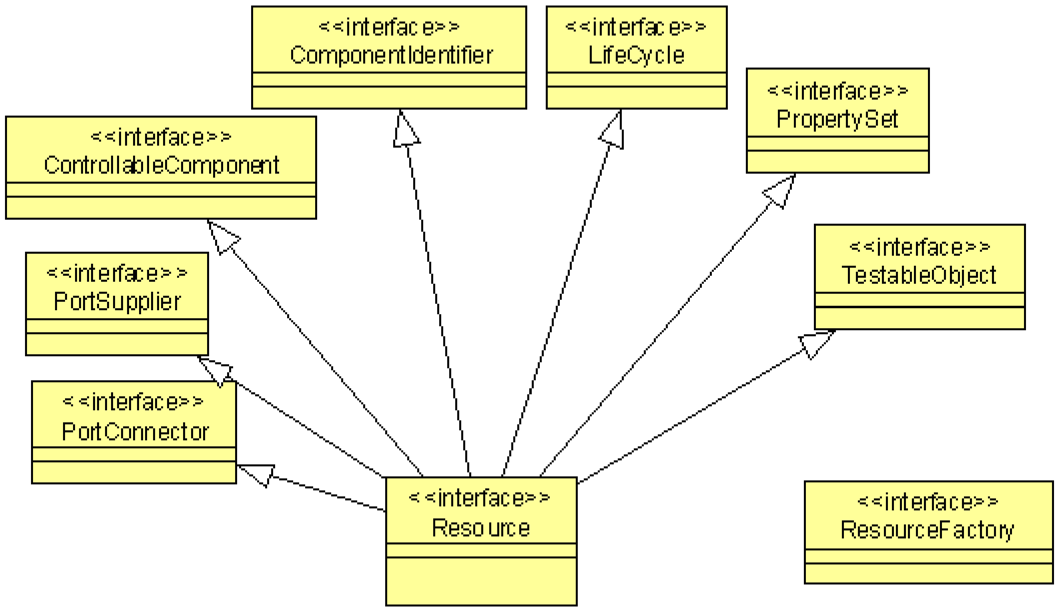


Figure 8-9 – Resource Interfaces Overview

Types and Exceptions

● `<<exception>>UnknownProperties (invalidProperties : Properties)`

The UnknownProperties <<exception>> indicates a set of properties unknown by the component.

8.1.5.1.1 ComponentIdentifier

Description

The ComponentIdentifier interface defines the identifier operations for a SWRadio's component.

Attributes

● `<<readonly>>identifier: String`

The unique identifier for a component.

8.1.5.1.2 ControllableComponent

Description

The ControllableComponent interface defines the generic operations for controlling a SWRadio's components.

Attributes

- `<<readonly>>started: Boolean`

  The started atribute indicates if a Resource has been started or not. A value of True indicates the start has been performed successfully. A value of False means stop mode of operation.

Operations

`Note - Issue 7904`

- `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. The behavior when entering into an operating condition is component implementation specific. The component implementation’s current internal state (e.g. current settings of data structures, memory allocations, hardware device configurations, etc.) is used as the operational starting point. This operation does not return a value. The start operation shall raise the StartError exception if an error occurs while starting the component.

`Note - Issue 7904`

- `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 puts a component in a non-operating condition. The behavior when exiting the operating state is component implementation specific. This operation does not return a value. The stop operation shall raise 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.

8.1.5.1.3 LifeCycle

Description

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

Operations

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

  The purpose of the initialize operation is to provide a mechanism to set a component to a known initial state. (e.g., 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 shall raise the InitializeError when an initialization error occurs.
- `releaseObject(): {raises = (ReleaseError)}`

  The purpose of releaseObject is to provide a means by which acomponent may be removed. The releaseObject operation shall release all internal memory allocated by the component. The releaseObject operation shall remove the component from the Operating Environment (OE). This operation does not return a value. The releaseObject operation shall raise the ReleaseError when a release error occurs.

Types and Exceptions

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

  The InitializeError exception indicates that an error occurred during component initialization. The errorMessage attribute is component-dependent and provides additional information describing why the error occurred.
- `<<exception>>ReleaseError(errorMessage: StringSequence)`

  The ReleaseError exception indicates that an error occurred during component releaseObject. The errorMessage attribute is component-dependent and provides additional information describing why the error occurred.

8.1.5.1.4 PortConnector

Description

The PortConnector interface provides operations for managing associations between ports. The PortConnector interface is used to connect a required port to a provided port.

Operations

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

  The connectPort operation shall make a connection to a component's provided port identified by its input parameters. This operation does not return a value. The connectPort operation shall support all of the required ports identified in the component's descriptor. The connectPort operation shall raise InvalidPort when connection is an invalid connection for this Port. The connectPort operation shall raise OccupiedPort when unable to accept the connections because the Port is already fully occupied.

Note – Issue 7580 Resolution

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

  The disconnectPort operation shall break the connection to the component. The connection is identified by requiredPortName and connectionId. The disconnectPort operation shall raise 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 ( errorCode: UnsignedShort, msg: String)`

  The InvalidPort exception indicates one of the following errors has occurred in the specification of a Port association:
  errorCode 1 means the connection (Provided Port) component is invalid or illegal object reference,
  errorCode 2 means the connectionId is not known (not used by this Port).
  errorCode 3 means the Required Port name does not exist for this component.
- `<<exception>>OccupiedPort`

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

Constraints

Note – Issue 7586, added requirements

- The PortConnector interface shall support all the used or required ports as specified in the component’s descriptor that relealizes this interface.

Semantics

Note – Issue 7580 Resolution (last three lines of paragraph)

A component realizes operations for transferring data and control. The component also establishes the meaning of its data and control values. Examples of how components may use ports include: push or pull, synchronous or asynchronous, mono- or bi-directional, and whether to use flow control (e.g., pause, start, stop). The nature of PortConnector, fan-in, fan-out, or one-to-one, is component dependent. A required port may support several connections. How components' ports are connected is described in a component assembly descriptor. The input connectionId is a unique identifier used by disconnectPort when breaking this specific connection from the required port identified by the input requiredPortName. The connectionId is unique at the required port level.

8.1.5.1.5 PortSupplier

Description

Note – Issue 7579 Resolution

This interface provides the getProvidedPorts operation for components that have provided ports.

Operations

Note – Issue 7579 Resolution

- `getProvidedPorts(inout ports: PortSequence): {raises = ( UnknownPorts )}`

  The getProvidedPorts operation provides a mechanism to obtain a component's provided ports in form of a sequence of name/value pairs, where each name corresponds to a provided port's name and the corresponding value is the provided port reference to be returned. The getProvidedPorts operation shall return all the component provided ports if the ports argument is zero size. The getProvidedPorts operation shall return only those provided ports specified in the ports argument if the ports argument is not zero size. The getProvidedPorts operation shall support all of the provided ports identified in the component's descriptor. The getProvidedPorts operation shall raise UnknownPorts when one or more provided port names being requested are not known by the component.

Types and Exceptions

Note – Issue 7579 Resolution

- `PortType (name: String, object: Port)`

  PortType defines a structure that associates a name with a port.
- `PortSequence`

  PortSequence provides an unbounded sequence of PortType.
- `<<exception>>UnknownPorts (invalidPorts: StringSequence)`

  The UnknownPorts exception is raised when one or more provided ports being requested are not known by the component. The invalidPorts attribute returned indicates the requested provided ports that were invalid.

Constraints

Note – Issue 7586, added requirements

- The PortSupplier interface shall support all the provided interfaces as specified in the component's descriptor that relealizes this interface

8.1.5.1.6 PropertySet

Description

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

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 shall assign values to the component's properties as indicated in the configProperties argument. This operation does not return a value. The configure operation shall raise PartialConfiguration when some configuration properties were successfully set and some configuration properties were not successfully set.

The configure operation shall raise 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 shall return all the component queryable properties if the input configProperties are zero size. The query operation shall return only those id/value pairs specified in the input configProperties if the configProperties are not zero size. The query operation shall raise UnknownProperties when one or more properties being requested are not known by the component.

Types and Exceptions

`Note - Issue 7695 - missing explanation for invalid propoerties`

- `<<exception>>InvalidConfiguration (msg: String, invalidProperties: Properties)`

  The InvalidConfiguration exception indicates the configuration of a component has failed (no configuration at all was done). The msg attribute is component-dependent, providing additional information describing the reason why the error occurred. The returned invalidProperties attribute indicates the properties that were not accepted by the component.

`Note - Issue 7695 - Missing explanation for invalid properties`

- `<<exception>>PartialConfiguration (reasons: StringSequence,invalidProperties: Properties)`

  The PartialConfiguration exception indicates the configuration of a Component was partially successful. The reasons attribute is component-dependent, providing additional information describing the reasons why the error occurred. The returned invalidProperties attribute indicates the properties that were not accepted by the component.

Constraints

Note – Issue 7586, added requirements

Valid properties for the configure operation shall be ConfigureProperty(s).

Valid properties for the query operation shall be:

- ConfigureProperty and QueryProperty properties, or
- ServiceProperty properties whose locallyManaged attribute value is True, or
- ExecutableProperty properties whose queryable attribute value is True.

The value attribute for each PropertyValue in the Properties shall be in its native form as specified by the RadioProperty definition.

The PropertySet interface shall support the configure and query type properties as specified in the component's descriptor that realizes this interface.

The mapping to the ConfigureProperty and QueryProperty types to the ResourceComponent's configure and query's operation properties parameter are:

1. A SimpleProperty or primitive type corresponds to a PropertyValue item in Properties list. The PropertyValue item's id matches the Configure or Query Property's name or integer attribute and the PropertyValue item's value matches the Configure or Query Property's value attribute but is converted to a format that agrees with the Configure or Query Property's type attribute.
2. A SimpleProperty sequence or primtive type sequence corresponds to a PropertyValue item in Properties list. The PropertyValue item's id matches the Configure or Query Property's name or integer attribute and the PropertyValue item's value matches the Configure or Query Property's values attribute that is converted to a primitive sequence type (as described in Section 8.1.1 Base Types), which agrees with the Configure or Query Property's type attribute.
3. A StructProperty corresponds to a PropertyValue item in Properties list. The PropertyValue item's id matches the Configure or Query Property's name or integer 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. StructProperty sequencecorresponds to a PropertyValue item in Properties list. The PropertyValue item's id matches the Configure or Query Property's name or integer 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.

The ExecutableProperty, CapacityProperty, and CharacteristicProperty shall follow the SimpleProperty format for the query operation. The CharacteristicSelectionProperty shall follow the SimpleProperty sequence format for the query operation. The CharacteristicSetProperty shall follow the StuctProperty sequence format for the query operation.

Semantics

The type of property is known by its name or integerId attribute.

8.1.5.1.7 Resource

Description

Note – Issue 7696 - missing reference to the figure

The Resource interface, as shown in Figure 8-9, provides a common API for the control and configuration of software radio components (applications and device). The Resource is a specialization of the ComponentIdentifier, ControllableObject, LifeCycle, PortSupplier, PortConnector, PropertySet, and TestableObject interfaces.

PortSupplier, PropertySet, Resource, ResourceFactory, and TestableObject that are described in the following subsections. These interfaces provide basic management interfaces for SWRadio

8.1.5 Resource Components

Note – Issue7588, 7904 Resolutions - Added figure, constraints and semantics sections

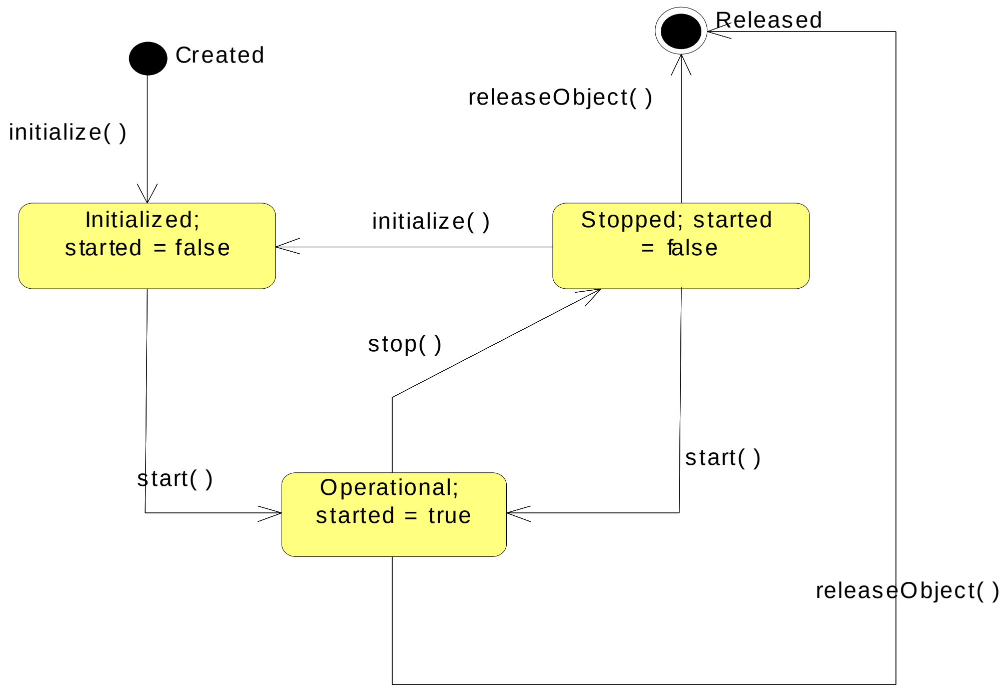


Figure 8-10 – Resource Interfaces Overview

Constraints

A realization of the Resource interface shall result in a specialized clarification of the inherited ControllableObject, Lifecycle, and PortConnector interface behaviors that is consistent with the following items and Figure 8-10:

- A StartError exception shall be generated if the start operation from ControllableObject is called before at least one call is made to the initialize operation from Lifecycle.
- An InitializeError shall be generated if initialize operation from Lifecycle is called while the Resource component's ControllableObject started attribute is true.
- The behavior of the stop operation from ControllableObject shall maintain the component's current configuration to allow subsequent start operations to resume from configuration present at stop, assuming no other LifeCycle, PortConnector, PropertySet, or TestableObject operations are exercised while stopped.
- The disconnectPort operation shall perform any cleanup associated with object being disconnected before completing the disconnect operation. The specific cleanup processing is Resource component implementation dependent.

- Use of the releaseObject operation from LifeCycle while the Resource component's ControllableObject started attribute is true shall result in a behavior consistent with first initiating the Resource's stop operation, then calls to disconnectPort on each of the Resource component's ports, and then the releaseObject behavior itself.

Semantics

The Resource PropertySet implementation is not inhibited by the stop operation. However, the configure behavior impacts to the Resource while stopped are limited in scope to updating internal data structures, in preparation for the next start operation.

The Resource Port Supplier and Port Connector implementation is not inhibited nor impacted by the stop operation, all methods of these two interfaces are supported, behavior unchanged. However, the behavior of the provided ports themselves is impacted. The impact of the stop is port implementation specific. The usual case is that port behavior is halted upon being stopped (started attribute is false). Resource component usage of connected port capabilities will cease and any access to provided port capabilities would result in error notification. An example would be an IO port that, upon being stopped, prevents further data from being pushed out and allows no further data to be pushed in. A notable exception would include status providing ports that would remain active even while stopped to maintain good standing with observing components.

Note – Issue 7742 Added ResourceFactory interface since the ResourceFactory stereotype contained M1 operations and types.

8.1.5.1.8 ResourceFactory

Description

The ResourceFactory interface provides an optional mechanism for the management of ResourceComponents.

Operations

- `createResource(in resourceId: String, in qualifiers: Properties, Return ResourceComponent): {raises = (CreateResourceFailure)}`

  The createResource operation provides the capability to create a ResourceComponent or retrieve an existing ResourceComponent. The resourceId parameter is the identifier for ResourceComponent. 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 a ResourceComponent created by a ResourceFactory.

  If no ResourceComponent exists for the given resourceId, the createResource operation shall create a ResourceComponent. otherwise the createResource operation shall return an existing ResourceComponent whose identifier attribute matches the input resourceId.The createResource operation shall assign the given resourceId to a new ResourceComponent's identifier attribute.

  The createResource operation shall raise the CreateResourceFailure exception when it cannot create the ResourceComponent and cannot find an existing ResourceComponent that contains the resourceId.

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

  The releaseResource operation provides the mechanism of releasing the Resource in the environment on the server side. The releaseResource operation shall release the ResourceComponent from the ResourceFactory as identified by the input resourceId parameter when the number of create requests matches the number of release requests for the input resourceId.

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

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

  The shutdown operation provides the mechanism for releasing the ResourceFactoryComponent from the environment on the server side. The shutdown operation results in the ResourceFactoryComponent being unavailable to any subsequent calls to its component reference (i.e. it is released from the environment). The shutdown operation shall raise the ShutdownFailure exception if unable to terminate the ResourceFactoryComponent.

Types and Exceptions

- `<<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., CF_NOTSET, CF_EBADMSG, CF_EINVAL, CF_EMSGSIZE, CF_ENOMEM). The message is component-dependent, providing additional information describing the reason for the error.
- `<<exception>>InvalidResourceId`

  The InvalidResourceId exception indicates the resourceId does not exist in the ResourceFactoryComponent.
- `<<exception>>ShutdownFailure (msg: String)`

  The ShutdownFailure exception indicates that the shutdown method failed to release the ResourceFactoryComponent from the operating environment due to the fact the Factory still contains ResourceComponents. The message is component-dependent, providing additional information describing why the shutdown failed.

Constraints

The created ResourceComponent's identifier attribute shall be the resourceId parameter value.

Semantics

A ResourceFactory interface is used to create and release a ResourceComponent.

8.1.5.1.9 TestableObject

Description

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

Operations

- `runTest(in testId:String, 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 shall use the testId parameter to determine which of its predefined test implementations should be performed. The testValues parameter Properties (id/value pair(s)) shall be used to provide additional information to the implementation-specific test to be run. The runTest operation shall return the result(s) of the test in the testValues parameter.

The runTest operation shall raise UnknownTest when there is no underlying test implementation associated with the input testId given.

The runTest operation shall raise UnknownProperties when the input parameter testValues contains input test parameters that are invalid. 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.

Constraints

Note – Issue 7586, modifed requirements

The TestableObject interface shall support all the TestProperty(s) as stated in the component’s descriptor that realizes this interface.

The format for a TestProperty’s InputValueProperty(s) and ResultValueProperty(s) for a test shall be as described for a SimpleProperty in the PropertySet section 8.1.5.1.6.

Semantics

The testid parameter corresponds to the name or integerId attribute of the TestProperty. Each TestProperty’s InputValueProperty maps to the testValues parameter for input to the test. Each TestProperty’s ResultValueProperty maps to the testValues, which are the return values for the test being performed.

Note – Issue 7742 added new subsection for M2 stereotypes. SubSections in this section were renumered to be a header 5. Updated figures to M1 Illustrations and changed Assoications header to be M1 Associations. For each component definition, a constraint was added for the interface that is realized by that component type. Removed Resource from table below.

8.1.5.2 Resource Components Stereotypes

Note – Issue 7586, Added constraints text in table

8.1.5 Resource Components

This section defines the SWRadio resource component stereotypes contained in the profile definition as shown in Table 8-4, which are: ResourceComponent, ResourceFactoryComponent, and SWRadioComponent that are described in the following subsections. These stereotypes provide the basic component definitions for SWRadio component developers. The SWRAPI stereotype denotes SWRadio interfaces that are used by or realized by Re-

Table 8-4 – Resource Components Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
ResourceComponent	Component	SWRadioComponent		See constraints in section below	Represents a component of a application.
ResourceFactoryComponent	Component	SWRadioComponent		See constraints in section below	Manages ResourceComponent(s)
SWRadioComponent	Component	N/A		See constraints in section below	Represents the base definition for all SWRadio Components.
swrapiRealization	Association	N/A			Represents an implemented or required application or logical device component interface.
swrapiUsage	Association	N/A			Describes an association that realizes a SWRAPI.

sourceComponent. SWRAPI interfaces are defined in the PIM Facilities of this specification.

8.1.5.2.1 ResourceComponent

Description

Note – Issue 7742 Updated Figure, added association for QueryProperty, updated wording in constraints

The ResourceComponent, as shown in Table 8-4, provides the component definition for software radio resource component. Figure 8-11 depicts the property associations for a ResourceComponent.

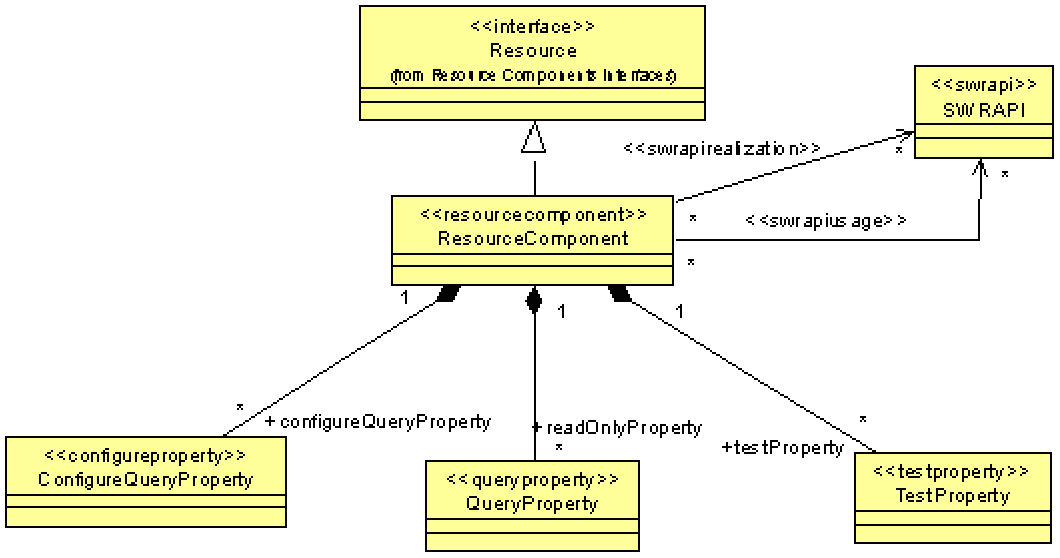


Figure 8-11 – ResourceComponent M1 Illustration

M1 Associations

Note – Issue 8873 Resolution - Replacement of non-existent ConfigureQuertyProperty type with the ConfigureProperty type

- `configureQueryProperty: ConfigureProperty [*]`

  A ResourceComponent may have zero to many configurable and queryable properties.
- `readOnlyProperty: QueryProperty [*]`

  A ResourceComponent may have zero to many query properties.
- `testProperty: TestProperty [*]`

  A 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.

Constraints

Note – Issue 7742, 7586 moved PropertySet constraints to PropertySet section, Issue 7742, added interface constraint

- The ResourceComponent shall realize the Resource interface.

Semantics

The ResourceComponent configure, query, and start operations are not inhibited by the stop operation.

8.1.5.2.2 Resource Factory Component

Description

Note – Issue 7742, updated figure and associations text, renamed component

The ResourceFactory class, as shown in Table 8-4, provides an optional mechanism for the management of Resources. Figure 8-12 depicts the associations for a ResourceFactoryComponent, which are described below.

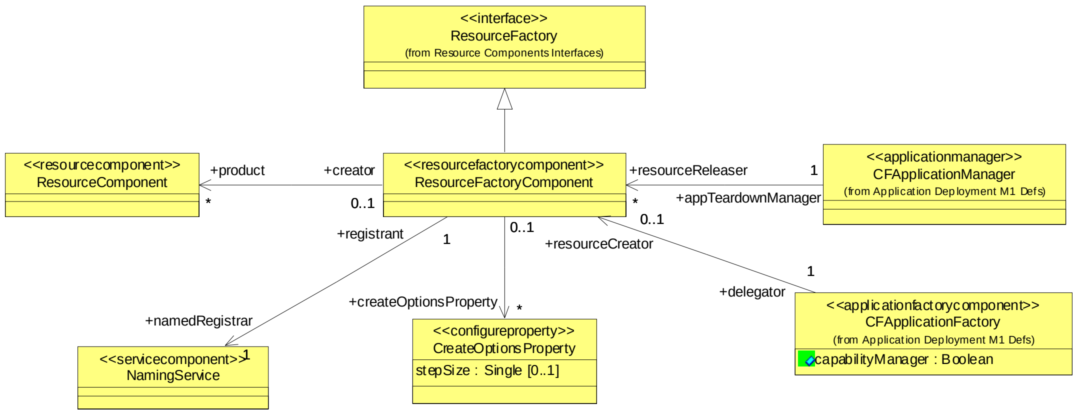


Figure 8-12 – ResourceFactoryComponent M1 Illustration

M1 Associations

Note – Issue 8873 Resolution - Replacement of non-existent ConfigureQueryProperty type with the ConfigureProperty type

- `createOptionsProperty: ConfigureProperty [*]`

  These are the properties that a ResourceFactoryComponent understands and can used when creating up a Resource.

- `namedRegistrar: NamingService`

  The NamingService that contains a named ResourceFactoryComponent reference.

- `product: Resource [*]`

  A ResourceComponent can be created from a ResourceFactoryComponent.

Constraints

The ResourceFactoryComponent shall realize the ResourceFactory interface.

Semantics

A ResourceFactoryComponent is used to create and release a ResourceComponent. When multiple clients have obtained a reference to the same ResourceComponent, the ResourceFactoryComponent must not release the ResourceComponent until release requests have been received from all the clients that issued the create request. Application and Waveform developers are not required to use ResourceFactoryComponents for their application definition. ResourceFactoryComponent provides the mechanism of creating separate process threads for each component created in the ResourceFactory.

8.1.5.2.3 SWRadioComponent

Description

Note – Issue 7742, updated figure to be a M1 type, added constraint for interface, updated assocations, removed attributes since this is M1 data.

The SWRadioComponent, as shown in Table 8-4, is extension of the UML component. Figure 8-13 depicts the relationships for any software radio component.

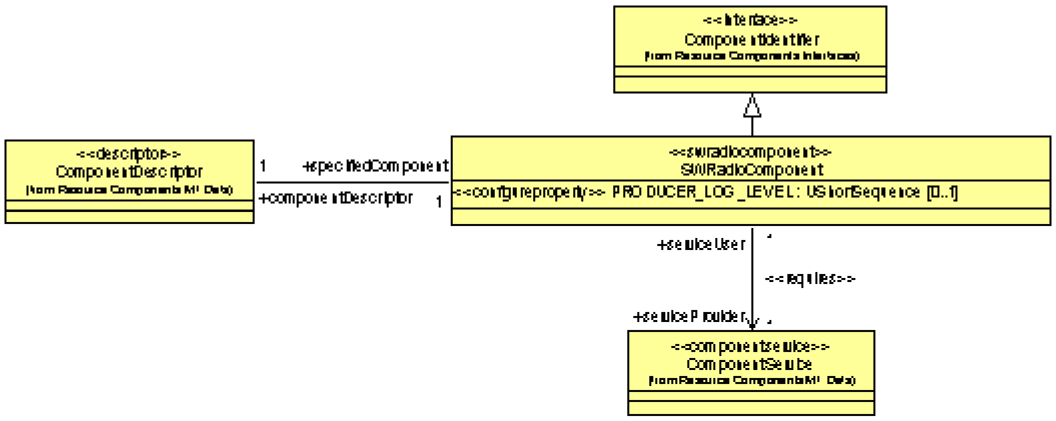


Figure 8-13 – SWRadioComponent M1 Illustration

M1 Associations

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

Constraints

A SWRadioComponent shall realize the ComponentIdentifier interface.

Semantics

SWRadioComponent may implement a ConfigureProperty with a name of "PRODUCER_LOG_LEVEL". The PRODUCER_LOG_LEVEL ConfigureProperty provides the ability to "filter" the log message output of a SWRadioComponent. This property may be configured via the PropertySet interface to output only specific log levels.

SWRadioComponents shall output only those log records to a LogService that correspond to enabled log level values in the PRODUCER_LOG_LEVEL attribute. Log levels that are not in the PRODUCER_LOG_LEVEL attribute are disabled. SWRadioComponents shall use their identifier attribute in the log record output to the LogService. SWRadioComponents shall operate normally in the case where the connections to a LogService are nil or an invalid reference.

8.1.6 Device Components

Note – Issue 7742 Broke the section into 2 subsections, one for model library (M1) interfaces defined in the profile and the second for component stereotypes (M2) defined in the profile.

The Device Components sections define the set of interfaces and component stereotypes used to communicate and manage SWRadio physical devices. The component stereotypes are depicted in the Table 8-5 below, which are extensions of the UML Component. The following subsections describe the details of logical device interfaces (8.1.6.1) and component stereotypes (8.1.6.2).

8.1.6.1 Device Component Interfaces

Note – Issue 8842 - rename DeviceAggregation to DeviceComposition

This section defines the modelLibrary device component interfaces contained in the profile definition as shown

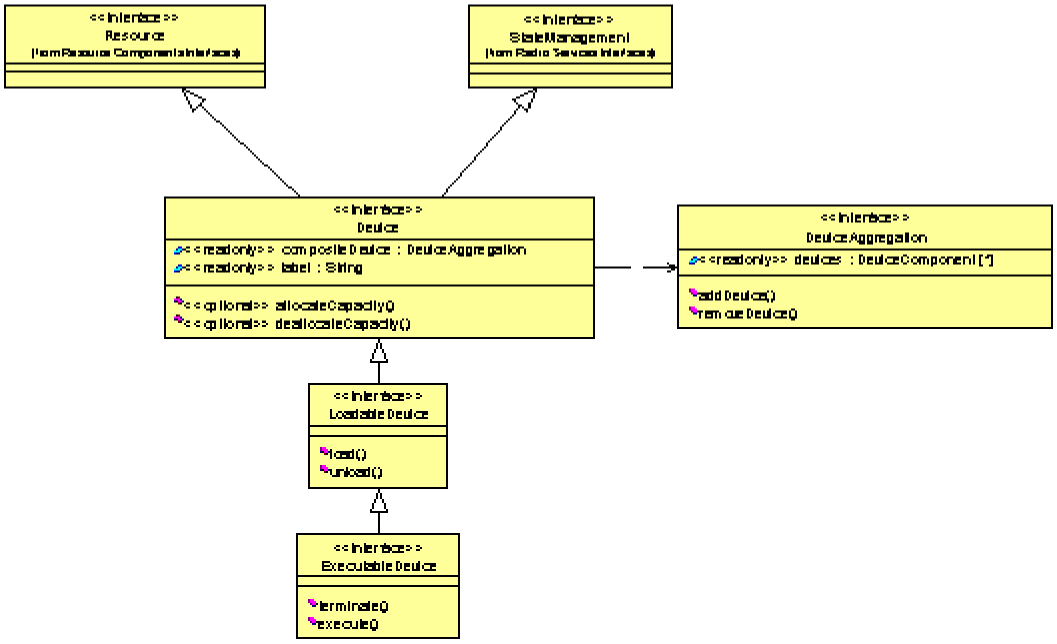

<<Interface>>
Resource
(from Resource Components Interfaces)
<<Interface>>
StateManagement
(from Radio Services Interfaces)
<<Interface>>
Device
<<readonly>> compositeDevice : DeviceAggregation
<<readonly>> label : String
<<optional>> allocateCapacity()
<<optional>> deallocateCapacity()
<<Interface>>
DeviceAggregation
<<readonly>> devices : DeviceComponent [*]
addDevice()
removeDevice()
<<Interface>>
LoadableDevice
load()
unload()
<<Interface>>
ExecutableDevice
terminate()
execute()

Figure 8-14 – Device Component Interfaces Definition

in Figure 8-14, which are: DeviceComposition, Device, LoadableDevice, and ExecutableDevice that are described in the following subsections. These interfaces provide basic management interfaces for SWRadio physical devices.

8.1.6.1.1 Device

Description

The Device, as shown in Figure 8-14, defines an interface that abstracts the underlying hardware. The Device is a specialization of Resource and ManagedServiceComponent interfaces with additional capacity behavior.

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

- State Management 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 a component's descriptor. A device may have multiple allocatable capacities, each having its own unique capacity model.

Note – Issue 8842 - rename DeviceAggregation to DeviceComposition

Attributes

- `<<readonly>>compositeDevice: DeviceCompositionComponent`

  The readonly compositeDevice attribute contains a DeviceCompositionComponent reference. This DeviceCompositionComponent reference refers to the object used by this device (e.g. in the context of the parent of the composite) to maintain the composite parts (includes a list of composite part devices, e.g. children) or is a nil component/object reference if no such composition 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>>softwareProfile: String`

  The profile descriptor (data/command uses and provides ports, configure and query properties, capacity properties, status properties, etc.).

Operations

- `<<optional>> allocateCapacity (in capacities: Properties, return Boolean): {raises = ( InvalidCapacity, InvalidState )}`

  The allocateCapacity operation provides the mechanism to request and allocate capacity from the DeviceComponent. The allocateCapacity operation shall reduce the current capacities of the DeviceComponent based upon the input capacities parameter. The allocateCapacity operation is valid when the adminState attribute is UNLOCKED, operationalState attribute is ENABLED, and usageState attribute is not BUSY.

  The allocateCapacity operation shall set the usageState attribute to BUSY, when the DeviceComponent determines that it is not possible to allocate any further capacity. The allocateCapacity operation shall set the usageState attribute to ACTIVE, when capacity is being used and any capacity is still available for allocation.

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

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

  The allocateCapacity operation shall raise the InvalidState exception, if the Device's adminState is not UNLOCKED or operationalState is DISABLED when invoked.
- `<<optional>> 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 shall adjust the current capacities of the Device based upon the input capacities parameter. The deallocateCapacity operation is valid when the adminState is UNLOCKED or SHUTTING_DOWN and operationalState is ENABLED and usageState is not IDLE.

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

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

The deallocateCapacity operation does not return any value.

The deallocateCapacity operation shall raise 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 shall raise the InvalidState exception, if the Device's adminState is LOCKED or operationalState is DISABLED or usageState is IDLE when invoked.

Note – Issue 8842

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

The following behavior is in addition to the LifeCycle releaseObject operation behavior.

If the compositeDevice attribute is not nil, the releaseObject operation shall call the releaseObject operation on all of the DeviceComponents managed b y the compositeDevice attribute referenced by the DeviceCompositionComponent (i.e., those DeviceComponents that are contained within the DeviceCompositionComponent's compositeParts attribute).

The releaseObject operation shall transition the DeviceComponent's adminState to SHUTTING_DOWN state when the DeviceComponent's adminState is UNLOCKED, and usageState is not IDLE or the compositeDevice attribute is not nil and the referenced DeviceCompositionComponent’s compositeParts attribute is not empty of devices.

The releaseObject operation shall transition the DeviceComponent's adminState to LOCKED when the DeviceComponent's adminSate is SHUTTING_DOWN and usageState attribute is IDLE and the compositeDevice attribute is nil or the compositeDevice attribute referenced DeviceCompositionComponent's compositeParts attribute is empty of devices; all composite parts have been removed.

The releaseObject operation shall transition the DeviceComponent’s adminState to LOCKED when the DeviceComponent’s adminState is UNLOCKED, and the usageState is IDLE and the compositeDevice attribute is nil or the referenced DeviceCompositionComponent’s compositeParts attribute is empty of devices; all composite parts have been removed.

The releaseObject operation shall release the DeviceComponent, when the Device's adminState transitions to LOCKED, ensuring that its usageState is IDLE and any composite parts have been removed.

If the DeviceComponent is a composite part or child of another DeviceComponent then the releaseObject operation shall cause the DeviceComponent to remove itself from the DeviceCompositionComponent (using the DeviceComposition reference provided as an execute property at the construction of the DeviceComponent).

If the DeviceComponent is registered with a DeviceManager, then the releaseObject operation shall unregister the DeviceComponent from its DeviceManager.

Types and Exceptions

- `<<exception>>InvalidCapacity (msg: String, capacities: Properties)`
  The InvalidCapacity exception indicates 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 operation being attempted due to its state(s) (e.g., admin, operational or usage).

8.1.6.1.2 ExecutableDevice

Description

The ExecutableDevice interface, as shown in Figure 8-14, extends the LoadableDevice by adding execute and terminate behavior.

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 or thread. A process or thread can be used to execute a runtime environment, function, or file.

- `terminate(in processId: ProcessID_Type): {raises = ( InvalidProcess, InvalidState )}`
  The terminate operation provides the mechanism for terminating the execution of a process or thread on a specific device that was started up with the execute operation. The terminate operation shall terminate the execution of the process or thread designated by the processId input parameter on the Device.

  When the last thread is terminated from a process, the terminate operation should terminate the process. The terminate operation shall raise the InvalidState exception if the Device's adminState is LOCKED or operationalState is DISABLED upon invocation.

The terminate operation shall raise the InvalidProcess exception when the processId is not executing on 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, is not executing 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 attribute 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 attribute is a list of invalid options specified in the execute operation.

`Note – Issue 7895, fix types, Issue 8949 changed type to Long`

- `ProcessID_Type: Long`

  This type, a specialization of Long, 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

---

Note – `Issue 7895, fix types`

- parameters. The value for a priority option parameter shall be an unsigned long.
- `STACK_SIZE_ID = "STACK_SIZE"`

  The STACK_SIZE_ID is the identifier for the ExecutableDevice's execute op

---

Note – `Issue 7895, fix types`

- tions parameter. The value for a stack size option parameter shall be an unsigned long.
- `CREATE_THREAD_REQUEST = "CREATE_THREAD"`
  The CREATE_THREAD_REQUEST is the identifier for the ExecutableDevice's execute options parameter. The value for create thread request option

---

Note – `Issue 7895, fix types`

- shall be an unsigned long that indicates the thread ID to be collocated with. A zero valid indicates no thread ID collocation is indicated. A non-zero indicates the thread ID to be collocated with.
- `RUNTIME_REQUEST = "RUNTIME_REQUEST"`
  The RUNTIME_REQUEST is the identifier for the ExecutableDevice's execute options parameter. The value for runtime request option shall be a string of the runtime name to be executed.
- `RUNTIME_OPTIONS = "RUNTIME_OPTIONS"`
  The RUNTIME_OPTIONS is the identifier for the ExecutableDevice's execute options parameter. The value for runtime options option shall be a BaseTypes::Properties. Each ID/value pair in the Properties represents a runtime option. The id indicates the option name and the value is the option value.

Semantics

The execute operation shall execute the function or file identified by the input name parameter using the input parameters and options parameters when no runtime or thread options are specified. Whether the input name parameter is a function or a file name is implementation-specific.

The execute operation shall pass the input parameters (ID/value string pairs) as arguments (array of strings) to the operating system “execute/thread” function, where argument (0) is the function name, argument (1) maps to input parameters (0) id and argument (2) maps to input parameters (0) value and so forth.

The execute operation shall create a thread when the options parameter is CREATE_THREAD_REQUEST is specified. The execute operation shall create thread in the same process as the thread ID identified by the CREATE_THREAD_REQUEST value when the CREATE_THREAD_REQUEST value is not zero.

The execute operation shall create a runtime process/thread when the RUNTIME_REQUEST options is specified in the options parameter. The execute operation create the runtime process/thread using the RUNTIME_REQUEST value. The execute operation shall pass the RUNTIME_OPTIONS as specified in the input options parameter, the input name, and arguments in the form that is compliant with runtime parameters syntax.

Note – Issue 8948, changed wording on processID

The execute operation shall use STACK_SIZE_ID and PRIORITY_ID options, when specified, to set the process/thread stack size and priority for the target executable. The execute operation returns a unique processID for the process or thread that it created.

The execute operation shall raise the InvalidState exception, if the Device's adminState is not UNLOCKED or operationalState is DISABLED when invoked. The execute operation shall raise the InvalidFunction exception when the function indicated by the input name parameter does not exist for the Device (e.g., not loaded on device). The execute operation shall raise the InvalidFileName exception when the file name indicated by the input name parameter does not exist for the Device (e.g., not loaded on device). The execute operation shall raise the InvalidParameters exception when the input parameters parameter item ID or value are not string types. The execute operation shall raise the InvalidOptions exception when the input options parameter does not comply with STACK_SIZE_ID, PRIORITY_ID, THREAD_CREATE_REQUEST, RUNTIME_CREATE_REQUEST, and RUNTIME_OPTIONS (described in Types section below). The execute operation shall raise the ExecuteFail exception when the operating system "execute" function for the device is not successful.

Note – Issue 8842 rename DeviceAggregation

8.1.6.1.3 DeviceComposition

Description

The DeviceComposition is an interface that provides the capability to construct a composite device definition.

Attributes

- `<<readonly>>compositeParts: DeviceComponent [*]`
  The readonly compositeParts attribute shall contain a list of DeviceComponents that have been added to DeviceComposition or a zero length sequence if the composition is empty (no devices have been added or all have been removed).

Operations

Note – Issue 8842 rename DeviceAggregation

- `addDevice (in associatedDevice: DeviceComponent): {raises = ( InvalidObjectReference )}`
  The addDevice operation provides the mechanism to associate a Device with a DeviceComposition. The addDevice operation shall add the input associatedDevice parameter to the compositeParts attribute when the associatedDevice does not already exist in the compositeParts attribute. The associatedDevice is ignored when duplicated. This operation does not return any value. The addDevice operation shall raise the InvalidObjectReference when the input associatedDevice is a nil DeviceComponent reference.

Note – Issue 8842 rename DeviceAggregation

- `removeDevice (in associatedDevice: DeviceComponent): {raises = ( InvalidObjectReference )}`
  The removeDevice operation provides the mechanism to disassociate a DeviceComponent from a DeviceComposition. The removeDevice operation shall remove the input associatedDevice parameter from the compositeParts attribute. This operation does not return any value. The removeDevice operation shall raise the InvalidObjectReference when the input associatedDevice is a nil DeviceComponent reference or does not exist in the compositeParts attribute.

Semantics

The DeviceComposition interface provides composite behavior that can be used to add and remove DeviceComponents from a composite relationship. Composite part DeviceComponents use this interface to introduce or remove an association between themselves and a DeviceComponent that manages the compositoin. When the aggregating DeviceComponent that manages the DeviceComposition changes state or is being released by the releaseObject operation, its associated DeviceComponents are affected. (all DeviceComponents added to the DeviceComposition)

8.1.6.1.4 LoadableDevice

Description

The LoadableDevice interface, as shown in Figure 8-14, extends the Device interface by adding software loading and unloading behavior.

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

  Multiple loads of the same input fileName do not result in an exception or a duplicate load, however the load operation should account for this attempt so that the unload operation behavior can be performed. The load operation shall raise the InvalidState exception if the Device's adminState is not UNLOCKED or operationalState is DISABLED upon invocation.

  The load operation shall raise the InvalidLoadKind exception if the input loadKind parameter is not supported.

  The load operation shall raise the InvalidFileName exception if the file designated by the input filename parameter cannot be found.

  The load operation shall raise the LoadFail exception if 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 shall unload the application software on the device based on the input fileName parameter. The unload operation shall perform the unload when the number of unload requests matches the number of load requests for the input filename.

  The unload operation shall raise the InvalidState exception if the adminState attribute is LOCKED or its operationalState attribute is DISABLED upon invocation.

  The unload operation shall raise the InvalidFileName exception if the file designated by the input filename parameter cannot be found.

Types and Exceptions

- `<<enumeration>>LoadType ( KERNEL_MODULE,  DRIVER, DLL, EXECUTABLE)`

  The LoadType defines the type of load to be performed which are:
  -KERNEL_MODULE,
  -DRIVER,

-DLL,
-EXECUTABLE, SHARED_LIBRARY

- `<<exception>>InvalidLoadKind`

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

- `<<exception>>LoadFail ( errorNumber : ErrorNumberType, msg : String)`

  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.

Note – Issue 7742 - added Device Component Stereoptypes subsection, changed names of loadable and executable device stereotypes by adding component at the end of the name. Updated stereotype table with name changes. Removed type, operations, attributes from definitions. Changed Assoications noheader to M1 Associations.

8.1.6.2 Device Component Stereotypes

Note – Issue 7586, added constraints table in table

The component stereotypes are depicted in the Table 8-5 below, which are extensions of the UML Component. The following subsections describe the details of logical device interfaces and component stereotypes.

Table 8-5 – Device Components Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
DeviceComponent	Component	ResourceComponent, ManagedServiceComponent		See constraints in section below	Represents a logical device that abstracts the underlying hardware.
DeviceDriver	Component	N/A			Represents a device driver that interfaces with the hardware.
ExecutableDeviceComponent	Component	LoadableDeviceComponent		See constraints in section below	Manages execution and termination of OS processes on a device.
DeviceComposition Component	Component	N/A		See constraints in section below	Provides the capability to construct composite devices.
LoadableDeviceComponent	Component	DeviceComponent		See constraints in section below	Represents a loadable device that manages loading behavior on a device.

8.1.6 Device Components

8.1.6.2.1 DeviceComponent

Description

Note – Issue 7742, updated figure to be M1 type,

The DeviceComponent, as shown in Figure 8-15, defines a component that abstracts the underlying hardware. The DeviceComponent is a type of ResourceComponent and ManagedServiceComponent with additional capacity behavior.

A DeviceComponent (e.g., 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 Attributes - This information describes the administrative, usage, and operational states of the device.
- Capacity Operations - In order to use a DeviceComponent, certain capacities (e.g., memory, performance, etc.) must be obtained from the DeviceComponent. The capacity properties will vary among DeviceComponents and are described in a component's descriptor. A DeviceComponent may have multiple allocatable capacities, each having its own unique capacity model.

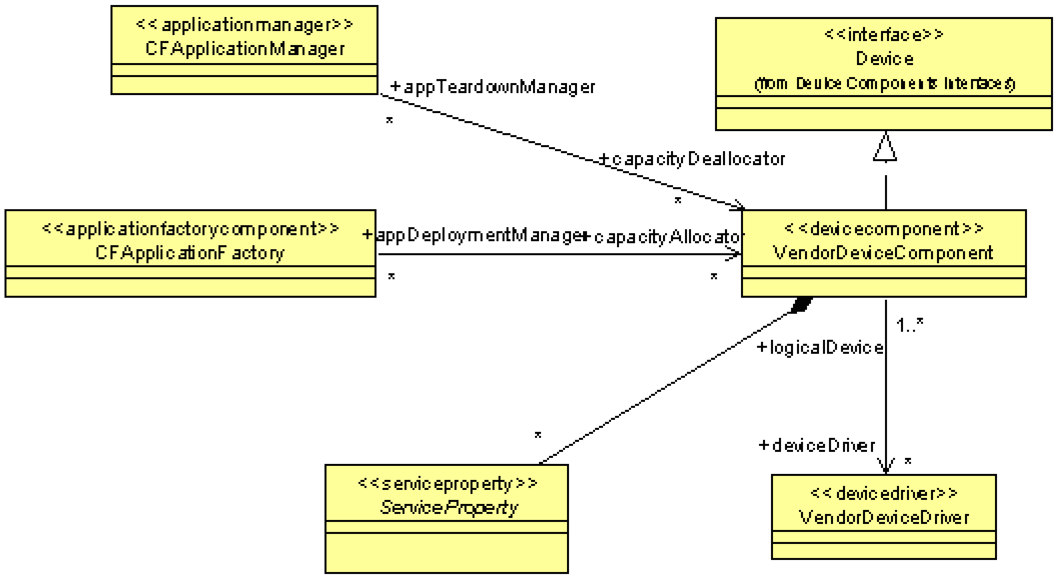


Figure 8-15 – DeviceComponent M1 Illustration

M1 Associations

- `deviceDriver: DeviceDriver [*]`

  The device drivers used by the logical device for communicating with the radio hardware.

Constraints

The DeviceComponent shall support the ServiceProperty capabilities and capacities properties as stated in the Device's component descriptor as specified by the softwareProfile attribute.

The ServiceProperty properties that are managed capacities shall be queryable from the DeviceComponent's query operation and managed by the allocateCapacity and deallocateCapacity operations.

The setAdminState operation shall be become disabled when the releaseObject operation is invoked.

The DeviceComponent shall provide the allocateCapacity and deallocateCapacity operations when the DeviceComponent contains ServiceProperty(s) whose locallyManaged attribute value is True.

The DeviceComponent shall realize the Device interface.

Semantics

The managed capacity properties are managed by the logical device through its capacity operations and reflected by its state attributes.

The DeviceComponent contains CapacityModel(s) when the DeviceComponent contains ServiceProperty(s) whose locallyManaged attribute value is True.

The BasicDeploymentRequirement corresponds to a PropertyValue item in the allocateCapacity or deallocateCapacity capacities parameter. The PropertyValue item's id matches the BasicDeploymentRequirement's identification and the PropertyValue item's value matches the BasicDeploymentRequirement's value attribute but is converted into a format that agrees with the ServiceProperty's type attribute.

The DeploymentRequirementQualifer (Infrastructure::SWRadio Deployment::SWRadio Artifacts) corresponds to a PropertyValue item in the allocateCapacity or deallocateCapacitycapacities parameter. The PropertyValue item's id matches the DeploymentRequirementQuailifer 's identification and the PropertyValue item's value contains a Properties type, where each DeploymentRequirementQuailifer's qualifier (id, value) corresponds to a Properties element in the list. The Properties element list size is based on the number of DeploymentRequirementQuailifer 's qualifier items. DeploymentRequirementQualifier's qualifier value attribute is converted into a format that agrees with the CharacteristicQuery value attribute.

8.1.6.2.2 DeviceDriver

Description

The DeviceDriver, as shown in Figure 8-16, represents a component that interfaces with the SWRadio communication equipment.

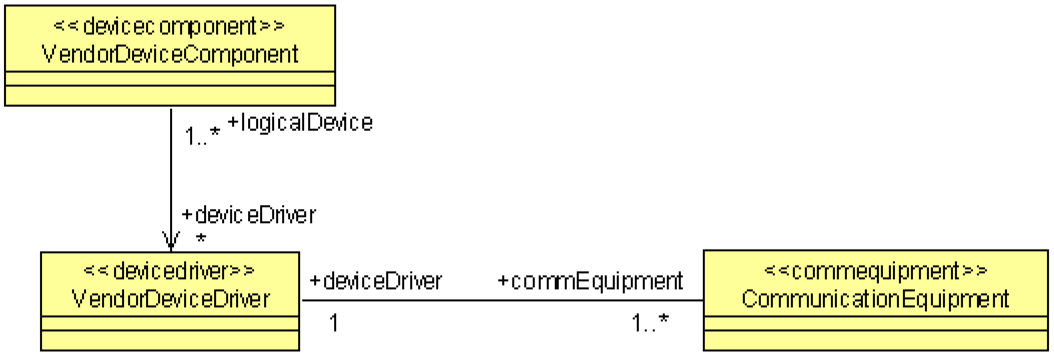


Figure 8-16 – DeviceDriver M1 Illustration

M1 Associations

- `CommEquipment: CommEquipment [1..*]`
  The device element the device driver is managing and controlling.

8.1.6.2.3 ExecutableDeviceComponent

Description

The ExecutableDeviceComponent, as shown in Figure 8-17, extends the LoadableDeviceComponent by adding execute and terminate process/thread behavior.

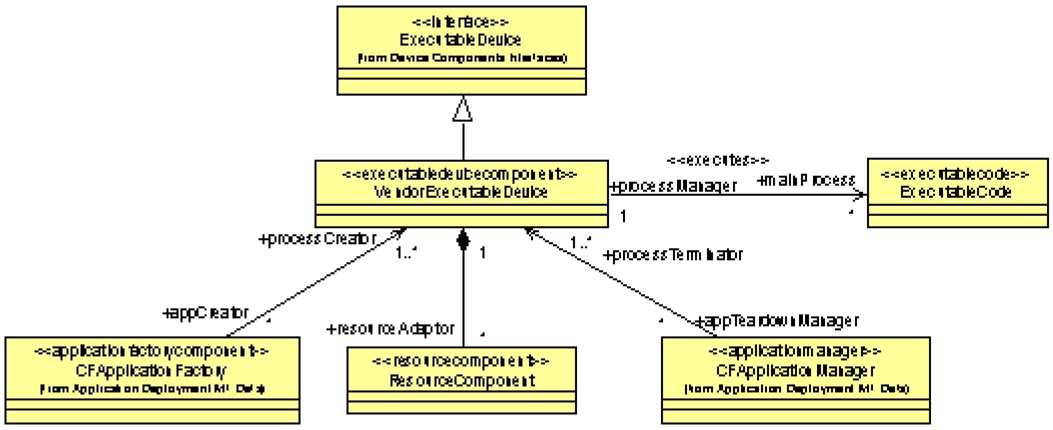


Figure 8-17 – ExecutableDeviceComponent M1 Illustration

M1 Associations

Note – Issue 7697 - Wrong Name

- `mainProcess: ExecutableCode [*]`
  Zero to many MainProcesses may be executed and terminated on a device.
- `resourceAdaptor: ResourceComponent [*]`
  A ResourceComponent can take on the role of an adaptor that communicates with components that are not implemented with a distributive component middleware environment.

Constraints

The ExecutableDeviceComponent shall realize the ExecutableDevice interface.

Note – Issue 8842

8.1.6.2.4 DeviceCompositionComponent

Description

The DeviceCompositionComponent is a component that provides the capability to construct a composite device definition.

Constraints

The DeviceCompositionComponent shall realize the DeviceComposition interface.

Semantics

The DeviceCompositionComponent component provides composite behavior that can be used to add and remove DeviceComponents from a composite relationship. Composite part DeviceComponents are provided with and use a reference to a DeviceCompositionComponent (as an instance of a DeviceComposition interface realization) to introduce or remove an association between themselves and a DeviceComponent that manages the composition. When the DeviceComponent that manages the DeviceComposition changes state or is being released by the releaseObject operation, its associated DeviceComponents are affected (all DeviceComponents added to the DeviceComposition).

8.1.6.2.5 LoadableDeviceComponent

Description

The LoadableDeviceComponent, as shown in Figure 8-18, extends the DeviceComponent component by adding software loading and unloading behavior.

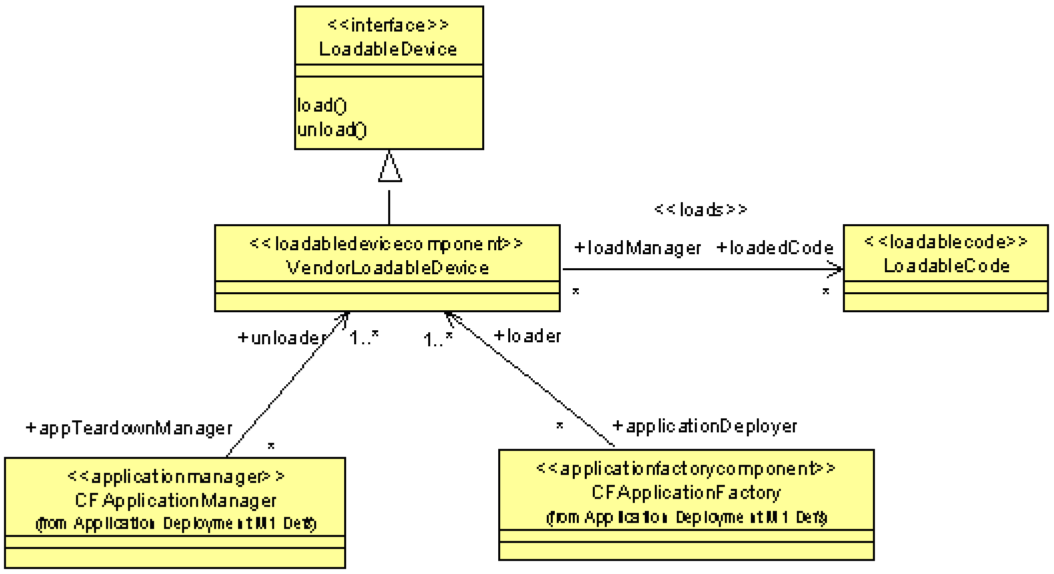


Figure 8-18 – LoadableDeviceComponent M1 Illustration

Attributes

- `<<characteristicselectionproperty>> loadKind(name = "Load Kind", type = string)`
  The <<characteristicselectionproperty>> (Application and Device Components::Properties) LoadKind defines the type of LoadKindTypes supported for the LoadableDevice load operation. Valid values are: DLL, DRIVER, EXECUTABLE, KERNEL MODULE, and SHARED LIBRARY. The valid values are device specifc.
- `<<characteristicsetproperty>> os(name = "OS", characteristic [1] = (qualifier [1] = ( name ="Name", value = ""), qualifier [2] = ( name ="Version", value = "")))`
  The <<characteristicsetproperty>> (Application and Device Components::Properties) os defines the type of Operating Systems supported for the LoadableDevice load operation. The OS name values are case sensitive. The value attributes are device specific.
- `·<<characteristicsetproperty>>runtime( name = "Runtime", characteristic [1] = ( qualifier [1] = ( name ="Name", value=""),qualifier[2]=(name="Version", value="")))`
  The <<characteristicsetproperty>> (Application and Device Compo-

nents::Properties) runtime defines the runtime environements supported for the LoadableDevice load operation. The value attributes are device specific.

- `<<characteristicsetproperty>>library( name = "Library", characteristic [1] = ( qualifier [1] = ( name ="Name", value = ""), qualifier [2] = (name ="Version", value = "")))`
  The <<characteristicsetproperty>> (Application and Device Components::Properties) library defines the libraries supported for the LoadableDevice load operation. The value attributes are device specific.

M1 Associations

- `ObjectCode: ObjectCode [*]`
  Zero to many ObjectCodes may be loaded on a device.

Constraints

The LoadableDeviceComponent shall support the load type capabilities identified in the Device's component descriptor as specified in the softwareProfile attribute.

When a LoadKind characteristic property is not defined for the LoadableDeviceComponent, the load operation shall support all load kinds.

The LoadableDeviceComponent shall realize the LoadableDevice interface.

Semantics

The loaded software may be subsequently executed on the LoadableDeviceComponent, if the component is also an ExecutableDeviceComponent.

8.1.7 Application Components

The Application Components sections define the set of components used to define applications and waveforms. The Application Components stereotypes are depicted in Table 8-6 below, which are extensions of the UML Component (UML2.0::Components::BasicComponents). The following subsections describe the details of these elements.

Note – Issue 7742, updated all figures to be M1 types and changed figure title, added constraints for the stereotypes as necessary that were depicted in the figures, Changed Associations noheader to M1 associations, Changed some of the cardinality on the associations., Issue 7586 added forward references for constraints

---

8.1.7 Application Components

Table 8-6 – Applications Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
Application	Component	N/A			Represents an assembly of ApplicationResources and SWRadioComponents.
ApplicationResourceComponent	Component	ResourceComponent		See constraints in section below	Provides a common definition for an application's ResourceComponent.
LinkLayerControlResource	Component	WaveformLayerResource		See constraints in section below	Represents a standard Link Layer Control component of the OSI layer model.
MediumAccessControlResource	Component	WaveformLayerResource		See constraints in section below	Represents a standard Medium Access Control component of the OSI layer model.
NeworkLayerResource	Component	WaveformLayerResource		See constraints in section below	Represents a standard Network Layer component of the OSI layer model.
PhysicalLayerResource	Component	WaveformLayerResource		See constraints in section below	Represents a standard Physical Layer component of the OSI layer model.
WaveformApplication	Component	Application			Represents a waveform application.
WaveformLayerResource	Component	ApplicationResourceComponent			Represents a standard component of the OSI layer model.

8.1.7.1 Application

Description

The Application, as shown in Figure 8-19, provides a component assembly definition for a set of ApplicationResourceComponent(s) and SWRadioComponent(s).

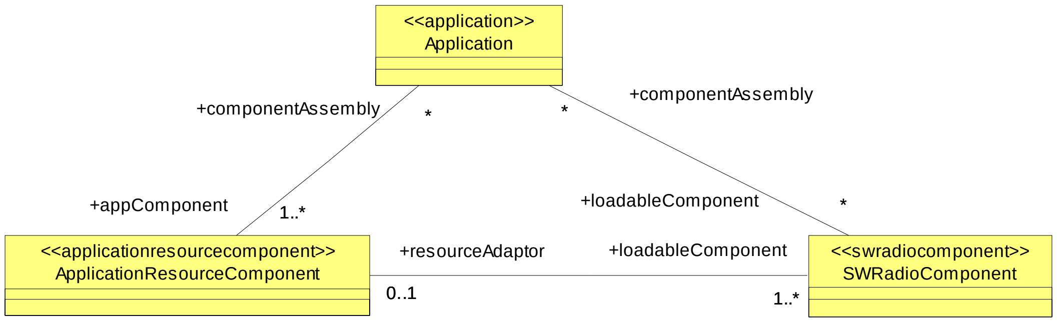


Figure 8-19 – Application M1 Illustration

M1 Associations

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

8.1.7.2 ApplicationResourceComponent

Description

The ApplicationResourceComponent, as shown in Table 8-6, provides a common API for control and configuration of an application resource component. Figure 8-20 depicts the associations for an ApplicaitonResourceCom-

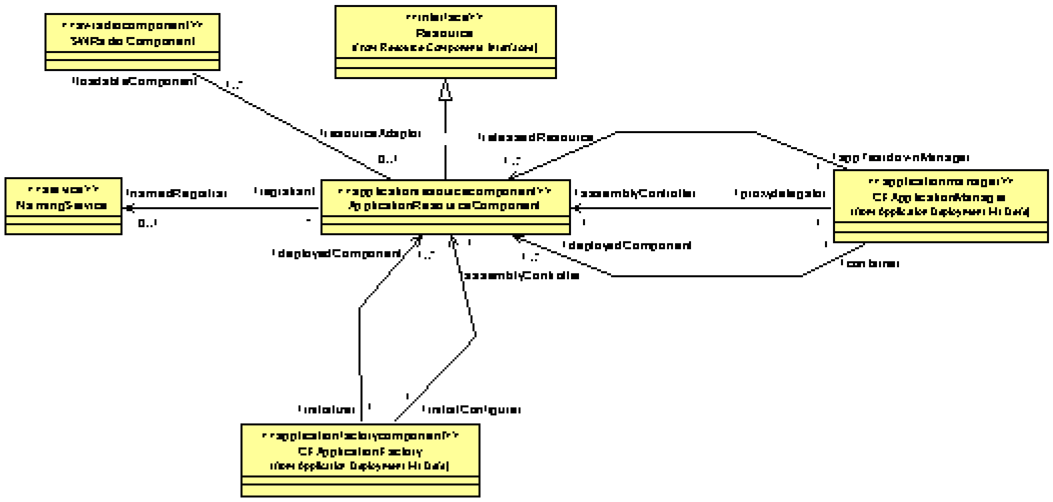


Figure 8-20 – ApplicationResourceComponent M1 Illustration

ponent.

M1 Associations

- `namedRegistrar: NamingService [0..1]`

  The optional NamingService that contains a named ApplicationComponent reference.

- `loadableComponent: SWRadioComponent [*]`

  A loadableComponent may be associated with a ApplicationResourceComponent when the component acts a resourceAdaptor. In this case, the loadableComponent cannot be communicated with unless through the resourceAdaptor.

Constraints

Note – Issue 7586

An ApplicationResourceComponent shall be registered with a NamingService when a ResourceFactoryComponent does not create the ApplicationResourceComponent.

Semantics

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

8.1.7.3 WaveFormLayerResource

Description

The WaveformLayerResource, as shown in Table 8-6, specializes the ApplicationResource stereotype. The WaveformLayerResource stereotype provides a mechanism to realize a waveform layer component, should a non-OSI layer is required. For standard OSI layers, the stereotypes that specialize the WaveformLayerResource should be used.

A WaveformLayerResource, can perform two types of communication as shown in Figure 8-21. 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.

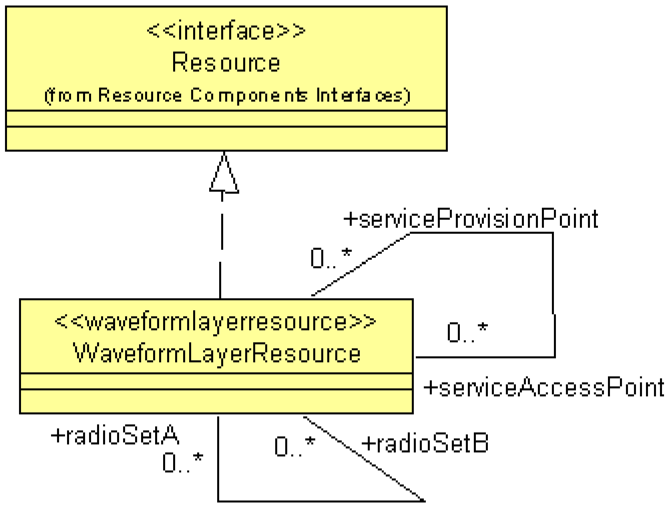


Figure 8-21 – WaveformLayerResource M1 Illustration

Note – Issue 7698 - Cardinality Problem

M1 Associations

Note – Issue 7698 - Cardinality Problem

- `radioSetA, radioSetB: WaveformLayerResource [0..*]`

  This association shows the data transfer between two WaveformLayerResources that reside in different radio sets. This is an example of horizontal communication.

Note – Issue 7698 - Cardinality Problem

- `serviceProvisionPoint, serviceAccessPoint: WaveformLayerResource [0..*]`

  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. Service Access Point is defined as a interface or a port on the client waveform component through which the client uses a service. Service Provision Point is defined as a port or interface on a server providing a service.

8.1.7.4 PhysicalLayerResource

Description

The PhysicalLayerResource, as shown in Table 8-6, specializes the WaveformLayerResource stereotype. The PhysicalLayerResource stereotype provides a mechanism to realize a standard Physical Layer component of the OSI layer model. A PhysicalLayerResource is associated with a medium access control or link layer control resource as shown in Figure 8-22. The standard facilities of a Physical Layer API are defined in the PIM facilities.

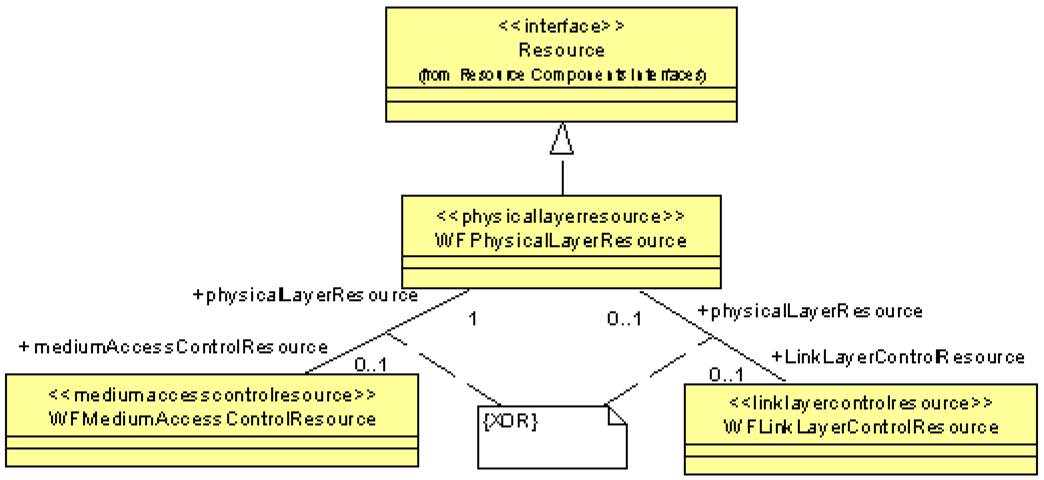

<<interface>>
Resource
(from Resource Components Interfaces)
<<physicallayerresource>>
WFPhysicalLayerResource
+physicalLayerResource
1
0..1
+physicalLayerResource
+mediumAccessControlResource
0..1
+LinkLayerControlResource
0..1
<<mediumaccesscontrolresource>>
WFMediumAccessControlResource
{XOR}
<<linklayercontrolresource>>
WFLinkLayerControlResource

Figure 8-22 – PhysicalLayerResource M1 Illustration

Note – Issue 7699 - Redundant Association (also removed the association description from the associations section below)

M1 Associations

- `MediumAccessControlResource : WFMediumAccessControlResource [0..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 : WFLinkLayerControllerResource [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)

Constraints

`A PhysicalLayerResource shall be associated with either a MediumAccessControlResource or a LinkLayerControlResource.`

8.1.7.5 MediumAccessControlResource

Description

The MediumAccessControlResource, as shown in Table 8-6, specializes the WaveformLayerResource stereotype. The MediumAccessControlResource stereotype provides a mechanism to realize a standard Medium Access Control component of the OSI layer model. A MediumAccessControlResource is associated with a physical layer resource and link layer control resource as shown in Figure 8-23. The standard facilities of a Medium Access Control API are defined in the PIM facilities.

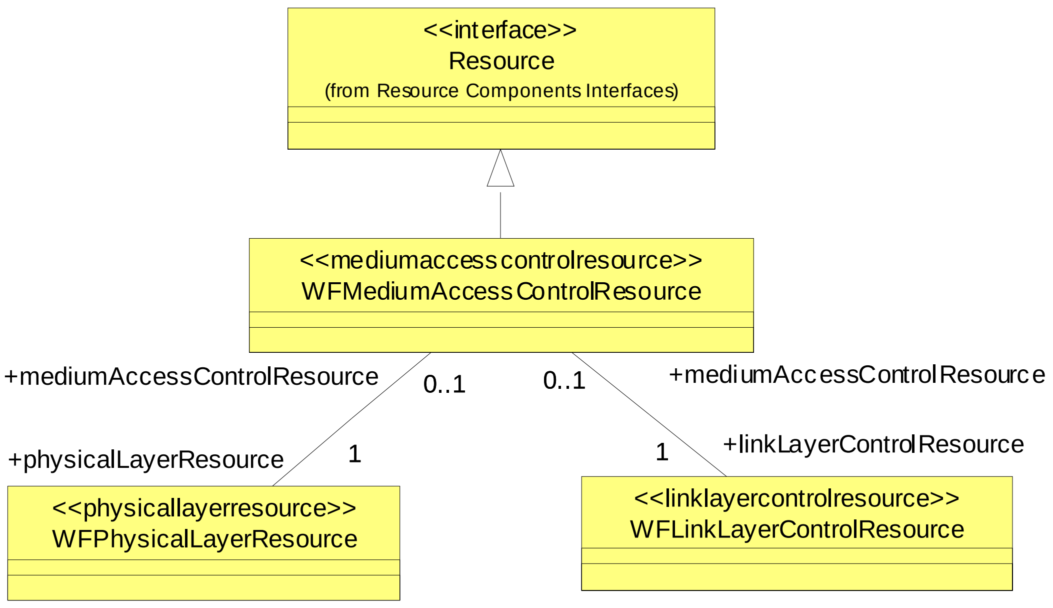

<<interface>>
Resource
(from Resource Components Interfaces)
<<mediumaccesscontrolresource>>
WFMediumAccessControlResource
+mediumAccessControlResource
0..1
0..1
+mediumAccessControlResource
+physicalLayerResource
1
1
+linkLayerControlResource
<<physicallayerresource>>
WFPhysicalLayerResource
<<linklayercontrolresource>>
WFLinkLayerControlResource

Figure 8-23 – MediumAccessControlResource M1 Illustration

Note – Issue 7700 - Redundant Association (also removed the association description from the associations section below)

M1 Associations

- `physicalLayerResource : WFPhysicalLayerResource [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 : WFLinkLayerControllerResource [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).

Constraints

A MediumAccessControlResource shall be associated with a PhysicalLayerResource and a LinkLayerControlResource.

8.1.7.6 LinkLayerControlResource

Description

The LinkLayerControlResource, as shown in Table 8-6, specializes the WaveformLayerResource stereotype. The LinkLayerControlResource stereotype provides a mechanism to realize a standard Link Layer Control component of the OSI layer model. A LinkLayerControlResource is associated with a physical layer resource or a medium access control resource as shown in Figure 8-24. The standard facilities of a Link Layer Control API are defined in the PIM facilities.

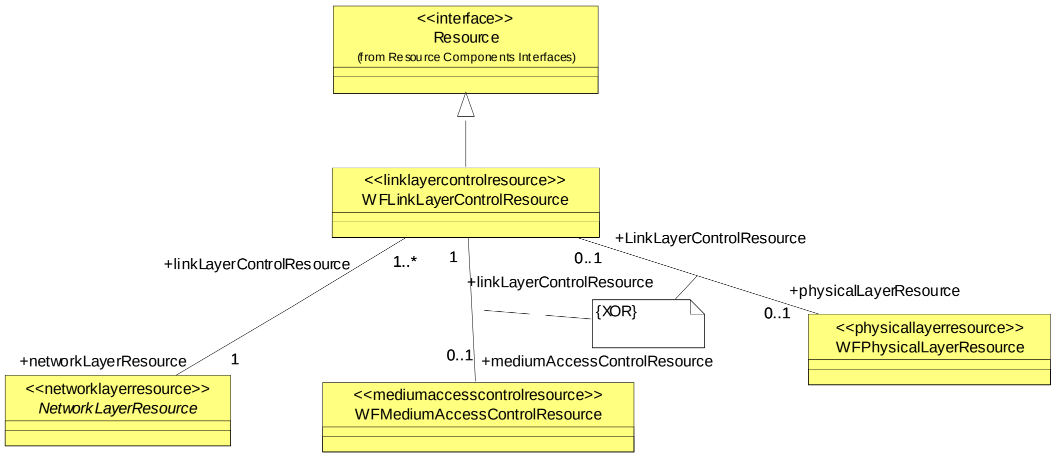

<<interface>>
Resource
(from Resource Components Interfaces)
<<linklayercontrolresource>>
WFLinkLayerControlResource
+LinkLayerControlResource
+linkLayerControlResource
1..*
1
0..1
+linkLayerControlResource
+physicalLayerResource
{XOR}
0..1
<<physicallayerresource>>
WFPhysicalLayerResource
+networkLayerResource
1
0..1
+mediumAccessControlResource
<<networklayerresource>>
NetworkLayerResource
<<mediumaccesscontrolresource>>
WFMediumAccessControlResource

Figure 8-24 – LinkLayerControlResource M1 Illustration

Note – Issue 7701 - Redundant Association (also removed the association description from the associations section below)

M1 Associations

- `physicalLayerResource : WFPhysicalLayerResource [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 Commnucation)

- `mediumAccessControlResource : WFMediumAccessControlResource [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)
- networkLayerResource: NetworkLayerResource [1]
  NetworkLayerResource communicates with LinkLayerControlResources.

Constraints

The LinkLayerControlResource shall be associated with a NetworkLayerResource.

Note – 7586, added “either”

The LinkLayerControlResource shall be either associated with a PhysicalLayerResource or a MediumAccess-ControlResource.

8.1.7.7 NetworkLayerResource

Description

The NetworkLayerResource, as shown in Figure 8-6, specializes the WaveformLayerResource stereotype. The NetworkLayerResource stereotype provides a mechanism to realize a standard Network Layer component of the OSI layer model. A NetworkLayerResource is associated with one or more link layer control resource and can also play the role of a gateway/translator as shown in Figure 8-25. The facilities of a network layer component is out of the scope of this specification and is not included in the PIM.

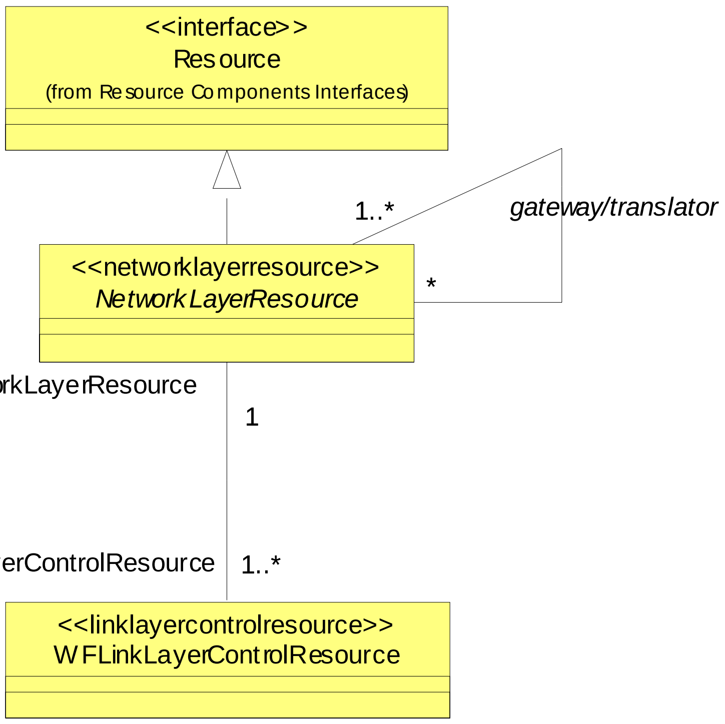

<<interface>>
Resource
(from Resource Components Interfaces)
1..*
gateway/translator
<<networklayerresource>>
NetworkLayerResource
*
+networkLayerResource
1
+linkLayerControlResource
1..*
<<linklayercontrolresource>>
WFLinkLayerControlResource

Figure 8-25 – NetworkLayerResource M1 Illustration

Note – Issue 7702 - Redundant Association (also removed the association description from the associations section below)

Associations

- `linkLayerControlResource : WFLinkLayerControlResource [1..*]`
  A NetworkLayerResource may communicate with one or more Link Layer components within the waveform.
- `Gateway/translator: NetworkLayerResource */[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 re-

sources to provide gateway/waveform translator functionality. (Horizontal Communication).

Constraints

A NeworkLayerResource shall be associated with one or more LinkLayerControlResources.

8.1.7 Application Components

8.2 Communication Equipment

Note – Issue 7785 - Waveform vs Application changes throughout Chapter 8

The Communication Equipment package contains device stereotypes that describe devices realized by a specific Communication Channel. The selected devices represent basic functions associated with software radio equipment. Additional stereotypes are defined for modeling the relationships between radio devices. However, this specification neither dictates, nor restricts, the arrangement of radio devices. Actual connection definitions between devices are left out to the implementer.

The purpose of the Communication Equipment package is twofold. It defines a language to describe a specific hardware platform upon which applications execute. This description can be 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 whether or not 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 application development and porting process since the actual hardware platform is not required for determining if a specific platform can support a specific waveform. The stereotypes providing this language are summarized in Table 8-7.

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

Note – Issue 7985 - Table updated to remove QueryProperty and ConfigureProperty properties,

Table 8-7 – Communication Equipment Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
Amplifier	Device	IODevice	dutyCycle, gain, maxGain, minGain	See constraints in section below	Increases the energy of signals passing through it.
AnalogInputPort	Port	N/A	inputImp, inputLevel, maxInputLevel, insertionLoss, inputVSWR	See constraints in section below	Receives an analog signal.
AnalogOutputPort	Port	N/A	maxOutputLevel, outputImp, outputVSWR	See constraints in section below	Transmits an analog signal.
Antenna	Device	IODevice	calibration, radiationPattern, polarization, type, maxRadiationPattern, minRadiationPattern, polarizationCapability	See constraints in section below	Converts an electrical signal into an electromagnetic wave and vice versa for carrying data over an air interface.
AudioDevice	Device	IODevice	N/A	There has to be at least one AnalogInput Port.	Converts electrical signals into sounds waves.
CommEquipment	Device	N/A	equipmentInformation, equipmentSize, equipmentWeight, powerConsumption, maxOperatingTemperature, minOperatingTemperature, radiationCapability, meanTimeBetweenFailures, lastMaintenanceCheck, maintenancePeriod, temperatureStatus	See constraints in section below	Represents a radio communication device.
CommEquipment CommunicationPa th	Commu nication Path	N/A	N/A	See constraints in section below	Represents an association between communication equipments through which signals and messages are exchanged.

Table 8-7 – Communication Equipment Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
CommEquipment Connector	Connect or	N/A	N/A	See constraints in section below	Represents a link that enables communication between two or more instances of communication equipment ports.
CryptoDevice	Device	CommEquipment	algorithm, keyLength	See constraints in section below	Performs encryption and decryption on a set of data.
DigitalConverter	Device	IODevice	sampleRate, maxSampleRate, minSampleRate, sampleSize, phaseNoise	See constraints in section below	Converts an analog signal into a digital signal and vice versa.
DigitalPort	Port	N/A	quantizationNoise, dataFlowDirection, streaming, maxThroughput	See constraints in section below	Receives or transmits a digital signal.
Filter	Device	IODevice	N/A	See constraints in section below	Alters the frequency spectrum of signals passing through it.
FrequencyConvert er	Device	IODevice	currentInputFrequency, currentOutputFrequency, maxInputFrequency, minInputFrequency, maxOutputFrequency, minOutputFrequency, loInputLeakagePower, loOutputLeakagePower, outputToInputLeakage, phaseNoise, loStability	See constraints in section below	Performs frequency translation in such a manner that the output frequencies are higher/lower in the spectrum than the input frequencies.
HoppingFrequenc yConverter	Device	FrequencyConve rter	nextInputFrequency, nextOutputFrequency	N/A	Performs hopping frequency conversion.
IODevice	Device	CommEquipment	maxPowerHandling, minPowerHandling, noiseFigure, maxOperatingVSWR, freqResponse, tunedFrequency, maxFrequencyResponse, minFreqencyResponse, maxFrequency, minFrequency, amplitudePhaseResponse	N/A	Operates on a signal.

8.2 Communication Equipment

Table 8-7 – Communication Equipment Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
Microphone	Device	IODevice	N/A	There has to be at least one AnalogOutputPort.	Converts sound waves into an electrical signal.
PowerSupply	Device	CommEquipment	type, efficiency	N/A	Provides electrical power to other devices.
Processor	Device	CommEquipment	processorArchitecture, maxOperatingFrequency, nonVolatileMemoryCapacity, volatileMemoryCapacity	See constraints in section below	Processes digital or analog data.
ProgrammableLogicDevice	Device	Processor	logicUnitCapacity, reconfigurability, timeForReconfiguration	N/A	Uses hardware logic to process data.
RadiatingElement	Device	IODevice	active, radiationPattern, polarization, type, positionInAntennaArray	See constraints in section below	Represents the part of an antenna that actually emits and receives electromagnetic waves.
SerialIODevice	Device	IODevice	N/A	N/A	Transmits and receives digital signals serially.
SoftwareProcessor	Device	Processor	operatingEnvironment	N/A	Uses software instructions to process digital data.
Switch	Device	IODevice	inputOutputIsolation, switchSetting	See constraints in section below	Connects two I/O ports to each other given a specific configuration.

Issue 7582 removed CharacteristicProperty from table

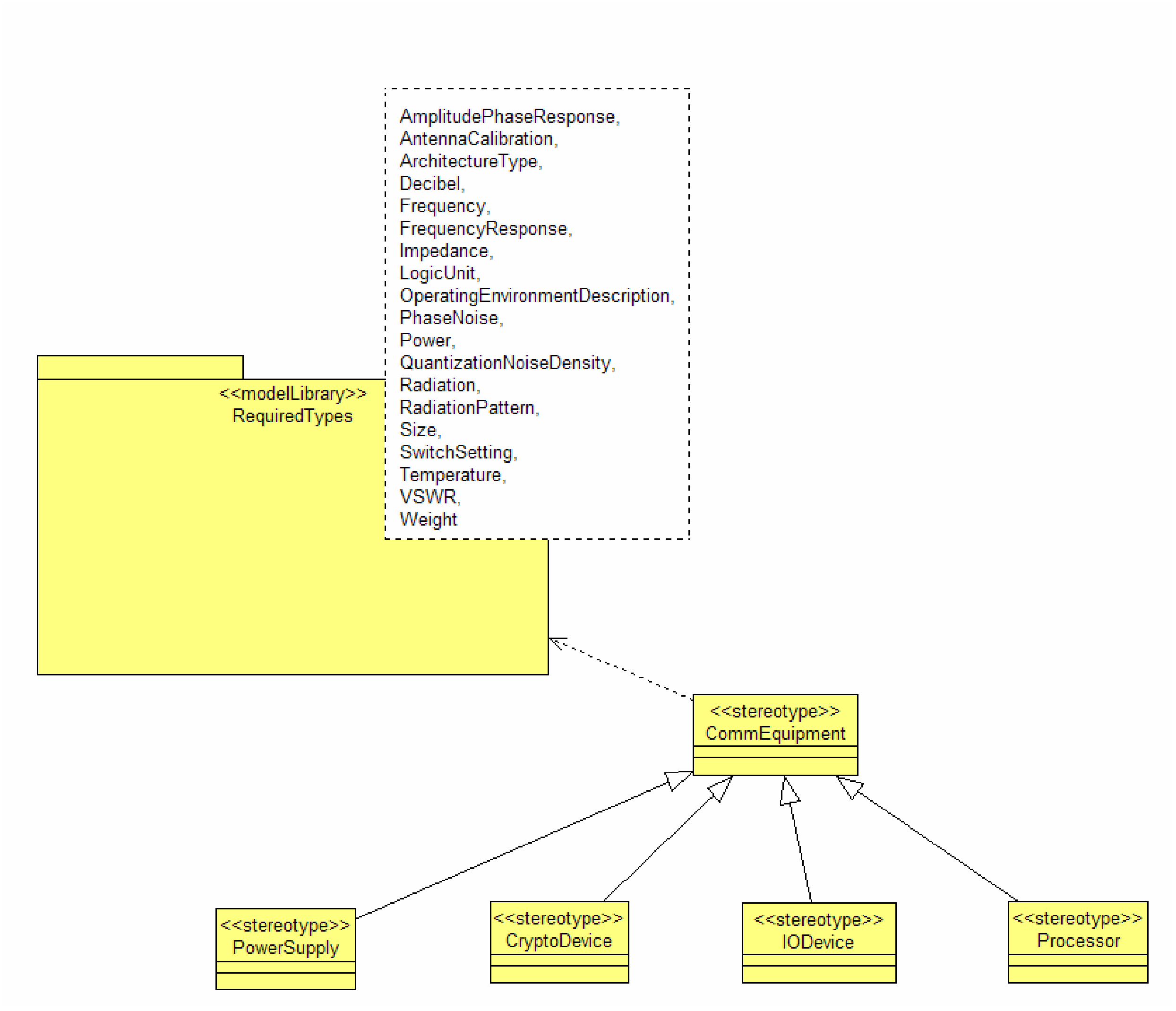


Figure 8-26 – Communication Equipment Overview

8.2.1 RequiredTypes Package

The RequiredTypes template package contains common types for the CommEquipment package. This template package must be bound to another package which substitutes the formal template parameters with concrete classes. The formal parameters are identified in the list below as formal template parameter. The other types are already defined.

Types and Exceptions

- `AmplitudePhaseResponse`

  An amplitude phase response with one point represents a, 1 dB compression point. In an amplitude phase response with two points, 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 the entire AM-to-AM and AM-to-PM curves. Typically, curves represent instantaneous power.

`Note - Issue 7895 provide a primitive type`

- `AntennaCalibration: OctetSequence`

  Antenna calibration data.

- `<<enumeration>>AntennaType (OMNI, DIRECTIONAL, OTHER)`

  The physical configuration of an antenna.

`Note - Issue 7895 provide a primitive type`

- `<<enumeration>>ArchitectureType(FPGA, CPLD, PPC, x86)`

  The architecture of the device (examples could be FPGA, CPLD, PPC, x86, etc).

- `CartesianCoordinates(x: Meter, y: Meter, z: Meter)`

  Three dimensional coordinates. This type is used to specify the location of an object from a given reference point.

`Note - Issue 7708 redundant class Name DigitalConverter`

- `<<enumeration>> ConverterType (ATOD, DTOA, BOTH)`

  The ConverterType defines the type of the converter. A converter can be an analog to digital converter (ATOD), digital to analog converter (DTOA) or can have both functionalities (BOTH).

- `<<enumeration>> CryptoAlgorithm (BLOWFISH, RSA, DES, 3DES, AES, HASH_MD5, OTHER)`

  Cryptographic algorithm.

`Note - Issue 7895 provide a primitive type (added Date & typed Decibel)`

- `Date(ULong day, ULong month, ULong year)`

  Date in days, months, and years.

- `Decibel`

  Decibel, a specialization of Float, denotes the ratio between two voltages, currents, or signal power levels.

- `<<enumeration>>Direction (INPUT, OUTPUT)`

  Direction of data flow.

`Note - Issue 7895`

- `<<enumeration>>DistributionType (GAUSSIAN, POISSON, RAYLEIGH, RICIAN, BINOMIAL, CHISQUARE, TDISTRIBUTION, WEIBULL, LOGNORMAL, NONE)`

  `Specifies the type of probability distribution.`

`Note - Issue 7895 provide a primitive type`

- `Frequency`

  Frequency, a specialization of Float, denotes the number of complete cycles per second of a signal.

`Note - Issue 7895 provide a primitive type`

- `FrequencyResponsePoint(frequency: Hertz, amplitude: dB, phase: Degrees)`
  A frequency response is the relation between signal amplitude and gain versus 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).

`Note - Issue 7895 provide a primitive type`

- `Impedance`
  Impendance, a specialization of Float, denotes the opposition that a device offers to an electric current. Impedance is composed of two components, resistance and reactance.

`Note - Issue 7895 provide a primitive type`

- `LogicUnit`
  LogicUnit, a specialization of UShort, denotes the description a basic logic blocks available inside the device.
- `Meter`
  Meter, a specialization of Double, denotes the fundamental unit of length in the metric system.

`Note - Issue 7895 provide a primitive type`

- `OperatingEnvironmentDescription`
  `OperatingEnvironmentDescription, a specialization of String, denotes a` description of the environment which the device is using (examples could be OS, middleware, etc).

`Note - Issue 7895 provide a primitive type (TBD)`

- `PhaseNoise`
  Random and short duration fluctuations in the phase of a signal.
- `PlugAndPlayInformation(manufacturerName: String, modelName: String, modelNumber: String, modelDescription: String, serialNumber: String, majorRevision: String, minorRevision: String)`
  Generic information about a hardware device.
- `<<enumeration>>PolarizationKind (VERTICAL, HORIZONTAL, RIGHT_CIRCULAR_POLARIZE, LEFT_CIRCULAR_POLARIZE)`
  The orientation of the RF energy radiated from the device.

`Note - Issue 7895 provide a primitive type`

- `Power`
  Power, a specialization of Float, denotes the Rate at which electrical energy is transformed to another type of energy.
- `<<enumeration>>PowerSupplyType (AC_DC, DC_DC)`
  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.

`Note - Issue 7895`

- `ProbabilityDensity (distribution: DistributionType, parameterList: double[*])`
  Specifies an exact or approximate value of a probability density function. In case distribution is NONE, parameterList refers to the expected values of the

random variable E(x), E(x^2), E(x^3), ... etc. Otherwise, parameters list contains the parameters required by the distribution type.

- `QuantizationNoiseDensity`

  Distribution function estimating the quantization noise resulting from using a specific quantization process.

- `<<enumeration>>RadiatingElementType (MONOPOLE, DIPOLE, PATCH, CONE, DISH, OTHER)`

  Physical configuration of a radiating element.

- `Radiation`

  Information about a specific radiation environment.

```
Note - Issue 7895
```

- `Range (minval: ULong, maxval: ULong)`

  Represents the allowable min and max values for a range of values.

```
Note - Issue 7895 provide a primitive type
```

- `RadiationPattern(gain: Decibel, angle: Degrees)`

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

- `<<enumeration>>ReconfigurabilityType (STATIC, DYNAMIC)`

  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.

```
Note - Issue 7895 provide a primitive type
```

- `Size(Float x, Float y, Float z)`

  Represents the physical size of an object in a given unit.

```
Note - Issue 7895 provide a primitive type (TBD)
```

- `SwitchSetting`

  Indicates the connections between the switch's ports.

```
Note - Issue 7895 provide a primitive type
```

- `Temperature`

  Temperature, a specialization of Float, represents the temperature of an object in a given unit (Celsius, Kelvin…).

```
Note - Issue 7895, add definition for Time - comment, it may be better to change
all references of "Time" to TimeType, Issue 8869, change all references to
TimeType and remove Time which is a specialization of Time
```

```
Note - Issue 7895 provide a primitive type
```

- `VSWR`

  VSWR, a specialization of Float, denotes the ratio of the device operating impedance to a desired characteristic impedance (usually 50 ohm characteristic impedance reference).

```
Note - Issue 7895 provide a primitive type
```

- `Weight`

  Weight, a specializtion of Float, represents the physical weight of an object in a given unit.

8.2.2 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 equipment elements, through which signals and messages may be exchanged.

Constraints

The association ends of a CommEquipmentCommunicationPath are of type CommEquipment.

8.2.3 CommEquipmentConnector

Description

The CommEquipmentConnector stereotype is an extension of the UML 2.0 Connector metaclass (from UML2.0::CompositeStructures::InternalStructures). A CommEquipmentConnector is a link that enables communication between two or more instances of communication equipments ports (see Section 8.2.4).

Constraints

The type attribute must be of CommEquipmentCommunicationPath type.

A CommEquipmentConnector connects compatible hardware ports. A set of compatible ports consists 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 must be the output port.

Note – Issue 7582 Removed section 8.2.4 Property

Note – Issue 7985 - Remove QueryProperty and ConfigureProperty subsections from Comm Equipment section 8.2.4 Property.

8.2.4 Port

Communication equipments communicate with each other through ports. Three extensions to the UML 2.0 Port metaclass (from UML2.0::CompositeStructures::Ports) are defined: AnalogInputPort, AnalogOutputPort and DigitalPort. By using the port stereotype, the implementer can customize, or extend a device with additional ports for exchanging control, status or any other information. An 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 can be constructed by aggregating one AnalogInputPort and one AnalogOutputPort. A bidirectional digital port can be constructed by aggregating two instances of DigitalPort.

8.2.4.1 AnalogInputPort

Description

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

Attributes

- `<<characteristicproperty>>inputImpedance: Impedance`
  The inputImpedance attribute represents the impedance of the port.
- `<<characteristicproperty>>inputLevel: Power`
  The inputLevel attribute represents the power level currently at the input of the port.
- `<<characteristicproperty>>inputVSWR: VSWR [0..1]`
  The inputVSWR attribute represents the voltage standing wave ratio of the port.
- `<<characteristicproperty>>insertionLoss: Decibel`
  The insertionLoss attribute represents the 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.
- `<<characteristicproperty>>maxInputLevel: Power`
  The maxInputLevel attribute represents the maximum input power that the port can sustain.

Constraints

An AnalogInputPort can only be connected to an AnalogOutputPort.

8.2.4.2 AnalogOutputPort

Description

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

Attributes

- `<<characteristicproperty>>maxOutputLevel: Power`
  The maxOutputLevel attribute represents the maximum output power that the port can provide.
- `<<characteristicproperty>>outputImpedance: Impedance`
  The outputImpedance attribute represents the impedance of the port.
- `<<characteristicproperty>>outputVSWR: VSWR [0..1]`
  The outputVSWR attribute represents the voltage standing wave ratio of the port.

Constraints

An AnalogOutputPort can only be connected to an AnalogInputPort.

8.2.4.3 DigitalPort

Description

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

Attributes

- `<<characteristicproperty>>dataFlowDirection: Direction`

  The dataFlowDirection attribute indicates whether the port is an input port or an output port.

```
Note – Issue 7895: Changed Integer to Float
```

- `<<characteristicproperty>>maxThroughput: Float`

  The maximum throughput of the port.

```
Note – Issue 7895 - Type is changed from QuantizationNoiseDensity to
ProbabilityDensity
```

- `<<characteristicproperty>>quantizationNoise: ProbabilityDensity`

  The quantizationNoise attribute represents the 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.

- `<<characteristicproperty>>streaming: Boolean`

  The streaming attribute indicates if the port is a streaming port.

Constraints

A DigitalPort with the dataFlowDirection attribute equal to INPUT can only be connected to a DigitalPort with the dataFlowDirection attribute equal to OUTPUT and vice versa.

8.2.5 CommEquipment

Note – Issue 7742 - Changed figure.

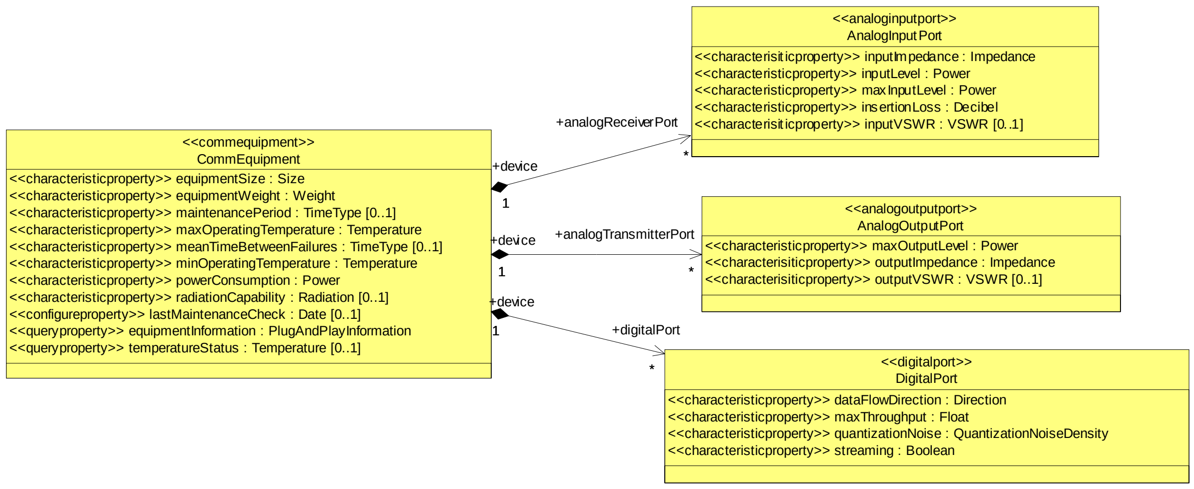

<<analoginputport>>
AnalogInputPort
<<characterisiticproperty>> inputImpedance : Impedance
<<characteristicproperty>> inputLevel : Power
<<characteristicproperty>> maxInputLevel : Power
<<characteristicproperty>> insertionLoss : Decibel
<<characterisiticproperty>> inputVSWR : VSWR [0..1]
+analogReceiverPort
<<commequipment>>
CommEquipment
<<characteristicproperty>> equipmentSize : Size
<<characteristicproperty>> equipmentWeight : Weight
<<characteristicproperty>> maintenancePeriod : TimeType [0..1]
<<characteristicproperty>> maxOperatingTemperature : Temperature
<<characteristicproperty>> meanTimeBetweenFailures : TimeType [0..1]
<<characteristicproperty>> minOperatingTemperature : Temperature
<<characteristicproperty>> powerConsumption : Power
<<characteristicproperty>> radiationCapability : Radiation [0..1]
<<configureproperty>> lastMaintenanceCheck : Date [0..1]
<<queryproperty>> equipmentInformation : PlugAndPlayInformation
<<queryproperty>> temperatureStatus : Temperature [0..1]
+device
1
*
+device
+analogTransmitterPort
1
*
<<analogoutputport>>
AnalogOutputPort
<<characteristicproperty>> maxOutputLevel : Power
<<characterisiticproperty>> outputImpedance : Impedance
<<characterisiticproperty>> outputVSWR : VSWR [0..1]
+device
1
+digitalPort
*
<<digitalport>>
DigitalPort
<<characteristicproperty>> dataFlowDirection : Direction
<<characteristicproperty>> maxThroughput : Float
<<characteristicproperty>> quantizationNoise : QuantizationNoiseDensity
<<characteristicproperty>> streaming : Boolean

Figure 8-27 – CommEquipment M1 Illustration

8.2.5 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. Antennas, amplifiers, CPUs and FPGAs are example CommEquipment Elements. They all contains the attributes of the CommEquipment stereotype.

Attributes

Note – Issue 8869 TimeType

- `<<characteristicproperty>>equipmentSize: Size`
  The size attribute indicates the size of the physical device.
- `<<characteristicproperty>>equipmentWeight: Weight`
  The weight attribute indicates the weight of the physical device.
- `<<characteristicproperty>>maxOperatingTemperature: Temperature`
  The maxOperatingTemperature attribute indicates the maximum sustainable operating temperature of the physical device.
- `<<characteristicproperty>>meanTimeBetweenFailures: TimeType [0..1]`
  The meantimeBetweenFailures attribute indicates the length of time a user may reasonably expect a component to work properly before an incapacitating fault occurs.
- `<<characteristicproperty>>minOperatingTemperature: Temperature`
  The minOperatingTemperature attribute indicates the minimum sustainable operating temperature of the physical device.
- `<<characteristicproperty>>powerConsumption: Power`
  The powerConsumption attribute indicates the power consumed by the device.
- `<<characteristicproperty>>radiationCapability: Radiation [0..1]`
  The radiationCapability attribute indicates the sustainable radiation level of the physical device. This attribute could is useful for radiation hardened devices.
- `<<configureproperty>>lastMaintenanceCheck: Date [0..1]`
  The lastMaintenanceCheck attribute indicates the date at which the last maintenance check was performed. Could be used for devices requiring manual calibration.
- `<<queryproperty>>equipmentInformation: PlugAndPlayInformation`
  The equipmentInformation attribute gives descriptive information about the physical device. This information could be used in a plug and play hardware environment.
- `<<queryproperty>>maintenancePeriod: TimeType [0..1]`
  The maintenancePeriod attribute indicates the time interval between required maintenance check. Could be used for components requiring manual calibration.
- `<<queryproperty>>temperatureStatus: Temperature [0..1]`
  The temperatureStatus attribute indicates the internal temperature of the device.

Note – Issue 7704 - Missing constraint (2nd Ballot)

Constraints

Note – 7586, changed aggregate to composite in text to agree with figure above

CommEquipment shall have a composite relationship to at least one of the following: AnalogInputPort, AnalogOutputPort or DigitalPort.

8.2.5.1 CryptoDevice

Description

The CryptoDevice stereotype represents 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.

Attributes

Note - Issue 7895 - change Integer to Ushort

- `<<queryproperty>>keyLength: UShort [0..*]`
  The keyLength attribute indicates the length of the cipher key supported by the device. It could be either 1024 bits long or more for Public Key or 128 bits or more for Symmetric Key. More than one key length can be supported depending on the algorithm.
- `<<queryproperty>>algorithm: CryptoAlgorithm [0..*]`
  The algorithm attribute identifies the cryptographic algorithms supported by the device.

Constraints

A CryptoDevice shall have at least two DigitalPorts.

A CryptoDevice's DigitalPort shall have its dataFlowDirection attribute equal to INPUT and the other CryptoDevice's DigitalPort shall have its dataFlowDirection attribute equal to OUTPUT.

8.2.5.2 PowerSupply

Description

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

Attributes

Note - Issue 7895 - change Single to Ushort

- `<<characteristicproperty>>efficiency: UShort`
  The efficiency attribute is the ratio of signal power output to total power input.
- `<<characteristicproperty>>type: PowerSupplyType`
  The type attribute indicates if the device converts AC power to DC power or DC power to DC power.

Constraints

Note – 7586 changed text

A PowerSupply shall have at least one AnalogInputPort representing the input voltage and one AnalogOutputPort for the output voltage.

The allowed PowerSupply input or output voltage value range for Efficiency is from 0 to 1 (it is expressed as a percentage).

8.2.5.3 Processor

Description

The Processor stereotype represents 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.

Examples of devices that are considered as processors can include but are not limited to: digital down converter, codec, interconnect, RAID subsystem, memory subsystem etc.

Due to the diverse nature of these devices, they are modeled by their communication capability, i.e. their ports and by their volatile and non-volatile memory capacities.

Attributes

- `<<characteristicproperty>>maxOperatingFrequency: Frequency`
  The maxOperatingFrequency attribute indicates the maximum frequency at which the device is able to operate.
- `<<queryproperty>>processorArchitecture: ArchitectureType`
  The architecture attribute indicates the specific type of programmable device.

`Note - Issue 7895 - change Integer to ULong for both of the following attributes`

- `<<queryproperty>>nonVolatileMemoryCapacity: ULong`
  The nonVolatileMemoryCapacity attribute indicates the total number of bytes of persistent memory available to the processor.
- `<<queryproperty>>volatileMemoryCapacity: ULong`
  The volatileMemoryCapacity attribute indicates the total number of bytes of volatile memory available to the processor.

Constraints

Note – 7586 changed text wording

A Processor shall have at least one DigitalPort.

8.2.5.3.1 ProgrammableLogicDevice

Description

The ProgrammableLogicDevice stereotype represents a device that processes 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.

Attributes

- `<<characteristicproperty>>logicUnitCapacity: LogicUnit`
  The logicUnitCapacity attribute is the total amount of logic units available inside the device.
- `<<characteristicproperty>>reconfigurability: ReconfigurabilityType`
  The reconfigurability attribute indicates whether the device is statically or dynamically reconfigurable.

Note – Issue 8869 TimeType

- `<<characteristicproperty>>timeForReconfiguration: TimeType`
  The timeForReconfiguration attribute indicates the duration of the reconfiguration process.

8.2.5.3.2 SoftwareProcessor

Description

The SoftwareProcessor stereotype represents a device that executes software instructions in order to execute specific algorithms. GPP and DSP processor are example devices of this type.

Attributes

- `<<queryproperty>>operatingEnvironment: OperatingEnvironmentDescription [1..*]`
  The operatingEnvironment attribute contains information regarding the operating environment that the device is using.

8.2.5.4 IODevice

Description

Note – Issue 7706 Name change (2nd Ballot)

The IODevice stereotype represents the base stereotype for all devices that provide analog or digital input/output capability for the RadioSet.

The IODevice class not only applies to the subscriber-side of the radio but also to the RF-side. The term subscriber-side does not imply a human actor. From a higher perspective, both ends of a radio can be considered as I/O. Filters, amplifiers, etc., can be found on both the subscriber-side and RF-side of the equipment.

The members of the IODevice class were conceived with this flexibility in mind. This implies that all devices can operate at non DC frequencies. The IODevice class includes a “tunedFrequency” parameter which can have any frequency as a valid entry.

Elements inheriting the IODevice class can used to construct more complex elements like receivers and exciters.

All of the attributes are optional to cover the specifics of both analog and digital IO devices.

Attributes

- `<<characteristicproperty>> noiseFigure: Decibel [0..1]`
  The noiseFigure attribute is the ratio of the noise power at the output to the noise power at the input, where the input noise temperature is equal to the reference temperature (290 K). The noise figure is expressed in decibels.

- `<<characteristicproperty>>amplitudePhaseResponse: AmplitudePhaseResponse [0..1]`
  The amplitudePhaseResponse attribute gives the amplitude/phase response plot 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.
- `<<characteristicproperty>>maxTunedFrequency: Frequency [0..1]`
  The maxTunedFrequency attribute is the maximum frequency of the bandwidth for which the device performance is rated.
- `<<characteristicproperty>>maxFrequencyResponse: FrequencyResponse[0..1]`
  The maxFrequencyResponse attribute is the maximum frequency response the device is able to achieve. The maximum amplitude and/or phase at a given frequency.
- `<<characteristicproperty>>maxOperatingVSWR: VSWR [0..1]`
  The maxOperatingVSWR attribute is the ratio of the device operating impedance to a desired characteristic impedance (usually 50 ohm characteristic impedance reference).
- `<<characteristicproperty>>maxPowerHandling: Power [0..1]`
  The maxPowerHandling attribute is the maximum power the device can sustain.
- `<<characteristicproperty>>minTunedFrequency: Frequency [0..1]`
  The minTunedFrequency attribute is the minimum frequency of the bandwidth for which the device performance is rated.
- `<<characteristicproperty>>minFrequencyResponse: FrequencyResponse [0..1]`
  The minFrequencyResponse attribute is the minimum frequency response the device is able to achieve. The minimum amplitude and/or phase at a given frequency.
- `<<characteristicproperty>>minPowerHandling: Power [0..1]`
  The minPowerHandling attribute is the minimum RF power the device must be supplied in order to work.
- `<<configureproperty>>freqResponse: FrequencyResponse [0..1]`
  The freqResponse attribute represents the frequency response plot for the device.
- `<<configureproperty>>tunedFrequency: Frequency [0..1]`
  The tunedFrequency attribute corresponds to the center frequency of the frequency response.

Constraints

An IODevice shall have at least one AnalogInputPort or one AnalogOutputPort or one DigitalPort.

8.2.5.4.1 Antenna

Description

The Antenna stereotype, shown in Figure 8-28 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.

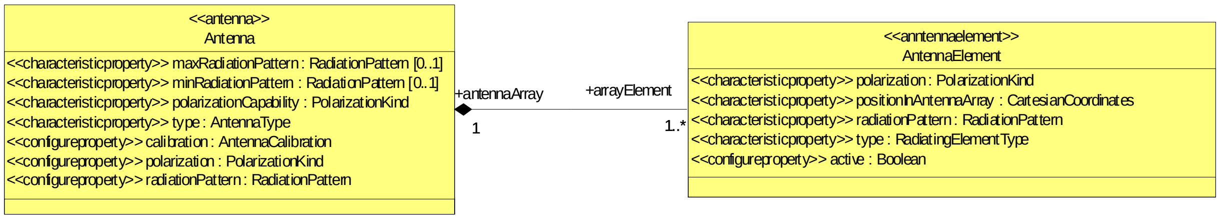

<<antenna>>
Antenna
<<characteristicproperty>> maxRadiationPattern : RadiationPattern [0..1]
<<characteristicproperty>> minRadiationPattern : RadiationPattern [0..1]
<<characteristicproperty>> polarizationCapability : PolarizationKind
<<characteristicproperty>> type : AntennaType
<<configureproperty>> calibration : AntennaCalibration
<<configureproperty>> polarization : PolarizationKind
<<configureproperty>> radiationPattern : RadiationPattern
+antennaArray
1
+arrayElement
1..*
<<anntennaelement>>
AntennaElement
<<characteristicproperty>> polarization : PolarizationKind
<<characteristicproperty>> positionInAntennaArray : CartesianCoordinates
<<characteristicproperty>> radiationPattern : RadiationPattern
<<characteristicproperty>> type : RadiatingElementType
<<configureproperty>> active : Boolean

Figure 8-28 – Antenna M1 Illustration

Attributes

- `<<characteristicproperty>>maxRadiationPattern: RadiationPattern`
  The maxRadiationPattern attribute indicates the maximum radiation pattern that the device is able to achieve.
- `<<characteristicproperty>>minRadiationPattern: RadiationPattern`
  The minRadiationPattern attribute indicates the minimum radiation pattern that the device is able to achieve.
- `<<characteristicproperty>>polarizationCapability: PolarizationKind`
  The polarizationCapability attribute gives the orientation options of the RF energy radiated from the antenna.
- `<<characteristicproperty>>type: AntennaType`
  The type attribute indicates the physical type of the antenna.
- `<<configureproperty>>calibration: AntennaCalibration`
  The calibration attribute contains calibration data for the antenna.
- `<<configureproperty>>polarization:PolarizationKind[0..1]`
  The polarization attribute indicates the current orientation of the RF energy radiated from the antenna.
- `<<configureproperty>>radiationPattern: RadiationPattern`
  The radiationPattern attribute represents the current radiation pattern configured in the device.

M1 Associations

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

Constraints

Note – Issue 7586, changed text description

An Antenna shall have at least one AnalogInputPort or one AnalogOutputPort.

8.2.5.4.2 Amplifier

Description

Note – Issue 7707 - Name change (2nd Ballot)

The Amplifiers stereotype represents a device that provides gain. Amplifiers include but are not limited to base band, RF, power and low noise amplifiers. Different Amplifier types are differentiated by the values of their attributes.

Attributes

`Note – Issue 7895 - change Single to ULong`

- `<<characteristicproperty>>dutyCycle: ULong`
  The dutyCycle attribute is the maximum continuous duty cycle the device can operate.
- `<<characteristicproperty>>maxGain: Decibel`
  The maxGain attribute is the maximum power amplification factor a device is able to apply to a signal.
- `<<characteristicproperty>>minGain: Decibel`
  The minGain attribute is the minimum power amplification factor a device is able to apply to a signal.
- `<<configureproperty>>gain: Decibel`
  The gain attribute is the current power amplification factor applied to the input signal by the device.

Constraints

An Amplifier shall have at least (one AnalogInputPort and one AnalogOutputPort) or two DigitalPorts.

Note – Issue 7708 - Redundant class name DigitalConverter (2nd Ballot)

8.2.5.4.3 Converter

Description

The Converter stereotype represents a device that performs analog-to-digital and / or digital-to-analog conversion of transmit and / or receive signal.

Attributes

- `<<characteristicproperty>> converterType:ConverterType`
  The converterType attribute represents the type of converter. It can be an ATOD, DTOA or BOTH.

`Note – Issue 7895 - change Single to Float`

- `<<characteristicproperty>>maxSampleRate: Float`
  The maxSampleRate attribute is the maximum sample rate the device is able to achieve.

- `<<characteristicproperty>>minSampleRate: Float`
  The minSampleRate attribute is the minimum sample rate the device is able to achieve.

`Note – Issue 7895 - Type is changed from QuantizationNoiseDensity to ProbabilityDensity`

- `<<characteristicproperty>>phaseNoise: ProbabilityDensity`
  The phaseNoise attribute represents the phase noise that the device introduces in the signal.

`Note – Issue 7895 - change Integer to ULong`

- `<<characteristicproperty>>sampleSize: ULong`
  The sampleSize attribute represents the size in bits of a sample.
- `<<configureproperty>>sampleRate: Frequency`
  The sampleRate attribute is the current number of samples per second converted by the device.

Constraints

Note – Issue 7586 changed text description

A Converter shall have at least (one AnalogInputPort or one AnalogOutputPort) and one DigitalPort.

8.2.5.4.4 Filter

Description

Note – Issue 7709 - Name change Filter

The Filter stereotype provides selective frequency gain or attenuation to the Communication Channel in both analog and digital domains. 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.

Constraints

Note – Issue 7586 changed text description

A Filter shall have at least one AnalogInputPort and one AnalogOutputPort.

8.2.5.4.5 FrequencyConverter

Description

Note – Issue 7710 - Name change FrequencyConverter

The FrequencyConverter stereotype represents an analog or digital device that 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, etc.).

In an analog FrequencyConverter, the local oscillator is assumed to be part of the device. Therefore the FrequencyConverter can be a device with two ports. 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 the entire exciter or receiver concept by itself. However, it is a key building block in the definition of higher level concepts.

Attributes

- `<<characteristicproperty>>loInputLeakagePower: Power [0..1]`
  The loInputLeakagepower attribute represents the local oscillator input leakage power.
- `<<characteristicproperty>>loOutputLeakagePower: Power [0..1]`
  The loOutputLeakagePower attribute represents the local oscillator output leakage power.

Note – Issue 7710 - Name change

`Note - Issue 7895 - change Single to Ushort`

- `<<characteristicproperty>>loStability: UShort [0..1]`
  The loStability attribute represents the local oscillator stability expressed in PPM.
- `<<characteristicproperty>>maxInputFrequency: Frequency [0..1]`
  The maxInputFrequency attribute represents the maximum input signal frequency the device is able to handle.
- `<<characteristicproperty>>maxOutputFrequency: Frequency [0..1]`
  The maxOutputFrequency represents the maximum output signal frequency the device is able to handle.
- `<<characteristicproperty>>minInputFrequency: Frequency [0..1]`
  The minInputFrequency represents the minimum input signal frequency the device is able to handle.
- `<<characteristicproperty>>minOutputFrequency: Frequency [0..1]`
  The minOutputFrequency represents the minimum output signal frequency the device is able to handle.

Note – Issue 7710 - Name change

- `<<characteristicproperty>>outputToInputLeakage: Decibel [0..1]`
  The outputToInputLeakage attribute indicates the amount of the output frequency which is found at the input.

- `<<characteristicproperty>>phaseNoise: PhaseNoiseType [0..1]`
  The phaseNoise attribute represents the phase noise that the device introduces in the signal.
- `<<configureproperty>>currentInputFrequency: Frequency [0..1]`
  The currentInputFrequency indicates the frequency of the signal currently at the input of the device.
- `<<configureproperty>>currentOutputFrequency: Frequency [0..1]`
  The currentOutputFrequency indicates the frequency of the signal currently at the output of the device,

Constraints

Note – Issue 7710 - Name change, Issue 7586 changed text description

A FrequenceConverter shall have (at least one AnalogInputPort and one AnalogOutputPort) or (at least two DigitalPorts).

8.2.5.4.6 HoppingFrequencyConverter

Description

The HoppingFrequencyConverter stereotype represents a device that performs frequency conversion while switching between predefined frequencies. It is a specialization of the FrequencyConverter stereotype.

Attributes

- `<<configureproperty>>nextInputFrequency: Frequency [0..1]`
  The nextInputFrequency attribute represents 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.
- `<<configureproperty>>nextOutputFrequency: Frequency [0..1]`
  The nextOutputFrequency attribute represents 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.

Note – Issue 7711 - Name change in RadiatingElement Section 8.2.6.4.7 (2nd Ballot)

8.2.5.4.7 AntennaElement

Description

The AntennaElement stereotype represents a device that translates electrical energy into an electromagnetic wave and vice-versa. An AntennaElement is a passive element. The AntennaElement acts as the transducer between the electrical world and the air interface. Typical examples can be cones, patches, dipoles, dishes, etc.

Figure 8-29 –

Note – Issue 7711 - Name change in RadiatingElement Section 8.2.6.4.7 (2nd Ballot)

Attributes

- `<<characteristicproperty>>polarization: PolarizationKind`
  The polarization attribute indicates the orientation of the RF energy radiated for the AntennaElement.
- `<<characteristicproperty>>positionInAntennaArray: CartesianCoordinates`
  The positionInAntennaArray attribute indicates its 3D position in array with re-pect to the geometric center of the array.
- `<<characteristicproperty>>radiationPattern: RadiationPattern`
  The radiationPattern attribute represents the current radiation pattern for this single AntennaElement.

Note – Issue 7711 - Name change in RadiatingElement Section 8.2.6.4.7 (2nd Ballot)

- `<<characteristicproperty>>type: RadiatingElementType`
  The type attribute indicates the physical configuration of the AntennaElement.
- `<<configureproperty>>active: Boolean`
  The active attribute indicates if the AntennaElement is currently active.

M1 Associations

Note – Issue 7711 - Name change in RadiatingElement Section 8.2.6.4.7 (2nd Ballot)

- `antennaArray: Antenna [1]`
  The antenna object which the AntennaElement is part of.

Constraints

Note – Issue 7586 changed text description

An AntennaElement shall have at least one AnalogInputPort and one AnalogOutputPort.

8.2.5.4.8 Switch

Description

Note – Issue 7712 - Change name Switch to AnalogSwitch.

The Switch stereotype represents a device that provides routing of signals between different devices. A Switch may have many input and output ports and it connects the chosen input port to one or many output ports. It may also be programmed to turn off the signal transmission. In this case, no input would be connected to any output port.

Attributes

- `<<characteristicproperty>>inputOutputIsolation: Decibel`
  The inputOutputIsolation attribute represents the amount of input port leakage on all unselected output ports.
- `<<configureproperty>>switchSetting: SwitchSetting`
  The switchSetting attribute indicates the current configuration matrix of the de-vice.

Constraints

Note – Issue 7712 - Change name Switch to AnalogSwitch.

A Switch shall have (at least one AnalogInputPort and one AnalogOutputPort) or (at least two DigitalPorts).

8.3 Infrastructure

Note – Issue 7785 - Waveform vs Application changes throughout Chapter 8

8.3.1 Radio Services

Note – Issue 7581 moved File Services here. Isssue 7742 added two subsections Radio Services Interfaces and Radio Services Stereotypes. Modified text for section 8.3.1 to reference the new subsections. Removed IStateManagement as a stereotype from Table below. Moved Capability-Model, CapacityModel, CharacteristicModel, and StateManagement interface to Radio Services Interfaces and made them a header 5.

This section defines the interfaces and stereotypes for SWRadio services. The following subsections provide the definitions for Radio Services Interfaces, Radio Services Stereotypes, and File Services.

8.3.1.1 Radio Services Interfaces

The following subsections provide the definitions for StateManagement and CapabilityModel(s).

8.3.1.1.1 CapabilityModel

Description

The abstract CapabilityModel, as shown Figure 8-30, provides the ability to determine if a ServiceProperty can satisfy a deployment requirement. As shown in the 8-30, there are two specializations of a CapabilityModel, which are: CharacteristicModel and CapacityModel.

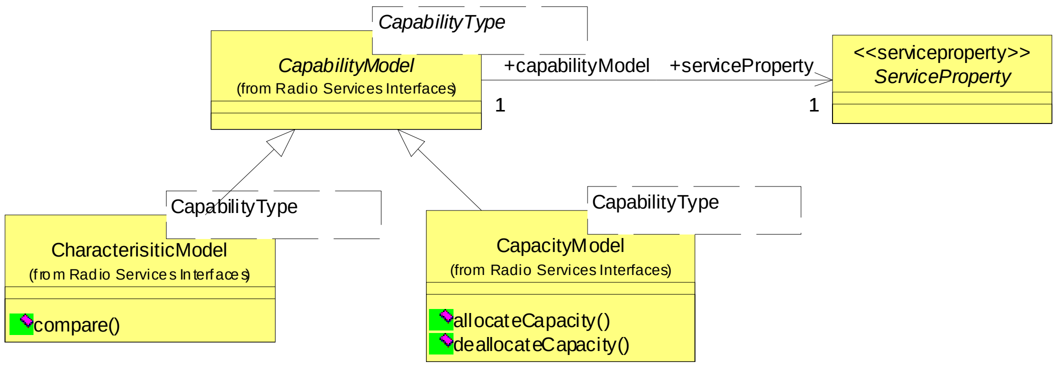


Figure 8-30 – CapabilityModels Definition

M1 Associations

- `serviceProperty: ServiceProperty [1]`
  A ServiceProperty is associated with a CapabilityModel that determines the feasibility of the ServiceProperty satisfying a deployment requirement.

Semantics

A CapabilityModel may be applicable for a set of ServiceProperty(s) depending on the ServiceProperty(s)' value type. For example, a CapabilityModel that compares string types may be applicable for a number of ServiceProperty(s) that are of a String type. CapabilityModel(s) are implemented or used by the deployment machinery such as ApplicationFactory(s) and ManagedServiceComponent(s).

8.3.1.1.2 CapacityModel

Description

The CapacityModel, as shown in Figure 8-30, provides the ability to manage allocation and deallocation of a CapacityProperty.

Operations

- `allocateCapacity (in requiredCapacity: CapabilityType, return Boolean)`
  The allocateCapacity operation provides the mechanism to request and allocate capacity. The allocateCapacity operation behavior is implementation specific.
- `deallocateCapacity (in requiredCapacity: CapabilityType)`
  The deallocateCapacity operation provides the mechanism to deallocate capacity. The deallocateCapacity operation behavior is implementation specific.

Constraints

A CapacityModel shall be associated with CapacityProperty.

Semantics

The types of CapacityProperty managed by a CapacityModel are usually numeric types. Examples of Capacity-Model(s) are: decrement and increment counter types and quantity subtraction and addition types.

8.3.1.1.3 CharacteristicModel

Description

The CharacteristicModel, as shown in Figure 8-30, provides the ability to determine the feasibility of a characteristic property that is static characteristic property.

Operations

- `compare (in requiredCharacterisitic: CapabilityType, return Boolean)`
  The compare operation provides a generic compare mechanism to determine whether a characteristic can satisfy a requiredCharacterisitic request. The implementation of the compare operation is implementation specific.

Constraints

A CharacteristicModel shall be associated with characteristic property type.

Semantics

The types of characterisitic property compared by a CharacteristicModel can be any type (e.g., CharacteristicProperty, CharacteristicSelectionProperty, and CharacteristicSetProperty). Examples of types of Characteristic-Model(s) are equality operators: “eq”, “ne”, “le”, “ge”, “gt”, and “lt” and also selection/matching operators.

8.3.1.1.4 StateManagement

Description

The **StateManagement** interface, as shown in Figure 8-31, provides the ability to retrieve state information and administratively manage a component. The **StateManagement** interface incorporates aspects of the Administrative, Operational, and Usage states described in ISO/IEC International Standard 10164-2.

Attributes

- `<<readonly>>adminState: AdminType [0..1]`
  The administrative state indicates the permission to use or prohibition against using the component. The adminState attribute contains the admin state value.
- `<<readonly>>operationalState : OperationalType`
  The readonly operationalState attribute contains the component's operational state (ENABLED or DISABLED). The operational state indicates whether or not a component is functioning.
- `<<readonly>>adminStateRequestSupportedCharacterisitic : AdminRequestSupportedType = UNLOCK_REQUEST`
  The adminStateRequestSupportedCharacterisitic attribute indicates the behavior requirement for the StateManagement's setAdminState operation and adminState attribute, which indirectly affect the admin states that have to be supported.

- `<<readonly>>states: StatesType`

  The readonly states attribute contains the values for all three states: admin, operational, and usage.
- `<<readonly>>usageState: UsageType`

  The readonly usageState attribute contains the component's usage state (IDLE, ACTIVE, or BUSY). UsageState indicates whether or not a component is in use, and if so, whether or not it has spare capacity for allocation.

Operations

- `<<optional>>setAdminState (in adminRequest: AdminRequestType): {raises = ( UnsupportedRequest )}`

  The setAdminState operation changes the adminState. attribute based upon the input adminRequest parameter.

  The setAdminState operation shall set the adminState attribute to UNLOCKED upon receipt of an UNLOCK admin request.

  The setAdminState operation shall set the adminState to LOCKED and usageState to IDLE upon receipt of a LOCK admin request, when the adminStateRequestSupportedCharacteristic attribute value is LOCK or ALL.

  The setAdminState operation shall set the adminState to SHUTTING_DOWN upon receipt of a SHUTDOWN admin request, when the adminState attribute is UNLOCKED, and adminStateRequestSupportedCharacteristic attribute value is SHUTDOWN or ALL.

  The setAdminState operation shall raise the UnsupportedRequest exception when the adminRequest is not supported.

Types and Exceptions

- `<<enumeration>>AdminRequestType (LOCK, SHUTDOWN, UNLOCK)`

  The AdminRequestType defines the administrative state request values for a component as follows:

  - LOCK - requests a forceful state transition to the LOCKED state.
  - SHUTDOWN - makes a graceful lock request that results in a transition to the SHUTTING_DOWN state
  - UNLOCK - requests a transition to the UNLOCKED state.
- `<<enumeration>>AdminRequestSupportedType (ALL, NOT_IMPLEMENED, LOCK, SHUTDOWN, UNLOCK)`

  The AdminRequestSupportedType defines the valid administrative state request values for a component as follows:

  - ALL - all requests are supported
  - NOT_IMPLEMENTED - no requests are supported
  - LOCK - only LOCK and UNLOCK
  - SHUTDOWN - only SHUTDOWN and UNLOCK
  - UNLOCK - only UNLOCK

- `<<enumeration>>AdminType (LOCKED, SHUTTING_DOWN, UNLOCKED)`

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

  - LOCKED - The component is reserved for administrative usage only. For example, no capacity requests are granted by a component such as device. Transitions to this state may originate from either the SHUTTING_DOWN or UNLOCKED state.
  - SHUTTING_DOWN - a transition state from UNLOCKED to LOCKED, this state does not allow capacity requests to be granted successfully and is maintained until all capacities are deallocated and the component is not in use.
  - UNLOCKED - The component is available for full usage providing the operationalState is ENABLED. A state transition from SHUTTING_DOWN or LOCKED can occur.
- `<<enumeration>>OperationalType`

  The OperationalType defines the operational state values for a component as follows:

  - ENABLED - the component is functional
  - DISABLED - the component is non-functional
- `StatesType( adminState : AdminType, operationalState : OperationalType, usageState : UsageType)`

  The StatesType contains the state values for a component.
- `<<enumeration>>UsageType`

  The UsageType defines the usage state values for a component as follows:

  - IDLE - not in use
  - ACTIVE - in use, with capacity remaining for allocation, or
  - BUSY - in use, with no capacity remaining for allocation

Constraints

When the adminStateRequestSupportedCharacteristic value attribute is NOT_IMPLEMENTED then a component that realizes this interface may not support setAdminState operation and adminState attribute. The NOT_IMPLEMENTED behavior is dependent of the PSM.

The adminState attribute shall transition to the LOCKED state from SHUTTING_DOWN state when the component's usageState attribute becomes IDLE and the adminStateRequestSupportedCharacteristic attribute value is SHUTDOWN or ALL.

Semantics

The interface is used by components that manage their own states and capacities, and provide administrative control. The relationships, values, and state transitions amongst the adminState, operationalState, usageState, and attributes are described in the ISO/IEC International Standard 10164-2.

8.3.1.2 Radio Services Stereotypes

The SWRadio service stereotypes, which are extensions of the UML 2.0 Component (UML2.0::Components::BasicComponents) and Interface (UML2.0::Classes::Interfaces) classifiers are depicted in the Table 8-8 below.

8.3.1 Radio Services

Table 8-8 – SWRadio Services Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
ManagedServiceComponent	Component	ServiceComponent		See constraints in section below	Offers Service(s) within the radio environment, which can be used by swradio components.
Service	Interface	N/A			Provides the parent interface definition for the services within the radio environment that can be used by SWRadio components.
ServiceComponent	Component	N/A			Provides the parent component definition for the service components within the radio environment that can be used by swradio components.

8.3.1.2.1 ManagedServiceComponent

Description

Note – Issue 7742 updated text description and replaced figure with M1 illustration

The **ManagedServiceComponent** is a type of **ServiceComponent** that provides state management behavior as shown in Table 8-8 and Figure 8-31.

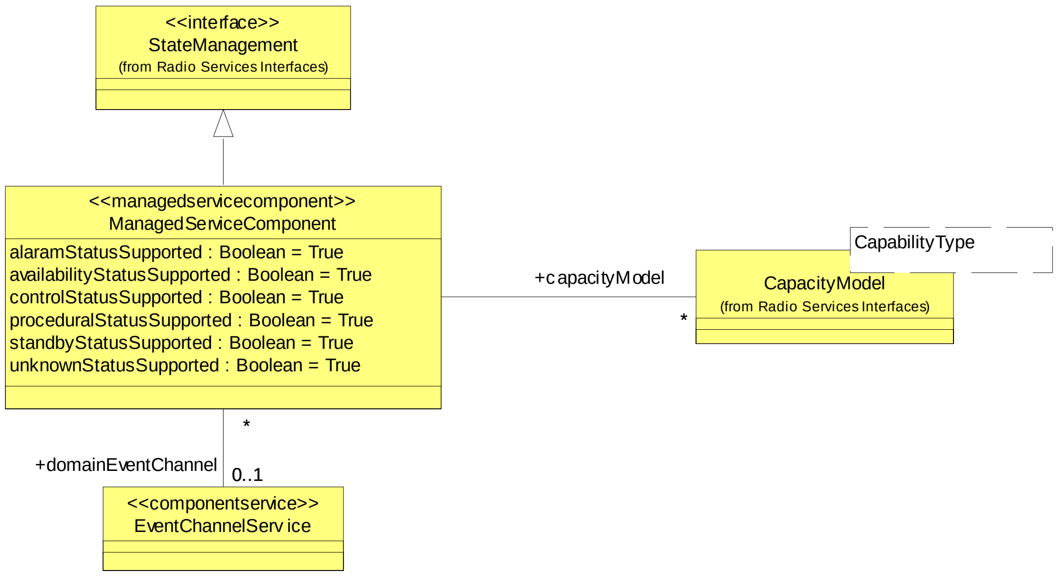

<<interface>>
StateManagement
(from Radio Services Interfaces)
<<managedservicecomponent>>
ManagedServiceComponent
alaramStatusSupported : Boolean = True
availabilityStatusSupported : Boolean = True
controlStatusSupported : Boolean = True
proceduralStatusSupported : Boolean = True
standbyStatusSupported : Boolean = True
unknownStatusSupported : Boolean = True
+capacityModel
*
CapabilityType
CapacityModel
(from Radio Services Interfaces)
*
+domainEventChannel
0..1
<<componentservice>>
EventChannelService

Figure 8-31 – ManagedServiceComponent M1 Illustration

M1 Associations

- `capacityModel: CapacityModel [*]`
  The capacityModel used by a managed service depends on the set of capacities managed by the component.
- `domainEventChannel: EventChannelService [0..1]`
  The event channel used by a managed service for indicating state changes.

Note – Issue 7713 - Wrong section name, Issue 7985

Attributes

Note – Issue 7742 changed the tags to be boolean to indicate if the optional configure and query properties are supported by a component. Added constraints to go with these attributes.

- `alarmStatusSupported: Boolean = false`
  The alarmStatusSupported attribute is used to indicate if the `<<configureproperty>>alarmStatus` is supported. A value of true means supported by the component. A alaramStatus can have zero or more of the following values, not all of which are applicable to every class of managed component. When the value of this attribute is empty set, this implies that none of the status conditions described below are present. AlarmStatus Ushort values are: under repair = 2, critical = 4, major = 8, minor = 16, and alarm outstanding = 32.

- `availabilityStatusSupported: Boolean = false`
  The availabilityStatusSupported attribute is used to indicate if the `<<queryproperty>>availabilityStatus` is supported. A value of true means supported by the component. The availabilityStatus attribute can have zero or more of the following values, not all of which are applicable to every class of managed component. When the value of this attribute is empty set, this implies that none of the status conditions described below are present. AvailabilityStatus UShort values are: in test = 2, failed = 4, power off = 8, off line = 16, off duty = 32, dependency = 64, degraded = 128, not installed = 256, and log full = 512.
- `controlStatusSupported: Boolean = false`
  The controlStatusSupported attribute is used to indicate if the `<<configureproperty>>controlStatus` is supported. A value of true means supported by the component. The controlStatus attribute that can have zero or more of the following values, not all of which are applicable to every class of managed component. When the value of this attribute is empty set, this implies that none of the status conditions described below are present. ControlStatus UShort values are: subject to test =2, part of services locked = 4, reserved for test = 8, and suspended = 16.
- `proceduralStatusSupported: Boolean = false`
  The procedurealStatusSupported attribute is used to indicate if the `<<queryproperty>>procedualStatus` is supported. A value of true means supported by the component. The proceduralStatus is supported only by those classes of managed components that represent some procedure (e.g., a test process) which progresses through a sequence of phases. Depending upon the managed component definition, the procedure may be required to reach certain phase for the resource to be operational and available for use (i.e. for the managed component to be enabled). Not all phases may be applicable to every class of managed component. If the value of this attribute is an empty set the managed component is ready, for example, the initialization is complete. When the value of this attribute is empty set, this implies that none of the status conditions described below are present. ProceduralStatus UShort values are: initialized required = 2, not initialized = 4, initializing = 8, reporting = 16, and terminating = 32.
- `standbyStatusSupported: Boolean = false`
  The standbyStatusSupported attribute is used to indicate if the `<<queryproperty>>standbyStatus` is supported. A value of true means supported by the component. The standbyStatus attribute is single-valued and is only meaningful when the back-up relationship role exists. StandbyStatus UShort values are: standby hot = 2, standby cold = 4, and providing service = 8.
- `unknownStatusSupported: Boolean = false`
  The unknownStatusSupported attribute is used to indicate if the `<<queryproperty>>unknownStatus` is supported. A value of true means supported by the component. The unknownStatus attribute is used to indicate that the state of the resource represented by the managed component is unknown. When the unknownStatus attribute Boolean value is True, the value of the state attributes may not reflect the actual state of the resource.

Constraints

A ManagedServiceComponent shall at a minimum support the UNLOCKED adminState value.

A ManagedServiceComponent shall at a minimum support the ENABLED operationalState value.

A ManagedServiceComponent shall support the IDLE, and ACTIVE and/or BUSY usageState values. The supported usageState values depend on the component's usage model.

A ManagedServiceComponent shall be associated with at least one CapacityProperty as described by the component's descriptor.

When alarmStatusSuuported attribute is true then the ManagedServiceComponent shall support the configuring and querying of the alarmStatus property.

When availabilityStatusSuuported attribute is true then the ManagedServiceComponent shall support the querying of the availabilityStatus property.

When controlStatusSuuported attribute is true then the ManagedServiceComponent shall support the configuring and querying of the controlStatus property.

When proceduralStatusSuuported attribute is true then the ManagedServiceComponent shall support the querying of the proceduralStatus property.

When standbyStatusSuuported attribute is true then the ManagedServiceComponent shall support the querying of the standbyStatus property.

When unknownStatusSuuported attribute is true then the ManagedServiceComponent shall support the querying of the unknownStatus property.

Semantics

Note – Typos

A ManagedServiceComponent may be associated with one to many ServiceProperty(s) (Characteristic Properties and CapacityProperty). The CapacityModel(s) associated with a ManagedServiceComponent depends on the CapacityProperty(s) managed by the component. The capacities associated with a ManagedServiceComponent may be managed or not managed by the component.

The adminStateRequestSupportedCharacterisitic attribute may be dynamically set at creation by a ConfigureProperty or ExecutableProperty.

Whenever the adminState attribute changes, a StateChangeEventType (Infrastructure::Radio Management::Event Channels) event may be issued to an event channel. The StateChangeEventType event data shall be populated as follows when issued:

1. The producerId field is the identifier attribute of the component.
2. The sourceId field is the identifier attribute of the component.
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.

Whenever the operationalState attribute changes, a StateChangeEventType event may be issued to an event channel. The event data shall be populated as follows when issued:

1. The producerId field is the identifier attribute of the component.
2. The sourceId field is the identifier attribute of the component.

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.

Whenever the usageState attribute changes, a StateChangeEventType event may be issued to an event channel. The StateChangeEventType event data shall be populated as follows when issued:

1. The producerId field is the identifier attribute of the component.
2. The sourceId field is the identifier attribute of the component.
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.

8.3.1.3 ServiceComponent

Description

Note – Issue 7714 - Invalid reference, Issue 7742 updated text and figure (M1 Illustration), renamed Associations noheader to be M1 Associations.

The abstract ServiceComponent, as shown in Figure 8-32, offers service(s) within the radio environment, which can be used by SWRadioComponent(s). The potential Service(s) offered by a ServiceComponent are depicted in the Types and Exceptions section below.

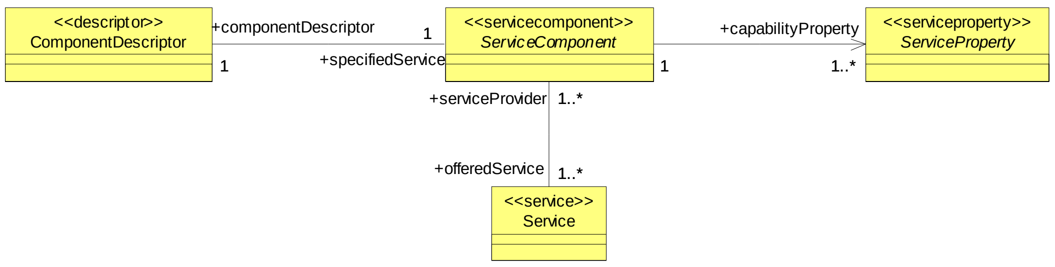


Figure 8-32 – ServiceComponent M1 Illustration

M1 Associations

- `componentDescriptor: ComponentDescriptor [1]`
  The Descriptor that provides the ServiceComponent definition.
- `capabilityProperty: ServiceProperty [1..*]`
  The capability properties that describe a Service's capabilities.
- `offerredService: Service [1..*]`
  An offeredService holds the set of Services that are provided by a ServiceComponent

Types and Exceptions

● `<<service>>EventService`	Provides capabilities for event generation and reception between decoupled components.
● `<<service>>LogService`	Provides capabilities for writing and retrieving system log information, and managing a log
● `<<service>>NamingService`	Provides a white page capability for component registration and retrieval.

Semantics

ServiceComponent(s) are registered with DeviceManager(s). A DomainManagerComponent knows the ServiceComponent(s) when DeviceManagerComponent(s) registers to a DomainManagerComponent.

Note – Issue 7581, Issue 7742 broke the section up into subsections, one for interfaces (model library) and one for stereotypes. Updated figure below.

8.3.1.4 File Services

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 those files on the various processors that they execute upon. The File Services interface are described in section 8.3.1.4.1 and the file services component stereotypes are described in section 8.3.1.4.5. The relationships between the interfaces and components are graphically depicted in Figure 8-33.

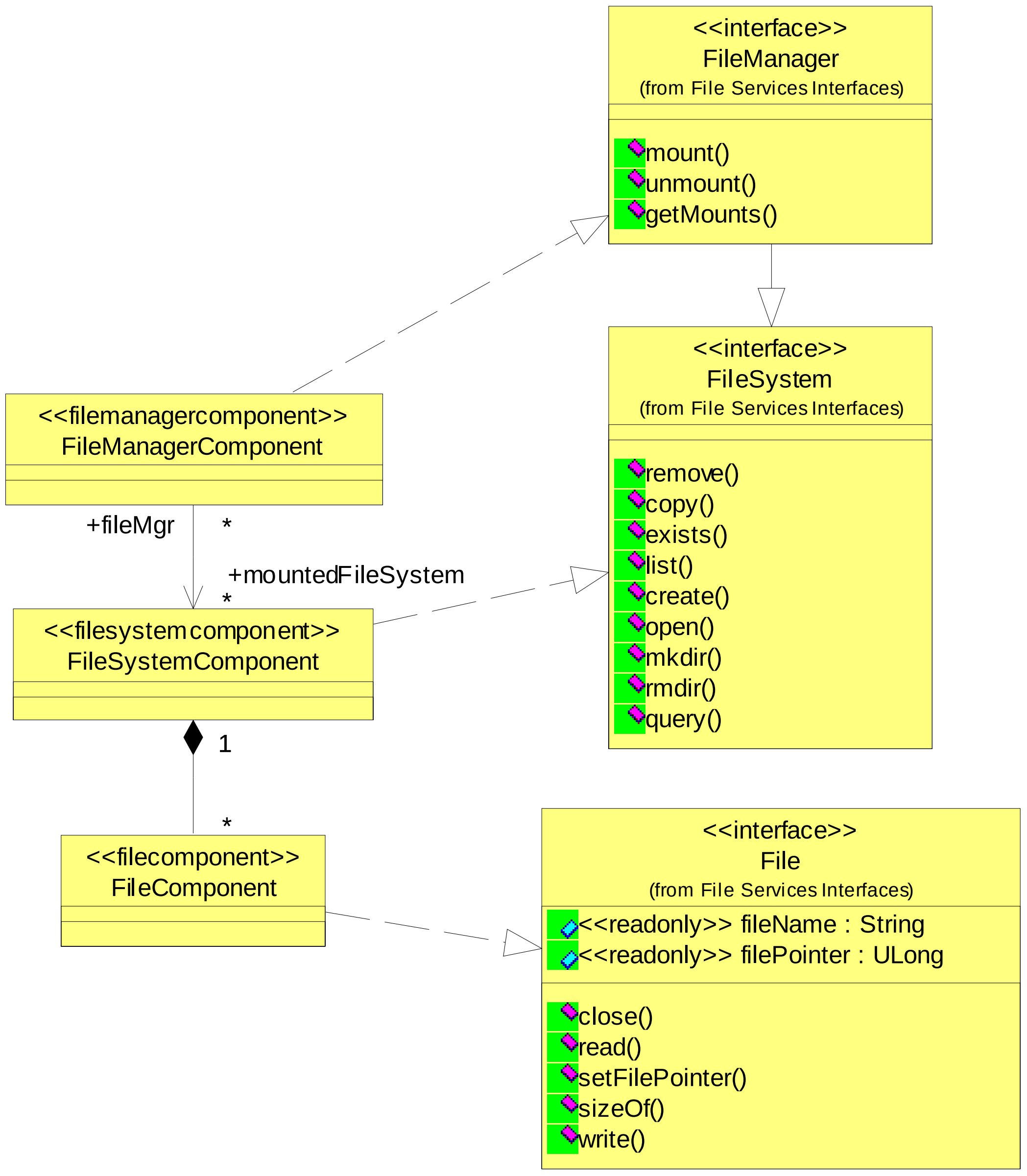


Figure 8-33 – File Services Overview

8.3.1.4.1 File Services Interfaces

This section defines the intefaces for File, FileSystem, and FileManager as shown in Figure 8-33. These interfaces provide basic file manupulation operations which are described in detail in the following subsections.

Types and Exceptions

- `<<exception>>FileException`

  The FileException, 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 FileException occurred

8.3.1.4.2 File

Description

The File interface, as shown in Figure 8-33, provides the ability to read and write files residing within a potentially distributed FileSystem. A file can be thought of conceptually as a sequence of Octet(s) 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.

Attributes

- `<<readonly>>fileName: String`

  The readonly fileName attributes shall contain the file name given to a file system open or create.
- `<<readonly>>filePointer: unsigned Long`

  The readonly filePointer attribute shall contain the file position where the next read or write will occur.

Operations

- `close(): {raises= ( FileException)};`

  The close operation shall close the file from the file system. The close operation shall remove the File component. The close operation shall raise the FileException when it cannot successfully close the file.
- `read(out data : OctetSequence, in length : UnsignedLong) : {raises= ( IOException)};`

  The read operation shall read the number of Octet(s) specified by the input length parameter and advance the value of the filePointer attribute by the number of Octet(s) read. The read operation will read less than the number of octets specified if an end of file is encountered before the input length number of octets is read.

  The read operation shall return via the out data parameter an OctetSequence that equals the number of octets actually read from the File. If the filePointer attribute value is at the end of the File prior to a read, the read operation shall return a 0-length OctetSequence. The read operation shall raise the IOException when a read error occurs.
- `setFilePointer(in filePointer : UnsignedLong) : {raises ( InvalidFilePointer, FileException )}`

  The setFilePointer operation shall set the filePointer attribute value to the input filePointer. The setFilePointer operation shall raise the FileException when the file pointer for the referenced file cannot be set to the value of the input filePointer parameter. The setFilePointer operation shall raise the InvalidFile-

Pointer exception when the value of the filePointer parameter exceeds the file size.

- `sizeOf( return UnsignedLong) : }raises ( FileException )}`

  The sizeOf operation shall return the number of octets stored in the file. The sizeOf operation shall raise the FileException when a file-related error occurs (e.g., file does not exist anymore).

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

  The write operation shall write data to the file. If the write is successful, the write operation shall update the filePointer attribute to reflect the number of octets written. If the write is unsuccessful, the filePointer attribute value shall maintain or be restored to its value prior to the write operation call. The write operation shall raise the IOException when a write error occurs.

Types and Exceptions

- `<<exception>>InvalidFilePointer`

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

- `<<exception>>IOException`

  The IOException exception, specialization of SystemException, indicates an error occurred during a read or writes operation to a File. The error number (e.g., CF_EFBIG, CF_ENOSPC, CF_EROFS and message is component-dependent. The message provides additional information describing the reason for the error.

Constraints

The filePointer shall be 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 shall be set at the end of the File.

8.3.1.4.3 FileSystem

Description

The FileSystem interface, as shown in Figure 8-33, defines common operations that enable access to a physical file system.

Operations

Note – Issue 8831 - copy () - Added behavior and exception requirements for when the target file exists.

- `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 shall copy the source file with the specified sourceFileName to the destination file with the specified destinationFileName. If the destination file already exists, and the sourceFileName and the destinationFileName are different, the copy operation shall overwrite the destination file. The copy operation shall raise the FileException when a file-related error occurs. The copy operation shall raise the InvalidFileName exception when the source or destination filename is not a valid file name or not an absolute pathname. The copy operation shall raise the InvalidFileName exception when the destination file name is identical to the source file name (i.e. attempting to copy on top of itself).

- `create(in fileName : String, return FileComponent) : {raises = ( InvalidFileName, FileException )}`
  The create operation provides the ability to create a new file on the FileSystem. The create operation shall create a new File based upon the fileName attribute. The create operation shall return a File component reference to the opened file. The create operation shall raise the FileException if the file already exists or another file error occurred. The create operation shall raise the InvalidFileName exception when a fileName is not a valid file name or not an absolute pathname or path prefix does not exist.
- `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 shall return True if the file exists, or False if it does not. The exists operation shall raise 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 return information for one file or a set of files. The list operation shall return a FileInformationSequence for files that match the wildcard criteria specified in the input pattern parameter. The list operation shall return a zero length sequence when no files are located that match the input pattern parameter. The list operation shall raise the InvalidFileName exception when the input pattern does not start with a slash "/" or cannot be interpreted due to unexpected characters. The list operation shall raise the FileException when a file-related error occurs.
- `open(in filename : String, in read_Only : Boolean, return : FileComponent) : {raises = ( InvalidFileName, FileException )}`
  The open operation provides the ability to open a file for read or write. The open operation shall open a file based upon the input fileName. The read_Only parameter indicates if the file should be opened for read access only. The open operation shall open the file for read only access when the read_Only parameter is True. The open operation shall return a File component parameter on successful completion. The returned File's filePointer attribute shall be set to the beginning of the file when the read_Only parameter is true, otherwise the File's filePointer attribute is set to the end of the file. 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 shall raise the FileException if the file does not exist or another file error occurred. The open operation shall raise 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 shall create all parent directories required to create the directoryName path given. The mkdir operation shall raise the FileException if a file-related error occurred during the operation. The mkdir operation shall raise 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 shall return file system information to the calling client based upon the given fileSystemProperties' ID. At a minimum, the query operation shall support the following property Ids in the fileSystemProperties parameter:

  1. SIZE - an ID value of “SIZE” causes query to return an unsigned long long containing the file system size (in octet(s) <<datatype>>).
  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 octet(s) <<datatype>>). The query operation shall raise the UnknownFileSystemProperties exception when the given file system property is not recognized.

- `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 shall remove the file with the given fileName. The remove operation shall raise the InvalidFileName exception when the fileName is not a valid fileName or not an absolute pathname. The remove operation shall raise the FileException when a file-related error occurs.

- `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 shall raise the FileException when the directory does not exist, if the directory is not empty, or another file-related error occurred. The rmdir operation shall raise the InvalidFileName exception when the directoryName is not a valid directory name.

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.

- `<<enumeration>>FileType (PLAIN, DIRECTORY, FILE_SYSTEM)`

  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 ( name : String, kind : FileType, size : UnsignedLongLong fileProperties : Properties)`

  The FileInformationType indicates the information returned for a file. Not all the fields in the FileInformationType are applicable for all file systems. The FileInformationType attributes are:

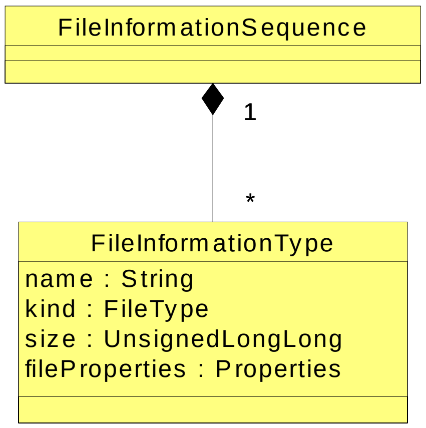
  - The name indicates the simple name of the file.
  - The kind indicates the type of the file entry.
  - The size indicates the size in Octet(s).

- The FileProperties indicates other optional File properties that could be given out.

- `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.
- `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.
- `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).
- `<<exception>>UnknownFileSystemProperties`

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

Constraints

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

At a minimum, the FileSystem shall support the FileInformationType attributes: name, kind, and size information for a file. Examples of other file properties that may be specified for fileProperties are created time, modified time, and last access time as stated in Types and Exceptions above. The value for these properties shall be unsigned long long and measured in seconds since 00:00:00 UTC, Jan. 1, 1970.

8.3.1.4.4 FileManager

Description

The FileManager, as shown in Figure 8-33, specializes the FileSystem 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 as 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 unmount operations.

Associations

- `mountedFileSystem:FileSystemComponent[*]`
  MountedFileSystem(s) that are associated with a FileManagerComponent.

Operations

- `getMounts( return MountSequence)`
  The getMounts operation shall return a sequence of Mount structures that describe the mounted FileSystems.
- `mount(in string mountPoint, in FileSystemComponent file_System) : {raises( InvalidFileName, InvalidFileSystem, MountPointAlreadyExists )}`
  The mount operation shall associate the specified FileSystem with the given mountPoint. A mountPoint name shall begin with a "/". A mountPoint name is a logical directory name for a FileSystem. The mount operation shall raise the InvalidFileName exception when the input file name is invalid. The mount operation shall raise the MountPointAlreadyExists exception when the mountPoint already exists in the file manager. The mount operation shall raise the InvalidFileSystem exception when the input FileSystem is a null object reference.
- `query(inout fileSystemProperties : Properties) : {raises( UnknownFileSystemProperties )}`
  The query operation shall return the combined mounted file systems information to the calling client based upon the given input fileSystemProperties' IDs. As a minimum, the query operation shall support the following input fileSystemProperties IDs:

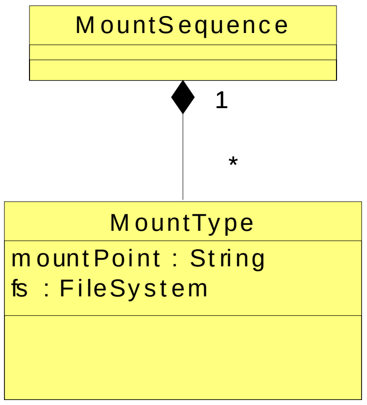
  1. SIZE - a property item ID value of “SIZE” will cause the query operation to return the combined total size of all the mounted file system as an unsigned long long property value.

  2. AVAILABLE_SPACE - a property item ID value of “AVAILABLE_SPACE” will cause the query operation to return the combined total available space (in Octet(s)) of all the mounted file system as unsigned long long property value.

- `unmount(in string mountPoint) : {raises( NonExistentMount )}`
  The unmount operation shall remove a mounted FileSystemComponent from the FileManagerComponent whose mounted name matches the input mountPoint name. The unmount operation shall raise the NonExistentMount exception when the mountPoint does not exist.

Types and Exceptions

- `MountType ( MountPoint : String, fs : FileSystem)`
  The MountType structure identifies the FileSystems mounted within the FileManager.
- `MountSequence`
  The MountSequence is an unbounded sequence of MountTypes.



- `<<exception>>InvalidFileSystem`
  The InvalidFileSystem exception indicates the FileSystemComponent is a null (nil) component reference.
- `.<<exception>>MountPointAlreadyExists`
  The MountPointAlreadyExists exception indicates the mount point is already in use in the FileManagerComponent.
- `<<exception>>NonExistentMount`
  The NonExistentMount exception indicates a mount point does not exist within the FileManagerComponent.

Constraints

The FileManager's operations shall remove the FileSystem mounted name from the input fileName before passing the fileName to an operation on a mounted FileSystem.

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

The FileManager shall propagate exceptions raised by a mounted FileSystem's operation.

Semantics

The FileManager's FileSystem operations behavior implements the requirements of the FileSystem operations against the mounted file systems. The FileManager's FileSystem operations ensure that the filename/directory arguments given are absolute pathnames relative to a mounted FileSystem.

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 file system components. From the client perspective, the File Manager may be used just like any other FileSystem component since the FileManager realizes the FileSystem interface.

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.

8.3.1.4.5 File Services Stereotypes

The File Services components are identified in Table 8-9 - File Services Stereotypes and their relationship are graphically depicted in Figure 8-33.

Note – Issue 7742 removed IFileSystem from stereotype table, moved interface to File Services Interfaces section. Added component to stereotype names to be consistent with spec and to be different than the interface names.

Table 8-9 – File Services Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
FileComponent	Component	N/A			Reads and writes a file within a FileSystem.
FileSystemComponent	Component	N/A			Remotely accesses a physical file system.
FileManagerComponent	Component	N/A			Provides a single interface to multiple, distributed FileSystems

8.3.1.4.6 FileComponent

Description

The File, as shown in Figure 8-33, provides the ability to read and write files residing within a potentially distributed FileSystem. A file can be thought of conceptually as a sequence of Octet(s) 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.

Constraints

A FileComponent shall realize the File interface.

8.3.1.4.7 FileManagerComponent

Description

The FileManagerComponent, as shown in Figure 8-33, realizes the FileManager interface and extends that interface by adding mount and unmount operations. This allows multiple, distributed FileSystemComponents to be accessed through a FileManagerComponent. The FileManagerComponent appears as a single file system 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 unmount operations.

M1 Associations

- `mountedFileSystem:FileSystemComponent[*]`
  MountedFileSystem(s) that are associated with a FileManagerComponent.

Constraints

The FileManagerComponent shall realize the FileManager interface.

8.3.1.4.8 FileSystemComponent

Description

A FileSystemComponent, as shown in Figure 8-33, realizes the FileSystem interface and may be associated with a File-Manager and consist of many FileComponents.

Associations

- `fileManager : FileManagerComponent[*]`
  A file system can be associated with many FileManagerComponents.

8.3.2 Communication Channel

Description

Note – Issue TBD , remove last sentence, Issue 7742 updated figure by removing associations since stereotypes cannot have associations. Updated text below.

A SWRadio provides a means to enable communications between physically separated users. A SWRadio has the capability to utilize its devices as needed for a particular communications scenario and to use them possibly in a different way in another instance. A Communication Channel is the data description for the collection and interconnection of the radio's devices necessary for a particular application to be able to provide communication.

The LogicalCommunicationChannel, shown in which inherits from an abstract Channel class is logically partitioned into partitioned into three groups: the LogicalPhysicalChannel (e.g., Radio Frequency (RF)), the LogicalProcessingChannel, and the LogicalIOChannel. LogicalPhysicalChannel bundles devices that provide communication over the physical medium, LogicalIOChannel bundles devices that provide I/O functionality for the platform and LogicalProcessingChannel for signal processing needs. 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 component stereotyped as CommChannel as defined in Infrastructure::Radio Management::Radio Set Management package needs visibility into the capabilities needed by its applications and the capabilities provided by its devices in order to deploy a usable communication channel.

Note – Issue 7582, updated figure and title below

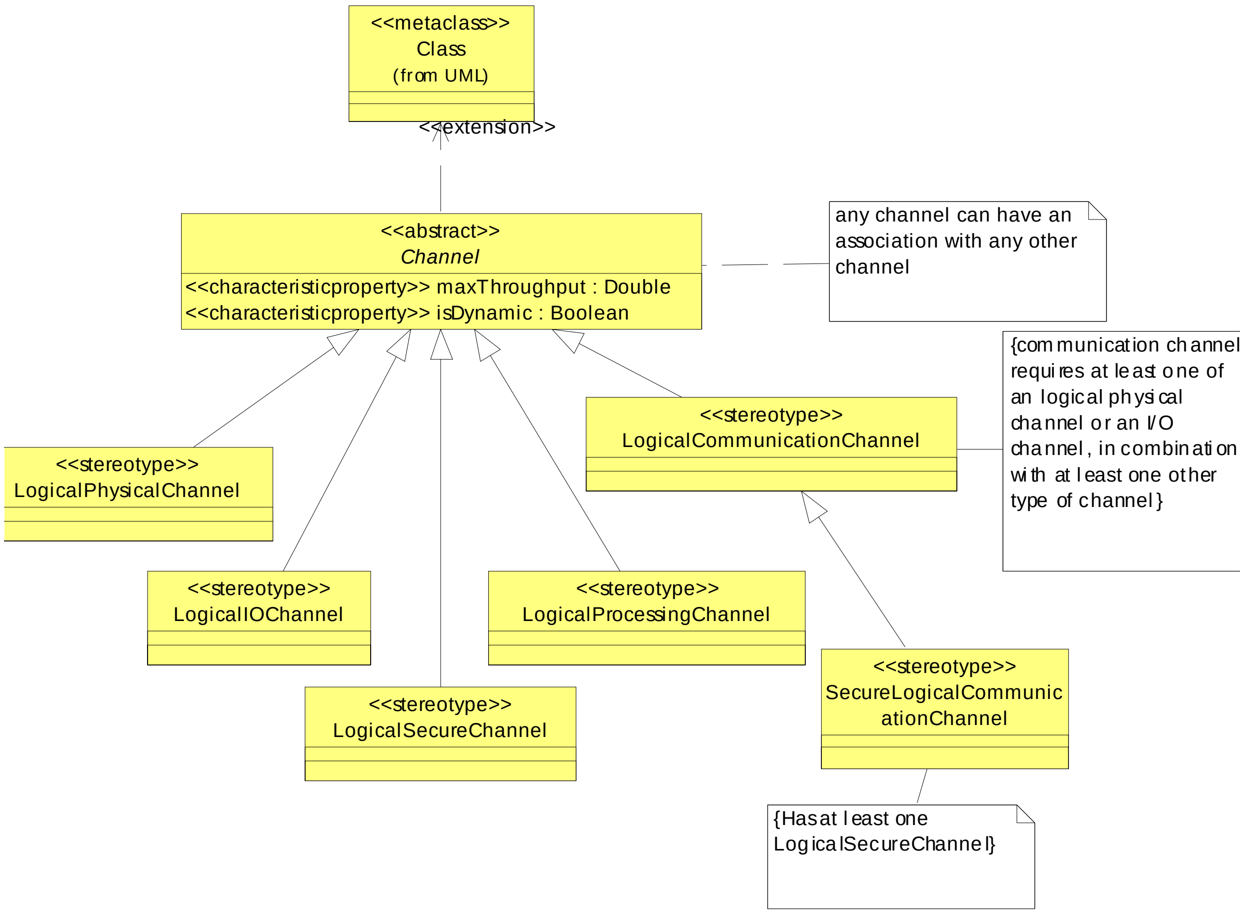

<<metaclass>>
Class
(from UML)
<<extension>>
<<abstract>>
Channel
<<characteristicproperty>> maxThroughput : Double
<<characteristicproperty>> isDynamic : Boolean
any channel can have an association with any other channel
{communication channel requires at least one of an logical physical channel or an I/O channel, in combination with at least one other type of channel}
<<stereotype>>
LogicalPhysicalChannel
<<stereotype>>
LogicalCommunicationChannel
<<stereotype>>
LogicalIOChannel
<<stereotype>>
LogicalProcessingChannel
<<stereotype>>
SecureLogicalCommunicationChannel
<<stereotype>>
LogicalSecureChannel
{Has at least one LogicalSecureChannel}

Figure 8-34 – Communication Channel Types Overview

8.3.2.1 Channel

Description

Note – Issue 7582 modified text

Channel provides an abstract class definition by extending the UML Class definition. This abstract class definition is specialized by all of the stereotype definitions in the Communication Channel section.

Attributes

Note – Issue 8125: throughput chaned to maxThroughput, Issue 7582 added stereotypes to attributes

- `<<characteristicproperty>>maxThroughput: Double`
  Data throughput of the channel
- `<<characteristicproperty>>isDynamic: Boolean`
  Specifies whether the channel is a dynamic channel or not. A Dynamic channel is one whose definition can be changed in run-time by the application

Note – Issue 7742 Changed Associations noheader to M1 Associations. Removed last sentence in Description.

M1 Associations

- `channel: Channel [*]`
  A channel can have associations to any number of channels

8.3.2.2 LogicalCommunicationChannel

Description

LogicalCommunicationChannel stereotype is a specialization of the abstract Channel and is a data descriptor for different types sub-channels. It is an aggregate of LogicalProcessingChannel, LogicalIOChannel and LogicalPhysical channel as shown in Figure 8-35.

Note – Issue 7742 updated Figure below to be M1 type. Changed Associations noheader to M1 Associations and changed associations descriptions to match figure. Updated Constraints

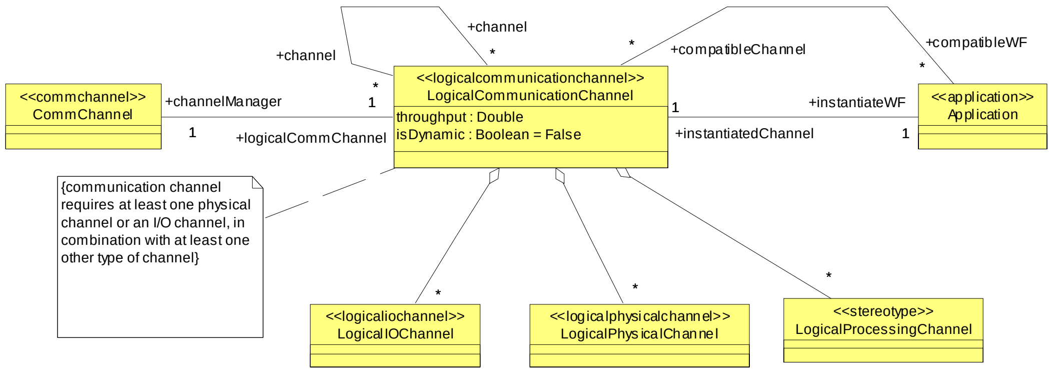


Figure 8-35 – LogicalCommunicationChannel M1 Illustration

M1 Associations

- `compatibleWF: Application [*]`
  A Logical Communication Channel may have all the capabilities required by a WaveformApplication.
- `instantiatedWF: Application [1]`
  An instantiated application runs on its associated CommunicationChannel
- `commManager: CommChannel [1]`
  A LogicalCommunicationChannel is managed by a CommChannel

Constraints

A LogicalCommChannel communication channel requires at least one LogicalPhysicalChannel or an LogicalIO-Channel and combination of any other channel type (LogicalPhysicalChannel, LogicalIOChannel, and LogicalProcessingChannel).

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.

8.3.2.3 SecureLogicalCommunicationChannel

Description

The SecureLogicalCommunicationChannel stereotype is an extnesion of LogicalCommunicationChannel stereotype and adds another aggregation relationship to the LogicalSecurityChannel stereotype as shown in Figure 8-34. Includes the relationships inherited from the LogicalCommunicationChannel, this stereotype provides a data descriptor definition that can be made of four different types of sub-channels: LogicalProcessingChannel, LogicalIOChannel, LogicalPhysicalChannel and LogicalSecurityChannel.

Note – Issue 7742 added constraints

Constraints

A SecureLogicalCommunicationChannel shall have at least one LogicalSecureChannel associated with it.

Note – Issue 7742 updated figure below and changed Associations noheader to M1 Associations noheader. Added Constraints section and one constraint

8.3.2.4 LogicalPhysicalChannel

Description

The LogicalPhysicalChannel stereotype extends the abstract Channel by consisting 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. Figure 8-36 shows the LogicalPhysicalChannel definition with attributes of its aggregated components.

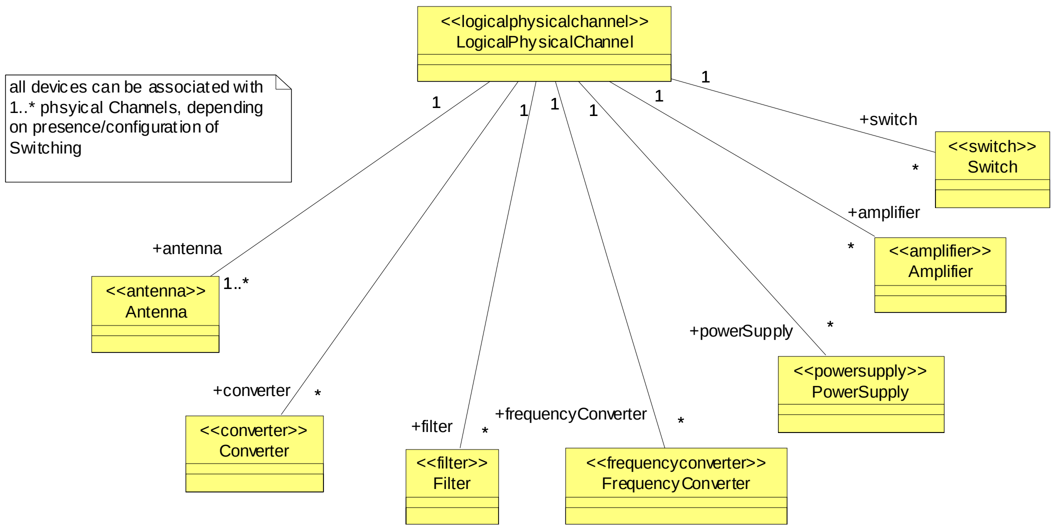

all devices can be associated with 1..* phsyical Channels, depending on presence/configuration of Switching
<<logicalphysicalchannel>> LogicalPhysicalChannel
<<antenna>> Antenna
+antenna
1..*
<<converter>> Converter
+converter
<<filter>> Filter
+filter
<<frequencyconverter>> FrequencyConverter
+frequencyConverter
<<powersupply>> PowerSupply
+powerSupply
<<amplifier>> Amplifier
+amplifier
<<switch>> Switch
+switch
1
*

Figure 8-36 – LogicalPhysicalChannel M1 Illustration

M1 Associations

- `anntenna: Antenna [1..*]`
  LogicalPhysicalChannel can be associated with an Antenna
- `converter: Converter [*]`
  LogicalPhysicalChannel can be associated with a Converter
- `filter: Filter [*]`
  LogicalPhysicalChannel can be associated with a Filter
- `amplifier: Amplifier [*]`
  LogicalPhysicalChannel can be associated with an Amplifier
- `frequencyConverter: FrequencyConverter [*]`
  LogicalPhysicalChannel can be associated with a FrequencyConverter
- `powerSupply: PowerSupply [*]`
  LogicalPhysicalChannel can be associated with a PowerSupply
- `switch: Switch [*]`
  LogicalPhysicalChannel can be associated with a Switch

Constraints

- `A LogicalPhysicalChannel shall be associated with at least one Antenna element.`

8.3.2.5 LogicalProcessingChannel

Description

The LogicalProcessingChannel stereotype extends the abstract Channel and provides the processing nodes for applications and radio services used by the waveforms 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 LogicalSecurityChannel. Figure 8-37 shows the LogicalProcessingChannel definition.

Note – Issue 7742 updated figure below to a M1 type and updated associations. Change Associations noheader to be M1 Associations. Addec Constraints sections and one constraint.

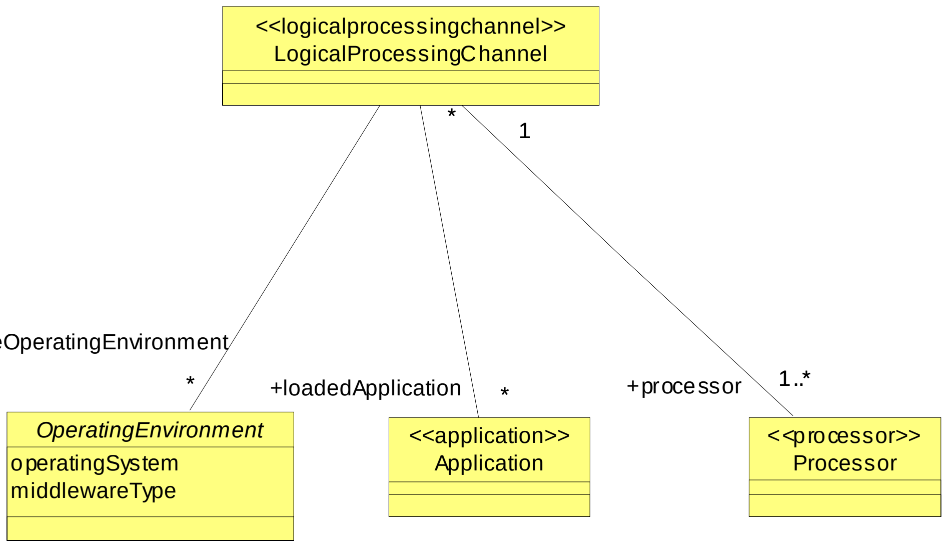


Figure 8-37 – LogicalProcessingChannel M1 Illustration

M1 Associations

- `availableOperatingEnvironment: OperatingEnvironment [*]`
  A LogicalProcessingChannel uses the interfaces provided by the operating environment
- `loadedApplication: Application [*]`
  A LogicalProcessingChannel can run multiple Applications on it
- `processor: Processor [1..*]`
  A LogicalProcessingChannel contains at least one processor to perform computations on.

Constraints

- A LogicalProcessingChannel shall be associated with at least one processor.

8.3.2.6 LogicalIOChannel

Description

The LogicalIOChannel stereotype extends the abstract Channel and provides for the baseband connection to the radio and consists of the devices that format, encode, decode, etc. the communication signals at that interface. Figure 8-38 shows the description of the Logical I/O Channel.

Note – Issue 7716 updated Figure 8-37 (below). Removed IO_Algorithm.

Note – Issue 7742 updated figure, Changed Associations noheader to M1 Associations noheader and modified associations.

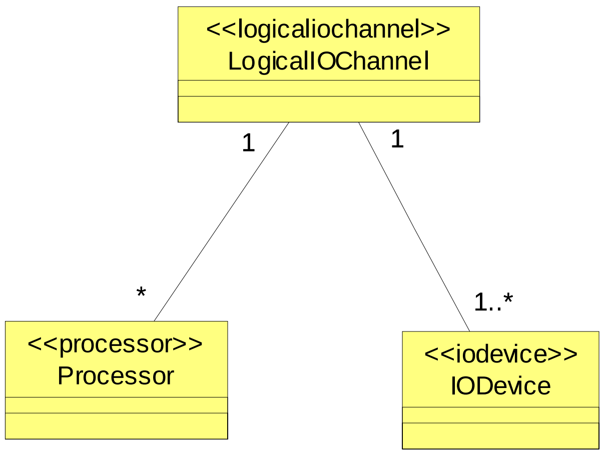


Figure 8-38 – LogicalIOChannel M1 Illustration

M1 Associations

Note – Issue 8122 and 8123 modified figure above for associations and text below. Added Constaints

- `processor: Processor [*]`
  A LogicalIOChannel may be associated with zero or more processors.
- `loadedAlgorithm: I/O_Algorithm [*]`
  There may be any number of IO algorithms loaded on the IO channel
- `ioDevice: IODevice [1..*]`
  Each LogicalIOChannel has only one IODevice

Constraints

- A LogicalIOChannel shall be associated with at least one IODevice.

Note – Issue 7742 Updated figure below, Replaced Associations noheader with M! associations no header. Updated constraints

Note – Issue 7716 - I/O Algorithm Description

Semantics

The LogicalIOChannel can include an IO algorithm which can be distinguished by the codec type and data conversion type. An IODevice and Processor can be associated with the algorithm that is a part of the channel. Processor acts as a data processor and an algorithm loader while the IODevice acts as the data processor which employs the IO algorithm to process data.

8.3.2.7 LogicalSecurityChannel

Description

The LogicalSecurityChannel stereotype extends the abstract Channel and provides the processing node(s) for security applications applicable to communications. The LogicalSecurityChannel is present in a logical channel definition only if the channel has security requirements. This channel may be used for separating between secure and insecure sides of the communication. (Red - Black separation). The LogicalSecurityChannel definition is shown in Figure 8-39.

Note – Issue 7717 - Removed SecurityAlgorithm, SecurityKey, SecurityPolicy from Figure 8-38 (below)

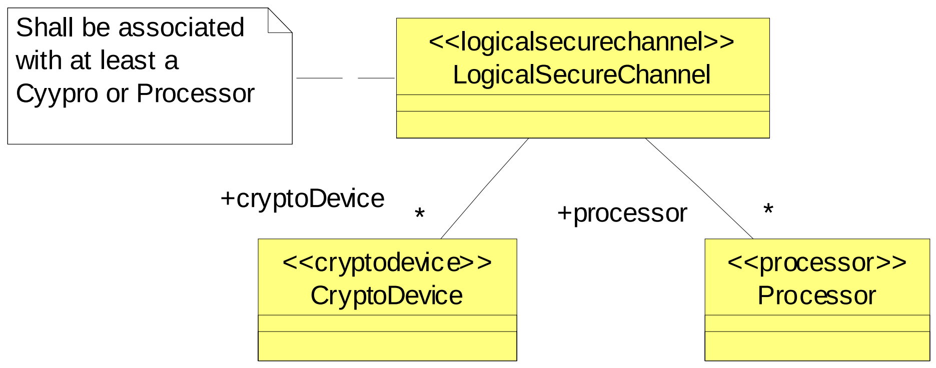


Figure 8-39 – LogicalSecurityChannel M1 Illustration

M1 Associations

- `cryptoDevice: CryptoDevice [1..*]`
  A LogicalSecurityChannel shall be associated with at least one CryptoDevice
- `processor: Processor [*]`
  A LogicalSecurityChannel may be associated with any number of processors
- `loadedAlgorithm: SecurityAlgorithm [1..*]`
  A LogicalSecurityChannel shall be associated with at least one SecurityAlgorithm. A security channel may support loading multiple algorithms at the same time

- `loadedKey: SecurityKey [1..*]`
  A LogicalSecurityChannel shall be associated with at least one SecurityKey.
- `loadedPolicy: SecurityPolicy [*]`
  A LogicalSecurityChannel may be associated with any number of security policies that direct its actions.

Constraints

LogicalSecurity Channel has either a crypto device or a processor that runs a security algorithm; it may have a processor(s) for other functions.

The LogicalSecurityChannel runs security algorithms on either a Processor or a dedicated Crypto Device

---

Note – Issue 7717 Resolution

Semantics

A LogicalSecurityChannel uses the Crypto and the Processor to provide security features of a waveform. A LogicalSecurityChannel may provide a security algorithm, security keys and a security policy in order to facilitate those features.

8.3.3 Radio Management

This section defines the stereotypes for radio management. Radio management involves the management of the radio, inclusive of its devices and services. The radio management stereotypes are categorized by domain and device management. The details of these categories are described in the following RadioSet Management and Device Management subsections.

8.3.3.1 RadioSet Management

Note – Issue 7742 created subsections RadioSet Management Interfaces and RadioSet Management Stereotypes. Moved the interfaces to RadioSet Management Interfaces. Made Interfaces sections a header 6. Added DomainManager interface definition that was based upon DomainManager component.

8.3.3.1.1 RadioSet Management Interfaces

This section defines the interfaces for RadioSet management. The types of capabilities offered by RadioSet management are categorized as follows:

1. Domain Registration Management - provides the mechanism for registering and unregistering DeviceManager’s services within a RadioSet.
2. Domain Installation Management - provides the mechanism for installing and nonstaining applications within a RadioSet.
3. Domain Manager - provides the mechanism for retrieving a radio domain's components.
4. Domain Event Channels- provides the mechanism for managing RadioSet's event channels connections.

Types and Exceptions

- `<<exception>>InvalidProfile`

  The InvalidProfile exception indicates an invalid profile error. A profile error indicates an invalid Descriptor (e.g., component description, Application description, etc.).

8.3.3.1.1.1 DomainEventChannels

Description

The DomainEventChannels interface, as shown in an association in Figure 8-41, provides radio domain event channel registration capabilities. The interface provides the capabilities of adding and removing connections to event channels in a radio domain.

Operations

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

  The registerWithEventChannel operation shall connect a consumer (registeringObject input parameter) to a domain's event channel as indicated by the input eventChannelName parameter.

  The registerWithEventChannel operation shall raise the InvalidObjectReference exception when the input registeringObject parameter contains an invalid reference to an event channel consumer type interface. The registerWithEventChannel operation shall raise the InvalidEventChannelName exception when the input eventChannelName parameter contains an invalid event channel name. The registerWithEventChannel operation shall raise the AlreadyConnected exception when the input eventChannelName parameter references an event channel that is connected to the consumer identified by the input registeringId parameter.
- `unregisterFromEventChannel(in unregisteringId: String, in eventChannelName: String): { raises = (InvalidEventChannelName, NotConnected)}`

  The unregisterFromEventChannel operation shall disconnect a consumer (unregisteringId input parameter) from a domain's event channel as indicated by the eventChannelName parameter.

  The unregisterFromEventChannel operation shall raise the InvalidEventChannelName exception when the input eventChannelName parameter contains an invalid reference to an event channel. The unregisterFromEventChannel operation shall raise the NotConnected exception when the input parameter unregisteringId parameter is not connected to the 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.

8.3.3.1.1.2 DomainInstallation

Description

The DomainInstallation interface, as shown in an association in Figure 8-41, defines radio domain application installation capabilities. The interface provides the capabilities of adding and removing applications from a radio domain.

Operations

Note – Issue 8839 - installApplication () - Added requirement for signalling duplicate installation requests as an error by raising the ApplicationInstallationError exception.

- `installApplication(in profileFileName: String): {raises = ( InvalidProfile, InvalidFileName, ApplicationInstallationError)}`

  The installApplication operation is used to install new application artifacts and descriptors in the domain. A SWRadioDeployment::Application Deployment::ApplicationFactory is created in the domain as a result of successful installation. The profileFileName is the absolute path of the profile filename. The installApplication operation shall verify that the Application's descriptor file exists in the domain and that all the files the Application depends on are also resident.

  The installApplication operation shall raise the ApplicationInstallationError exception when the installation of the Application file(s) is not successfully completed. The installApplication operation shall raise the ApplicationInstallationError exception when the to-be-installed application’s identifier (specified in the application’s descriptor referenced by the profileFileName input parameter) is the same as a previously registered application. The installApplication operation shall raise the InvalidFileName exception when the input file or any referenced file name does not exist in the file system as defined in the absolute path of the input profileFileName. The installApplication operation shall raise the InvalidProfile exception when the input file or any referenced descriptor file is not compliant with its descriptor definition.

Note – Issue 8837 - uninstallApplication () - Removed requirement for removing all files associated with the installed application.

- `uninstallApplication(in applicationId: String): {raises = (InvalidIdentifier, ApplicationUninstallationError)}`

  The uninstallApplication operation is used to uninstall an application from the domain. The applicationId corresponds to the identifier in the SWRadioDeployment::Application Deployment::ApplicationFactory. The uninstallApplication operation shall remove the ApplicationFactory from the domain.

The uninstallApplication operation shall raise the InvalidIdentifier exception when the applicationId is invalid. The uninstallApplication operation shall raise the ApplicationUninstallationError exception when an internal error causes an unsuccessful uninstallation of the Application.

Types and Exceptions

Note – Issue 8839 - Editorial change to ApplicationInstallationError exception synopsis by adding missing exception attributes.

- `<<exception>>ApplicationInstallationError ( ErrorNumberType: errorNumber, msg: String )`

  The ApplicationInstallationError exception, a type of System Exception, is raised when an application installation has not completed correctly. The error number indicates the type of error (e.g., CF_EINVAL, CF_ENAMETOOLONG, CF_ENOENT, CF_ENOMEM, CF_ENOSPC, CF_ENOTDIR, CF_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.
- `<<exception>> InvalidIdentifier`

  The InvalidIdentifier exception indicates an application identifier is invalid.

Semantics

Note – Issue 8837 - Clarified that creation/removal of installation files to/from the domain occurs separately from and before/after the installApplication/uninstallApplication operations.

An installer service typically invokes these operations when adding or removing an ApplicationFactory (installed Application) after creating or prior to deleting the associated files.

8.3.3.1.1.3 DeviceManagerRegistration

Description

The DeviceManagerRegistration interface, as shown in an association in Figure 8-41, defines radio domain service registration capabilities. The interface provides the capabilities of adding and removing DeviceManager’s services from a radio domain.

Operations

Note – Issue 8840 - Clarification of registerDeviceManager connection behavior - any entity can satisfy a domain registration request, pending connection is established until required service is registered with the domain, services from a different DeviceManager may satisfy connection requirements from another DeviceManager.

- `registerDeviceManager(in  deviceMgr: DeviceManager):  {raises = (InvalidObjectReference, InvalidProfile, RegisterError )}`

  The registerDeviceManager operation shall add the input deviceMgr to the domain, if it does not already exist. The registerDeviceManager operation shall add each deviceMgr's Service attributes (e.g., identifier, name/label, characteristic and capacity properties, etc.) to the domain as specified in the deviceMgr's registeredServices attribute. The registerDeviceManager operation shall establish any connections specified in the DeviceManager's descriptor that are possible with the current set of domain registered services--others are left unconnected pending future service registrations. The registerDeviceManager operation shall establish any pending connections from previously registered DeviceManagers if any of the newly registered services complete these connections.

For connections established for an EventService's EventChannel, the registerDeviceManager operation shall connect to the event channel as specified in the deviceMgr's descriptor. If the EventChannel does not exist, the registerDeviceManager operation shall create the EventChannel.

The registerDeviceManager operation may obtain all the Descriptor(s) from the registering DeviceManager's FileSystem.

The registerDeviceManager operation shall mount the deviceMgr's FileSystem to the domain. The mounted FileSystem name shall have the format, "/DomainName/HostName", where DomainName is the name of the domain and HostName is the input deviceMgr's label attribute.

The registerDeviceManager operation shall raise the InvalidObjectReference exception when the input parameter deviceMgr contains an invalid reference to a DeviceManager interface.

The registerDeviceManager operation shall raise the RegisterError exception when an internal error exists which causes an unsuccessful registration.

Note – Issue 8840 - Clarification of unregisterDeviceManager disconnection behavior.

● `unregisterDeviceManager(in  deviceMgr: DeviceManager):  {raises  = (InvalidObjectReference, UnregisterError) }`

The unregisterDeviceManager operation is used to unregister a DeviceManager component from domain. The unregisterDeviceManager operation shall unregister input deviceMgr component and its Service(s) from the domain. The unregisterDeviceManager operation shall disconnect the established connections (including those made to an EventService's Event Channel) involving the unregistering DeviceManager. Any such broken connections to components remaining from other DeviceManager's DCD files shall be considered as "pending" future connections. The unregisterDeviceManager operation may destroy the EventService EventChannel when no more consumers and producers are connected to it.

The unregisterDeviceManager operation shall unmount the deviceMgr's FileSystem from the domain.

The unregisterDeviceManager operation shall raise the InvalidObjectReference when the input parameter deviceMgr contains an invalid reference to a DeviceManager interface. The unregisterDeviceManager operation shall raise the UnregisterError exception when an internal error exists which causes an unsuccessful unregistration.

Note – Issue 8839 - Changed registeringService parameter type from Service to ServiceComponent.

Note – Issue 8840 - The registering service may satisfy pending connection requests from other DeviceManagers.

- `registerService(in registeringService: ServiceComponent, in registeredDeviceMgr: DeviceManager, in name: String): {raises = (InvalidObjectReference, DeviceManagerNotRegistered, RegisterError) }`

  The registerService operation registers a DeviceManager's Service to a domain. The registerService operation shall add the registeringService and the registeringService's attributes (e.g., identifier, name/label, descriptor's characteristic and capacity properties, etc.) to the domain, if the registeringService does not already exist and registeredDeviceManager exists in the domain.

  The registerService operation shall establish any required port connections pending from any previously registered DeviceManagers that are satisfied by the newly registered service, using the IPortConnector and IPortSupplier interfaces.

  The registerService operation shall raise the InvalidProfile exception when registeringService's descriptor is invalid. The registerService operation shall raise a DeviceManagerNotRegistered exception when the input registeredDeviceMger is not registered with the domain. The registerService operation shall raise the InvalidObjectReference exception when input parameters registeringService or registeredServiceMgr contain an invalid reference. The registerService operation shall raise the RegisterError exception when an internal error exists which causes an unsuccessful registration.

Note – Issue 8840 - Clarification of disconnect behavior for unregisterService () - broken connections during the service unregistration process are recognized as future pending connections.

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

  The unregisterService operation shall remove a Service entry from the domain. The unregisterService operation shall disconnect the unregisteringService's port connections. Such connections to the remaining components shall then be considered as “pending” future connections. The unregisterService operation may destroy the EventService's EventChannel when no more consumers and producers are connected to it.

  The unregisterService operation shall raise the InvalidObjectReference exception when the input parameter contains an invalid reference to a Service interface. The unregisterService operation shall raise the UnregisterError exception when an internal error exists which causes an unsuccessful unregistration.

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 service registration 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.

- `<<exception>>UnregisterError`

  The UnregisterError exception, a type of SystemException, indicates that an internal error has occurred to prevent domain unregistration operations from successful completion.

Semantics

Note – Issue 8839 - Added semantics for duplicate DeviceManager and ServiceComponent registeration requests. Changed references to Service(s) with ServiceComponent(s).

The DeviceManagerRegistration interface provides the mechanisms for components, such as DeviceManagers, to register their ServiceComponent(s) (e.g., ManagedServiceComponent or DeviceComponent) for a specific domain. As ServiceComponent(s) are removed from the environment, the interface provides the capability of removing them from the domain. Setting up connections for a registeringService is usually done for DeviceComponents that need an EventChannel, LogService, etc. As ServiceComponent(s) are made available to a domain, they become available for the domain's Application(s) deployment usage.

The behavior for duplicated registering DeviceManager or ServiceComponent may result in a RegisterError exception being thrown or the duplicated registration may be ignored. The duplicated registration behavior is left up to PSMs, PIMs, or profiles that extend the SWRadio profile.

8.3.3.1.2 DomainManager

Description

The DomainManager interface as shown in Figure 8-41 describes the definition and relationships that are common for all SWRadio domain managers. The DomainManager provides the capabilities of retrieving a domain's components and profile.

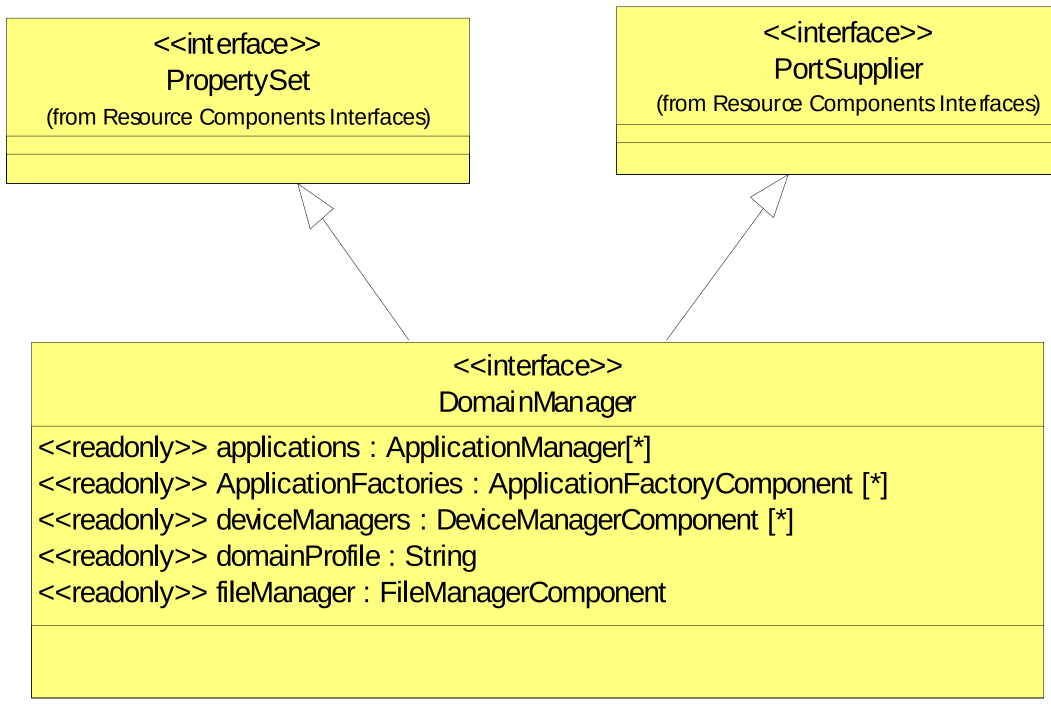

<<interface>>
PropertySet
(from Resource Components Interfaces)
<<interface>>
PortSupplier
(from Resource Components Interfaces)
<<interface>>
DomainManager
<<readonly>> applications : ApplicationManager[*]
<<readonly>> ApplicationFactories : ApplicationFactoryComponent [*]
<<readonly>> deviceManagers : DeviceManagerComponent [*]
<<readonly>> domainProfile : String
<<readonly>> fileManager : FileManagerComponent

Figure 8-40 – DomainManager Definition

Note – Issue 7905 Resolution (Above Diagram - Removal of portExists () from DomainManager).

Attributes

- `<<readonly>>applications : ApplicationManager [*]`
  The readonly applications attribute contains a sequence of instantiated applications in the domain. The applications attribute shall contain the list of SWRadioDeployment::Application Deployment::Application Deployment Stereotypes::ApplicationManager(s) that have been instantiated (e.g., created by ApplicationFactory) within the domain.
- `<<readonly>>applicationFactories : ApplicationFactoryComponent [*]`
  The readonly applicationFactories attribute contains a list of ApplicationFactories in the domain. The applicationFactories attribute shall contain a list of one SWRadioDeployment::Application Deployment::Application Deployment

Stereotypes::ApplicationFactory per Application successfully installed (i.e. no exception raised by the DomainInstallation::install).

- `<<readonly>>deviceManagers : DeviceManagerComponent [*]`

  The readonly deviceManagers attribute contains a sequence of DeviceManagerComponents in the domain. The deviceManagers attribute shall contain a list of registered DeviceManagerComponents that have registered with the DomainManagerComponent.
- `<<readonly>>domainManagerProfile : String`

  The readonly domainManagerProfile attribute contains a file reference to the domain configuration descriptor. The descriptor provides configuration information for a domain. Files referenced within the descriptor will have to be obtained from the domain's fileMgr attribute.
- `<<readonly>>fileMgr : FileManagerComponent`

  The readonly fileMgr attribute contains the DomainManager's RadioServices::File Services::File Services Stereotypes::FileManagerComponent.

Note – Issue 7905 Resolution - Deleted the Operations section due to removal of the portExists operation.

8.3.3.1.3 RadioSet Management Stereotypes

Note – Issue 7742 renamed Associations noheader to M1 Associations throughout. Renamed DomainManager to DomainManagerComponent. Updated RadioSet Management to a M1 Illustration.

This section defines the stereotypes for RadioSet management. The RadioSet management stereotypes depicted in Table 8-10 and Figure 8-41 are extensions of the UML 2.0 Component (UML2.0::Components::BasicComponents).

Table 8-10 – DomainManagement Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
CommChannel	Component	N/A			Represents a component that manages a LogicalCommunicationChannel.
DomainManagerComponent	Component	SWRadioComponent			Provides Domain Retrieval capability and represents a component that manages a domain.
RadioManager	Component	DomainManagerComponent			Represents a component that manages a RadioSet.
RadioSystemManager	Component	N/A			Represents a component that manages RadioManagers.

8.3.3 Radio Management

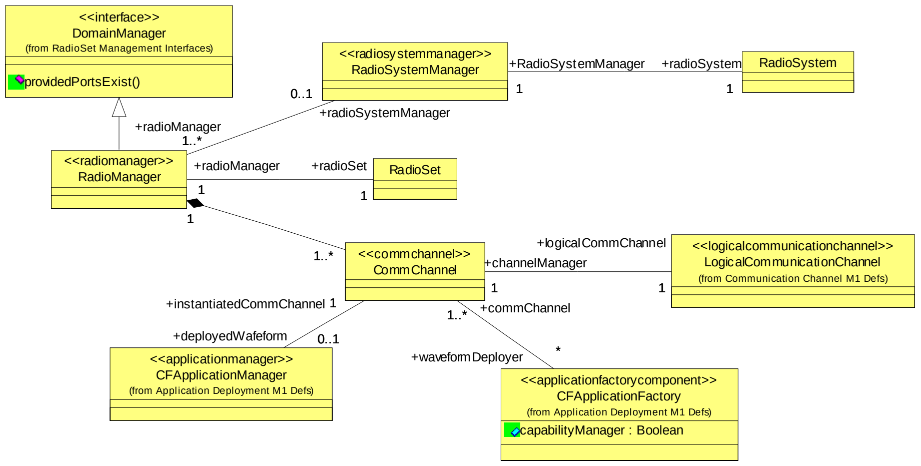


Figure 8-41 – RadioSet Management M1 Illustration

8.3.3.1.3.1 CommChannel

Description

The CommChannel, as shown in an association in Figure 8-41 represents a component that provides communication channel management.

M1 Associations

- `deployedWaveform: ApplicationManager [0..1]`
  The LogicalCommunicationChannel represents the set of devices that provide the communication path for the communication channel.
- `logicalCommunicationChannel: LogicalCommunicationChannel [1]`
  The LogicalCommunicationChannel represents the set of devices that provide the communication path for the communication channel.
- `waveformDeployer:ApplicationFactoryComponent[*]`
  The ApplicationFactory that deploys the waveform onto the communication channel.

Semantics

The CommChannel may be associated with static or a dynamic LogicalCommunicationChannel. The devices associated with a static LogicalCommunicationChannel do not vary over the life cycle of the communication channel. For dynamic LogicalCommunicationChannel the devices can vary during the life cycle of the communication channel.

8.3.3.1.3.2 DomainManagerComponent

Description

Note – Issue 7742 updated Figure to a M1 Illustration, removed attributes and operations, changed DomainManager to DomainManagerComponent.

The DomainManagerComponent as shown in Table 8-10 and Figure 8-42 describes the definition and relationships that are common for all SWRadio domain managers. The DomainManagerComponent provides the capabilities of retrieving a domain's components and profile.

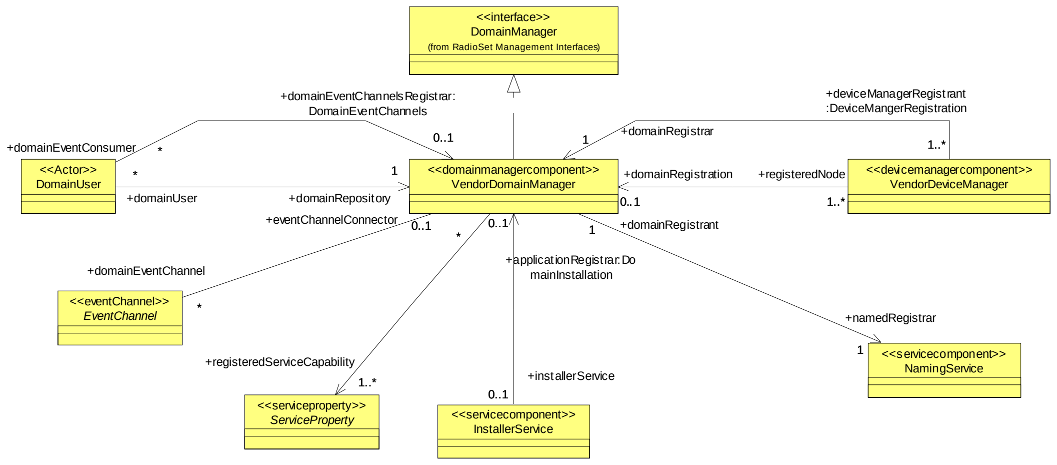


Figure 8-42 – DomainManagerComponent M1 Illustration

Note – Issue 7578 and 7579 Resolution (Above Diagram) - Note to Tansu from Neli: These issues do not affect the above diagram. The diagram change is for issue 7742?

M1 Associations

- `domainEventChannel: EventChannel [*]`
  A DomainManagerComponent may be associated with many event channels. Refer to Domain Event Channels section below for description of domain event channels and event types.
- `namedRegistrar: NamingService [1]`
  The NamingService that contains a named DomainManagerComponent reference.
- `registeredServiceCapability: ServiceProperty [1..*]`
  The registedServiceCapability(s) are the set of ServiceComponent's ServiceProperty(s) registered and/or known by the DomainManagerComponent.

Constraints

The identifier attribute shall contain a unique identifier for a DomainManagerComponent. The identifier shall be identical to the id attribute of the DomainManagerComponent's configuration descriptor as specified by the domainProfile attribute.

During component construction the DomainManagerComponent shall register itself with the NamingService. During NamingService registration the DomainManagerComponent shall create a “naming context” using “/DomainName” as its name.ID component and "" (Null string) as its name.kind component, then create a “name binding” to the “/DomainName” naming context using “/DomainManager” as its name.ID component, "" (Null string) as its name.kind component, and the DomainManager's object reference.

The DomainManagerComponent shall create its own FileManager component that consists of all registered DeviceManager's FileSystems.

The DomainManagerComponent shall restore ApplicationFactories after startup for Application(s) that were previously installed by the DomainManagerComponent installApplication operation. The DomainManagerComponent shall add the restored ApplicationFactories to the DomainManagerComponent's applicationFactories attribute.

If the DomainManagerComponent has the association role of a domainEventChannelsRegistrar then the DomainManagerComponent shall provide a port that provides the IDomainEventChannels service and the port name is DomainEventChannelsPort, and support the DomainOutgoingEventChannel and DomainIncomingEventChannel..

If the DomainManagerComponent has the association role of a domainRegistration then the DomainManagerComponent shall provide a port that provides the DeviceManagerRegistration service and the port name is DomainDeviceMgrRegPort.

If the DomainManagerComponent has the association role of a applicationRegistrar then the DomainManagerComponent shall provide a port that provides the ApplicationInstallation service and the port name is DomainInstallationPort.

If the DomainManagerComponent has a LogService required port then the DomainManagerComponent shall provide a port that provides LogService admin service and the port name is DomainLogAdminPort.

If the DomainManagerComponent has a LogService required port then the DomainManagerComponent shall provide a port that provides LogService consumer service and the port name is DomainLogConsumerPort.

Each registered ServiceProperty with the same identification shall be the same type within DomainManagerComponent.

Semantics

Note – Issue 7719 - wrong semantics description

The set of interfaces realized by a DomainManagerComponent depends on the system the DomainManagerComponent is built for. As shown in Figure 8-42 above, the DomainManagerComponent could realize PortSupplier and PropertySet interfaces and inherits from SWRadioComponent. The types of ServiceArtifactProperty(s) supported by a DomainManagerComponent are implementation specific. A DomainManagerComponent implementation may constraint the types of ServiceArtifactProperty(s) which would be reflected in its registration behavior.

Note – Issue 7905 Resolution - Deleted references to the portExists operation usage due to removal of this operation.

The DomainManagerComponent may upon successful add of an component to its applications, applicationFactories, or deviceManagers attribute or registered Service send an event to the Outgoing Domain ManagementEventChannel with event data consisting of a DomainManagementObjectAddedEventType. The DomainManagementObjectAddedEventType event data shall be populated as follows when issued:

1. The producerId is the identifier attribute of the DomainManagerComponent.
2. The sourceId is the identifier attribute of the created (ApplicationManager), installed (ApplicationFactory), or registered (DeviceManager or Service) component to the domain.
3. The sourceName is the name attribute of the added component to the domain.
4. The sourceHandle is the component reference added to the domain.
5. The sourceCategory is the type of component added to the domain.

The DomainManagerComponent may upon successful removal of a component from its applications, applicationFactories, or deviceManagers attribute or unregistered Service send an event to the Outgoing Domain ManagementEventChannel with event data consisting of a DomainManagementObjectRemovedEventType. The DomainManagementObjectRemovedEventType event data shall be populated as follows when issued:

1. The producerId is the identifier attribute of the DomainManagerComponent.
2. The sourceId is the identifier attribute of the component uninstalled (ApplicationFactory), released (ApplicationManager), or unregistered (DeviceManager or Service).
3. The sourceName is the name attribute of the removed component from the domain.
4. The sourceCategory is the type of component removed from the domain.

The DomainManagerComponent shall produce DomainManagementObjectRemovedEventType and DomainManagementObjectAddedEventType when the DomainManagerComponent has the association role of a domainEventChannelsRegistrar.

8.3.3.1.3.3 RadioManager

Description

Note – Issue 7720 - Irrelevant description

The RadioManager component, as shown in Figure 8-41, describes the definition and relationships that are common for RadioSet manager. The RadioSystemManager is responsible for control and management tasks for the RadioSystem. It may be associated with one or more RadioManagers which are used to control the RadioSets the RadioSystem consists of.

M1 Associations

- `radioSet: RadioSet [1]`

  A RadioManager is associated with one RadioSet

8.3.3.1.3.4 RadioSystemManager

Description

The RadioSystemManager component, as shown in Figure 8-41, describes the definition and relationships that are common for RadioSystem managers. The RadioManager extends the DomainManagerComponent by providing communication channel management within the RadioSet.

M1 Associations

- `radioManager: RadioManager[1..*]`
  The associated RadioManager provides the capability to manage a RadioSet within a RadioSystem.
- `radioSystem: RadioSytem [1]`
  The RadioSystem provides a set of communication equipment that their relationships.

8.3.3.2 Device Management

Note – Issue 774 added two subsections, Device Management Interfaces and Device Management Stereotypes. Moved the interfaces to Device Management Interfaces section.

This section defines the stereotypes and interfaces for radio device management, whcich are described in the two subsections below: 8.3.3.2.1 and .8.3.3.2.2.

8.3.3.2.1 Device Management Interfaces

This section defines the interfaces for radio device management. The types of capabilities offered by radio device management are categorized as follows:

1. Service Registration Management - provides the mechanism for registering and unregistering services within a device.
2. DeviceManager - provides the mechanism for retrieving SWRadio's node services and information.

8.3.3.2.1.1 DeviceManager

Description

The DeviceManager interface as shown in Figure 8-44 defines the attributes and operations relationships that are common for all SWRadio node managers. A DeviceManager interface is used to manage the ServiceComponent(s) on a node and to retrieve information about a node or device manager.

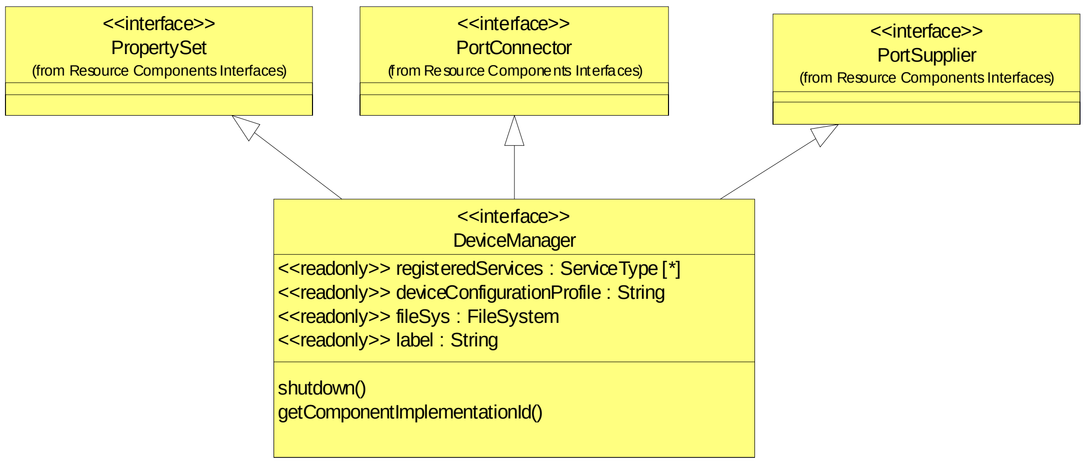

<<interface>>
PropertySet
(from Resource Components Interfaces)
<<interface>>
PortConnector
(from Resource Components Interfaces)
<<interface>>
PortSupplier
(from Resource Components Interfaces)
<<interface>>
DeviceManager
<<readonly>> registeredServices : ServiceType [*]
<<readonly>> deviceConfigurationProfile : String
<<readonly>> fileSys : FileSystem
<<readonly>> label : String
shutdown()
getComponentImplementationId()

Figure 8-43 – DeviceManager Definition

Note – Issue 7905 Resolution (Above Diagram - Removal of portExists () from DeviceManager).

Attributes

- `<<readonly>>label: String`
  The readonly label attribute contains a node's meaningful name.
- `<<readonly>>fileSys: FileSystemComponent`
  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>>deviceConfigurationProfile : String`
  The readonly deviceConfigurationProfile 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 a node or a sequence length of zero if no Services have registered with the node.

Operations

- `getComponentImplementationId (in componentInstantiationId: String, return String):`
  The getComponentImplementationId operation shall return the implementation ID used to create the component identified by the input componentInstantiatio-

nId parameter. The implementation ID corresponds to the ServiceExecutableCode's implementation id in a component implementation descriptor that was used to manifest a ServiceComponent. The getComponentImplementationId operation shall return an empty string when the input componentInstantiationId parameter does not match a ServiceExecutableCode's implementation ID deployed by the DeviceManager. This operation does not raise any exceptions.

Note – Issue 7905 Resolution - Deteled the portExists operation.

● `shutdown ()`

The shutdown operation provides the mechanism to terminate a DeviceManager. The shutdown operation shall unregister the DeviceManager from the DomainManagerComponent.

The shutdown operation shall release all of the DeviceManager's registered ServiceComponent(s) (DeviceManager's registeredServices attribute) that support the Ilifecycle interface and started up by the DeviceManager.

The shutdown operation shall terminate the execution of each ExecutableCode that was created as specified in the deviceConfigurationProfile attribute. For a released (releaseObject operation) ServiceComponent, the termination shall take place after the ServiceComponent has unregistered with the DeviceManager.

The shutdown operation shall remove the DeviceManager from the environment when all of the released registered Services are unregistered from the DeviceManager. This operation does not return any value and does not raise any exceptions.

Types and Exceptions

● `ServiceType( serviceObject: ServiceComponent, serviceName: String)`

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

8.3.3.2.1.2 ServiceRegistration

Description

The ServiceRegistration interface defines SWRadio node service registration capabilities. The interface provides the capabilities of adding and removing ServiceComponent(s) from a SWRadio node.

Operations

● `registerService(in  registeringService: ServiceComponent, in name: String): {raises = (InvalidObjectReference) }`

The registerService operation provides the mechanism to register a ServiceComponent with a node. The registeringService is ignored when duplicated. The registerService operation shall raise the InvalidObjectReference exception when the input registeringService is a nil component reference.

● `unregisterService(in unregisteringService: ServiceComponent, in name: String): {raises = ( InvalidObjectReference)}`

The unregisterService operation provides the mechanism to unregister a ServiceComponent from a node. The unregisterService operation shall raise the InvalidObjectReference when the input registeredService is a nil component reference or does not exist in the node.

Semantics

The ServiceRegistration interface provides the mechanisms for ServiceComponent(s) started up on a node to register to a node or device manager that is managing a node. As ServiceComponent(s) are removed from node environment, the interface provides the capability of removing them from a node or device manager. Services managed by a node manager can also include managed services such as DeviceComponent(s).

8.3.3.2.2 Device Management Stereotypes

This section defines the stereotypes for radio device management. The device management stereotypes are depicted in Table 8-11 below, which are extensions of UML 2.0 Component (UML2.0::Components::BasicComponents). The details of each classifier are described in the following subsections.

Note – Issue 7742 added component to the DeviceManager name, added contraints for the interfaces realized by a DeviceManagerComponent, updated figure to be a M1 Illustration, Changed Assoication noheader to be M1 Associations.

Table 8-11 – Node Management Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
DeviceManagerComponent	Component	SWRadioComponent			Manages a node and its services.

8.3.3.2.2.1 DeviceManagerComponent

Description

The DeviceManagerComponent a type of SWRadioComponent as shown in Table 8-11. Figure 8-44 denotes the relationships that are common for all SWRadio node managers. A DeviceManagerComponent manages the ServiceComponent(s) on a node.

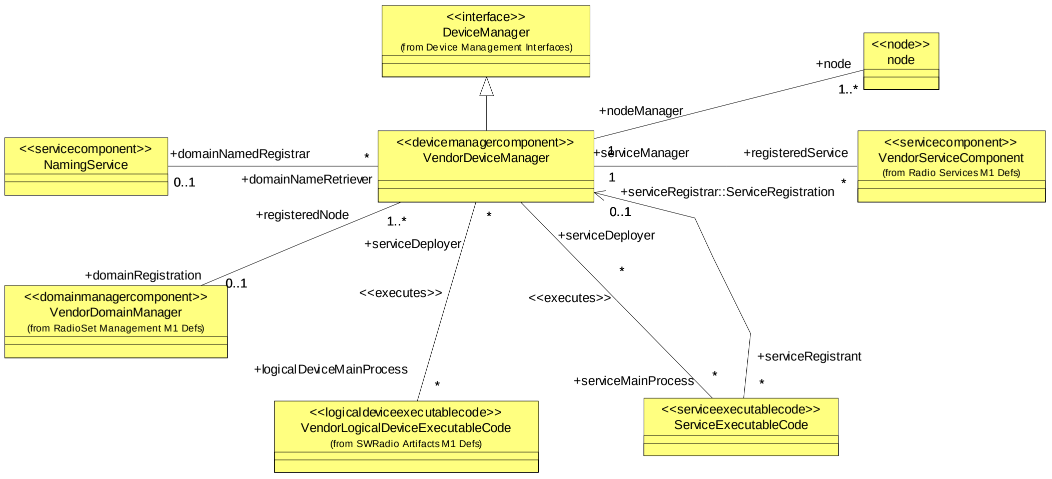


Figure 8-44 – DeviceManager M1 Illustration

Note – Issue 7578 and 7579 Resolution (diagram above) - Note to Tansu: These issues do not affect the above diagram. The diagram change is for issue 7742?

M1 Associations

- `node: Node[1..*]`

  One to many nodes can be managed by a DeviceManagerComponent.

- `domainNamedRegistrar: NamingService [0..1]`

  A NamingService contains the named DomainManagerComponent's component reference that is obtained by a DeviceManager that needs to register to a DomainManagerComponent.

- `domainNodeRegistrar: DomainManagerComponent [0..1]`

  A DomainManagerComponent harnesses all nodes' services and capabilities within the domain.

- `logicalDeviceMainProcess: LogicalDeviceExecutableCode [*]`

  A logical device main process that manifests a DeviceComponent can be executed by a DeviceManagerComponent.

- `registeredService: ServiceComponent[1..*]`

  A set of ServiceComponent(s) (e.g., DeviceComponents, etc.) registered to a DeviceManagerComponent.

- `serviceMainProcess: ServiceExecutableCode [*]`
  A service component's main process that manifests a ServiceComponent can be executed by a DeviceManagerComponent.

Constraints

The registeredServices attribute shall contain a list of registered Radio Services::ServiceComponent(s) that have registered with the DeviceManagerComponent. Each registeredservice in the registeredServices attribute shall be registered with the DeviceManagerComponent's associated DomainManagerComponent when the DeviceManagerComponent registers with theDomainManagerComponent.

Each unregistered ServiceComponent shall be unregistered from the DeviceManagerComponent's associated DomainManagerComponent when the unregistered ServiceComponent is registered with the DeviceManagerComponent and the DeviceManagerComponent is not shutting down. If a SWRadio Deployment::SWRadio Artifacts::ServiceExecutableCode was started up by the DeviceManagerComponent as specified in the deviceConfiguration attribute, then the DeviceManagerComponent shall terminate the ServiceExecutableCode and deallocate capacity to the device the ServiceExecutableCode was deployed on.

If the DeviceManagerComponent has the association role of a serviceRegistrar, then the DeviceManagerComponent shall provide a port that provides the ServiceRegistration service and the port name is ServiceRegistration.

The DeviceManagerComponent shall realize the DeviceManager interface.

Semantics

The DeviceManagerComponent provides the capability of starting up services' main processes on a given node by the deviceConfigurationDescriptor attribute. Services started up also include DeviceComponent(s) that are a type of ManagedServiceComponent. A DeviceManagerComponent registers to a DomainManagerComponent using the deviceConfigurationProfile descriptor information.

The DeviceManagerComponent upon start up shall register itself with a DomainManagerComponent as specified in the deviceConfigurationProfile attribute. A DeviceManagerComponent shall use its deviceConfigurationProfile attribute for determining:

1. Services to be deployed for this DeviceManagerComponent (for example, LogService, DeviceComponent),
2. DeviceComponents to be created for this DeviceManagerComponent,
3. Services to be deployed on (executing on) another Device,

Note – Issue 8842

---

4. DeviceComponents to be part of another DeviceComponent's composition definition,
5. Mount point names for File Systems,
6. The DeviceManagerComponent's identifier attribute value, and
7. The DeviceManagerComponent's label attribute value.

The DeviceManagerComponent shall create file system components implementing the FileSystem interface. If multiple FileSystems are to be created, the DeviceManagerComponent shall mount created FileSystemComponents to a FileManagerComponent (widened to a FileSystem through the FileSys attribute). Each mounted FileSystemComponent name shall be unique within the DeviceManagerComponent.

If the DeviceManagerComponent deploys a ServiceComponent, the DeviceManagerComponent shall supply execute operation parameters as stated for SWRadio Deployment::SWRadio Artifacts::ServiceExecutableCode. The Service Name executable parameter shall be Service's usagename element as specified in the deviceManager's DCD. The DeviceManagerComponent shall use an SWRadio Deployment::SWRadio Artifacts::ExecutableCode's user-defined executable parameters as specified in the component's implementation descriptor. The DeviceManagerComponent shall use an ExecutableCode's stacksize, priority, runtime executable options when specified in the component's implementation descriptor. The DeviceManagerComponent shall allocate the ExecutableCode's capacity requirements against the device the ExecutableCode is deployed on.

If the DeviceManagerComponent deploys a DeviceComponent, the DeviceManagerComponent shall supply execute operation parameters as stated for SWRadio Deployment::SWRadio Artifacts::LogicalDeviceExecutableCode. The following LogicalDeviceExecutableCode's execute operation parameters values shall be:

- The Device Identifier executable parameter shall be Service's id element as specified in the DeviceManager's DCD.
- The Profile Name executable parameter shall be Service's component implementation descriptor as specified in the DeviceManagerComponent's DCD. The file name shall be the DeviceManagerComponent’s full mounted file system file path name.
- Composite Device Component Reference executable parameter shall be a registered DeviceComponent's compositeDevice attribute that corresponds to the composite part relationship as specified in the DeviceManagerComponent's DCD. This parameter is only used when the composite part relationship is specified in the DeviceManagerComponent's DCD.

Note – Issue 8858 Resolution

Note – Issue 8873 Resolution - Replacement of non-existent ConfigureQuertyProperty type with the ConfigureProperty type

The DeviceManagerComponent shall initialize and configure ServiceComponents that are started by the DeviceManagerComponent after they have registered with the DeviceManagerComponent provided they realize the Lifecycle and PropertySet interfaces. The DeviceManagerComponent shall configure a registered Service, provided the registered Service has ConfigureProperty(s).

A ServiceComponent’s configuration property values shall only come from the deviceConfigurationProfile descriptor, not from the component’s implementation or component definition descriptors.

Note – Issue 7905 Resolution - Removed references to the portExists operation usage due to removal of this operation.

8.3.3.3 Domain Event Channels

For radio and domain management, a domain may support a number of event channels. Two event channels that a domain may support are:

- IncomingDomainEventChannel - This event channel receives events from the domain's registered components.
- OutgoingDomainEventChannel - This event channel sends events from the domain out to registered domain event consumers.

The types of domain events that could be issued are depicted in the Figure 8-45 and described in the Types and Exceptions section below.

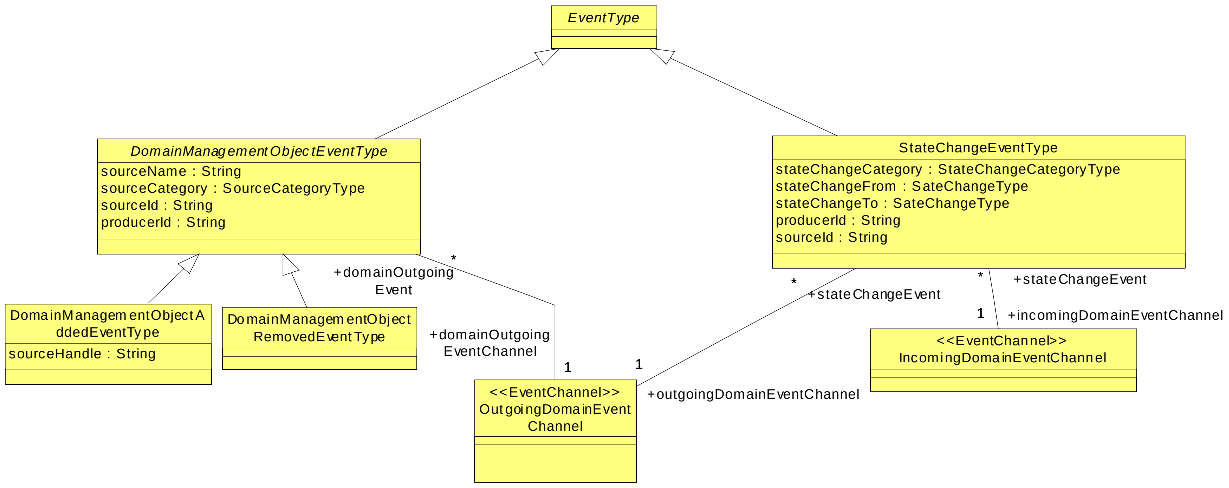

EventType
DomainManagementObjectEventType
sourceName : String
sourceCategory : SourceCategoryType
sourceId : String
producerId : String
StateChangeEventType
stateChangeCategory : StateChangeCategoryType
stateChangeFrom : SateChangeType
stateChangeTo : SateChangeType
producerId : String
sourceId : String
DomainManagementObjectAddedEventType
sourceHandle : String
DomainManagementObject RemovedEventType
+domainOutgoing Event
+domainOutgoing EventChannel
<<EventChannel>> OutgoingDomainEvent Channel
+stateChangeEvent
+outgoingDomainEventChannel
+incomingDomainEventChannel
<<EventChannel>> IncomingDomainEventChannel
*
1

Figure 8-45 – Domain Events Overview

Types and Exceptions

- `DomainManagementObjectAddedEventType( sourceHandle: String)`

  The DomainManagementObejectAddedEventType is a specialization of DomainManagementObjectEventType. This event type indicates a component has been added to the domain. The sourceHandle attribute indicates the component reference of the component added to the domain.
- `DomainManagementObjectEventType ( sourceName: String, sourceCategory: SourceCategoryType, sourceId: String, producerId: String)`

  The DomainManagementObjectRemovedEventType is a specialization of event, which contains information about the event that occurred in the domain.

  - SourceName attribute is the name of source that caused the event.
  - SourceCategory attribute indicates the type of component that caused the event.
  - SourceID is the identifier of the source that caused the event.
  - ProducerId is the identifier of the producer of the event.
- `DomainManagementObjectRemovedEventType`

  The DomainManagementObjectRemovedEventType is a specialization of DomainManagementObjectEventType. This event type indicates a component has been removed from the domain.
- `<<enumeration>>SourceCategoryType(COMM_CHANNEL,DEVICE_MANAGER,DEVICE,DOMAIN_MANAGER, APPLICATION, APPLICATION_FACTORY, SERVICE))`

  The SoureCategoryType defines the types of components within the domain that can be added or removed. A ManagedService in the domain can also have its state changed.

- `<<enumeration>>StateChangeCategoryType( Administrative_Event_State,Operational_Event_State, Usage_Event_State)`
  The StateChangeCategoryType indicates either an admin, operational, or usage state change.
- `<<enumeration>>StateChangeCategoryType( Administrative_Event_State, Operational_Event_State, Usage_Event_State)`
  The StateChangeCategoryType indicates either an admin, operational, or usage state change.
- `StateChangeEventType( stateChangeCategory:StateChangeCategoryType, stateChangeFrom:StateChangeType,  stateChangeTo: StateChangeType, producerId: String, sourceId: String)`
  The StateChangeEventType indicates either an admin, operational, or usage state change. The stateChangeCategory attribute indicates the type of state change. The sourceId attribute indicates the source component's state change and the producerId indicates the component that produced the event.
- `<<enumeration>>StateChangeType( LOCKED, UNLOCKED, SHUTTING_DOWN, ENABLED, DISABLED, IDLE, BUSY, ACTIVE)`
  The StateChangeType indicates the values for state changes. LOCKED, UNLOCKED, and SHUTTING_DOWN values are associated with admin state changes. ENABLED and DISABLED values are associated with operational state changes. IDLE, BUSY, and ACTIVE values are associated with usage state changes.

8.3.4 SWRadio Deployment

SWRadio deployment describes the SWRadio executable artifacts that are involved in the deployment of applications (e.g, applications), logical devices, and services within a SWRadio environment. This section also describes the components that are involved in the deployment of and management of SWRadio Applications in the Application Deployment subsection.

8.3.4.1 SWRadio Artifacts

This section defines the types and stereotypes for SWRadio artifacts which are described in the following teo ssubsections respectivey, 8.3.4.1.1 and 8.3.4.1.2.

Note – Issue 7742 added two subsections, one for model library types in the profile and the second one for SWRadio artifact stereotypes. Moved the types into the SWRadio Aritfacts Types section. Removed duplicate DeploymentRequirement section.

8.3.4.1.1 SWRadio Artifacts Types

This section defines the types for SWRadio artifacts which are: BasicDeploymentRequirement, DeploymentRequirement, and DeploymentRequirementQualifier that are depicted in Figure 8-49. These types are used to specify a deployment requirement on a device with in a radio set.

8.3.4.1.1.1 BasicDeploymentRequirement

Description

The BasicDeploymentRequirement, as shown in Figure 8-49 on Page 191, specializes the DeploymentRequirement. The BasicDeploymentRequirement is used to define the deployment requirement value for a ServiceProperty.

Attributes

- `value: String`

  The value attribute indicates the deployment requirement needed (service characteristic or capacity).

Constraints

The value attribute shall conform to the string format that is valid for the referenced ServiceProperty type definition (e.g., string, integer, etc.).

8.3.4.1.1.2 DeploymentRequirement

Description

The DeploymentRequirement abstraction, as shown in Figure 8-49 on Page 191, is used to provide the common definition for deployment requirements. The DeploymentRequirement is used to define the deployment requirement value for a ServiceProperty.

Attributes

- `name: String`

  The name attribute indicates the name of the ServiceProperty the deployment requirement is against.

Semantics

The deployment requirement is used to specify the type of service needed and/or the capacity needed from a service. These deployment requirements are evaluated against the registered Service's ServiceProperty(s) within a domain.

8.3.4.1.1.3 DeploymentRequirementQualifier

Description

The DeploymentRequirementQualifier, as shown in Figure 8-49 on Page 191, specializes the DeploymentRequirement. The DeploymentRequirementQualifer is used to define characteristic qualifiers for a deployment requirement such as name and version for complier, operating system, library, runtime, and interface.

Attributes

Note – Issue TBD not CharacterisiticQualifier not defined, definition was removed for properties issue 7984 but still is applicable here. Need to add a types section.

- `qualifier: CharacteristicQualifier [1..*]`
  The qualifier attribute indicates characteristic qualifier for a deployment requirement.

8.3.4.1.2 SWRadio Artifacts Stereotypes

This section defines the stereotypes for SWRadio artifacts. The SWRadio artifacts stereotypes are depicted in Table 8-12 and Figure 8-46 below, which are extensions of the UML Artifact. The details of each artifact are described in the following subsections.

Table 8-12 – SWRadio Artifacts Stereotypes

Stereotype	Base Class	Parent	Tags	Constraints	Description
BITStream	Artifact	LoadableCode			Indicates an artifact that is object code for a prologic device (e.g., Field Programmable Gate Array device).
depends on	Association	N/A			Indicates an association that depicts a coding dependency where the code pointed needs to be loaded first.
Descriptor	Artifact	N/A			Indicates an artifact that is a deployment or component specification that conveys information on the element to be deployed.
executes	Association	N/A			Indicates an association that denotes execution code on a device
Library	Artifact	LoadableCode			Indicates an artifact that is loadable static or dynamic object code.
loads	Association	N/A			Indicates an association that denotes the loading of code on a device
LogicalDeviceExecutableCode	Artifact	ServiceExecutableCode		Minimum Executable parameters supported: Identifier, Software Profile, Composite Device IOR	Indicates an artifact that is an executable operating system main process that manifest a LogicalDevice component.
ExecutableCode	Artifact	LoadableCode	EntryPointName, ProcessPriority, stackSize	Executable parameters conform to argv of the POSIX exec family of functions	Indicates an artifact that is an executable operating system main process or entry point.

Table 8-12 – SWRadio Artifacts Stereotypes

Stereotype	**Base Class**	**Parent**	**Tags**	**Constraints**	**Description**
LoadableCode	Artifact	N/A			Indicates an artifact that defines the base SWRadio artifact definition and relationships for any type of SWRadio artifact.
ResourceExecutableCode	Artifact	ExecutableCode		Minimum Executable parameters supported: identifier, naming context component reference, name binding	Indicates an artifact that is an executable operating system main process that manifests either an SWRadioResource or/and ResourceFactory component.
ServiceExecutableCode	Artifact	ExecutableCode		Minimum Executable parameters supported: DeviceManager Registration Reference, Service Name	Indicates an artifact that is an executable operating system main process that manifests a Service.
terminates	Association	N/A			Indicates an association that denotes termination executable code on a device
unloads	Association	N/A			Indicates an association that denotes the unloading of code from a device

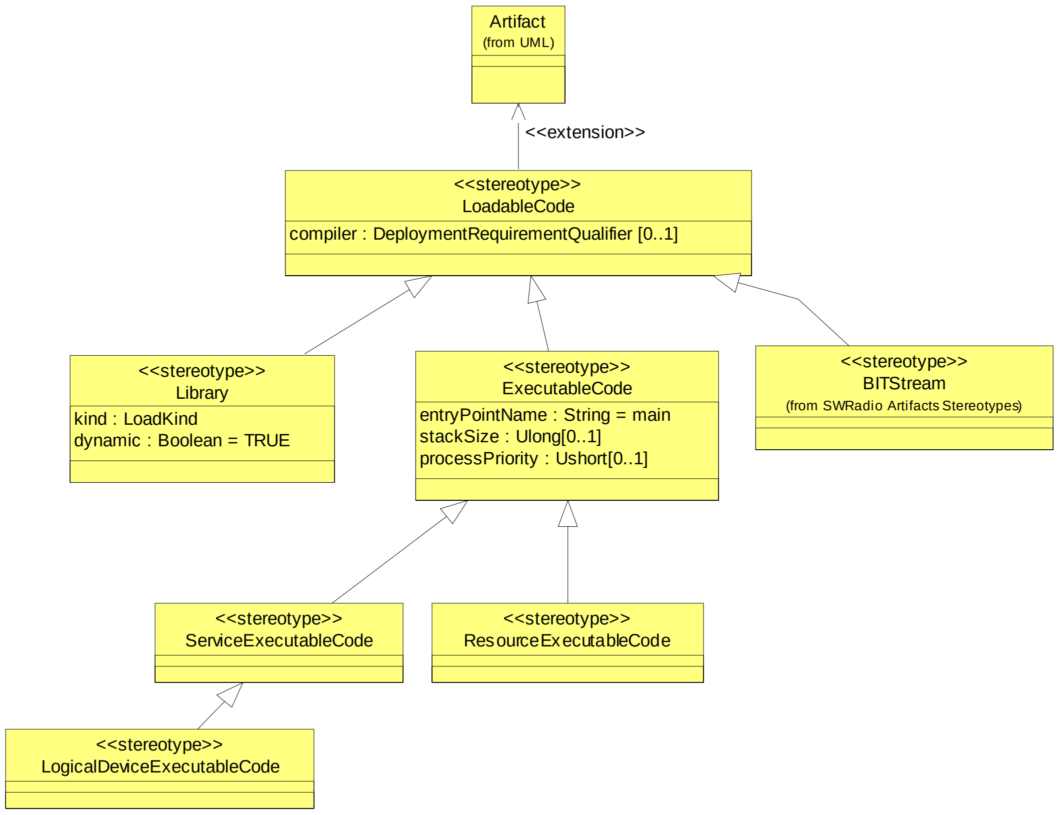


Figure 8-46 – SWRadio Artifacts Relationships

Note – Issue 7742 Updated Figure Above., removed Association

8.3.4.1.2.1 ExecutableCode

Description

The ExecutableCode artifact, as shown in Figure 8-47, defines the definition and relationships that are common for SWRadio main process types.

Note – Issue 7742 updated figure, Made figure at the M1 level. Changed Associations to be M1

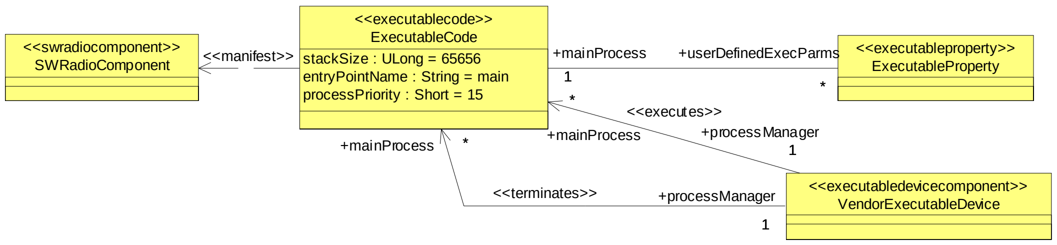


Figure 8-47 – ExecutableCode M1 Illustration

Associations

Attributes

Note – removed Latency attribute since this is part of RadioProperty, Issue 7895 changed types for processPriority and stackSize.

- `entryPointName: String = main`

  The entryPointName attribute indicates the name of the process to be created within the operating system.

- `processPriority: Ushort [0..1]`

  The processPriority attribute indicates the priority of the process to be created within the operating system

- `stackSize: Ulong [0..1]`

  The stackSize attribute indicates the stack size of the process to be created within the operating system

M1 Associations

- `userDefinedExecParms: ExecutableProperty [*]`

  For any main process or entry point, a user can specify executable parameters that are to be passed to the process upon creation.

Constraints

The ExecutableCode entry point's input parameters (id/value string pairs) shall conform to the standard argv of the POSIX exec family of functions, where argv(0) is the function name followed by id/value string pairs.

8.3.4.1.2.2 Library

Description

The Library artifact defines the LoadableCode artifact that is a loadable image.

Attributes

- `kind: LoadKind`
  The kind attribute indicates the type of load to be performed.
- `dynamic: Boolean = TRUE`
  The dynamic attribute indicates if the library to be loaded is dynamic or static. TRUE means dynamic loadable.

Types and Exceptions

- `<<enumeration>> LoadKind (KERNEL_MODULE, DLL, DRIVER, SHARED_LIBRARY, EXECUABLE)`
  The LoadKind defines the type of load to be performed.

8.3.4.1.2.3 LogicalDeviceExecutableCode

Description

The LogicalDeviceExecutableCode artifact, a type of ServiceExecutableCode (see Figure 8-48), defines the operating system main process that manifests a LogicalComponent component, which is specific type of ManagerServiceComponent.

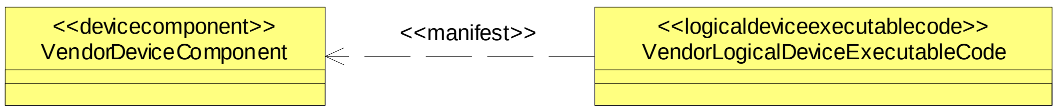

<<devicecomponent>>
VendorDeviceComponent
<<manifest>>
<<logicaldeviceexecutablecode>>
VendorLogicalDeviceExecutableCode

Figure 8-48 – LogicalDeviceExecutableCode M1 Illustration

Note – Issue7742 updated figure above to be at the M1 Level.

Constraints

Note – Issue 8842 (thruout constraints section)

The LogicalDeviceExecutableCode shall accept following executable parameters in addition to the ServiceExecutableCode parameters:

- Profile Name - The ID is “PROFILE_NAME” and the value is a string that is the full mounted file system file path name that is used for DeviceComponent profile attribute.
- Device Identifier - The ID is “DEVICE_ID” and the value is a string that is used for the SWRadioComponent's identifier attribute.
- Composite Device Component Reference - The ID is “Composite_DEVICE_IOR” and the value is a string that is a DeviceCompositionComponent reference.

The LogicalDeviceExecutableCode Service Name executable parameter shall be used for the DeviceComponent's label attribute value.

The LogicalDeviceExecutableCode shall add the DeviceComponent manifested by the process to the device composition using the Composite Device Component Reference executable parameter when specified.

The LogicalDeviceExecutableCode Device Identifier executable parameter shall be used for the DeviceComponent's identifier attribute value.

The LogicalDeviceExecutableCode Profile Name executable parameter shall be used for the DeviceComponent's softwareProfile attribute value.

8.3.4.1.2.4 LoadableCode

Description

The SWRadio LoadableCode artifact, as shown in Figure 8-49, describes the definition and relationships that are common for all SWRadio artifact types.

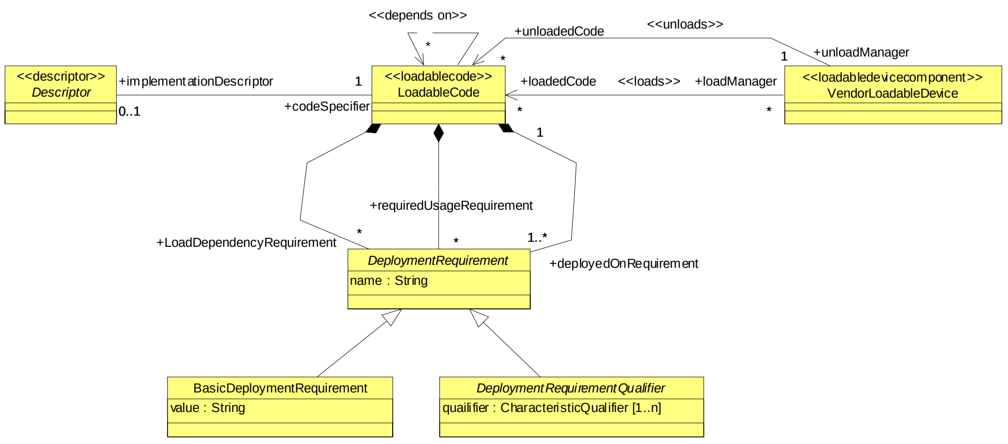


Figure 8-49 – SWRadio LoadableCode M1 Illustration

Note – Issue 7742, updated figure above to be at the M1 Level. Changed Attributes header to M1 Attributes. Changed Associations header to M1 Associations

---

M1 Attributes

- `compiler: DeploymentRequirementQualifier [0..1]`
  The compiler attribute indicates the complier name and version used to create the loadable code.
- `deployedOnRequirement: DeploymentRequirement [1..*]`
  A SWRadio LoadableCode artifact needs to specify "deployed on" deployment requirements in order for the artifact code gets deployed on the right SWRadio communication channel's device. The set of deployedOnRequirements specifies the device required and the capacity needed from the device.
- `loadDependencyRequirement: DeploymentRequirement [*]`
  A SWRadio LoadableCode artifact can have dependency to other object code that requires loading and or execution first. The set of loadDependencyRequirements specifies the object code dependency.

- `requiredUsageRequirement: DeploymentRequirement [*]`
  A SWRadio LoadableCode artifact requires usage of a SWRadio service. The set of requiredUsageRequirements specifies the service required.

M1 Associations

- `implementationDescriptor: Descriptor [1]`
  SWRadio LoadableCode artifacts are described by an implementation descriptor, which captures the deployment requirements and the type of artifact to be deployed

Semantics

A LoadableDevice manages the loading and unloading of SWRadio LoadableCode on a loadable device.

8.3.4.1.2.5 ResourceExecutableCode

Description

The ResourceExecutableCode artifact, as shown in Figure 8-50, defines the operating system main process that manifests an Resource or ResourceFactory component.

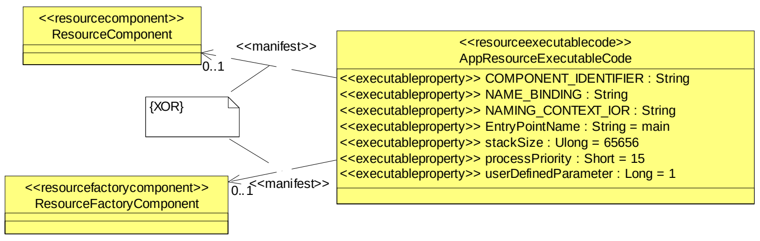

<<resourcecomponent>>
ResourceComponent
<<manifest>>
0..1
{XOR}
<<resourceexecutablecode>>
AppResourceExecutableCode
<<executableproperty>> COMPONENT_IDENTIFIER : String
<<executableproperty>> NAME_BINDING : String
<<executableproperty>> NAMING_CONTEXT_IOR : String
<<executableproperty>> EntryPointName : String = main
<<executableproperty>> stackSize : Ulong = 65656
<<executableproperty>> processPriority : Short = 15
<<executableproperty>> userDefinedParameter : Long = 1
<<resourcefactorycomponent>>
ResourceFactoryComponent
0..1
<<manifest>>

Figure 8-50 – ResourceExecutableCode M1 Illustration

Note – Updated Figure above to be at the M1 Level.

Constraints

ResourceExecutableCode shall accept the following three executable parameters:

- NamingContext Component Reference - The ID is “NAMING_CONTEXT_IOR” and the value is a stringified naming context reference. The reference is used to obtain the NamingContext path. The format of a NamingContext path is sequence of NamingContext where each “slash” (/) represents a separate naming context. A NamingContext is made up of ID and kind pair, which is indicated by “id.kind” in the NamingContext string. The NamingContext kind is optional and when not specified a null string will be used. For example, the structure of the naming context path is “/ SomeName / [optional naming context sequences]”. There is at least one “slash” (/) in the Naming Context string.
- Name Binding - The ID is “NAME_BINDING” and the value is a string that corresponds to the name used to bind the manifested component to the naming service.

- Identifier - The ID is “COMPONENT_IDENTIFIER” and the value is a string that is used for the SWRadioComponent's identifier value.

ResourceExecutableCode shall register the Resource or ResourceFactory component manifested by the process to the NamingService as specified by the NamingContext Component Reference executable parameter. The name binding registered to the NamingService shall be as specified by the Name Binding executable parameter.

The ResourceExecutableCode Identifier executable parameter shall be used for the manifested SWRadioComponent's identifier attribute value.

8.3.4.1.2.6 ServiceExecutableCode

Description

The ServiceExecutableCode artifact, as shown in Figure 8-51, defines the operating system main process that manifests a Service component.

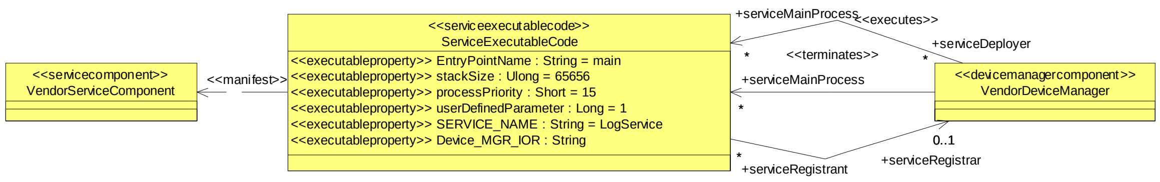

<<servicecomponent>>
VendorServiceComponent
<<manifest>>
<<serviceexecutablecode>>
ServiceExecutableCode
<<executableproperty>> EntryPointName : String = main
<<executableproperty>> stackSize : Ulong = 65656
<<executableproperty>> processPriority : Short = 15
<<executableproperty>> userDefinedParameter : Long = 1
<<executableproperty>> SERVICE_NAME : String = LogService
<<executableproperty>> Device_MGR_IOR : String
+serviceMainProcess
<<executes>>
+serviceDeployer
*
<<terminates>>
*
+serviceMainProcess
*
<<devicemanagercomponent>>
VendorDeviceManager
0..1
*
+serviceRegistrant
+serviceRegistrar

Figure 8-51 – ServiceExecutableCode M1 Illustration

Note – Updated the above Figure to be at the M1 Level.

Constraints

The ServiceExecutableCode shall accept the following user-defined executable parameters:

- DeviceManager Registration Reference - The ID is “DEVICE_MGR_IOR” and the value is a string that is the INodeRegistration component reference.
- Service Name - The ID is “SERVICE_NAME” and the value is a string that corresponds to the service name.

The ServiceExecutableCode shall register the ServiceComponent’s Service(s) manifested by the process to the DeviceManager using the DeviceManager Registration Reference executable parameter.

8.3.4.2 Applications Deployment

Note – Issue 7742, broke this section into two subsection, interfaces at the m1 level (model library) and component stereotypes

This section defines the interfaces (8.3.4.2.1) and components (8.3.4.2.3) that perform the deployment behavior within a SWRadio.

8.3.4.2.1 Applications Deployment Interfaces

This section defines the interfaces that perform the deployment behavior within a SWRadio. The SWRadio deployments interfaces are Application and ApplicationFactory, which are described in the following subsections.

Types and Exceptions

- `DeviceAssignmentType( componentId: String, assignedDeviceId: String)`
  DeviceAssignmentType defines a structure that associates a component with the DeviceComponent upon which the component must execute.
- `DeviceAssignmentSequence`
  DeviceAssignmentSequence provides an unbounded sequence of DeviceAssignmentType.

8.3.4.2.1.1 Application

Description

The Application interface provides for the control, configuration, and status of an instantiated application or waveform in the radio domain. The Application interface is specialization of the Resource interface that provides

Figure 8-52 – Application Definition

generic opeations for controlling the deployed application.

Attributes

- `<<readonly>>componentDevices: DeviceAssignmentSequence`
  The readonly componentDevices attribute contains a list of DeviceComponents, which each ApplicationAssembly's component uses, is loaded on or is executed on.
- `<<readonly>>componentImplementations: ComponentElementSequence`
  The readonly componentImplementations attribute contains the list of components' implementation IDs within the Application for those components created.
- `<<readonly>>componentNamingContexts: ComponentElementSequence`
  The readonly componentNamingContexts attribute contains the list of components' Naming Context using Naming Service.
- `<<readonly>>componentProcessIds: ComponentProcessIdSequence`
  The readonly componentProcessIds attribute contains the list of components' process IDs within the Application for components that are manifested within an ExecutableCode on an ExecutableDeviceComponent.
- `<<readonly>>name: String`
  The readonly name attribute contains the name of the created Application. The ApplicationFactory's create operation name parameter provides the name content.
- `<<readonly>>profile: String`
  The readonly profile attribute contains the instantiated Application descriptor file reference.

Operations

Note – Issue 7579 Resolution

- `getProvidedPorts (inout ports: PortSequenceType): {raises ( UnknownPorts )}`

  The getProvidedPorts operation returns object references only for input port names that match the external provided port names that are in the Application's component assembly descriptor. In the ports name/value pair sequence, each name corresponds to an external provided port name and each value corresponds to the object reference of the external provided port to be returned. The getProvidedPorts operation shall return all the external provided ports if the ports argument is zero size. The getProvidedPorts operation shall return only those provided ports specified in the ports argument if the ports argument is not zero size. The getProvidedPorts operation shall raise an UnknownPorts exception when one or more requested provided ports are invalid.
- `releaseObject(): (raises (ReleaseError)}`

  The releaseObject operation terminates execution of the ApplicationManager.

Types and Exceptions

- `ComponentProcessIdType( componentId: String, processed: unsigned long)`

  The ComponentProcessIdType defines a type for associating a component with its process ID. This type can be used to retrieve a process ID for a specific component.
- `ComponentProcessIdSequence`

  The ComponentProcessIdSequence type defines an unbounded sequence of componentProcess IdTypes.
- `ComponentElementType( componentId:String, processed: unsigned long)`

  The ComponentElementType defines a type for associating a component with an element (e.g., naming context, implementation ID).
- `ComponentElementSequence`

  The ComponentElementSequence defines an unbounded sequence of ComponentElementType.

8.3.4.2.2 ApplicationFactory

Description

The ApplicationFactory interface provides the dynamic mechanism to create a specific type of Application (e.g. waveform) in the SWRadio domain. Figure 8-56 depicts the ApplicationFactory's capacity constraints and Figure 8-57 depicts the relationships that are common for all ApplicationFactory(s).

Figure 8-53 – ApplicationFactory Capacity Overview

8.3.4 SWRadio Deployment

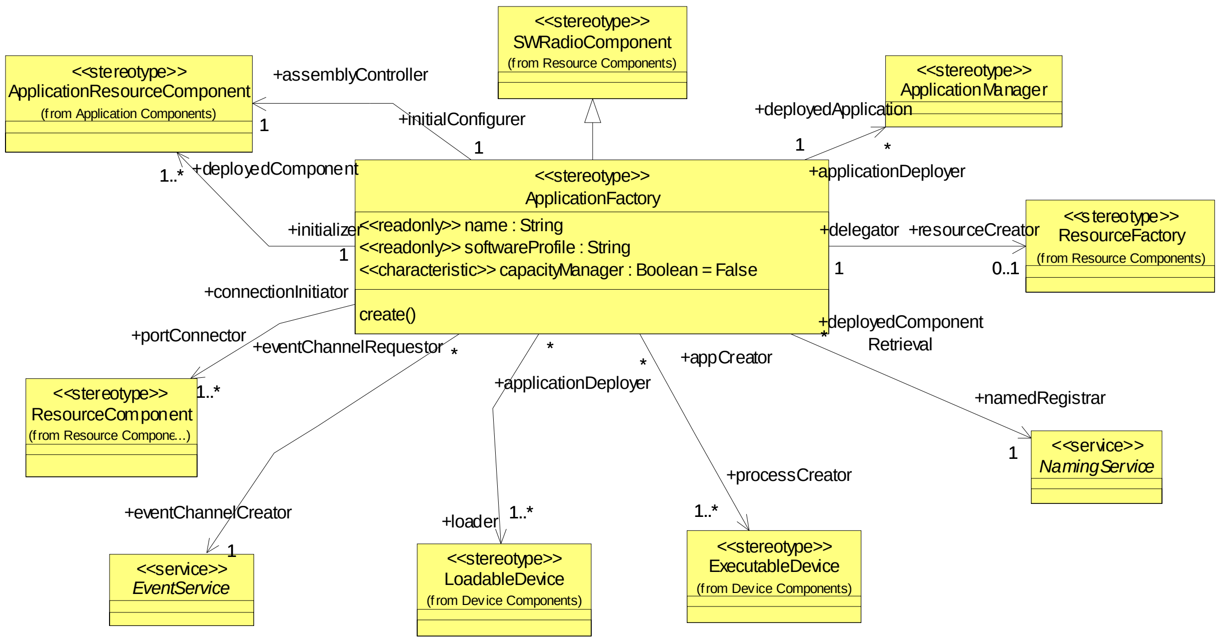


Figure 8-54 – ApplicationFactory Definition

Attributes

- `<<characteristic>>capabilityManager: Boolean = False`

  The characterisitic capabilityManager attribute indicates the ApplicationFactory behavior in regards to CapacityProperty(s). A value of True means the ApplicationManager manages CapacityProperty(s), otherwise it does not.

- `<<readonly>>name:`

  The readonly name attribute contains the name of the installed Application.

- `<<readonly>>softwareProfile:`

  The readonly softwareProfile attribute contains the installed Application description file reference.

Operations

- `create(in name: String, in initConfiguration: Properties, in deviceAssignments: DeviceAssignmentSequence, return ApplicationManager):{raises ( CreateApplicationError, CreateApplicationRequestError, InvalidInitConfiguration )}`

  The create operation provides the capability of creating an ApplicationManager.

Types and Exceptions

- `<<exception>>CreateApplicationRequestError`

  This exception is raised when the DeviceAssignmentSequence contains 1 or more invalid Application component-to-device assignment(s)

- `<<exception>>CreateApplicationError`

  The CreateApplicationError exception, specialization of SystemException, is raised the create operation is unsuccessfully due to internal processing errors. The error number indicates an ErrorNumberType value (e.g., E2BIG, ENAMETOOLONG, ENFILE, ENODEV, ENOENT, ENOEXEC, ENOMEM, ENOTDIR, ENXIO, EPERM). The message is component-dependent, providing additional information describing the reason for the error.
- `<<exception>> InvalidInitConfiguration`

  The InvalidInitConfiguration exception is raised when the input initConfiguration parameter is invalid.

Semantics

The create operation provides the capability of deploying an Application by making dynamic decisions on which DeviceComponent(s) the Application's components are deployed on and which Service(s) are used by an Application. The create operation determines which registered domain Services can satisfy the Application's deployment requirements as stated in the installed Application descriptor. The create operation also provides the mechanism to direct which DeviceComponent(s) are to be used for Application deployment instead of the ApplicationFactoryComponent making the DeviceComponent decision.

The create operation shall use descriptor information as referenced by the softwareProfile to determine the Application deployment requirements.

If input deviceAssignments (not zero length) are provided, the create operation verifies each device assignment, for the specified component, against the Application's deployment requirements.

Note – Issue 8841 - Clarification of create () behavior for deploying application components. Use the term deployment and partitioning requirement instead of allocation requirements for application components. Also, specify that capability and characteristic requirements are specified in a Service component’s ServiceProperties.

The create operation shall allocate the Application's deployment requirements against candidate Service(s) to determine which candidate Service(s) satisfy all the Application's deployment requirements and partitioning requirements (e.g., components HostCollocation, components process thread collocation, etc.). The create operation shall only use Service(s) whose capability and capacity characteristics (expressed in the Service(s)’ ServiceProperties) satisfy the allocation requirements (i.e. deployment and partitioning requirements) specified for the Application components. The create operation shall use the ServiceProperty's CapabilityModel or delegate as specified by Service's ServiceProperty's capabilityModel and locallyManaged attributes for determining Service(s) that can satisfy an Application's deployment requirement. The actual Service(s) chosen will reflect changes in capacity based upon component deployment requirements allocated to them, which may also cause state changes for the Service(s).

The create operation shall load the Application components (including all of the Application-dependent components) to the chosen DeviceComponent(s).

The create operation shall execute the application components (including all of the application-dependent components) as specified in the application's descriptor. The create operation shall use each component's implementation stack size and priority attributes, when specified, for the ExecutableDevice's execute options parameters.

The create operation shall pass the mandatory execute parameters as specified for ResourceExecutableCode for each Application component's implementation that has an entry point.

The create operation shall pass ResourceExecutableCode's executable parameters (NamingContext, Name Binding, Identifier) to an ExecutableDevice's execute formal parameter named parameters. The create operation creates any naming contexts that do not exist to which the component will bind to. The structure of the naming context path shall be “/ DomainName / [optional naming context sequences]”. In the naming context path, each “slash” (/) represents a separate naming context.

The ResourceExecutableCode's executable Name Binding parameter value shall be set to a string in the format of “ComponentName_UniqueIdentifier”. The ComponentName value shall be the component instantiation naming-service's name attribute in the Application's component assembly descriptor. The UniqueIdentifier is determined by the implementation.

The create operation uses “ComponentName_UniqueIdentifier” to retrieve the component's object reference from the Naming Context. Due to the dynamics of bind and resolve to NamingService, the create operation should provide sufficient attempts to retrieve component object references from NamingService prior to generating an exception.

The ResourceExecutableCode's executable Identifier parameter value shall be set to a string in the format of “Component_Instantiation_Identifier: Application_Name”. The Component_Instantiation_Identifier shall be the component's instantiation id attribute in the Application's component assembly descriptor. The Application_Name field shall be identical to the create operation's input name parameter. The Application_Name field provides a specific instance qualifier for executed ResourceComponent(s).

The create operation shall pass a component's implementation's ExecutableProperty(s) to an ExecutableDevice's execute formal parameter named parameters. The create operation shall pass Executable Property values as string values.

Note – Issue 8857 resolution (Next THREE paragraphs)

Note – Issue 8873 resolution (Usage of ConfigureProperty type instead of the non-existent ConfigureQueryProperty type)

The create operation shall, in order, initialize application components that support the LifeCycle interface, then establish connections for application components that support the PortConnector interface, and finally configure the application components that support the PropertySet interface and have ConfigureProperty(s) described in the application’s component assembly descriptor. The Application’s assemblycontroller shall be configured after the configuration of all application components.

The create operation uses the PortSupplier interface for obtaining provider interfaces for a connection.

The create operation input initConfiguration properties shall only apply to the assemblycontroller component of the deployed Application as defined in the Application's descriptor.   A deployed component’s configuration property values shall only come from the Application's descriptor, not from the component’s implementation or component definition descriptors.

The create operation configures an Application's assemblyController component provided the assemblyController has ConfigureProperty(s). The create operation shall use the union of the input initConfiguration properties of the create operation and the assemblyController’s ConfigureProperties. The input initConfiguration parameters shall have precedence over the assemblyController’s configure property values.

The create operation shall, when creating a ResourceComponent from a ResourceFactory, pass the associated ResourceFactory's ConfigureProperty(s) as qualifiers parameters to the referenced ResourceFactory component's createResource operation.

Note – Issue 7579 resolution (renamed getProvidedPorts in the last sentence)

The create operation shall interconnect ResourceComponent(s) (Application's components or DeviceComponents') ports in accordance with the Application's component assembly descriptor. The create operation obtains provider ports in accordance with the Application's component assembly via PortSupplier's getProvidedPorts operation.

The connections to domain Service(s) such as LogService, FileManager, FileSystem, Event Service, and NamingService are specified as component's connections using domainfinder in the Application's component assembly descriptor. Domain Service(s) are services that have been registered to the domain.

For connections established for a EventService's event channel, the create operation shall connect a PushConsumer or PushSupplier object to the event channel as specified in the component's connection in the Application's component assembly descriptor. If the event channel does not exist, the create operation shall create the event channel.

The create operation establishes connections to ResourceComponent(s) using the PortConnector::connectPort operation. The create operation shall use the connection id attribute as the unique identifier for a specific connection when provided in the Application's component assembly descriptor. The create operation shall create a connection ID when no connection id is specified for a connection in the Application's component assembly descriptor.

If the Application is successfully created, the create operation returns an ApplicationManager component reference for the created Application. A sequence of created Application references can be obtained using the DomainManagerComponent's readonly applications attribute (getApplications operation).No additional semantics.

The create operation shall raise the CreateApplicationRequestError exception when the DeviceAssignmentSequence parameter contains one (1) or more invalid Application component to device assignment(s).

The create operation shall raise the CreateApplicationError exception when the Application cannot be successfully instantiated due to internal processing error(s).

The create operation shall raise the InvalidInitConfiguration exception when the input initConfiguration parameter is invalid. The InvalidInitConfiguration invalidProperties identifies the properties that are invalid.

The created ApplicationManager's name attribute shall be identical to the input name parameter.

8.3.4.2.3 Application Deployment Stereotypes

This section defines the components that perform the deployment behavior within a SWRadio. The SWRadio stereotypes are depicted in the Table 8-13, which are extensions of the UML Component. The details of each component are described in the following subsections.

Note – Issue7742 Renamed ApplicationFactory to ApplicationFactoryComponent

Table 8-13 – SWRadio Applications

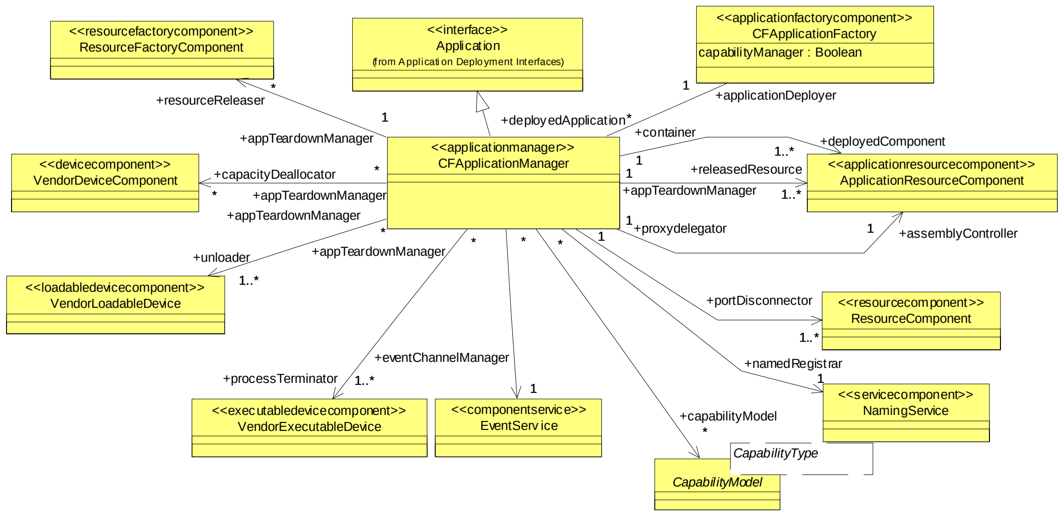
Stereotype	Base Class	Parent	Tags	Constraints	Description
ApplicationManager	Component	Resource Component			Represents the proxy for deployed Application.
ApplicationFactory Component	Component	SWRadioComponent	Name, SoftwareProfile		Represents the deployment machinery for Application's Descriptor.

8.3.4.2.3.1 ApplicationManager

Description

Note – Issue 7742 removed attributes and operations, updated figure to be M1 type, renamed Associations noheader to be M1 Assoications noheader. Updated associations. Added constraint for interface.

The ApplicationManager is a type of ResourceComponent as shown in Figure 8-55. The ApplicationManager provides the interface for the control, configuration, and status of an instantiated application or waveform in the radio domain. The ApplicationManager is the proxy for the deployed Application component.



<<resourcefactorycomponent>> ResourceFactoryComponent
<<interface>> Application (from Application Deployment Interfaces)
<<applicationfactorycomponent>> CFApplicationFactory
capabilityManager : Boolean
+resourceReleaser
+appTeardownManager
+deployedApplication
+applicationDeployer
+container
+deployedComponent
<<devicecomponent>> VendorDeviceComponent
<<applicationmanager>> CFApplicationManager
<<applicationresourcecomponent>> ApplicationResourceComponent
+capacityDeallocator
+releasedResource
+appTeardownManager
+proxydelegator
+assemblyController
+unloader
<<loadabledevicecomponent>> VendorLoadableDevice
+portDisconnector
<<resourcecomponent>> ResourceComponent
+namedRegistrar
<<servicecomponent>> NamingService
+eventChannelManager
+processTerminator
<<executabledevicecomponent>> VendorExecutableDevice
<<componentservice>> EventService
+capabilityModel
CapabilityType
CapabilityModel

Figure 8-55 – ApplicationManager Definition

M1 Associations

- `assemblyController: ApplicationResourceComponent [1]`

  The Application's assemblyController that ApplicationManager delegates operations to.

- `capacityDeallocator: VendorDeviceComponent [*]`
  The DeviceComponents that were allocated capacities during the Application deployment are used to return capacities back to these devices.
- `capabilityModel: CapabilityModel [*]`
  The capabilityModels for ServiceProperty(s) used and managed by the Application.
- `deployedComponent: ApplicationResourceComponent [1..*]`
  The Application's ApplicationResourceComponent that are deployed.
- `eventChannelManager: EventService [1]`
  The EventService that manages the creation and destruction of EventChannels.
- `portDisonnector: ResourceComponent [1..*]`
  The ResourceComponent(s) provide the capability to disconnect provided ports from their required ports.
- `namedRegistrar: NamingService [1]`
  The NamingService that has the deployed Application components references.
- `releasedResource: ApplicationResourceComponent [1..*]`
  The ApplicationResourceComponent(s) that were manifested during the Application deployment are released.
- `resourceReleaser: ResourceFactoryComponent [*]`
  The ResourceFactory(s) that were used during the Application deployment are used to release ApplicationResourceComponent(s) from these ResourceFactory(s).
- `processTerminator: VendorExecutableDevice [1..*]`
  The ExecutableDevices that executed ExecutabeCode during Application deployment are used to terminate these processes.
- `unloader: LoadableDevice [1..*]`
  The LoadableDevices that were loaded with ObjectCode during the Application deployment are used to unload the ObjectCode from these devices.

Constraints

The ApplicationManager shall be either associated with DeviceComponent(s) or CapabilityModel(s) for the deallocate capacity behavior.

The ApplicationManager shall realize the Application interface.

Semantics

The ApplicationManager shall delegate the implementation of the inherited ResourceComponent operations (runTest, start, stop, configure, and query) to the Application's assembly controller's ApplicationResourceComponent. The ApplicationManager shall propagate exceptions raised by these operations. The initialize operation is not propagated to the Application's assembly controller. The initialize operation causes no action within an ApplicationManager.

Note – Issue 8931 (Paragraph Below)

The ApplicationManager releaseObject operation shall deallocate Application's required capacities against the ServiceComponent(s) that were obtained from or associated with. The actual DeviceComponent(s) may reflect changes in capacity based upon component capacity requirements deallocated from them, which may also cause state changes for the DeviceComponent(s). The releaseObject operation shall release all references to the Application components. The releaseObject operation shall disconnect port connections to non-application component providers that have been connected based upon the Application's component assembly descriptor. The releaseObject operation shall disconnect Application's components consumers and producers from an EventService's event channel. The releaseObject operation may destroy an EventService's event channel when no more consumers and producers are connected to it. For components (e.g., Resource, ResourceFactory) that are registered with Naming Service, the releaseObject operation unbinds those components and destroys the associated naming contexts as necessary from the NamingService. The releaseObject operation shall disconnect ports first, then release the Application's ResourceComponent(s) and ResourceFactory(s), terminate Application component's ExecutableCode, and lastly unload Application component's LoadableCode from the DeviceComponent(s). The releaseObject operation shall raise a ReleaseError exception when the releaseObject operation unsuccessfully releases the Application components due to internal processing errors.

Note – Issue7742 renamed ApplicationFactory to ApplicationFactoryComponent, removed operations and some attributes, updated figures to M1 types, removed semantics, renamed Associations noheader to be M1 Associations noheader, and updated some associations. Added constraint for interface.

8.3.4.2.4 ApplicationFactoryComponent

Description

The ApplicationFactoryComponent provides the dynamic mechanism to create a specific type of Application (e.g. waveform) in the SWRadio domain. Figure 8-56 depicts the ApplicationFactory's capacity constraints and Figure 8-57 depicts the relationships that are common for all ApplicationFactory(s).

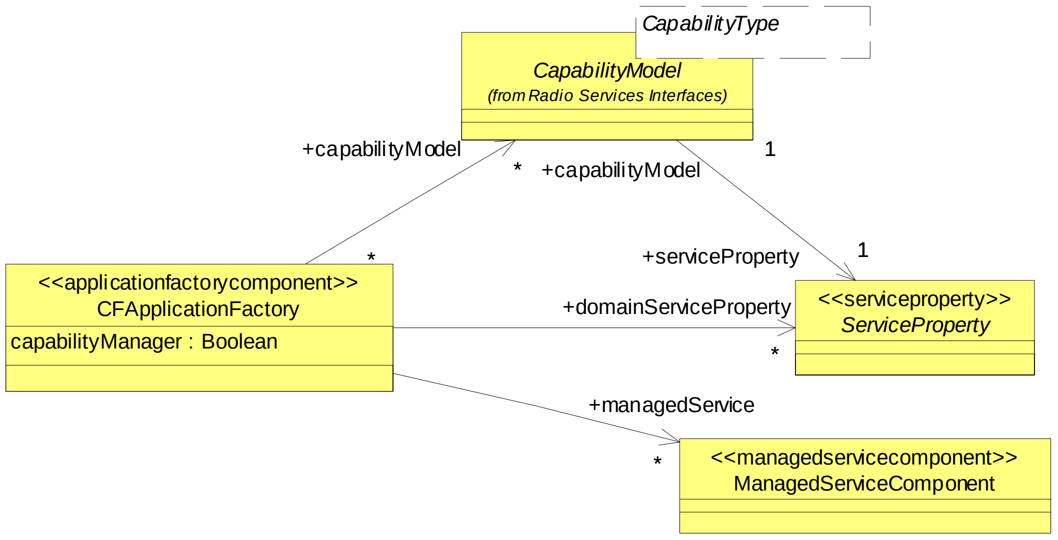


Figure 8-56 – ApplicationFactoryComponent Capacity Overview

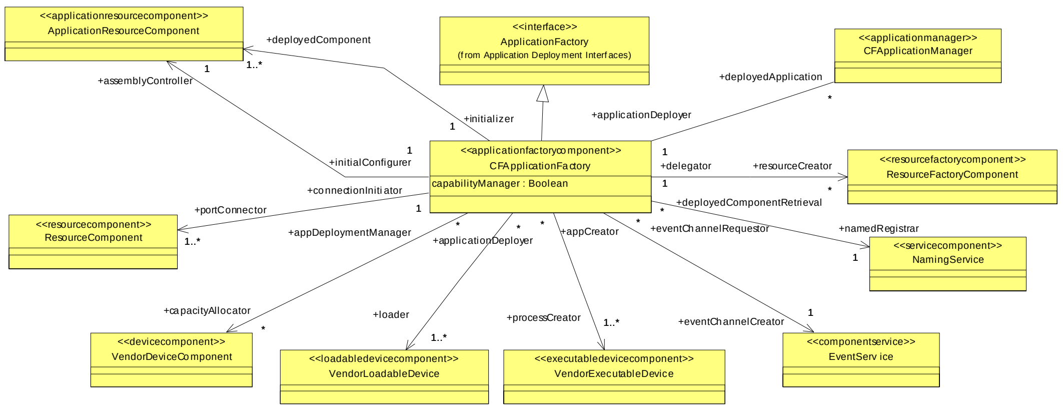


Figure 8-57 – ApplicationFactory M1 Illustration

Attributes

● `capabilityManager: Boolean = False`

The capabilityManager attribute indicates the ApplicationFactoryComponent behavior in regards to CapacityProperty(s). A value of True means the ApplicationManagerComponent manages CapacityProperty(s), otherwise it does not.

M1 Associations

Note – Issue 8873 Resolution - Replacement of non-existent ConfigureQuertyProperty type with the ConfigureProperty type

---

● `assemblyController: ApplicationResourceComponent [1]`

The ApplicationResourceComponent that is the assemblyController for the instantiated Application, which acts as the initialConfigurer of the ConfigureProperty(s) for the instantiated Application.

● `capabilityModel: CapabilityModel [*]`

The CapabilityModel(s) used for determining which domainServiceProperty(s) can satisfy the Application's deployment requirements.

● `portConnector: ResourceComponent [1..*]`

The ResourceComponent(s) that are connected to Services and to other ResourceComponent(s) by the ApplicationFactory.

● `deployedApplication: CFApplicationManager [*]`

The ApplicationManager that manages the instantiated Application.

● `deployedComponent: ApplicationResourceComponent [1..*]`

The instantiated Application's deployed ApplicationResourceComponents that are initialized.

● `domainServiceProperty: ServiceProperty [1..*]`

The registered Services' ServiceProperty(s) that are used for determining Services to be used for Application deployment.

- `eventChannelCreator: EventService [1]`
  The EventService that manages EventChannels.
- `loader: VendorLoadableDevice [1..*]`
  The LoadableDeviceComponent(s) are used to load ObjectCode during the Application deployment.
- `managedService: ManagedServiceComponent [*]`
  A managedService manages its own CapacityModel(s).
- `NamedRegistrar: NamingService[1]`
  The NamingService that contains deployed component object references.
- `processCreator: VendorExecutableDevice [1..*]`
  The ExecutableDeviceComponent(s) are used to execute Application main processes during Application deployment.
- `resourceCreator: ResourceFactoryComponent [*]`
  The ResourceFactory can be used during the Application deployment as an optional means of creating ApplicationResourceComponent(s).

Constraints

The identifier attribute shall be identical to the installed Application's descriptor id attribute.

The name attribute shall be identical to the installed Application's descriptor name attribute.

When capacityManager attribute is False the ApplicationFactoryComponent shall only use ServiceCapacityProperty(s) that are locally managed by a ManagedServiceComponent, otherwise the ApplicationFactory manages the ServiceCapacityProperty(s). Characteristic properties can be either managed or unmanaged by the ApplicationFactoryComponent.

The ApplicationFactoryComponent shall realize the ApplicationFactory interface.

9 Platform Independent Model (PIM)

Note – Issue 7845

The SWRadio PIM specified in this section is a normative specification of the SWRadio profile. It may be realized using many technologies. The CORBA reference PSM in Section 10 is one such realization.

Note – Issue 7785 - Waveform vs Application changes throughout Chapter 9

The SWRADIO PIM Components are made of:

- Common Layer Facilities - This facility defines the set of interfaces that all components (regardless of any layering) within the radio can realize. Examples of these types of interfaces are flow control, packet, and stream interfaces.

Note – Issue 8201 Resolution (Removal of references to File Services from Common Radio Facilities)

- 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, and event service.
- Data Link Layer Facilities - These facilities define Link Layer Control (LLC) and Media Access Control (MAC) layer functionality for communication needs.
- 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. Physical layer facilities also include functionality for baseband I/O such as serial and audio devices.
- Radio Control Facilities - These facilities define the functionality to configure, get status, and control the radio domain and channels within the radio.

9.1 Common Radio Facilities

Note – Issue 8201 Resolution (Addition of all of section 9.1.1 (Lightweight Services)

9.1.1 Lightweight Services

9.1.1.1 NamingService

The NamingService provides a white page capability for component registration and retrieval. This white page capability provides the means to have a centralized repository of component references in the system. Servers (components that provide services) register their component references with the NamingService under a unique name so that clients (components that require these services) can find them. Clients find the desired component references distributed throughout the system by their assigned name as published within this while page capability. Once a client finds the desired server component, the client can start requesting the desired services.

Note – Issue 7586

Semantics

The NamingService's NameComponent structure is made up of an id-and-kind pair. The "id" element of each NameComponent is a string value that uniquely identifies a NameComponent.

9.1.1.2 EventService

The EventService decouples the communication between consumer and producer components, where consumer components are unaware of producer components, and vice versa. Consumer components process event data that are produced by producer components. The OMG Lightweight Event Service as required by this specification is restricted to support the canonical Push Model approach where producers push events to event channels and event channels in turn push these events to consumers.

The CosLightweightEventComm package is used by consumers for receiving events and by producers for generating events. A component that consumes events shall implement the CosLightweightEventComm PushConsumer interface. A component that produces events shall implement the CosLightweightEventComm PushSupplier interface and use the CosLightweightEventComm PushConsumer interface for generating the events. A producer component shall handle all cases, without raising any exceptions outside of the producer component, due to the connections to a CosEventComm PushConsumer being nil or an invalid reference. The EventService will have the capability to create event channels. An event channel allows multiple suppliers to communicate with multiple consumers asynchronously. An event channel is both a consumer and a producer of events. For Example, event channels can be standard CORBA objects and communication with an event channel is accomplished using standard CORBA requests.

Note – Issue 7586

9.1.1.3 LogService

The OMG Lightweight Log Service Specification contains the interfaces and the types necessary for the use of a log. These interfaces consist of the LogProducer, LogConsumer and LogAdministrator. Using the LogProducer interface, a log producer may generate log records conformant to this specification. Using the LogConsumer interface, a log consumer may retrieve records from a log. Using the LogAdministrator interface, a log administrator may control the operation of a log. Throughout this specification, use of the term Log, Log Service, or LogService refers to any one of these interfaces based upon the context it is used in. Additionally, these interfaces provide operations that may be used to obtain the status of a log. The OMG Lightweight Log Service Specification also defines the types necessary to control the logging output of a log producer. SWRadioComponents that produce logs are required to implement ConfigureProperty(s) and QueryProperty(s) that allow the component to be configured and queried as to what log records it will output.

Note – Issue 8873 Resolution - Replacement of non-existent ConfigureQuertyProperty type with the ConfigureProperty and QueryProperty types (last sentence in above paragraph)

A LogService may be provided in a software radio installation. The optional aspect of the LogService is restricted to its implementation and deployment. A software radio provider may deliver a product conformant to this specification without a LogService implementation. For instance, a handheld platform with limited resources may choose not to deploy a LogService as part of its domain. Several Infrastructure components contain requirements to write log records using the log service. Components that are required to write log records are also required to account for the absence of a LogService and otherwise operate normally.

Note – Issue 7586, removed cosntraints since it is already in SWradioComponents semantics section.

Note – Issue 8873, Constraints section added back as the SWRadioComponents semantics section provides optional and limited requirements for SWRadioComponents that MAY be log producers. The LogService constraints section provides complete and mandatory requirements for log producers.

Constraints

A log producer is a SWRadioComponent that produces log records using the LogProducer interface. (A component that calls the writeRecord(s) operation of the LogProducer interface.)

Note – Issue 8873 Resolution - Replacement of non-existent ConfigureQuertyProperty type with the ConfigureProperty and QueryProperty types

---

A standard record type is defined for all log producers to use when writing log records. The log producer may be configured via the PropertySet interface to output only specific log levels. Log producers shall implement a ConfigureProperty and a QueryProperty with an ID of "PRODUCER_LOG_LEVEL". The PRODUCER_LOG_LEVEL ConfigureProperty provides the ability to "filter" the log message output of a log producer. The type of the PRODUCER_LOG_LEVEL ConfigureProperty and QueryProperty shall be a Lightweight LogService LogLevelSequence. The LogLevelSequence will contain all log levels that are enabled. Only the messages that contain an enabled log level shall be sent by a log producer to a Log. Log levels that are not in the LogLevelSequence are disabled.

Log producers shall use their component identifier (identifier attribute of the ComponentIdentifier interface) in the producerId field of the CosLwLog ProducerLogRecord.

Log producers shall operate normally in the case where the connections to a Log are nil or an invalid reference.

Log producers shall output only those log records that correspond to enabled CosLwLog LogLevel values.

9.2 Common Layer Facilities

This section defines the Common Layer Facilities, which provide interfaces that cross cut through facilities that correlate to layers. These interfaces can be viewed as building blocks for waveform components that realize multiple interfaces. Figure 9-58 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 Facilities - defines the quality of service related facilities.
- Flow Control Facilities - provides means to control communication flow so that a sender does not transmit more packets than a receiver can process.
- Measurement Facilities - specifies facilities to set up waveform related measurement parameters and schedule the measurement.
- Error Control Facilities -- 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.
- PDU Facilities - 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 Facilities - 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.

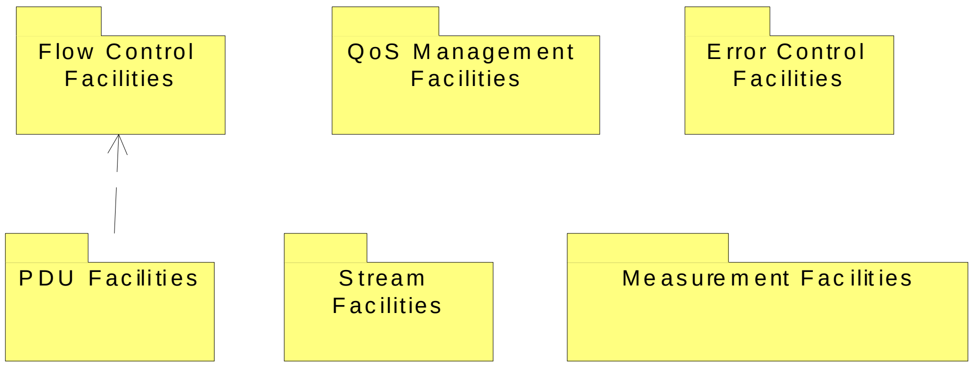


Figure 9-58 – Common Layer Facilities Overview

Types and Exceptions

Note – Issue 7895: add primitive type

- `SduSizeType (maxSduSize : ULong, minSduSize : ULong)`

  SduSizeType defines the maxSduSize and minSduSize attributes as positive longs. Those two values together define the range of values sduSize can take.

- `AddressType`

  AddressType is an OctetSequence that represents the source or destination address.

Note – Issue 7895: Issue TBD remove integer, remove type defined in BaseTypes

9.2.1 QoS Management Facilities

Quality of Service (QoS) Management Facilities define the facilities that can be used to control quality of service related parameters. The QoS parameters that can be set up are given in the DLPI specification document. Figure 9-59 shows an overview of QoS facilities.

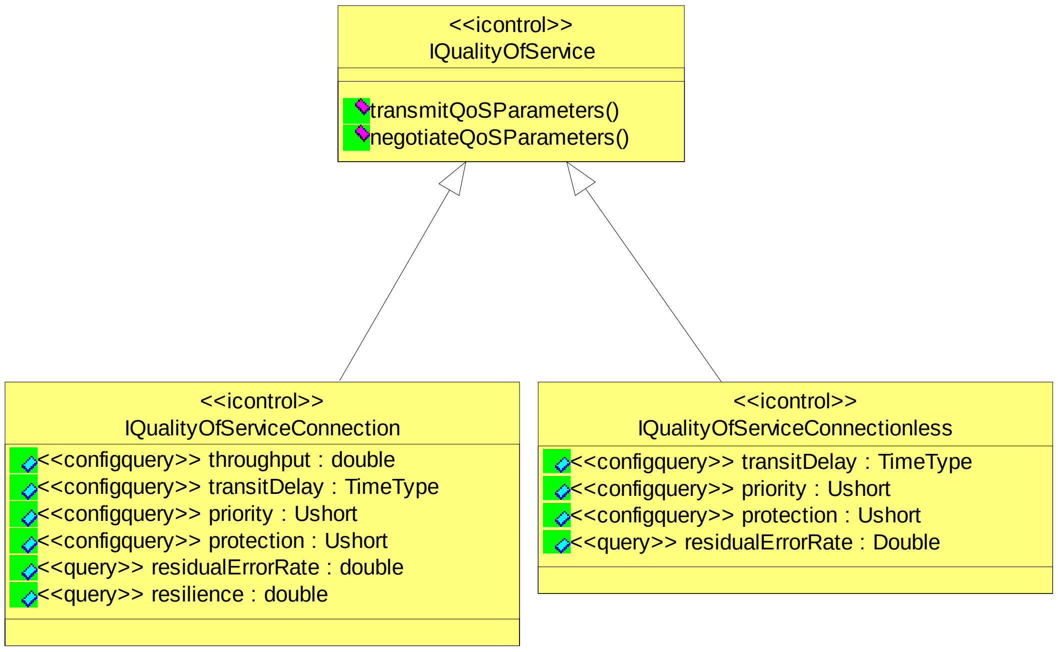

<<icontrol>>
IQualityOfService
transmitQoSParameters()
negotiateQoSParameters()
<<icontrol>>
IQualityOfServiceConnection
<<configquery>> throughput : double
<<configquery>> transitDelay : TimeType
<<configquery>> priority : Ushort
<<configquery>> protection : Ushort
<<query>> residualErrorRate : double
<<query>> resilience : double
<<icontrol>>
IQualityOfServiceConnectionless
<<configquery>> transitDelay : TimeType
<<configquery>> priority : Ushort
<<configquery>> protection : Ushort
<<query>> residualErrorRate : Double

Figure 9-59 – Quality of Service Facilities Overview

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

9.2.1.1 IQualityOfService

Description

Note – Issue 7878 Resolution (Renamed IErrorControl to IError_Control)

IQualityOfService, as shown in Figure 9-59, 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 and connectionless). This interface provides the capabilities of signalling and negotiating QoS parameters with waveform components. A component realizing the IQualityOfService interface depends on other components that realize transmission interfaces (Common Layer Facilities::IPdu) for transferring control and user data, flow control interfaces (Common Layer Facilities::IFlowControl) that allows a QoS controller component to change flow control parameters such as data rate, buffer size, measurement interfaces (Common Layer Facilities::IMeasurement) for monitoring QoS related system performance parameters and error control interfaces (Common Layer Facilities::IError_Control) for controlling error control coding parameters.

Operations

- `transmitQoSParameters( )`

  This operation signals the quality of service parameters to the requester.

- `negotiateQoSParameters( )`

  This operation provides a generic interface to negotiate the quality of service parameters with the peer receiver/transmitter.

Semantics

IQualityOfService interface depends on the IStream and IPdu interfaces to communicate QoS parameters using transmitQoSParameters operation. A component that plays the AssemblyController role within the same radio set, or the peer receiving radio set may acquire the QoS parameters.

The negotiateQoSParameters operation implies implementation of an encapsulated underlying bidirectional communication protocol. This operation 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.

9.2.1.2 IQualityOfServiceConnection

Description

IQualityOfServiceConnection specializes the IQualityOfService interface to provide QoS attributes for connection oriented communication establishment. The definition of IQualityOfServiceConnection is shown in Figure 9-59.

Attributes

- `<<configquery>> throughput : Double`

  Throughput is a connection-mode QoS parameter that has end-to-end significance. It is defined as the total number of Service Data Unit (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 SDUs.

Note – Issue 8869 TimeType

- `<< configquery >> transitDelay : TimeType`

  The transitDelay 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.

Note – Issue 7895, change primitive type to not be an integer

---

- `<< configquery >> priority : UShort`

  The specification of priority is concerned with the relationship between connections. This attribute 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. Priority attribute is of UShort type. A lower value means lower priority and vice versa.

Note – Issue 7895, change primitive type to not be an integer

---

● `<< configquery >> protection : UShort`
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. Protection attribute is of UShort type. A lower value means lower protection and vice versa. Protection has local significance only.

● `<<query>> residualErrorRate : Double`
Residual Error Rate 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.

● `<<query>> resilience : Double`
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.

Semantics

IQualityOfServiceConnection interface inherits transmitQoSParameters and negotiateQoSParameters operations from its base class IQualityOfService. Those operations are used respectively transmit and negotiate all of the attributes of the IQualityOfServiceConnection interface.

Throughput attribute is specified and negotiated for 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.

Transit delay attribute is negotiated on an end-to-end basis during connection establishment. For each connection, transit delay is negotiated for 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

Priority attribute is negotiated locally between each user and the provider in connection-mode service. 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.

Protection attribute is negotiated locally between each user and the provider in connection mode. Provider protects against modification, replay, addition, or deletion of user data. Each user negotiates a particular value with the provider during connection establishment. This attribute only has local significance.

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

9.2.1.3 IQualityOfServiceConnectionless

Description

IQualityOfServiceConnectionless specializes the IQualityOfService interface to provide QoS attributes for connectionless communication establishment. Figure 9-59 shows the IQualityOfServiceConnectionless definition.

Attributes

- `<< configquery >> transitDelay : Double`
  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. This attribute is of Time class as defined in the Communication Equipment section of the UML Profile.

Note – Issue 7895, change primitive type to not be an integer

---

- `<< configquery >> priority : UShort`
  This attribute specifies the relative importance of a connectionless communication service. Priority is determined locally for each user in connectionless mode service. A lower value means lower priority and vice versa.
- `<< configquery >> protection: UShort`
  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. Protection attribute is of UShort type. A lower value means lower protection and vice versa. Protection has local significance only.
- `<<query>> residualErrorRate : Double`
  Residual Error Rate 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.

Semantics

In determining the transitDelay attribute, 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.

The specification of priority attribute 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.

Protection attribute has local significance only. Provider protects against modification, replay, addition, or deletion of user data. The transmitting radio set selects a particular value within the supported range, and the value may be changed for each SDU submitted for transmission.

9.2.2 Flow Control Facilities

Flow Control facilities define interfaces that relate to flow control of data transmission and reception. Those facilities control 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 facilities are separated into two interfaces for signaling and control management behavior, namely: IFlowControl-Signalling and IFlowControlManagement interfaces. The goal of flow-control mechanisms is to prevent dropped packets that must be retransmitted. Figure 9-60 shows an overview of Flow Control facilities. Flow Control can be implemented between peer layers for both connection oriented and connectionless communication modes, or at the service boundary between different layers within the same Software Defined Radio (SDR) set.

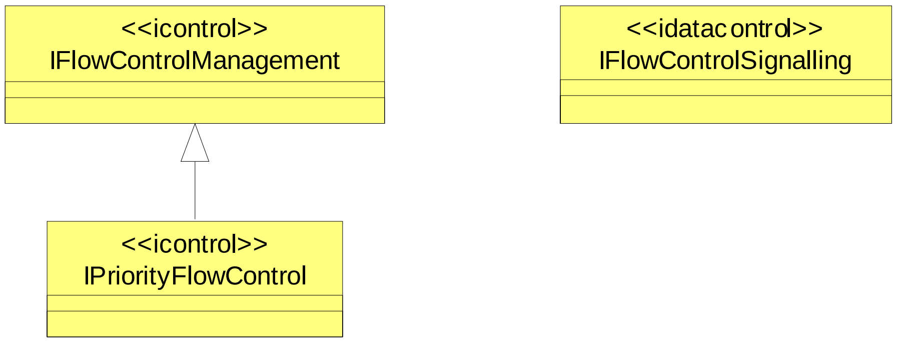


Figure 9-60 – Flow Control Facilities Definition

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

9.2.2 Flow Control Facilities

9.2.2.1 IFlowControlManagement

Description

IFlowControlManagement provides an interface for flow control manager component to control and manage flow control related arguments. The interface provides the capabilities of enabling and disabling flow control signaling, enabling priority based queueing and negotiating flow control parameters with the peer flow controller. Figure 9-61 shows the definition of IFlowControlManagement.

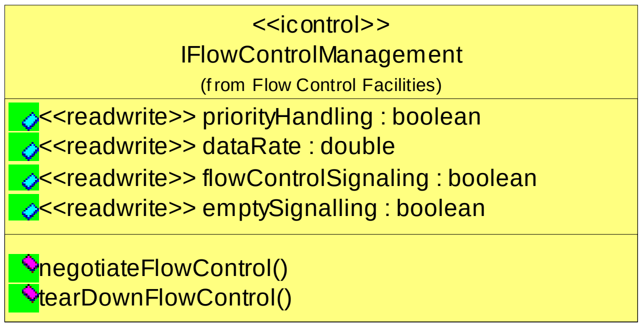

<<icontrol>>
IFlowControlManagement
(from Flow Control Facilities)
<<readwrite>> priorityHandling : boolean
<<readwrite>> dataRate : double
<<readwrite>> flowControlSignaling : boolean
<<readwrite>> emptySignalling : boolean
negotiateFlowControl()
tearDownFlowControl()

Figure 9-61 – IFlowControlManagement Definition

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

Attributes

- `<<readwrite>> flowControlSignalling: Boolean`
  This attribute indicates whether flow control signalling is currently enabled or not.
- `<<readwrite>> priorityHandling: Boolean`
  This attribute indicates whether priority queue handling is currently enabled or not.
- `<<readwrite>> dataRate : Double`
  Target data rate.
- `<<readwrite>>> emptySignaling : Boolean`
  This attribute indicates whether the flow controller should signal when a queue is empty.

Operations

- `negotiateFlowControl ( )`
  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.

Semantics

negotiateFlowControl and tearDownFlowControl operations indicate an underlying protocol mechanism that allows for two-way handshaking between components in order to negotiate and tear down flow control.

A component realizing IFlowControlManagement interface shall communicate with the component that realizes IStream or IPdu interface in order to transmit flow control related data, and with the component that realizes the IPriorityQueue interface in order to setup and teardown a priority queue.

9.2.2.2 IFlowControlSignalling

Description

IFlowControlSignalling provides an interface for sending flow control related signals. The interface provides the signalling capabilities for data congestion, high and low watermark, empty buffer, acknowledgement and negative acknowledgement events. Figure 9-62 shows the definition of IFlowControlSignalling.

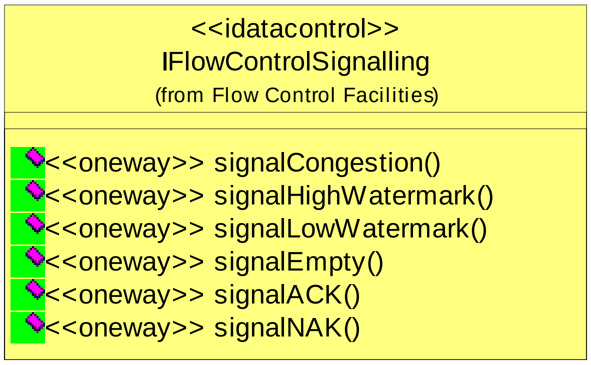

<<idatacontrol>>
IFlowControlSignalling
(from Flow Control Facilities)
<<oneway>> signalCongestion()
<<oneway>> signalHighWatermark()
<<oneway>> signalLowWatermark()
<<oneway>> signalEmpty()
<<oneway>> signalACK()
<<oneway>> signalNAK()

Figure 9-62 – IFlowControlSignalling Definition

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

Operations

Note – Issue 7658

- `<<oneway>>signalCongestion (in priorityQueueID : Octet)`

  This operation signals a congestion (the user can not handle incoming packets and they are being dropped.

- `<<oneway>>signalHighWatermark (in priorityQueueID : Octet)`

  The signalHighWaterThreshold operation is used to alert the peer entity that high watermark threshold has been reached.

- `<<oneway>>signalLowWatermark (in priorityQueueID : Octet)`
  The signalLowWaterThreshold operation is used to alert the peer entity that low watermark threshold has been reached.
- `<<oneway>>signalEmpty (in priorityQueueID : Octet)`
  The signalEmpty operation signals that the buffer is empty and ready to receive data.
- `<<oneway>>signalACK (in priorityQueueID : Octet)`
  The signalACK operation is used to acknowledge successful reception of a PDU sent by the provider.
- `<<oneway>>signalNAK (in priorityQueueID : Octet)`
  The signalNAK operation is used to acknowledge unsuccessful reception of a PDU sent by the provider.

Semantics

The component that consumes data shall use this interface to indicate to the sender the condition of the data consumer. A component realizing IFlowControlSignalling interface shall receive signals from the data consumer through this interface.

9.2.2.3 IPriorityFlowControl

Description

IPriorityFlowControl interface specializes the IFlowControlManagement interface and extends it by adding priority queue handling behavior. This interface can be used to create and destroy both PriorityQueue and WindowedPriorityQueue structures.

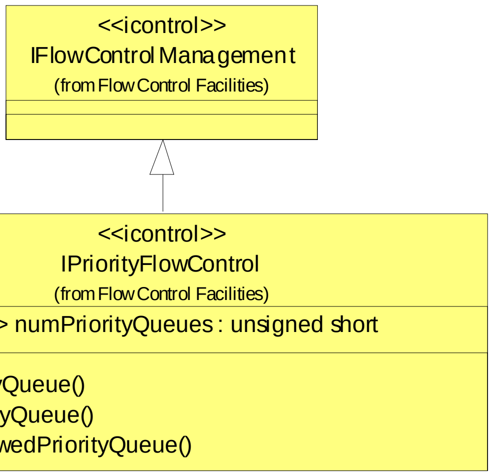

<<icontrol>>
IFlowControlManagement
(from Flow Control Facilities)
<<icontrol>>
IPriorityFlowControl
(from Flow Control Facilities)
<<readonly>> numPriorityQueues : unsigned short
createPriorityQueue()
destroyPriorityQueue()
createWindowedPriorityQueue()

Figure 9-63 – IPriorityFlowControl Definition

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

Attributes

Note – Issue 7895, change primitive type to not be an integer

- `<<readonly>> numPriorityQueues : UShort`
  Number of priority queues that the flow controller component is managing. This attribute can only be queried, setting the number of priority queues is done by creating and destroying priority queues using the related operations.

Operations

Note – Issue 7658, Issue 7895, change primitive type to not be an integer

- `createWindowedPriorityQueue (in priority: UShort, in queueSize: ULong, in highWatermarkThreshold: ULong, in lowWaterMarkThreshold: ULong, return Octet)`
  This operation creates a windowed priority queue bound to the flow control manager, and returns the priorityQueueId for it.

Note – Issue 7895, change primitive type to not be an integer

- `createPriorityQueue (in priority: UShort, in queueSize: ULong, in highWatermarkThreshold: ULong, in lowWaterMarkThreshold: ULong, return Octet)`
  This operation creates a priority queue bound to the flow control manager, and returns the priorityQueueId for it.
- `destroyPriorityQueue (in priorityQueueID : Octet)`
  This operation destroys a previously instantiated priority queue. This interface can also be used to destroy a WindowedPriorityQueue, which is a specialization of PriorityQueue.

Types and Exceptions

Note – Issue 7895, change primitive type to not be an integer

- `PriorityQueue (priority: UShort, queueSize: ULong, highWatermarkThreshold: ULong, lowWatermark-Threshold: ULong)`
  PriorityQueue provides a struct definition to keep priority queues parameters. Priority queues are used by the flow control mechanism to direct incoming Protocol Data Units (PDU) 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 values of highWatermarkThreshold and lowWatermarkthreshold attributes are application dependent and usually are determined by the flow controller of the QoS controller after negotiating with the remote entity. The interface provides the capability for configuring various parameters of a priority queue such as the queue size, priority level and high and low watermark levels. The attributes of PriorityQueue class is defined as:

  - *priority : UShort*
    Priority defines the relative importance of queues. A lower value means a lower value and vice versa.

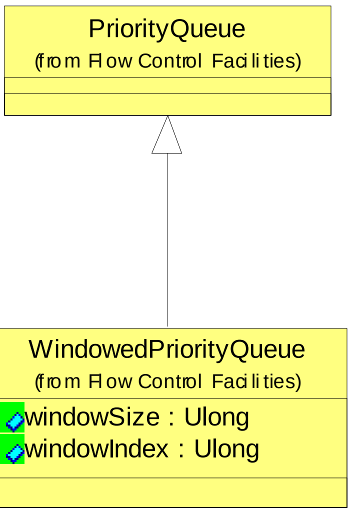
  - *queueSize : ULong*
    The maximum number of elements the queue can hold.

- *highWatermarkThreshold : ULong*
  High watermark threshold shows the number of elements in the queue where a dangerous occupancy is reached in the buffer and probability of dropping PDUs has increased.
- *lowWatermarkThreshold : ULong*
  Low watermark threshold shows the number of elements 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.
- *spaceAvailable : ULong*
  The size of available buffer space in the priority queue in terms of queue elements.
- *priorityQueueID : Octet*
  This attribute is assigned during the instantiation of a priority queue component and is used by other components to uniquely identify the priority queue.

Note – Issue 7895

- `WindowedPriorityQueue( windowSize : ULong, windowIndex : ULong)`

WindowedPriorityQueue specializes the PriorityQueue class in order to provide a mechanism for windowed acknowledgement in a priority queue. .



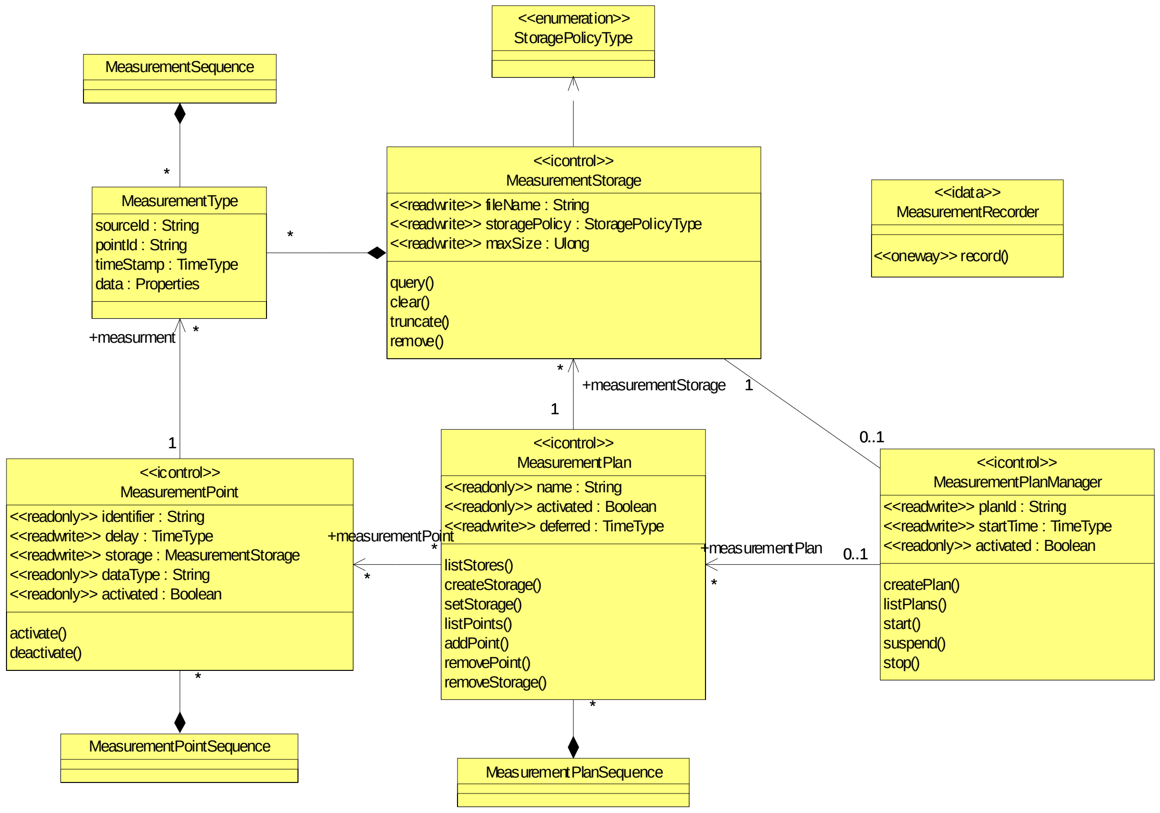
- *windowSize : ULong*
  Size of the window. This attribute can be changed during initialization and/or after the communication has been established.
- *windowIndex: ULong*
  Index of the current data window that is being acknowledged. Every time a window is acknowledged, the index is incremented.

9.2.3 Measurement Facilities

Measurement facilities 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 MeasurementPlan. Measurement Facilities interfaces are shown in Figure 9-64.

Note – Issue 7985 - fix type

Note – Issue 7878 Resolution (Diagram below - Renamed "Measurement" type to "MeasurementType")



<<enumeration>> StoragePolicyType
MeasurementSequence
<<icontrol>> MeasurementStorage
<<readwrite>> fileName : String
<<readwrite>> storagePolicy : StoragePolicyType
<<readwrite>> maxSize : Ulong
query()
clear()
truncate()
remove()
<<idata>> MeasurementRecorder
<<oneway>> record()
MeasurementType
sourceId : String
pointId : String
timeStamp : TimeType
data : Properties
+measurment
+measurementStorage
<<icontrol>> MeasurementPoint
<<readonly>> identifier : String
<<readwrite>> delay : TimeType
<<readwrite>> storage : MeasurementStorage
<<readonly>> dataType : String
<<readonly>> activated : Boolean
activate()
deactivate()
<<icontrol>> MeasurementPlan
<<readonly>> name : String
<<readonly>> activated : Boolean
<<readwrite>> deferred : TimeType
listStores()
createStorage()
setStorage()
listPoints()
addPoint()
removePoint()
removeStorage()
+measurementPoint
+measurementPlan
<<icontrol>> MeasurementPlanManager
<<readwrite>> planId : String
<<readwrite>> startTime : TimeType
<<readonly>> activated : Boolean
createPlan()
listPlans()
start()
suspend()
stop()
MeasurementPointSequence
MeasurementPlanSequence

Figure 9-64 – Measurement Facilities Overview

Types and Exceptions

Note – Issue 7878 Resolution (Renamed "Measurement" type to "MeasurementType")

● `MeasurementSequence`	The MeasurementSequence type represents an unbounded sequence of MeasurementTypes.
● `MeasurementPlanSequence`	The MeasurementPlanSequence type represents an unbounded sequence of MeasurementPlans.
● `MeasurementPointSequence`	The MeasurementPointSequence type represents an unbounded sequence of MeasurementPoints.
● `MeasurementStorageSequence`	The MeasurementStorageSequence type represents an unbounded sequence of MeasurementStorages.

Note – Issue 7878 Resolution (Renamed “Measurement” type to “MeasurementType”)

9.2.3.1 MeasurementType

Description

Note – Issue 7878 Resolution (Renamed “Measurement” type to “MeasurementType”)

MeasurementType represents the information captured or measured for a MeasurementPoint.

Attributes

● `sourceId: String`	The sourceId attribute represents the component that contains the measurement point that made the measurement.
● `pointId: String`	The pointId attribute represents the measurement point that made the measurment.
● `timeStamp TimeType`	The timeStamp represents the time the measurement was made.
● `data: Properties`	The data attribute represents the measurement data. The measurement point dataType attribute indicates the type of measurement data captured in the measurement.

Semantics

Note – Issue 7878 Resolution (Renamed “Measurement” type to “MeasurementType”)

As MeasurementPoints are activated they create MeasurementTypes which are recorded in a MeasurementStorage.

9.2.3.2 MeasurementPlan

Description

The Measurement Plan interface is used to manage a measurement plan, to configure it, and to manage its measures.

Attributes

- `<<readonly>name: String`
  The name attribute is the name of the measurement plan.
- `<<readonly>>activated: Boolean`
  The activated attribute indicates if the plan is activated. A value of True indicates the plan is activated.
- `<<readwrite>>deferred: TimeType`
  The deferred attribute represents when the activation should take place.

Associations

- `measurementCommunicator: ITransmission [0..1]`
  IMeasurement interface may be associated with an ITransmission interface.

Operations

- `addPoint (in point: MeasurementPoint)`
  The addPoint operation shall add a MeasurementPoint to the plan.
- `createStorage (in fileName: String, return MeasurementStorage`
  The createStorage operation creates a new MeasurementStorage for a plan.
- `listPoints (return MeasurementPointSequence)`
  The listPoints operation shall return all MeasurementPoint(s) attached to this plan by either through the addPoint operation.
- `listStores (return MeasurementStoreSequence)`
  The listStoresoperation shall return all MeasurementStore(s) attached to this plan by either through the create or by the set operations.
- `removePoint (in pointId : String)`
  The removePoint operation shall remove a MeasurementPoint from the plan as specified by the input.
- `removeStorage (in storageId : String)`
  The removeStorage operation shall remove a MeasurementStorage from the plan as specified by the input.
- `setStorage (in storage: MeasurementStorage`
  The setStorage operation sets the current storage for the plan.

9.2.3.3 IMeasurementPoint

Description

The MeasurementPoint interface is used to manage a measurement point, to set and to get its configuration, to control its activation and its deactivation, and to set its storage

Attributes

- `<<readonly>> activated: Boolean`
  The activated attribute indicates if the MeasurementPoint is activated or not. A value of True means the MeasurementPoint is activated.
- `<<readonly>> identifier: String`
  The identifier attribute uniquely identifies a MeasurementPoint.
- `<<readwrite>> delay : TimeType`
  The delay attribute indicates the delay for deferred/immediate measurement.

- `<<readwrite>> storage : MeasurementStorage`

  The storage attribute indicates the current MeasurementStorage associated with measurement point.
- `<<readonly>> dataType : String`

  The dataType attribute indicates the type of data issued from Measurement Point.

Associations

Note – Issue 7878 Resolution (Renamed "Measurement" type to "MeasurementType")

- `measurement: MeasurementType [*]`

  The MeasurementTypes produced by a MeasurementPoint.

Operations

Note – Issue 7878 Resolution (Renamed "Measurement" type to "MeasurementType")

- `activate( )`

  The activate operation activates the MeasurementPoint to start collecting MeasurementTypes.

Note – Issue 7878 Resolution (Renamed "Measurement" type to "MeasurementType")

- `deactivate( )`

  The deactivate operation deactivates the MeasurementPoint from collecting MeasurementTypes.

9.2.3.4 IMeasurementPlanManager

Description

The MeasurementPlan interface is used to control a measurement plan.

Attributes

- `<readonly>> activated: Boolean`

  The activated attribute indicates if a MeasurementPlan is activated or not. A value of True indicates a plan is activated.
- `<<readwrite>> planId: String`

  The planId attribute indicates the MeasurementPlan that can be activated or is activated.
- `<<readwrite>> startTime: TimeType`

  The startTime attribute indicates the time to activate the plan.

Associations

- `measurementPlan: MeasurementPlan [*]`

  A measurement plan is associated with one MeasurementPlanManager.

Operations

- `createPlan (in name: String, return MeasurementPlan)`
  The createPlan operation shall create a MeasurementPlan with the specified input name.
- `listPlans (return MeasurementPlanSequence)`
  The listPlans operation shall return all MeasurementPlans that have been created, which have not been removed since there creation.
- `start ()`
  The start operations shall activate or restart to execute the plan as specified by the planId attribute.
- `suspend ()`
  The suspend operation shall halt the plan measurement execution.
- `stop ()`
  The stop operation shall stop the plan measurement execution.

9.2.3.5 MeasurementRecorder

Description

The MeasurementRecorder interface is used to record measurements.

Operations

Note – Issue 7878 Resolution (Renamed “Measurement” type to “MeasurementType”)

- `record:(in in_measurement: MeasurementType)`
  The record operation records a MeasurementType.

9.2.3.6 MeasurementStorage

Description

The MeasurementStorage interface is used to control and retrieve measurements.

Attributes

- `<<readwrite>> fileName: String`
  The fileName attribute indicates the name of file that actually store records.
- `<<readwrite>> storagePolicy StoragePolicyType`
  he storagePolicy attribute indicates the storage policy.

Note – Issue 7895, change primitive type to not be an integer

- `<<readwrite>> maxSize : ULong`
  The maxSize attribute indicates the allocated size for measurement storage.

Operations

- `record:()`
  The record operation enables to record a measure.

- `query (in queryProperties: Properties, return MeasurementSequence`

  The query operation enables to retrieve a set of measurements based upon the input queryProperties.

- `clear ()`

  The clear operation shall clear all recorded measurements from storage.

Note – Issue 7895, change primitive type to not be an integer

---

- `truncate (in size : ULong`

  The truncate operation truncates storage file to new size. The truncate operate shall set the maxSize attribute to the input size value.

- `remove()`

  This operation deletes the storage. The component is no longer available for service.

Types and Exceptions

- `THIS SHOULD BE A BULLET WITH ALIGNMENT LIKE PREVIOUS ONES <<enumeration>>StoragePolicyType (ONE-SHOT, CIRCULAR)`

  The StoragePolicyType indicates how the storage should be performed.

9.2.4 Error Control Facilities

Error Control facilities allow the Data User (consumer) to tell the Data Provider about the protocol data units damaged or lost during transmission, and coordinate the re-transmission of those data units by the Provider. Since the Flow Control Facilities provide the User's acknowledgement (ACK) of correctly-received data units, it is closely linked to error control. The Error Control interface attributes are communicated to the component through a class of type ErrorControlType. Error Control Facilities also provides a mechanism for receiving status asynchronously by the StatusSignal interface.

Note – Issue 7878 Resolution (Renamed IErrorControl to IError_Control)

9.2.4.1 IError_Control

Description

Note – Issue 7878 Resolution (Renamed IErrorControl to IError_Control)

The IError_Control 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.

Attributes

- `<<readwrite>> errorControlParams: ErrorControlParamsType`

  This attribute defines which error control attributes are currently enabled to execute for the existing communication link.

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 requestRetransmit operation allows the user to request a retransmission of a recently arrived PDU, which contained an 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

- `ErrorControlParamsType (ARQStopWait: boolean, errorControl: boolean, forwardErrorCorrection: boolean, slidingWindowARQ: boolean)`

  The ErrorControlParamsType is a struct that defines the error control attributes that can be enabled as a part of the error control facility. It contains ARQStopWait, errorControl, forwardErrorCorrection and slidingWindowARQ Boolean attributes.

Constraints

Note – Issue 7586

If errorControl parameter of errorControlParams attribute is set to be False (no error control at all), then all other parameters of the errorControlParams shall be set to False.

9.2.4.2 IStatusSignal

Description

Note – Issue 7657 (2nd Ballot)

The IStatusSignal interface provides a mechanism to asynchronously indicate a status from one component to another component. Figure 9-65 shows the definition of IStatusSignal.

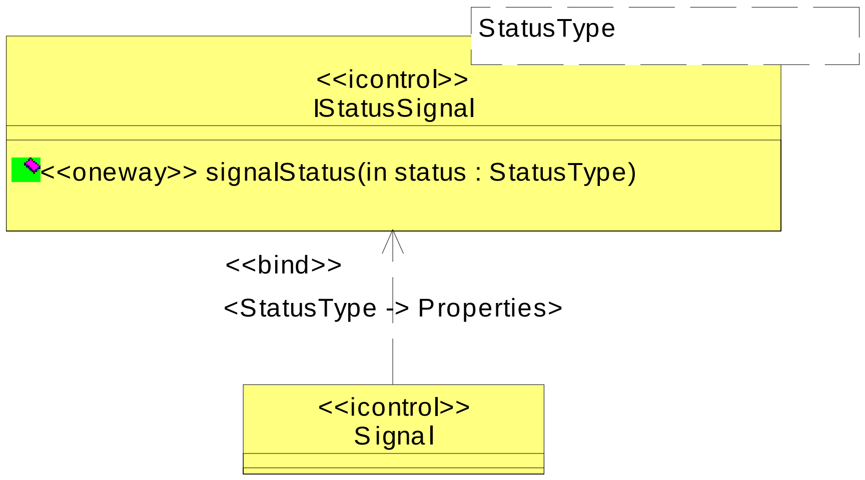

StatusType
<<icontrol>>
IStatusSignal
<<oneway>> signalStatus(in status : StatusType)
<<bind>>
<StatusType -> Properties>
<<icontrol>>
Signal

Figure 9-65 – IStatusSignal Definition

Operations

Note – Issue 7657 (2nd Ballot)

- `<<oneway>>signalStatus(in status : statusType)`

  The signalStatus operation provides a mechanism to send a status.

Semantics

The StatusSignal is a template interface. To use this interface one must form a new interface by binding to this interface with a specific StatusType.

9.2.4.3 Signal

Description

The Signal interface provides a generic mechanism to asynchronously indicate a status from one component to another component. Figure 9-65 shows the definition of Signal.

Operations

- `<<oneway>>signalStatus( )`

  The signalStatus operation provides a mechanism to send a status.

Semantics

The StatusSignal is a template interface. To use this interface one must form a new interface by binding to this interface with a specific StatusType.

9.2.5 Protocol Data Unit Facilities

This facility defines the Protocol Data Unit (PDU) concept that can be used as the smallest data unit element in any waveform layer. PDUs 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. 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. The PDU facilities define IBasePdu, ISimplePdu, IPdu, IDataPdu and IPriorityPdu interfaces as shown in Figure 9-66. In order to provide flexibility of usage, those interfaces are specified as parametrized classes. This package also provides two concrete interface recommendations that realize IDataPdu and IPdu through binding concrete data types as parameters. The operations and attributes for the interfaces are not shown in Figure 9-66.

9.2.5 Protocol Data Unit Facilities

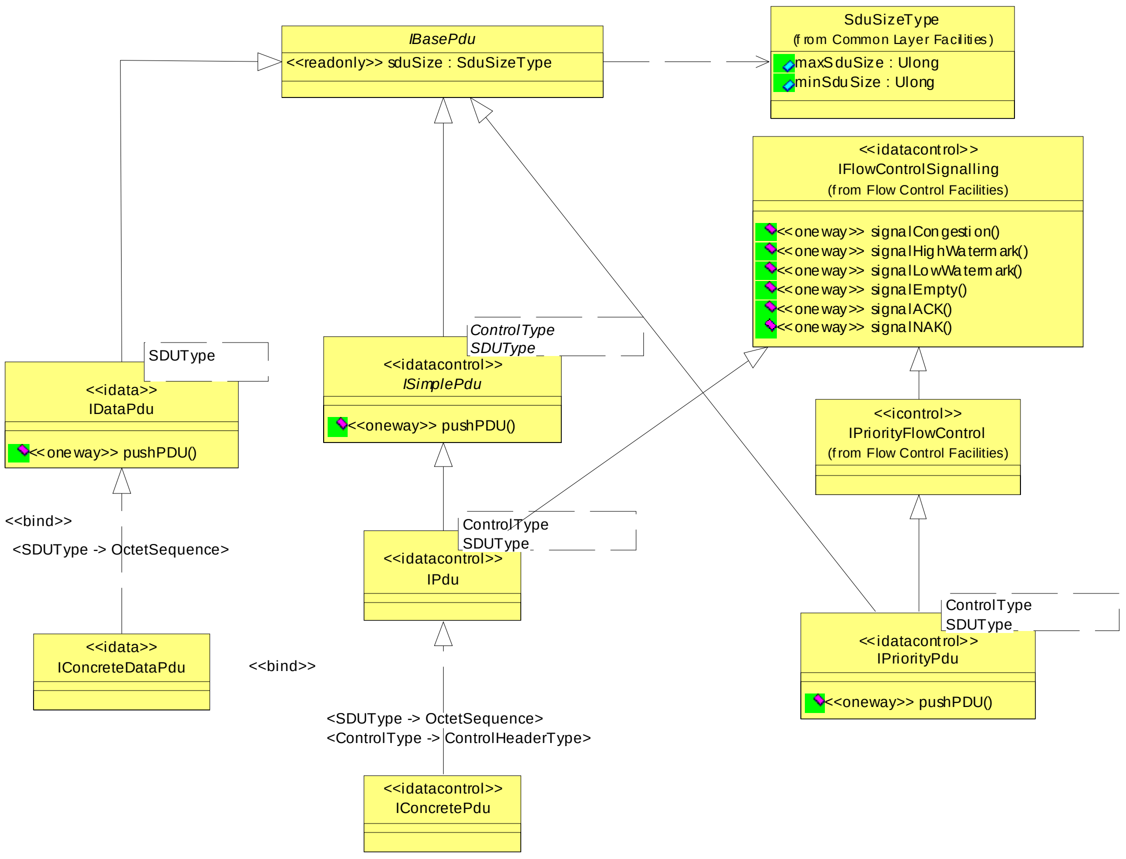


Figure 9-66 – PDU Facilities Overview

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

Note – Issue 7789 IPdu specialization

9.2.5.1 IBasePdu

Description

The IBasePdu interface is an abstract class that can be specialized by any PDU definition, whether it is used for data, control, or both. This class forms the basis for ISimplePdu and IPriorityPdu interfaces. It only defines common attributes that any PDU can have, and does not specify any operations. Those types are defined as dependencies of the IBasePdu interface. This interface can be used for both vertical and horizontal communication links. Figure 9-66 shows the definition of IBasePdu interface.

Attributes

- `<<readonly>> SduSize`

  The SduSize attribute is of type SduSizeType and it specifies the minimum and maximum payload size that can be stored in a single PDU.

Types and Exceptions

- `SduSizeType`

  SduSizeType defines the maxSduSize and minSduSize attributes as positive longs. Those two values together define the range of values sduSize can take.

9.2.5.2 ISimplePdu

Description

The ISimplePdu interface is a parametrized class that specializes the IBasePdu interface and adds a pushPDU behavior to it. ISimplePdu interface is shown in Figure 9-66.

Operations

Note – Issue 7658

- `<<oneway>>pushPDU(in control : ControlType, in sdu : SDUType )`

  The pushPDU interface is used to create and send protocol data units through the existing communication link.

9.2.5.3 IPdu

Description

Note – Issue 7789

The IPdu interface is a parametrized class which specializes ISimplePdu interface and can be implemented using different header and service data unit (SDU) types. IPdu interface also specializes IFlowControlSignalling interface, so it supports flow control signalling. This interface can be used for both vertical and horizontal communication links. Figure 9-66 shows the definition of IPdu interface.

9.2.5.4 IPriorityPdu

Description

The IPriorityPdu interface is a parameterized class that specializes the IBasePdu and IPriorityFlowControl interfaces. A component realizing the IPriorityPdu interface shall contain priority queuing behavior besides the functionalities of an IPdu interface. IPriorityPdu also defines a pushPDU behavior which also takes priority information into account. IPriorityPdu interface is shown in Figure 9-66.

Operations

Note – Issue 7658

- `<<oneway>>pushPDU(in priority : Octet, in control : ControlType, in sdu : SDUType )`
  The pushPDU interface is used to create and send protocol data units through the existing communication link.

9.2.5.5 IDataPdu

Description

The IBasePdu interface is a parametrized class which specializes IBasePdu interface and can be implemented using different SDU types. IDataPdu does not have any header information, so it can be used when there is a stream of data is to be transferred in frames, with no header requirements. This interface can be used for both vertical and horizontal communication links. Figure 9-66 shows the definition of IDataPdu interface.

Operation

Note – Issue 7658

- `<<oneway>>pushPDU(in sdu : SDUType )`
  The pushPDU interface is used to create and send protocol data units through the existing communication link.

9.2.5.6 IConcretePdu

Description

IConcretePdu interface realizes the IPdu by binding the SDUType with UML Profile for SWRadio::Application and Device Components::BaseTypes::OctetSequence and ControlType with ControlHeaderType.

Types and Exceptions

Note – Issue 7895, change primitive type to not be an integer

- `ControlHeaderType (sourceAddress: AddressType, destinationAddress: AddressType, priority: UShort, sduSize: sduSizeType, sequenceNumber: ULong)`
  ControlHeaderType is defined in this package in order to provide a concrete PDU definition. This class defines the sourceAddress and destionationAddress fields for the PDU, priority attribute as an ULong, sduSize as the allowed minimum and maximum values and the sequenceNumber, which shows the sequence number of a PDU in a given stream of data packets.

9.2.5.7 IConcreteDataPdu

Description

IConcreteDataPdu interface provides a concrete interface by realizing the parameterized IDataPdu and binding the SDUType with UML Profile for SWRadio::Application and Device Components::BaseTypes::OctetSequence.

9.2.6 Stream Facilities

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.

9.2.6.1 IStream

Description

The IStream interface, as shown in Figure 9-67, 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

Note – Issue 7895 Changed .priority to UShort

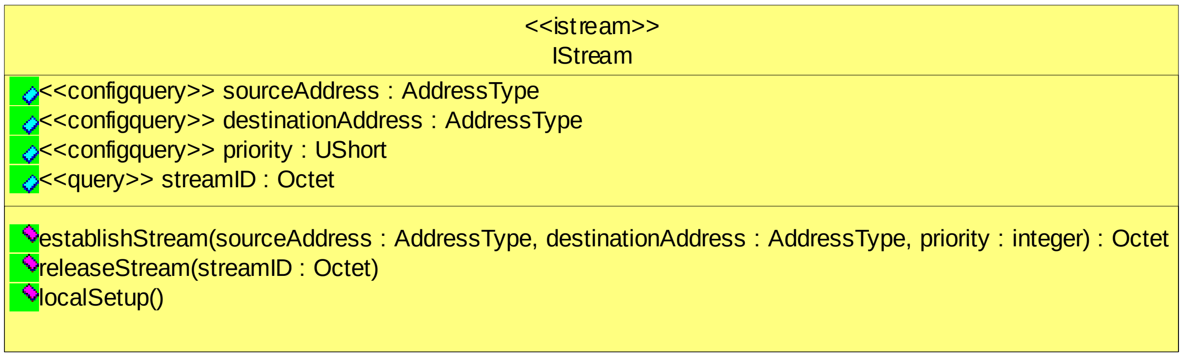

<<istream>>
IStream
<<configquery>> sourceAddress : AddressType
<<configquery>> destinationAddress : AddressType
<<configquery>> priority : UShort
<<query>> streamID : Octet
establishStream(sourceAddress : AddressType, destinationAddress : AddressType, priority : integer) : Octet
releaseStream(streamID : Octet)
localSetup()

Figure 9-67 – IStream Definition

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

Attributes

- `<<configquery>> sourceAddress : AddressType`
  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.
- `<<configquery>> destinationAddress : AddressType`
  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.

Note – Issue 7895, change primitive type to not be an integer

- `<<configquery>> priority : UShort`

  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.

- `<<query>> streamID : Octet`

  streamID attribute is the unique ID that the system assigns to the stream. This attribute can only be queried, since it is set by the establishStream operation.

Operations

Note – Issue 7895, change primitive type to not be an integer

- `establishStream (in sourceAddress : AddressType, in destinationAddress : AddressType, in priority : UShort, return streamID : Octet)`

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

- `releaseStream (in streamID : Octet)`

  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.

- `localSetup( )`

  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

- `AddressType (address : OctetSequence)`

  AddressType provides a definition for a generic addressing mechanism.

9.3 Data Link Layer Facilities

9.3.1 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.

9.3.1.1 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.

9.3.1.1.1 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 9-68. Every logical link is referenced by a ConnectionID that describes the service SAPs that the link is bound to.

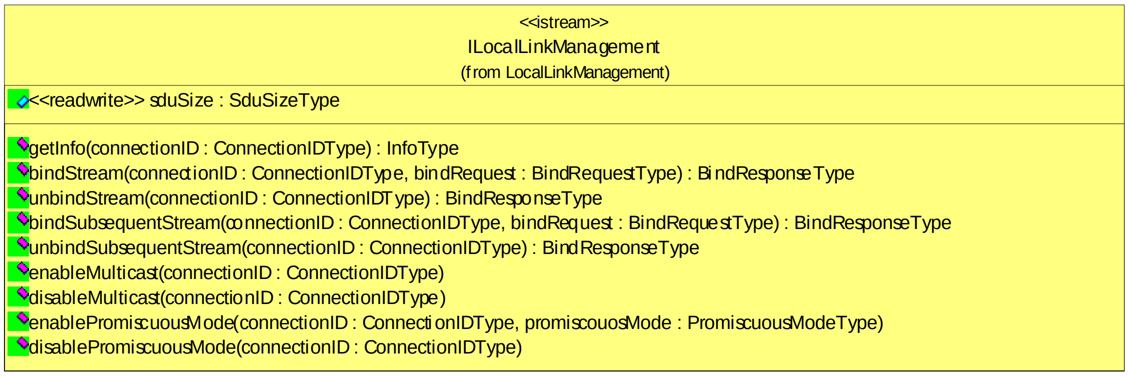

<<istream>>
ILocalLinkManagement
(from LocalLinkManagement)
<<readwrite>> sduSize : SduSizeType
getInfo(connectionID : ConnectionIDType) : InfoType
bindStream(connectionID : ConnectionIDType, bindRequest : BindRequestType) : BindResponseType
unbindStream(connectionID : ConnectionIDType) : BindResponseType
bindSubsequentStream(connectionID : ConnectionIDType, bindRequest : BindRequestType) : BindResponseType
unbindSubsequentStream(connectionID : ConnectionIDType) : BindResponseType
enableMulticast(connectionID : ConnectionIDType)
disableMulticast(connectionID : ConnectionIDType)
enablePromiscuousMode(connectionID : ConnectionIDType, promiscouosMode : PromiscuousModeType)
disablePromiscuousMode(connectionID : ConnectionIDType)

Figure 9-68 – ILocalLinkManagement Definition

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

Note – Issue 7725 - 9.3.1.1.1 IlocalLinkManagement Unbind

Note – Issue 7729 - Replace void in getInfo operation with a return type.

Attributes

- `<<readwrite>> sduSize : sduSizeType`

  This attribute specifies the minimum and maximum service data unit size the LLC resource can transfer. If incoming data is less that this amount, the LLC resource waits to transmit until more data comes in. (or until timeout occurs, depending on the implementation)

Operations

Note – Issue 7658

Note – ISSUE 7729 - IlocalLinkManagement: The getInfo operation does return any information and exceptions listed in text description are undefined.

- `getInfo(in connectionID : ConnectionIDType, return InfoType): {raises = (InvalidPort, SystemError)}`

  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 (as defined in Section 8.1.5.1.4) or SystemException (as defined in Section 8.1.1.26).

Note – Issue 7658

- `bindStream(in connectionID : ConnectionIDType, in bindRequest : BindRequestType, return BindResponseType)`

  The bindStream operation associates a 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.

Note – Issue 7658

- `bindSubsequentStream(in connectionID : ConnectionIDType, in bindRequest : BindRequestType, return 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 and exception.

Note – Issue 7658

Note – Issue 7725 - 9.3.1.1.1 IlocalLinkManagement

- `unbindStream(in connectionID : ConnectionIDType)`

  The 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.

Note – Issue 7658

Note – Issue 7725 - 9.3.1.1.1 IlocalLinkManagement

- `unbindSubsequentStream(in connectionID : ConnectionIDType)`

  The unbindSubsequentStream requests the logical link provider to unbind the subsequently bound SAP.

Note – Issue 7658

- `enableMulticast(in connectionID : ConnectionIDType)`

  enableMulticast operation requests the logical link provider to enable specific multicast addresses on a per stream basis.

Note – Issue 7658

- `disableMulticast(in connectionID : ConnectionIDType)`

  disableMulticast operation requests the logical link provider to disable specific multicast addresses on a per stream basis.

Note – Issue 7658

- `enablePromiscuousMode(in connectionID : ConnectionIDType, in promiscouosMode : PromiscuousModeType)`

  This operation requests the provider to enable promiscuous mode on a per Stream basis, either at the physical level or at the SAP level.

Note – Issue 7658

- `disablePromiscuousMode(in connectionID : ConnectionIDType)`

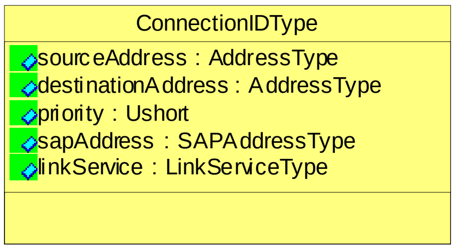
  This operation requests the provider to disable promiscuous mode on a per Stream basis.

Types and Exceptions

- `<<enumeration>> PromisciousModeType (PHYSICAL, SAP, MULTIPLE)`

  Promiscuous mode can be enabled on a per connection basis, either at the physical level (PHYSICAL) or at the SAP level, or at a multiple (MULTIPLE) level.
- `ConnectionIDType (sourceAddress: AddressType, destinationAddress: AddressType, priority: UShort, sapAddress: SAPAddressType, linkService: LinkServiceType)`

  ConnectionIDType class completely specifies a logical link that is established at the LLC layer. It specifies the sourceAddress and destionationAddress for radio sets, the sapAddress that the logical link is bound to within the local radio set, as well as the linkService type (connection, connectionless, ack connectionless).


ConnectionIDType
sourceAddress : AddressType
destinationAddress : AddressType
priority : Ushort
sapAddress : SAPAddressType
linkService : LinkServiceType

- `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. The attributes of BindRequestType are: sapAddress, maxConnectionId, linkService (type of link service, connection, ack connection or connectionless), isListenStream (Boolean), autoXID (Boolean), autoTest (Boolean)
- `BindResponseType`

  This class defines the BindResponse header attributes. The attributes of BindResponseType are: sapAddress, maxConnectionId, autoXID (Boolean), autoTest (Boolean)

9.3.1.2 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, IConnectionlessLink 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.

9.3.1.2.1 ConnectionlessLink Component

Description

Note – Issue 7726 (change below)

ConnectionlessLink component as shown in Figure 9-69, 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 ConnectionlessLink component for unacknowledged connectionless communication scenario. This component realizes the IQualityOfServiceConnectionless interface for quality of service related facilities, IFlowControlSignaling for flow control interfaces, and IIndicator and IRequestPdu interfaces for PDU based communication. Connectionless link component 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 port(s) that the link is bound to. Local link management interface establishes the links and bounds them to vertical (internal) streams. This component can have a user role, a provider role, or both; depending on the waveform scenario.

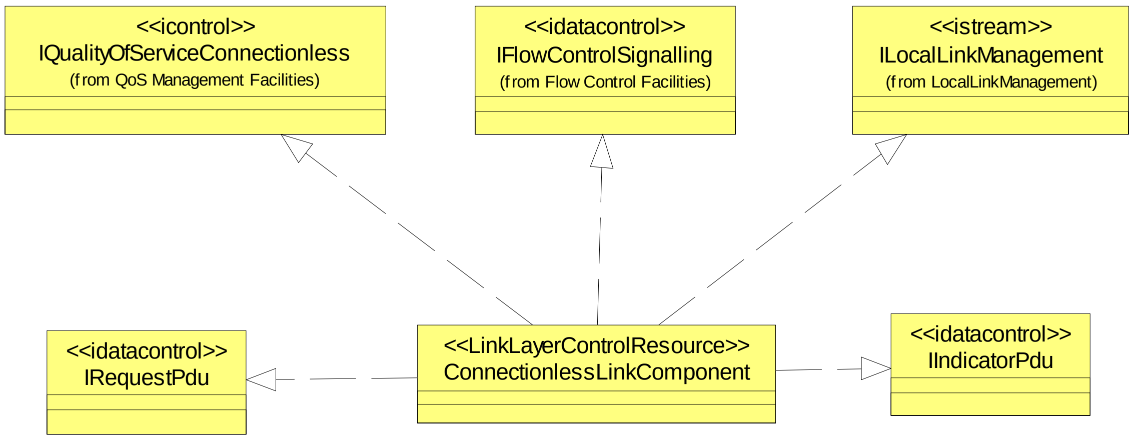

<<icontrol>>
IQualityOfServiceConnectionless
(from QoS Management Facilities)
<<idatacontrol>>
IFlowControlSignalling
(from Flow Control Facilities)
<<istream>>
ILocalLinkManagement
(from LocalLinkManagement)
<<idatacontrol>>
IRequestPdu
<<LinkLayerControlResource>>
ConnectionlessLinkComponent
<<idatacontrol>>
IIndicatorPdu

Figure 9-69 – ConnectionlessLink Component Definition

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

Note – Issue 7726 ConnectionlessLink Component: realize IFlowControlSignalling instead of IFlowControlManagement

Associations

- `localLinkManager: ILocalLinkManagement [1]`
  ILocalLinkManagement interface can act as a local link manager to the links that are used by the IConnectionlessLink interface.

Types and Exceptions

Note – Issue 7895 - correct the types

- `ConnectionIDType: (sourceAddress: AddressType, destinationAddress: AddressType, priority: UShort, sapAddress: SAPAddressType, linkService: LinkServiceType)`
  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)

Constraints

Note – Issue 7584: Specify which interfaces are mandatory

ConnectionlessLinkComponent shall provide one ControlPort, at least one input DataControl port and at least one output DataControl port.

9.3.1.2.2 IIndicatorPdu

Description

IIndicator interface, as shown in Figure 9-70, realizes the IPdu interface from the Common Layer Facilities::PDU Facilities, by binding ControlHeaderType to MediumAccessControlHeaderType and SDUType to OctetSequence.

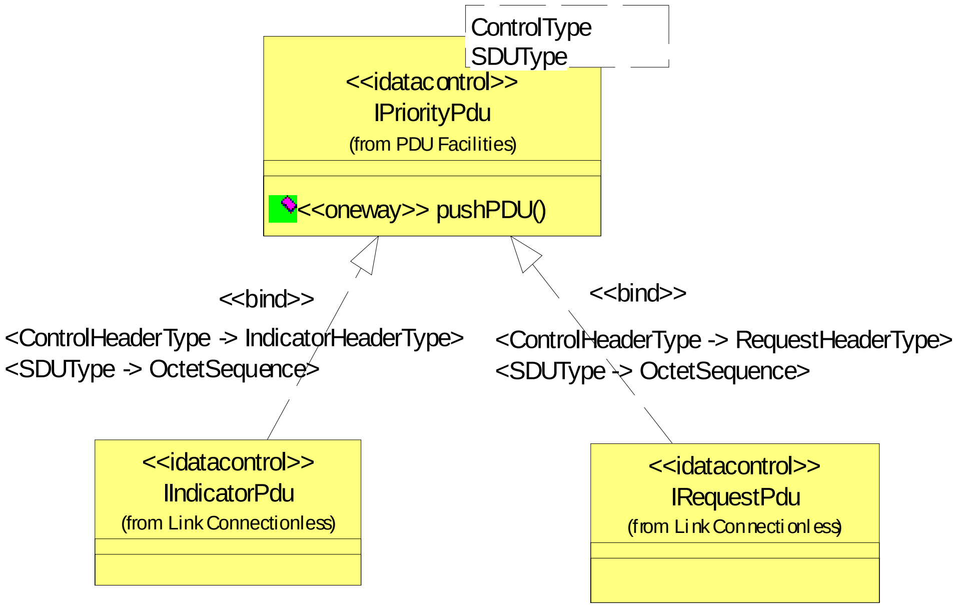

ControlType
SDUType
<<idatacontrol>>
IPriorityPdu
(from PDU Facilities)
<<oneway>> pushPDU()
<<bind>>
<ControlHeaderType -> IndicatorHeaderType>
<SDUType -> OctetSequence>
<<bind>>
<ControlHeaderType -> RequestHeaderType>
<SDUType -> OctetSequence>
<<idatacontrol>>
IIndicatorPdu
(from Link Connectionless)
<<idatacontrol>>
IRequestPdu
(from Link Connectionless)

Figure 9-70 – IIndicatorPdu and IRequestPdu Definitions

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

Types and Exceptions

- `IndicatorHeaderType (isGroupAddress: Boolean)`

  Indicator header is used to conveys one SDU from the LLC provider to the LLC user. IndicatorHeaderType inherits from ControlHeaderType defined in the Common Layer Facilities::PDU Facilities package.

  isGroupAddress attribute defines whether the destination address is a group address.

9.3.1.2.3 IRequestPdu

Description

IRequestPdu interface, as shown in Figure 9-70, realizes the IPdu interface from the Common Layer Facilities::PDU Facilities, by binding ControlHeaderType to MediumAccessControlHeaderType and SDUType to OctetSequence.

Types and Exceptions

● `RequestHeaderType`	This header type conveys one SDU from the LLC user to the LLC provider for transmission to a peer LLC user. RequestHeaderType inherits from ControlHeaderType defined in the Common Layer Facilities::PDU Facilities package.

9.3.1.3 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.

9.3.1.3.1 IAckConnectionless

Description

IAckConnectionlessLink interface as shown in Figure 9-71, provides the extra functionality to control parameters that are related to acknowledged connectionless link establishment and management.

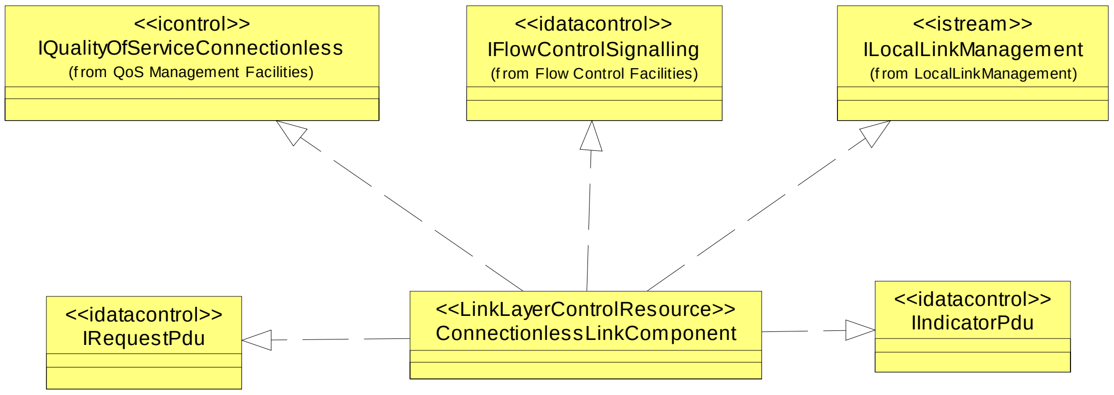

<<icontrol>>
IQualityOfServiceConnectionless
(from QoS Management Facilities)
<<idatacontrol>>
IFlowControlSignalling
(from Flow Control Facilities)
<<istream>>
ILocalLinkManagement
(from LocalLinkManagement)
<<idatacontrol>>
IRequestPdu
<<LinkLayerControlResource>>
ConnectionlessLinkComponent
<<idatacontrol>>
IIndicatorPdu

Figure 9-71 – IAckConnectionlessLink Definition

Note – Issue 7661: Lack Consistent use of SWRadio Stereotypes in Facilities interface

---

Note – Issue 7727: Replace specialization of ConnectionlessLinkComponent with realization of IQualityOfServiceConnectionless, IFlowControlSignaling, and ILocalLinkManagement.

Operations

Note – Issue 7658

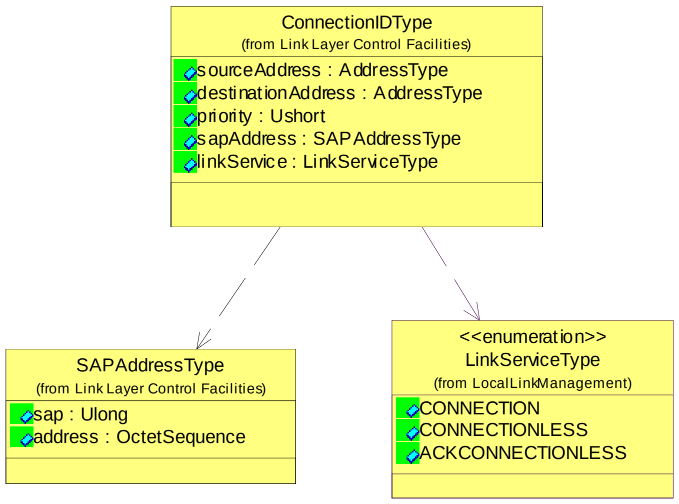
- `ackReception (in sequenceNumber : Octet)`
  Acknowledgement of received PDU.
- `nakReception(in sequenceNumber : Octet)`
  Negative acknowledgement of PDU. This operation indicates that an expected data packet was not received at all, or it was received in error.

Types and Exceptions

Note – Issue 7895, fix types

- `ConnectionIDType (sourceAddress: AddressType, destinationAddress: AddressType, priority: UShort, sapAddress: SAPAddressType, linkService: LinkServiceType)`
  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 pa-

rameter 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 Layer Control Facilities)
sourceAddress : AddressType
destinationAddress : AddressType
priority : Ushort
sapAddress : SAPAddressType
linkService : LinkServiceType
SAPAddressType
(from Link Layer Control Facilities)
sap : Ulong
address : OctetSequence
<<enumeration>>
LinkServiceType
(from LocalLinkManagement)
CONNECTION
CONNECTIONLESS
ACKCONNECTIONLESS

- `<<enumeration>> PacketIndicatorType (PI_ONEWAY, PI_TWOWAY)`

PacketIndicatorType specifies whether one way or two way packet indication will be used.

9.3.1.3.2 IAckReplyPdu

Description

IAckReplyPdu interface, as shown in Figure 9-72, realizes the IPriorityPdu interface from the Common Layer Facilities::PDU Facilities, by binding ControlHeaderType to RequestHeaderType and SDUType to OctetSequence. This PDU interface shall be used when replying to a data request.

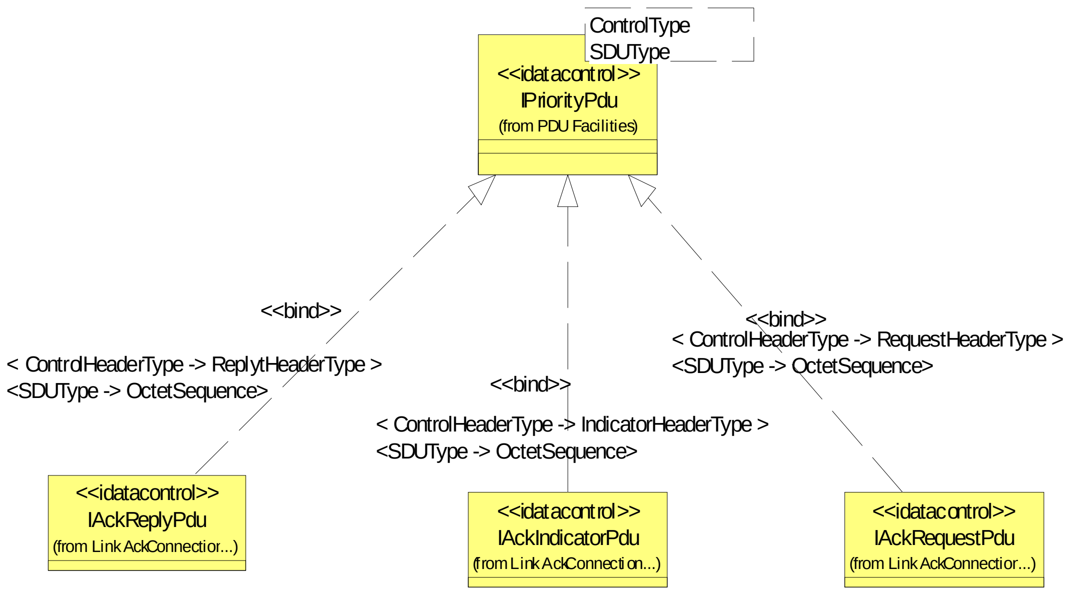

ControlType
SDUType
<<idatacontrol>>
IPriorityPdu
(from PDU Facilities)
<<bind>>
< ControlHeaderType -> ReplytHeaderType >
<SDUType -> OctetSequence>
<<bind>>
< ControlHeaderType -> IndicatorHeaderType >
<SDUType -> OctetSequence>
<<bind>>
< ControlHeaderType -> RequestHeaderType >
<SDUType -> OctetSequence>
<<idatacontrol>>
IAckReplyPdu
(from LinkAckConnectior...)
<<idatacontrol>>
IAckIndicatorPdu
(from LinkAckConnection...)
<<idatacontrol>>
IAckRequestPdu
(from LinkAckConnectior...)

Figure 9-72 – IAckReplyPdu, IAckIndicatorPdu and IAckRequestPdu Definitions

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

Note – Issue 7728 - AckReplyPdu definition: Change RequestHedaerType to ReplyHeaderType in IAckReplyPdu bind definition

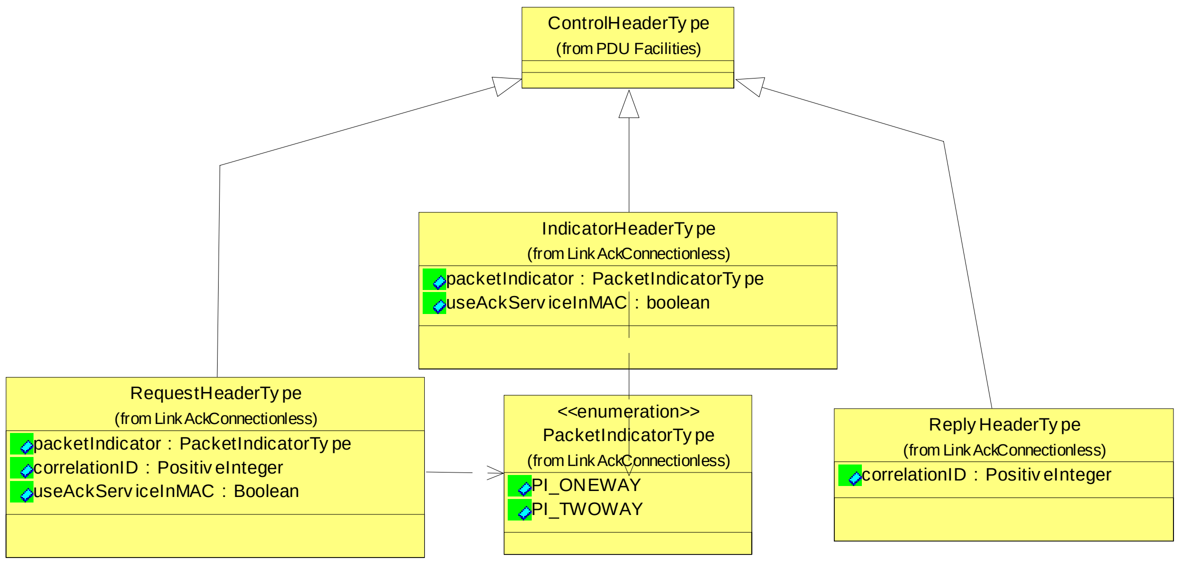

ControlHeaderType
(from PDU Facilities)
IndicatorHeaderType
(from LinkAckConnectionless)
packetIndicator : PacketIndicatorType
useAckServiceInMAC : boolean
RequestHeaderType
(from LinkAckConnectionless)
packetIndicator : PacketIndicatorType
correlationID : PositiveInteger
useAckServiceInMAC : Boolean
<<enumeration>>
PacketIndicatorType
(from LinkAckConnectionless)
PI_ONEWAY
PI_TWOWAY
ReplyHeaderType
(from LinkAckConnectionless)
correlationID : PositiveInteger

Figure 9-73 – IAckConnectionLink Header Types

Types and Exceptions

- `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. replyHeaderType is shown in Figure 9-73.

9.3.1.3.3 IAckIndicatorPdu

Description

IAckIndicatorPdu interface, as shown in Figure 9-72, realizes the IPriorityPdu interface from the Common Layer Facilities::PDU Facilities, by binding ControlHeaderType to IndicatorHeaderType and SDUType to OctetSequence. This PDU interface shall be used when indicating successful or unsuccessful data request or transfer.

Types and Exceptions

- `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. indicatorHeaderType is shown in Figure 9-73.

9.3.1.3.4 IAckRequestPdu

Description

IAckRequestPdu interface, as shown in Figure 9-72, realizes the IPriorityPdu interface from the Common Layer Facilities::PDU Facilities, by binding ControlHeaderType to RequestHeaderType and SDUType to OctetSequence. This PDU interface shall be used when indicating successful or unsuccessful data request or transfer.

Types and Exceptions

- `requestHeaderType`

  (packetIndicator: PacketIndicatorType, correlationID: ULong, useAckServiceInMac: Boolean)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. requestHeaderType is shown in Figure 9-72.

9.3.1.3.5 AckConnectionlessLinkComponent

Description

Note – Issue 7727

AckConnectionlessLinkComponent as shown in Figure 9-71, realizes IAckConnectionlessLink, IErrorControl, IQualityOfServiceConnectionless, IFlowControlSignaling, and ILocalLinkManagement interfaces. With those relationships, this component provides functionality to control parameters that are related to acknowledged connectionless link establishment and management. After the connection is established, data can be transferred using IAckIndicatorPdu, IAckRequestPdu and IackReplyPdu interfaces for acknowledged connectionless communication scenario. IErrorControl is realized for detecting and reporting errors in the reception or transmission. Acknowledged connectionless link component may realize 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 port(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.

Constraints

Note – Issue 7584: Specify which interfaces are mandatory

AckConnectionlessLinkComponent shall provide one ControlPort, at least one input DataControl port and at least one output DataControl port.

9.3.1.4 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.

9.3.1.4.1 IConnectionLink

Description

IConnectionLink interface as shown in Figure 9-74, 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 SAPs 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.

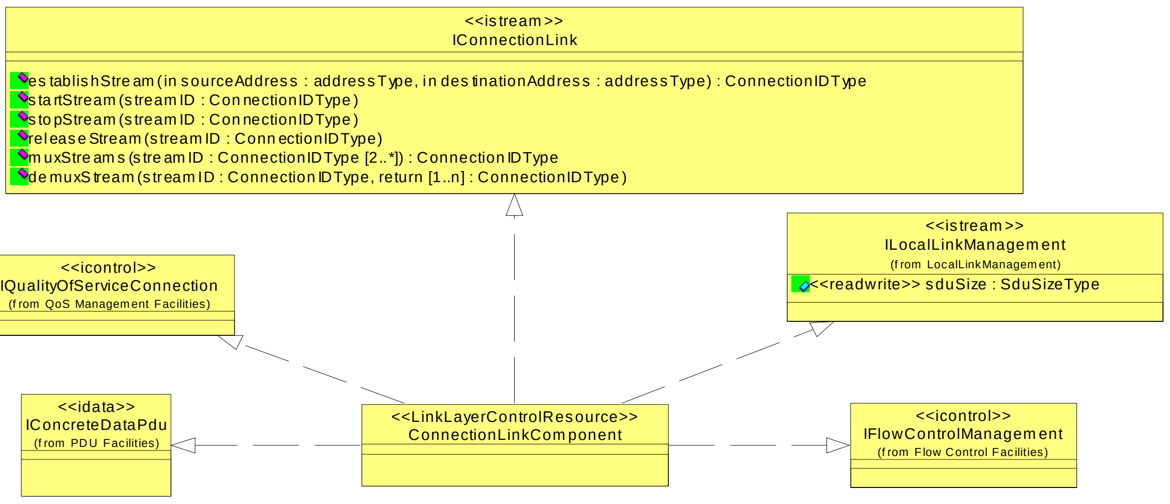

<<istream>>
IConnectionLink
establishStream(in sourceAddress : addressType, in destinationAddress : addressType) : ConnectionIDType
startStream(streamID : ConnectionIDType)
stopStream(streamID : ConnectionIDType)
releaseStream(streamID : ConnectionIDType)
muxStreams(streamID : ConnectionIDType [2..*]) : ConnectionIDType
demuxStream(streamID : ConnectionIDType, return [1..n] : ConnectionIDType)
<<icontrol>>
IQualityOfServiceConnection
(from QoS Management Facilities)
<<istream>>
ILocalLinkManagement
(from LocalLinkManagement)
<<readwrite>> sduSize : SduSizeType
<<idata>>
IConcreteDataPdu
(from PDU Facilities)
<<LinkLayerControlResource>>
ConnectionLinkComponent
<<icontrol>>
IFlowControlManagement
(from Flow Control Facilities)

Figure 9-74 – IConnectionLink Definition

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

---

Note – Issue 7787 IConnectionLink Operations: establishStream operation should take source and destionation address as input parameters. Furthermore, muxStreams should return a streamID that is a reference to the multiplexed stream. demuxSteams should return the streamID's of the demultiplexed streams.

Operations

Note – Issue 7658

Note – Issue 7787

- `establishStream(in sourceAddress : AddressType, in destinationAddress : AddressType, return : ConnectionIDType)`

  This operation allows the LLC service user to initialize a stream.

Note – Issue 7658

- `startStream(in streamID : ConnectionIDType)`

  This operation starts data transfer through a previously established stream

Note – Issue 7658

- `stopStream(in streamID : ConnectionIDType)`

  This operation stops data transfer through the given stream

Note – Issue 7658

- `releaseStream(in streamID : ConnectionIDType)`

  The releaseStream operation destroys the stream and releases all of the resources associated with it.

Note – Issue 7658

Note – Issue 7787

- `muxStreams(in streamID [2..n] : ConnectionIDType, return ConnectionIDType)`

  This operation multiplexes multiple (two or more) streams into a single stream. This can be done by both the receiving or transmitting entity.

Note – Issue 7658

Note – Issue 7787

- `demuxStream(in streamID : ConnectionIDType, return [1..n] : ConnectionIDType)`

  This operation demultiplexes a stream that is composed of multiple data streams.

Types and Exceptions

Note – Issue 7895, fix types

- `ConnectionIDType (sourceAddress: AddressType, destinationAddress: AddressType, priority: UShort, sapAddress: SAPAddressType, linkService: LinkServiceType)`

  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)

9.3.1.4.2 ConnectionLinkComponent

Description

ConnectionLinkComponent as shown in Figure 9-74, 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, IFlowControlManagement for flow control interfaces, ILocalLinkManagement for link management tasks, IConnectionLink for managing connection oriented streams and IConcreteDataPdu for transferring data over a stream on a frame-by-frame basis with no control information.. Every logical link is referenced by a ConnectionID that describes the SAPs 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

Constraints

Note – Issue 7584: Specify which interfaces are mandatory

ConnectionLinkComponent shall provide one ControlPort and at least one StreamPort.

9.3.2 Medium Access Control Layer Facilities

This section defines the MAC Layer facilities. MAC Layer provides facilities to upper layers, for both data transmission and control purposes. In that manner, LLC layer, Radio resource Control (RRC) layer, and other layers that can by-pass the waveform stack to communicate with the MAC layer. MAC layer uses the facilities offered by the physical layer in order to perform medium access control tasks. DLPI specification, OSI reference model X.200e, IEEE 802 series, 3GPP UMTS and GSM specifications were used when defining this interface.

The 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. A MAC component communicates with a SAP in order to transfer data and control information between components within the same radio set. (Vertical communication) It also provides interfaces to communicate with the remote radio set MAC layer. (Horizontal communication)

Due to the complexity and variety of waveforms, defining a single MAC API capable of satisfying all waveform requirements would result in significant processing and memory inefficiencies. For these reasons, most of the main MAC layer interface is defined as a bundle of building blocks as defined in the Common Layer Facilities. Several services provided by a MAC interface is listed as follows:

- Flow control and priority queueing (from Common Layer Facilities::Flow Control Facilities)
- Quality of Service (from Common Layer Facilities::Quality of Service Facilities)
- Error Control (from Common Layer Facilities::Error Control Facilities)
- Measurement and reporting of requested traffic parameters (from Common Layer Facilities::Measurement Facilities)

- Handling of data and control channels
- Scheduling of transmission (Common Radio Facilities::Scheduling Facilities)
- Reordering, Assembly, Multiplexing of data

Figure 9-75 shows an example medium access layer component definition for a CDMA system. The example CDMA parameter type is bound to the MediumAccessParameter definition.

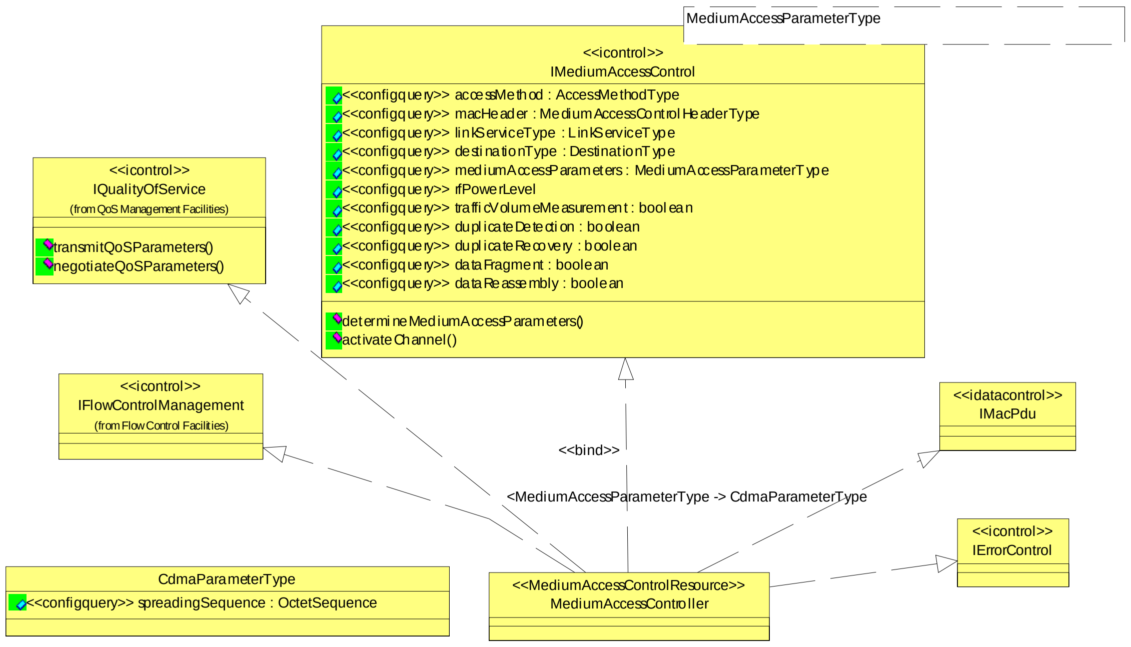

MediumAccessParameterType
<<icontrol>>
IMediumAccessControl
<<configquery>> accessMethod : AccessMethodType
<<configquery>> macHeader : MediumAccessControlHeaderType
<<configquery>> linkServiceType : LinkServiceType
<<configquery>> destinationType : DestinationType
<<configquery>> mediumAccessParameters : MediumAccessParameterType
<<configquery>> rfPowerLevel
<<configquery>> trafficVolumeMeasurement : boolean
<<configquery>> duplicateDetection : boolean
<<configquery>> duplicateRecovery : boolean
<<configquery>> dataFragment : boolean
<<configquery>> dataReassembly : boolean
determineMediumAccessParameters()
activateChannel()
<<icontrol>>
IQualityOfService
(from QoS Management Facilities)
transmitQoSParameters()
negotiateQoSParameters()
<<icontrol>>
IFlowControlManagement
(from Flow Control Facilities)
<<idatacontrol>>
IMacPdu
<<bind>>
<MediumAccessParameterType -> CdmaParameterType
<<icontrol>>
IErrorControl
CdmaParameterType
<<configquery>> spreadingSequence : OctetSequence
<<MediumAccessControlResource>>
MediumAccessController

Figure 9-75 – MAC Facilities Overview

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

9.3.2.1 IMediumAccessControl

Description

The IMediumAccessControl Facility as shown in Figure 9-75 is a parameterized interface that defines operations for activating (or selecting) a transport channel and setting the control mode of the medium access parameters. The MediumAccessParameterType is the parameter type that is dependent on the physical medium and the multiple access mechanism. The IMediumAccessControl interface is realized by the MediumAccessController component.

For the best software re-use practice, several facilities that are provided by a MAC component are defined as common interfaces in the common layer facilities. Those facilities can be realized by components other than a MAC component, and they may be used in a non-OSI specific waveform layer implementation. The IMediumAccessControl interface provides extra MAC layer specific functionality.

Attributes

- `<<configquery>> accessMethod: AccessMethodType`
  This attribute defines the access method mechanism MAC component is using. Possible values are defined in the AccessMethodType class definition.
- `<<configquery>> macHeader: MediumAccessControlHeaderType`
  MacHeader attribute defines the in-band control parameters that will be embedded in the MAC PDU header. Possible fields are defined in the MediumAccessControlHeaderType definition.
- `<<configquery>> linkServiceType: LinkServiceType`
  This attribute provides a mechanism for setting the link service type (connectionless, ack connectionless, connection oriented). LLC layer can set this parameter, and request MAC services related to the link type.
- `<<configquery>> destinationType: DestinationType`
  This attribute determines whether the destination is a single entity (unicast), multiple entities (multicast) or the entire network of radio sets (broadcast).
- `<<configquery>> mediumAccessParameters: MediumAccessParameterType`
  This is an abstract definition of a mediumAccessParameter type. Implementation of this parameter is dependent upon the waveform that implements the MAC component. It may consist of the spreading and scrambling codes on case of WCDMA, allowed time slots for TDMA, frequency bandwidth and hop-set for hopping FDMA, etc.
- `<<configquery>> rfPowerLevel`
  rfPowerLevel attribute is used to get/set the RF power output level. MAC component can communicate the RF power level to the physical layer API, if instructed by a higher layer component. Also in certain MAC layer specifications, MAC layer has the ability to extract power control bits from the incoming MAC PDU and set the RF power level accordingly.
- `<<configquery>> trafficVolumeMeasurement: Boolean`
  This attribute specifies whether the traffic control measurement is enabled or not. Measurement parameters are communicated to the MAC layer using the Measurement Facilities.
- `<<configquery>> duplicateDetection: Boolean`
  duplicateDetection attribute is used to specify whether duplicate PDU detection is enabled in the MAC layer or not.
- `<<configquery>> duplicateRecovery: Boolean`
  duplicateRecovery attribute is used to specify whether duplicate PDU recovery is enabled in the MAC layer or not.
- `<<configquery>> dataFragment: Boolean`
  dataFragment attribute is used to specify whether data fragmenting is enabled in the transmission chain of the MAC layer or not. If data fragmenting is enabled, related parameters (SDU size etc.) should be defined in the mediumAccessParameters attribute.
- `<<configquery>> dataReassembly: Boolean`
  dataReassembly attribute is used to specify whether data reassembling is enabled in the reception chain of the MAC layer or not. If data reassembly is enabled, related parameters (SDU size etc.) should be defined in the mediumAccessParameters attribute.

Associations

- `serviceAccessPoint: IServiceAccessPoint[1..n]`
  IServiceAccessPoint is used to provide MAC service to higher waveform layers.
- `measurementProvider: IMeasurement`
  IMeasurement interface is used by other layers in order to communicate measurement requests to the MAC layer. MAC layer also uses this interface to report measurement results.

Note – Issue 7849 - IScheduling removed

Operations

Note – Issue 7658

- `determineMediumAccessParameters (return Boolean)`
  This operation is realized differently depending on the medium the waveform is trying to access to. It can be the ethernet address for an ethernet type connection, or spreading code for UMTS waveform.

Note – Issue 7658, Issue 7895, fix types

- `activateChannel (in presetNum: UShort, return 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.

Types and Exceptions

- `<<enumeration>> DestinationType`
  DestinationType class defines the type of destination that is being addressed. Possible values are:
  - UNICAST: For addressing a single recipient.
  - MULTICAST: For addressing multiple recipients.
  - BROADCAST: For addressing entire network of possible recipients.
- `AccessMethodType : String`
  This class is used to define the access method that the MAC layer is using. Some possible values are: CSMACD (Carrier Sense Multiple Access / Collision Detect), ETHER (Ethernet), ISDN, ATM, LOOP (Software Loopback), etc. For a full listing, see DLPI specification.
- `<<abstract>> MediumAccessParameterType`
  Implementation of this class is dependent upon the waveform that implements the MAC component. It may consist of the spreading and scrambling codes on case of WCDMA, allowed time slots for TDMA, frequency bandwidth and hopset for hopping FDMA, etc.

Note – Issue 7586

Semantics

Transmission Security is either implemented by the LLC or the MAC.

9.3.2.2 IMacPdu

Description

IMacPdu interface, as shown in Figure 9-76, realizes the IPdu interface from the Common Layer Facilities::PDU Facilities, by binding ControlHeaderType to MediumAccessControlHeaderType and SDUType to OctetSequence.

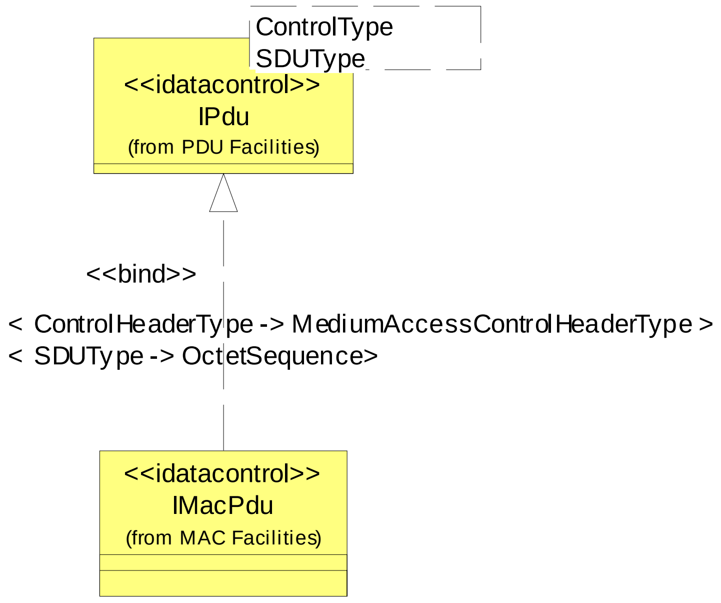


Figure 9-76 – IMacPdu Definition

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

Types and Exceptions

Note – Issue 7895, fix types

- `MediumAccessControlHeaderType (receiverAddress : AddressType, transmitterAddress : AddressType, CRC : OctetSequence, frameType : ULong, frameSubType : ULong, moreFlag : Boolean, retryFlag : Boolean, powerManagementCommands : OctetSequence, privacyKey : OctetSequence)`

MediumAccessControlHeaderType class inherits and extends the ControlHeaderType class as defined in the PDU Facilities. Attributes defined by this class are:

- receiverAddress: Address information of the receiver. This field may be different than the destionationAddress defined by the control header, in case of retransmission/bridging of a PDU over multiple radio sets before reaching its final destination.
- transmitterAddress: Address information for the transmitter. This field may be different than the sourceAddress defined by the control header, in case of retransmission/bridging of a PDU over multiple radio sets before reaching its final destination.
- CRC: cyclic redundancy check code for error checking.
- frameType: this is an abstract definition and defines the type of frame that is being transferred.
- frameSubType: this is an abstract definition and defines the sub-type of frame (if exists) that is being transferred.
- moreFlag: specifies whether there is more data that will be sent as a part of current transmission.
- retryFlag: specifies whether current packet is a retransmission or not.
- powerManagementCommands: this abstract attribute is used to convey the power management commands to the receiver. Power management is especially required in spread spectrum systems in order to overcome the near/far problem.
- privacyKey: This key is used in case transmission involves security features.

9.3.2.3 MediumAccessController Component

Description

Note – IScheduling removed

The MediumAccessController Component as shown in Figure 9-75 realizes IMediumAccessControl, IErrorControl, IFlowControlManagement, IMeasurement, IMacPdu and IQualityOfService interfaces. Any extra functionality that is not defined by the interfaces from the common layer facilities package is defined by the IMediumAccessControl interface. In order to realize IMediumAccessControl interface the implementer shall bind a specific medium access parameter type to MediumAccessParameterType. For example, in a code division multiple access (CDMA) system, users are distinguished by their orthogonal spreading sequences, therefore the MediumAccessParameterType is bound to CdmaParameterType for a CDMA MediumAccessController component as shown in Figure 9-75. Through realizing above mentioned interfaces, this component provides operations for activating (or selecting) a transport channel and setting the control mode of the medium access parameters. MAC Facilities provide Service Users with methods to send non-real-time control and data between software resources and methods to signal the Service User that an event has occurred. Real-time control and signals are communicated via the packet interface. The MediumAccessControlResource is defined in the UML Profile.

Constraints

Note – Issue 7584: Specify which interfaces are mandatory

MediumAccessController component shall provide one ControlPort, at least one input DataControl port and at least one output DataControl port.

9.4 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 equipment,
- 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).

This package defines two types of IO mechanisms: Serial IO and Audio IO.

9.4.1 Serial IO Package

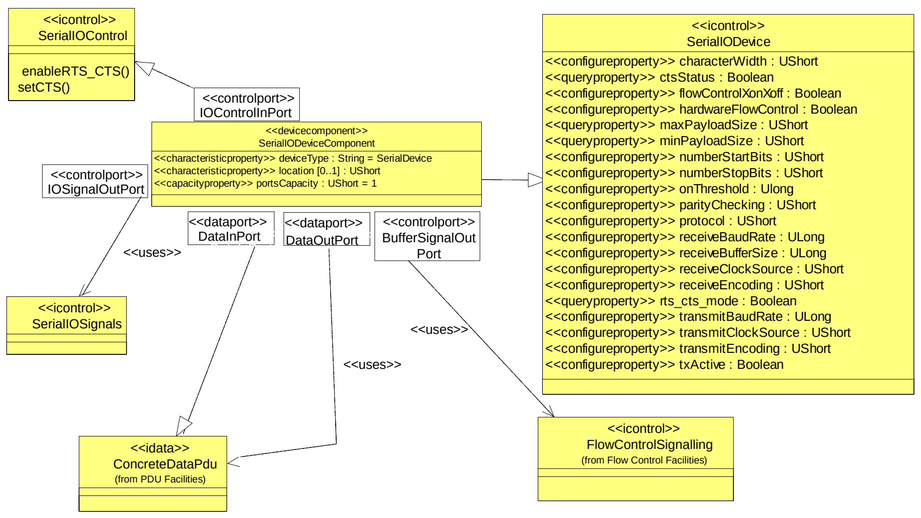
The Serial IO services are realized by a SerialIODeviceComponent that provides and uses the following set of interfaces:

- SerialIOSignals,
- SerialIODevice
- SerialIOControl

Those interfaces are summarized on the Figure 9-77 below:

Note – Issues 7868 incomplete definitions, 7985 property stereotypes

9.4.1 Serial IO Package



<<icontrol>>
SerialIOControl
enableRTS_CTS()
setCTS()
<<controlport>>
IOControlInPort
<<devicecomponent>>
SerialIODeviceComponent
<<characteristicproperty>> deviceType : String = SerialDevice
<<characteristicproperty>> location [0..1] : UShort
<<capacityproperty>> portsCapacity : UShort = 1
<<controlport>>
IOSignalOutPort
<<uses>>
<<dataport>>
DataInPort
<<dataport>>
DataOutPort
<<controlport>>
BufferSignalOut
Port
<<icontrol>>
SerialIOSignals
<<uses>>
<<uses>>
<<icontrol>>
SerialIODevice
<<configureproperty>> characterWidth : UShort
<<queryproperty>> ctsStatus : Boolean
<<configureproperty>> flowControlXonXoff : Boolean
<<configureproperty>> hardwareFlowControl : Boolean
<<queryproperty>> maxPayloadSize : UShort
<<queryproperty>> minPayloadSize : UShort
<<configureproperty>> numberStartBits : UShort
<<configureproperty>> numberStopBits : UShort
<<configureproperty>> onThreshold : Ulong
<<configureproperty>> parityChecking : UShort
<<configureproperty>> protocol : UShort
<<configureproperty>> receiveBaudRate : ULong
<<configureproperty>> receiveBufferSize : ULong
<<configureproperty>> receiveClockSource : UShort
<<configureproperty>> receiveEncoding : UShort
<<queryproperty>> rts_cts_mode : Boolean
<<configureproperty>> transmitBaudRate : ULong
<<configureproperty>> transmitClockSource : UShort
<<configureproperty>> transmitEncoding : UShort
<<configureproperty>> txActive : Boolean
<<idata>>
ConcreteDataPdu
(from PDU Facilities)
<<icontrol>>
FlowControlSignalling
(from Flow Control Facilities)

Figure 9-77 – Serial IO Framework

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface

9.4.1.1 Serial IO Control Interfaces

9.4.1.1.1 SerialIOControl

This interface is used for in-band control of Serial IO

Description

The SerialIOControl interface is used to control flow on customer's side.

Operations

- `enableRTS_CTS (in enable : Boolean)`

  Enable Clear To Send (CTS) and Request To Send (RTS).
- `setCTS (in cts : Boolean)`

  Force CTS.

9.4.1.1.2 SerialIOSignals

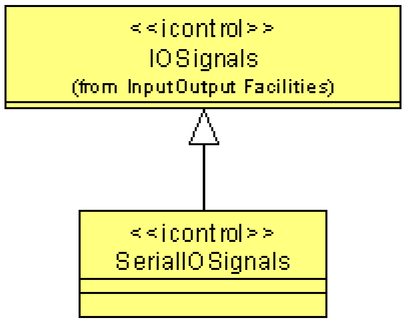


Figure 9-78 – Serial IO Signal

Note – Issue 7661 Lack Consistent use of SWRadio Stereotypes in Facilities interface, Issue 7868 incomplete definitions

---

Description

This interface is used by IO device to signal clients when RTS signal is up.

9.4.1.1.3 SerialIODevice

Description

The Serial IODevice is the control interface used for out-band control of serial lines.

Attributes

Note – Issue 7985 configure and query property stereotype

- `<<configureproperty>> characterWidth : UShort`
  (Asynchronous protocol only) Number of bits in character (5, 6, 7, or 8).
- `<<queryproperty>> ctsStatus: Boolean`
  Indicates the CTS status.
- `<<configureproperty>> flowControlXonXoff: Boolean`
  Controls whether flow Control signals should be generated. True means Xon and False means Xoff.
- `<<configureproperty>> hardwareFlowControl : Boolean`
  To enable/disable use of RTS/CTS hardware signals used for flow control.
- `<<queryproperty>> maxPayloadSize : UShort`
  Maximum size of payload for the pushPDU() method in ConcreteDataPDU interface.
- `<<queryproperty>> minPayloadSize : UShort`
  Minimum size of payload for the pushPDU() method in ConcreteDataPDU interface.
- `<<configureproperty>> numberStartBits : UShort`
  (Asynchronous protocol only) Number of start bits (0 or 1).

- `<<configureproperty>> numberStopBits : UShort`
  (Asynchronous protocol only) Number of stop bits (1 or 2).
- `<<configureproperty>> onThreshold : ULong`
  Optional, used only for receive flow control. IDLE time that Serial I/O waits before data received through the serial port must be forwarded to the component connected to the DataOutPort. IDLE time in number of not received characters unit.
- `<<configureproperty>> parityChecking : UShort`
  Type of parity checking (Even = 0, odd = 1).
- `<<configureproperty>> protocol: UShort`
  Sets asynchronous serial data protocol (`Asynchronous=0 and Synchronous = 1).`
- `<<configureproperty>> receiveBaudRate: ULong`
  Baud rate for Receive data
- `<<configureproperty>> receiveBufferSize : ULong`
  Size of packets to buffer before any data is written to device caller.
- `<<configureproperty>> receiveClockSource : UShort`
  Clock source for Receive data: internal Receive baud rate generator, external clock line, and Transmit clock source, respectively. Predefined values for coding scheme are 0=Internal Receive and 1=External clock.
- `<<configureproperty>> receiveEncoding : UShort`
  Sets the encoding method for Transmission of serial data to NRZ, NRZI Mark, FM0, Manchester, and Differential Manchester, respectively. Predefined values for coding scheme are 0=NRZ, 1=NRZI Mark, 2=FM0, 3=Manchester, and 4=Differential Manchester, respectively.
- `<<queryproperty>> rts_cts_mode: Boolean`
  Retrieves the RTS/CTS mode.
- `<<configureproperty>> transmitBaudRate : ULong`
  Baud rate for transmit data.
- `<<configureproperty>> transmitClockSource : UShort`
  Clock source for Transmission of data: internal Transmit baud rate generator, external clock line, Receive clock source, and clock recovery, respectively. Predefined values for coding scheme are 0=Internal Receive and 1=External clock.
- `<<configureproperty>> transmitEncoding : UShort`
  Sets the encoding method for Transmission of serial data to NRZ, NRZI Mark, FM0, Manchester, and Differential Manchester, respectively. Predefined values for coding scheme are 0=NRZ, 1=NRZI Mark, 2=FM0, 3=Manchester, and 4=Differential Manchester, respectively.
- `<<configureproperty>> txActive : Boolean`
  Set if on-going transmission.

Note – Issue7868 missing definitions

9.4.1.2 SerialIODeviceComponent

Description

The <<devicecomponent>> SerialIODeviceComponent contains the basic definition, ports and properties, for a logical serial I/O device.

Attributes

- <<characteristicproperty>> DeviceType : String = "SerialDevice"
  Defines the type of device.
- <<capacityproperty>> portsCapacity : UShort = 1
  Specifies the number of serial ports for a device.
- <<characteristicproperty>> Location : UShort [0..1]
  Defines if the device is on red (unencrypted boundary) or black side (encrypted boundary) of an encryption boundary (Black/Encrypted = 0, Red/Unencrypted = 1).

Ports

Table 9-14 – SerialIODeviceComponent Required Ports

Required Port Name	Required Interface	Connections	Purpose
DataOutPort	<<idata>> ConcreteDataPDU	SerialIODeviceComponent can only be connected to one component by this port	This port is used by SerialIODeviceComponent to connect to a component to which data, coming from a host connected to the serial port, are forwarded.
BufferSignalOutPort	<<icontrol>> FlowControlSignaling interface (SWRadio Facilities::Common Layer Facilities::Flow Control Facilities)	SerialIODeviceComponent can be connected to any number of components by this port	This port is used by SerialIODeviceComponent to connect to components in order to notify serial data buffer signal events.
IOSignalOutPort	<<icontrol>>SerialIOSignals	SerialIODeviceComponent can be connected to any number of components by this port	This port is used by SerialIODeviceComponent to connect to components in order to notify Request To Send (RTS) change.
TraceOutPort	OMG LightWeight Log Service	SerialIODeviceComponent can only be connected to one log service by this port	This port is used by SerialIODeviceComponent to connect to log service in order to send log information.

Table 9-15 – SerialIODeviceComponent Provided Ports

Provided Port Name	Provided Interface	Purpose
DataInPort	<<idata>>ConcreteDataPDU (SWRadio Facilities::Common Layer Facilities::PDU Facilities)	The SerialIODeviceComponent provides this port so that components can send data to this component using this port.
IOControlInPort	SerialIOControl	The SerialIODeviceComponent provides this port so that clients can control RTS/ Clear To Send (CTS) signals on the serial device.

Constraints

Note – Issue 7586

- Number of Start Bits value shall be 0 or 1.
  The corresponding OCL is as follows:
  context SerialIODeviceComponent
  inv: self.numberStartBits in Set {0,1}
- Number of Stop Bits value shall be 1 or 2.
  The corresponding OCL is as follows:
  context SerialIODeviceComponent
  inv: self.numberStopBits in Set {1,2}
- Location value shall be 0 or 1.
  The corresponding OCL is as follows:
  context SerialIODeviceComponent
  inv: self.location in Set {0,1}
- ParityChecking value shall be 0 or 1.
  The corresponding OCL is as follows:
  context SerialIODeviceComponent
  inv: self.parityChecking in Set {0,1}
- Protocol value shall be 0 or 1.
  The corresponding OCL is as follows:
  context SerialIODeviceComponent
  inv: self.protocol in Set {0,1}
- Receive Clock Source value shall be 0 or 1.
  The corresponding OCL is as follows:
  context SerialIODeviceComponent
  inv: self.receiveClockSource in Set {0,1}
- Receive Encoding value shall be 0 or 1.
  The corresponding OCL is as follows:
  context SerialIODeviceComponent
  inv: self.receiveEncoding in Set {0,1,2,3,4}

- Trasnmit Clock Source value shall be 0 or 1.
  The corresponding OCL is as follows:
  context SerialIODeviceComponent
  inv: self.transmitClockSource in Set {0,1}
- Transmit Encoding value shall be 0 or 1.
  The corresponding OCL is as follows:
  context SerialIODeviceComponent
  inv: self.transmitEncoding in Set {0,1}

Semantics

The SerialIODeviceComponent provides a basic standard definition of a logical serial I/O device. The <<icontrol>> SerialIODevice interface defines configuration and query properties based upon industry. The PropertySet interface (UML Profile for SWRadio::Application and Device Components::Resource Components) is used to configure and query these properties for a serial device, which can occur at initial setup of the serial I/O device or during runtime by application using the serial device.

The SerialIODeviceComponent supports a provided port named DataInPort and a required port named DataOutPort, which are both based upon the same <<idata>> ConcreteDataPDU interface (SWRadio Facilities::Common Layer Facilities::PDU Facilities).

The SerialIODeviceComponent supports RTS/CTS management by a provided port named IOControlInPort and a required port named IOSignalOutPort.

The SerialIODeviceComponent also uses the <<icontrol>> FlowControlSignaling interface (SWRadio Facilities::Common Layer Facilities::Flow Control Facilities) to indicate the serial data buffer state as follows:

● signalHighWatermark	The signalHighWatermark indicates that the serial I/O data buffer is full and that no more data can be processed until its state is changed to data buffer not full data buffer empty.
● signalLowWatermark	The signalLowWatermark indicates that the serial I/O data buffer is capable of receiving and processing more data.
● signalEmpty	The signalEmpty indicates that the serial I/O data buffer is empty and capable of receiving and processing more data.
● signalCongestion	The signalCongestion indicates that the serial I/O data buffer is full and data is being dropped not processed.

9.4.2 Audio Interfaces

The Audio IO services are realized by a AudioIODeviceComponent that provides and uses the following set of interfaces:

- AudioIOControl,
- AudioIODevice.

9.4.2 Audio Interfaces

Note – Issues 7868 incomplete definitions, 7985 property stereotypes, Issue 8205 Updated Figure Below

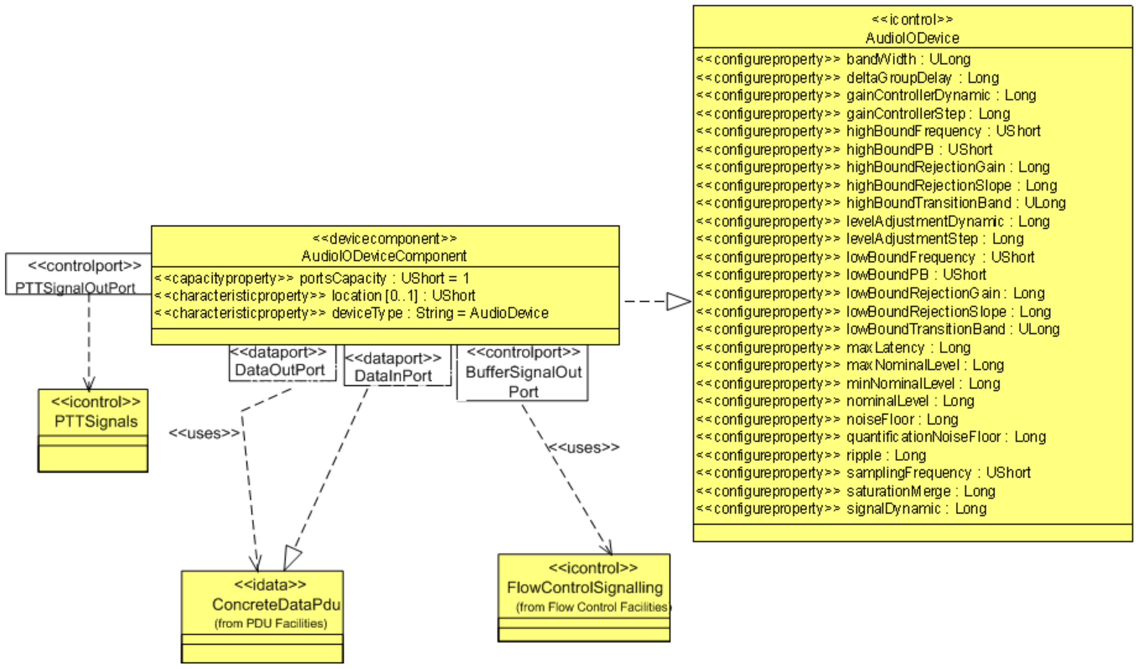

<<icontrol>>
AudioIODevice
<<configureproperty>> bandWidth : ULong
<<configureproperty>> deltaGroupDelay : Long
<<configureproperty>> gainControllerDynamic : Long
<<configureproperty>> gainControllerStep : Long
<<configureproperty>> highBoundFrequency : UShort
<<configureproperty>> highBoundPB : UShort
<<configureproperty>> highBoundRejectionGain : Long
<<configureproperty>> highBoundRejectionSlope : Long
<<configureproperty>> highBoundTransitionBand : ULong
<<configureproperty>> levelAdjustmentDynamic : Long
<<configureproperty>> levelAdjustmentStep : Long
<<configureproperty>> lowBoundFrequency : UShort
<<configureproperty>> lowBoundPB : UShort
<<configureproperty>> lowBoundRejectionGain : Long
<<configureproperty>> lowBoundRejectionSlope : Long
<<configureproperty>> lowBoundTransitionBand : ULong
<<configureproperty>> maxLatency : Long
<<configureproperty>> maxNominalLevel : Long
<<configureproperty>> minNominalLevel : Long
<<configureproperty>> nominalLevel : Long
<<configureproperty>> noiseFloor : Long
<<configureproperty>> quantificationNoiseFloor : Long
<<configureproperty>> ripple : Long
<<configureproperty>> samplingFrequency : UShort
<<configureproperty>> saturationMerge : Long
<<configureproperty>> signalDynamic : Long
<<controlport>> PTTSignalOutPort
<<devicecomponent>>
AudioIODeviceComponent
<<capacityproperty>> portsCapacity : UShort = 1
<<characteristicproperty>> location [0..1] : UShort
<<characteristicproperty>> deviceType : String = AudioDevice
<<dataport>> DataOutPort
<<dataport>> DataInPort
<<controlport>> BufferSignalOut Port
<<icontrol>> PTTSignals
<<uses>>
<<uses>>
<<idata>> ConcreteDataPdu (from PDU Facilities)
<<icontrol>> FlowControlSignalling (from Flow Control Facilities)

Figure 9-79 – Audio Framework

9.4.2.1 Audio Control Interfaces

Note – Issues 7868 missing definitions, 7868 alphabetical order and made consistent with figure

9.4.2.1.1 AudioIODevice

Description

The AudioIODevice is the control interface used to configure and control Acquisition and Restitution Audio device.

Attributes

Note – Issue 7985 configure and query property stereotyoes, Issue 8205 converted audio types to integer types

- `<<configureproperty>> bandWidth : ULong`
  Width of frequency band.

- `<<configureproperty>> deltaGroupDelay : Long`
  Delta group delay.
- `<<configureproperty>> gainControllerDynamic : Long`
  Define Gain
- `<<configureproperty>> gainControllerStep : Long`
  Defines granularity of gain.
- `<<configureproperty>> highBoundFrequency : UShort`
  High bound sampling frequency in order to satisfy the Shannon sampling criterion.
- `<<configureproperty>> highBoundPB : UShort`
  Defines the high bound rejection limit in low frequencies to avoid continuous component (pass band).
- `<<configureproperty>> highBoundRejectionGain : Long`
  High bound of rejection gain.
- `<<configureproperty>> highBoundRejectionSlope : Long`
  High bound of rejection slope.
- `<<configureproperty>> highBoundTransitionBand : ULong`
  High bound of transition band.
- `<<configureproperty>> levelAdjustmentDynamic : Long`
  capability of the gain.
- `<<configureproperty>> levelAdjustmentStep : Long`
  granularity of the gain.
- `<<configureproperty>> lowBoundFrequency : UShort`
  Low bound sampling frequency in order to satisfy the Shannon sampling criterion.
- `<<configureproperty>> lowBoundPB : UShort`
  Defines the low bound rejection limit in low frequencies to avoid continuous component (pass band).
- `<<configureproperty>> lowBoundRejectionGain : Long`
  Low bound of rejection gain.
- `<<configureproperty>> lowBoundRejectionSlope : Long`
  Low bound of rejection slope.
- `<<configureproperty>> lowBoundTransitionBand : ULong`
  Low bound of transition band.
- `<<configureproperty>> maxLatency : Long`
  Maximum allowed latency.
- `<<configureproperty>> maxNominalLevel : Long`
  Defines maximum bound of nominal level.
- `<<configureproperty>> minNominalLevel : Long`
  Defines minimal bound of nominal level.
- `<<configureproperty>> nominalLevel : Long`
  Defines the instruction for output analog signal nominal level.
- `<<configureproperty>> NoiseFloor : Long`
  Defines the level of noise (assumed white) present in audio frequency samples as inputting inside (resp. being output from) Audio. Expressed in dBFS/Hz. Possible spurious are integrated in this value.
- `<<configureproperty>> QuantificationNoiseFloor : Long`
  Defines the level of quantification noise present in digital samples as inputting inside (resp. being output from) ADC. Expressed in dBFS.
- `<<configureproperty>> ripple : Long`
  Ripple

- `<<configureproperty>> SamplingFrequency : UShort`
  Defines the sampling frequency of the audio frequency signal.
- `<<configureproperty>> saturationMerge : Long`
  Avoid gain saturation (in dBfs).
- `<<configureproperty>> SignalDynamic : Long`
  Expresses the expected variations of signal magnitude around the nominal level.

Note – Issue 7868 Missing Definitions

9.4.2.1.2 PTTSignals

Description

This interface is used by audio IO device to signal clients when Pushed-To-Talk (PTT) is pushed or released.

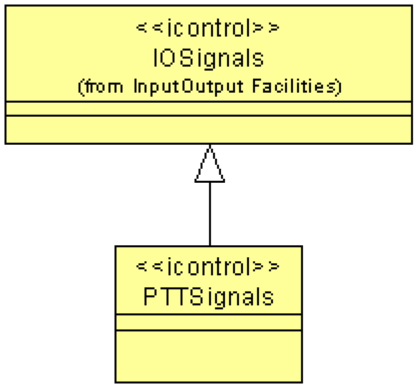


Figure 9-80 – PTTSignals

9.4.2.2 AudioIIODeviceComponent

Description

The <<devicecomponent>> AudioIODeviceComponent contains the basic definition, ports and properties, for a logical audio I/O device.

Attributes

- <<characteristicproperty>> DeviceType : String = "AudioDevice"
  Defines the type of device.
- <<capacityproperty>> portsCapacity : UShort = 1
  Specifies the number of audio ports for a device.
- <<characteristicproperty>> Location : UShort [0..1]
  Defines if the device is on red (unencrypted boundary) or black side (encrypted boundary) of an encryption boundary (Black/Encrypted = 0, Red/Unencrypted = 1).

Ports

Table 9-16 – - AudioIODeviceComponent Required Ports

Required Port Name	Required Interface	Connections	Purpose
DataOutPort	<<idata>> ConcreteDataPDU (SWRadio Facilities::Common Layer Facilities::PDU Facilities)	AudioIODeviceComponent can only be connected to one component by this port	This port is used by AudioIODeviceComponent to connect to a component to which data, coming from a host connected to the audio port, are forwarded.
BufferSignalOutPort	<<icontrol>> FlowControlSignaling interface (SWRadio Facilities::Common Layer Facilities::Flow Control Facilities)	SerialIODeviceComponent can be connected to any number of components by this port	This port is used by AudioIODeviceComponent to connect to components in order to notify audio data buffer signal events.
PTTSignalOutPort	<<icontrol>>PTTSignals	AudioIODeviceComponent can be connected to any number of components by this port	This port is used by AudioIODeviceComponent to connect to components in order to notify Push To Talk (PTT) change.
TraceOutPort	OMG LightWeight Log Service	AudioIODeviceComponent can only be connected to one log service by this port	This port is used by AudioIODeviceComponent to connect to log service in order to send log information.

Table 9-17 – AudioIODeviceComponent Provided Ports

Provided Port Name	Provided Interface	Purpose
DataInPort	<<idata>>ConcreteDataPDU (SWRadio Facilities::Common Layer Facilities::PDU Facilities)	The AudioIODeviceComponent provides this port so that components can send data to this component using this port.

Semantics

The AudioIODeviceComponent provides a basic standard definition of a logical Audio I/O device. The <<icon-trol>> AudioIODevice interface defines configuration and query properties based upon industry. The PropertySet interface (UML Profile for SWRadio::Application and Device Components::Resource Components) is used to configure and query these properties for a Audio device, which can occur at initial setup of the Audio I/O device or during runtime by application using the Audio device.

The AudioIODeviceComponent supports a provided port named DataInPort and a required port named DataOutPort, which are both based upon the same <<idata>> ConcreteDataPDU interface (SWRadio Facilities::Common Layer Facilities::PDU Facilities).

The AudioIODeviceComponent supports PTT management by a required port named IOSignalOutPort.

The AudioIODeviceComponent also uses the <<icontrol>> FlowControlSignaling interface (SWRadio Facilities::Common Layer Facilities::Flow Control Facilities) to indicate the Audio data buffer state as follows:

● signalHighWatermark	The signalHighWatermark indicates that the serial I/O data buffer is full and that no more data can be processed until its state is changed to data buffer not full data buffer empty.
● signalLowWatermark	The signalLowWatermark indicates that the serial I/O data buffer is capable of receiving and processing more data.
● signalEmpty	The signalEmpty indicates that the serial I/O data buffer is empty and capable of receiving and processing more data.
● signalCongestion	The signalCongestion indicates that the serial I/O data buffer is full and data is being dropped not processed.

9.4.3 IOSignals

Description

This interface is used by IO device to signal clients when a request to send data condition has occurred.

Operations

- ·<<oneway>> signalRTS (in rts: Boolean):The signalRTS operation indicates whether a request to send data condition exists. True means the condition does exist to send data. False means the condition does not exist for sending data.

Note – Issue 7985: replace configquery with configureproperty stereotype replace query with queryproperty stereotype throughout Physical Facilities

9.5 Physical Layer Facilities

Note – Issue 7656 Typos and Acronyms

According to Open System Interconnection (OSI) model, the purpose of the physical layer is "…provides the mechanical, electrical, functional and procedural means to activate, maintain, and de-activate physical-connections for bit transmission between data-link-entities [4]". It is the stated goal of the physical layer facilities to provide the necessary interfaces required to implement the functionality specified by the OSI physical layer. Due to the proposed facilities partitioning, interfaces in the Common Layer Facilities and in the Common Radio Facilities may be required to achieve this objective. 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. Various types of components, such as resources or devices, can implement an interface. As with all other interfaces, this specification does not restrict a particular implementation. Finally although the Physical Layer Facilities were designed with the OSI model as framework, it does not impose such a layering on any waveform implementation.

The approach supporting this specification separates the interfaces required to implement the physical layer into two sets 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.

Like in the communication equipment package, the Physical Layer Facilities provide the required interfaces to interact with both the subscriber-side and RF-side of the radio. Protocol specific combination, like Ethernet, USB, RS232, GSM, CDMA2000, Bluetooth etc. can be build using the facilities presented here in conjunction with the other higher layer services.

9.5.1 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.

9.5.2 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 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 increases the ease and speed at which waveform applications can be developed.

Note – Issue 7656 Typos & Acronyms

4.ITU-T Recommendation X.200 p.49

Two functionally separate facilities are part of the Physical Layer Facilities. The first set of facilities is responsible for modem operation. The modem includes all signal processing components involved in the translation of bits into symbols and vice-versa. In this context, bits are composed of data and any and all overhead information. Symbols are defined as the points of any n dimensional constellation. This definition applies on both the subscriber-side and the RF-side. Common subscriber-side modulations include Manchester, Non-Return to Zero (NRZ), Non-Return to Zero Inverted (NRZI), Return-to Zero (RZ) etc. Common RF-side modulations include Amplitude Modulation (AM), Frequency Modulation (FM), Quadriture Amplitude Modulation (QAM), Phase Shift Keying (PSK), Continuous Phase Modulation (CPM) etc. Channel coding also equally applies to both sides.

The second set of facilities is used to control the basic devices of the channel adaptation chain. This is called the Radio Frequency/Intermediate Frequency (RF/IF) chain. The chain's purpose it to adapt the symbol stream to the transmission channel by adjusting the frequency response, power, and centre frequency of the signal. In subscriber-side applications this may translate in adjusting the pitch, doing echo cancellation, and setting the volume. On the subscriber-side the center frequency is Direct Current (DC) or 'close' to DC. On the RF-side this can be pulse shaping, equalization and power control. On the RF-side the center frequency is normally not DC.

9.5.2.1 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 9.5.2.2.

Note – Issue 7656

The modem is not concerned with the functionalities of the Medium Access Control (MAC) layer. It will, in some cases (CDMA), provide services (i.e. PN sequence generator), which can be used by the various MACs to achieve their objectives.

The modem should have the facilities to implement all of today's baseband and passband digital modulation schemes: RZ, NRZ, Manchester, Direct sequence spread spectrum, QAM, PSK, Frequency Shift Keying (FSK), Amplitude Shift Keying (ASK), CPM, Gaussian Minimum Shift Keying (GMSK), Orthogonal Frequency Division Multiplex (OFDM), and Multiple-Input/Multiple-Output (MIMO) as well as digitally represented analog modulation schemes: Amplitude Modulation (AM), Frequency Modulation (FM), and Phase Modulation (PM). The preceding list is not exhaustive and is not indented to limit the scope of the modem functionality.

The modem facilities include more than just simple modulation; they also provide support for Forward Error Correction (FEC), differential encoding, interleaving, direct sequence spreading, scrambling, 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, Trellis Coded Modulation (TCM) is viewed as a special arrangement of modulation and FEC.

.

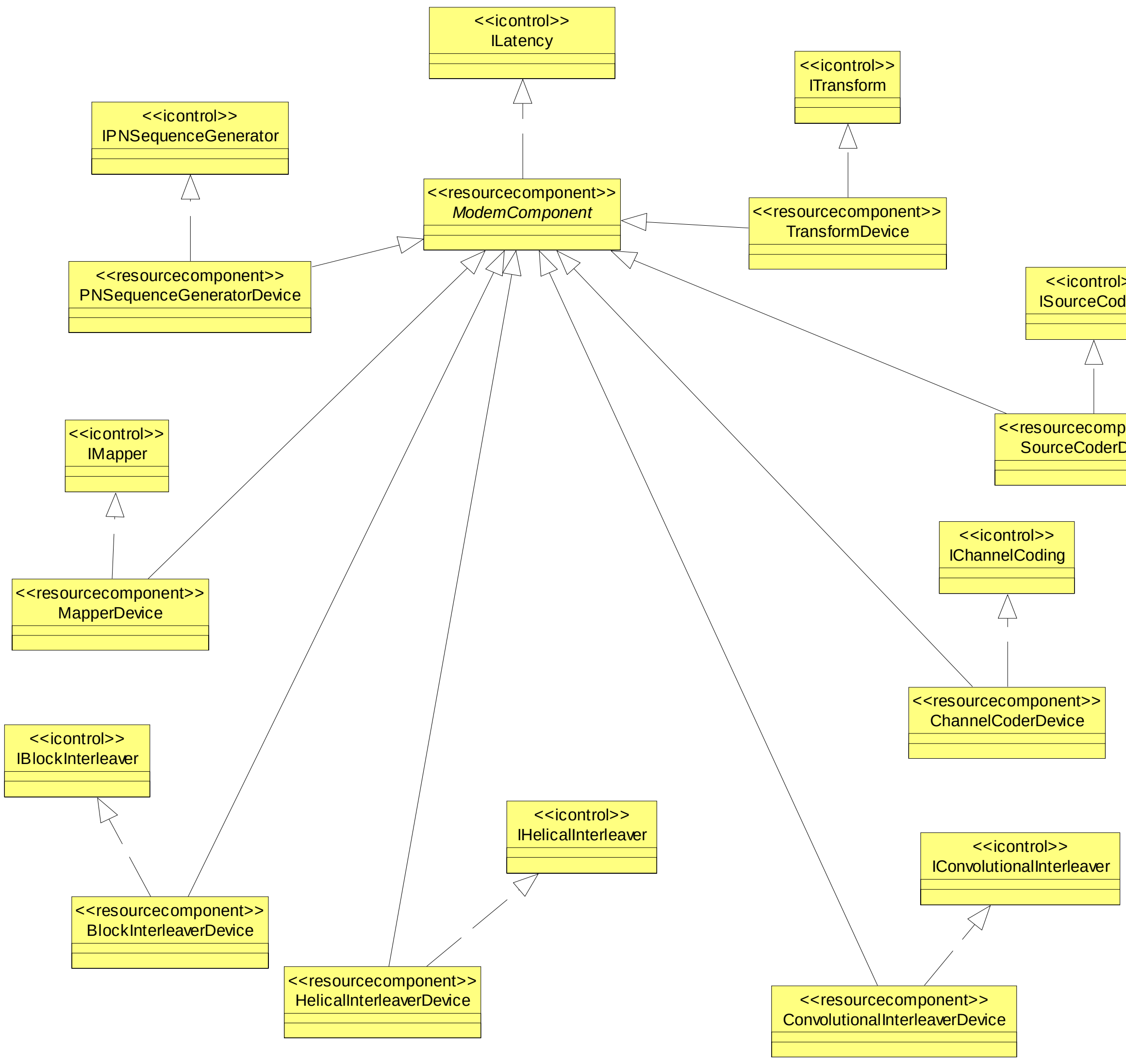


Figure 9-81 – Modem Facilities Overview

9.5.2.1.1 ModemComponent

Description

The ModemComponent component is an abstract component that realizes the ILatency interface. All components in the modem facilities inherit from this component. Components are stereotyped as <<resourcecomponent>> to indicate that they could either be implemented purely in software or in hardware via a <<devicecomponent>> component.

Note – Issue 7584 - Specify which interfaces are mandatory

Constraints

Note – Issue 7586

ModemComponent shall provide one ControlPort and at least one DataControlPort or DataPort.

9.5.2.1.2 ILatency

Description

The ILatency interface is used for specifying the processing latency of all digital signal processing components.

Attributes

- `<<querypropoerty>>processingLatency: Seconds`
  The processingLatency attribute represents the time it takes for an input data element to be carried out to the output of the component.

9.5.2.1.3 IBlockInterleaver

Description

This interface is used to control a block 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.

Attributes

Note – Issue 7895, fix types

- `<<configureproperty>>columns: UShort`
  The columns attribute is the number of columns of the block interleaver.
- `<<configureproperty>>rows: UShort`
  The rows attribute is the number of rows of the block interleaver.

9.5.2.1.4 IConvolutionalInterleaver

Description

This interface is used to control a convolutional 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.

Attributes

Note – Issue 7895, fix types

- `<<configureproperty>>delays: UShort [1..*]`

  The delays attribute is the delays applied by the convolutional interleaver to the bit stream.

9.5.2.1.5 IHelicalInterleaver

Description

This interface is used to control a helical 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.

Attributes

Note – Issue 7895, fix types

- `<<configureproperty>>columns: UShort`

  The columns attribute is the number of columns of the helical interleaver.
- `<<configureproperty>>groupSize: UShort`

  The groupSize attribute is the size of each group of input symbols.
- `<<configureproperty>>stepSize: UShort`

  The stepSize attribute is the number of rows between consecutive input groups in their respective columns.

9.5.2.1.6 IMapper

Description

This interface is used to control a mapper. The mapper executes 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

Note – Issue 7895, fix types

- `<<configureproperty>>baudRate: UShort`
  The baudRate attribute represents the current baud rate.
- `<<configureproperty>>constellation: String`
  The constellation attribute is the constellation type used by the mapper.
- `<<configureproperty>>bitPatternMapping: BitsToSymbolsMapping [1..*]`
  The bitPatternMapping attribute represents the actual definition of the constellation. Each input bit pattern is mapped to one or more dimensional quantities.

Types and Exceptions

Note – Issue 7895, fix types

- `BitsToSymbolsMapping ( bitPattern: ULong,dimensions: UShort [1..*])`
  bitPattern: The actual bit pattern as an UShort.
  dimensions: The quantity to which the bit pattern is mapped.

9.5.2.1.7 IPNSequenceGenerator

Description

This interface is used to control a Pseudo Noise (PN) Sequence Generator. PN sequences are commonly used in scramblers, spreaders and data sources. The output rate of the PN sequence will be called chip rate. Note that when used as a scrambler, the chip rate matches the data rate + overhead rate, when used as a data source, then it is simply the data rate.

The interface assumes that the PN sequence generated multiplies another incoming 'data' stream to produce the output 'data' stream.

There are many techniques used to generate PN sequences. Much like interleavers, each technique has its own mathematical description method. The generic formula for describing a random sequence generator is[5]:

$$X_n = (a_1 X_{n-1}^{j1} + a_2 X_{n-2}^{j2} \ldots + a_k X_{n-k}^{jk}) \bmod m$$

Figure 9-82 – Sequence number generator formula

Note – Issue 7656 (formula above)

Attributes

Note – Issue 7895, fix types

- `<<configureproperty>>chipRate: Float`
  The chipRate attribute represents the rate of encoding of the spreader. In other words, the chip rate is the rate at which the information bits are transmitted as a pseudo-random sequence of chips.

5.Knuth, Donald E., The Art of Computer Programming, Volume 2

- `<<configureproperty>>polynomial: Polynomial [1..*]`
  The polynomial attribute is the polynomial used to generate the pseudo-random sequence.

Note – Issue 7895, fix types

- `<<configureproperty>>modulus: UShort`
  The modulus attribute represents the value by which the polynomial is divided i.e. m in Figure 9-82 - Sequence number generator formula.
- `<<configureproperty>>seed: ULongLong`
  The seed attribute is the first value (X0) used to calculate the remaining pseudo-random sequence.

Types and Exceptions

Note – Issue 7895, fix types

- `Polynomial  ( multiplier: ULongLong, exponent: ULongLong )`
  (refer to Figure 9-82 - Sequence number generator formula)
  multiplier: a
  exponent: j

9.5.2.1.8 ITransform

Description

Note – Issue 7656

This interface is used to control the transform. The transformations included at this point are Fast Fourier Transform (FFT) and Inverse Fourier TRansform (IFFT). These transformations are commonly used for the generation and reception of OFDM and Coded OFDM (COFDM) waveforms as well as for frequency domain filtering.

Attributes

Note – Issue 7895, fix types

- `<<configureproperty>>blockSize: ULong`
  The blockSize attribute is the block size used by the transform.
- `<<configureproperty>>transform: TransformType`
  The transform type attribute indicates which type of transform is performed by the implementation.

Note – Issue 7895, fix types

- `<<configureproperty>>overlap: ULong`
  The overlap attribute is the amount of overlap of the transform in number of points.

Types and Exceptions

- `<<enumeration>>TransformType (FFT, IFFT, OTHER)`
  FFT: Fast Fourier Transform
  IFFT: Inverse Fast Fourier Transform

9.5.2.1.9 IChannelCoding

Description

This interface is used to represent a channel coder or decoder. A coder applies some transformation on the incoming data. Common examples of coders are differential encoders, and Forward Error Correction (FEC) encoders. Decoders reverse the transformation. The output of the coder is a coded sequence. In differential encoders the output rate is usually the same as in input rate where as in FEC coders, the output rate is higher than the input rate. The code rate is the ratio of the input rate over the output rate.

Channel coders and decoders have very different structures and mathematical formulas that describe them. Due to these differences, the interface provided here is not sufficient for a waveform developer. The goal of the interface is to provide system simulator with the minimum number of parameters to be able to model the communication path.

Attributes

Note – Issue 7895, fix types

- `<<configureproperty>>codeRate: Float`

  The code rate attribute, R, represents the ratio of the input rate, N, over the output rate K. R = N / K.

9.5.2.1.10 ISourceCoding

Description

This interface is used to control a source coder or decoder. Source coding essentially represents the compression of input data for better efficiency during transmission.

Source coders and decoders have very different structures and mathematical formulas that describe them. Due to these differences, the interface provided here is not sufficient for a waveform developer. The goal of the interface is to provide system simulator with the minimum number of parameters to be able to model the communication path.

Attributes

Note – Issue 7895, fix types

- `<<configureproperty>>codeRate: Float`

  The code rate attribute, R, represents the ratio of the input rate, N, over the output rate K. R = N / K.

9.5.2.2 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 could elect to configure each device with the correct frequency response. These design choices are left to the implementer. The same scenario could be applied to all interface defined in this package.

The components of the RF/IF Facilities maps to the concepts defined in the CommEquipment package. Components stereotyped as <<resourcecomponent>> indicates that they are either implemented in software or via hardware devices. Components stereotyped as <<devicecomponent>> indicates that they are implemented via hardware devices.

.

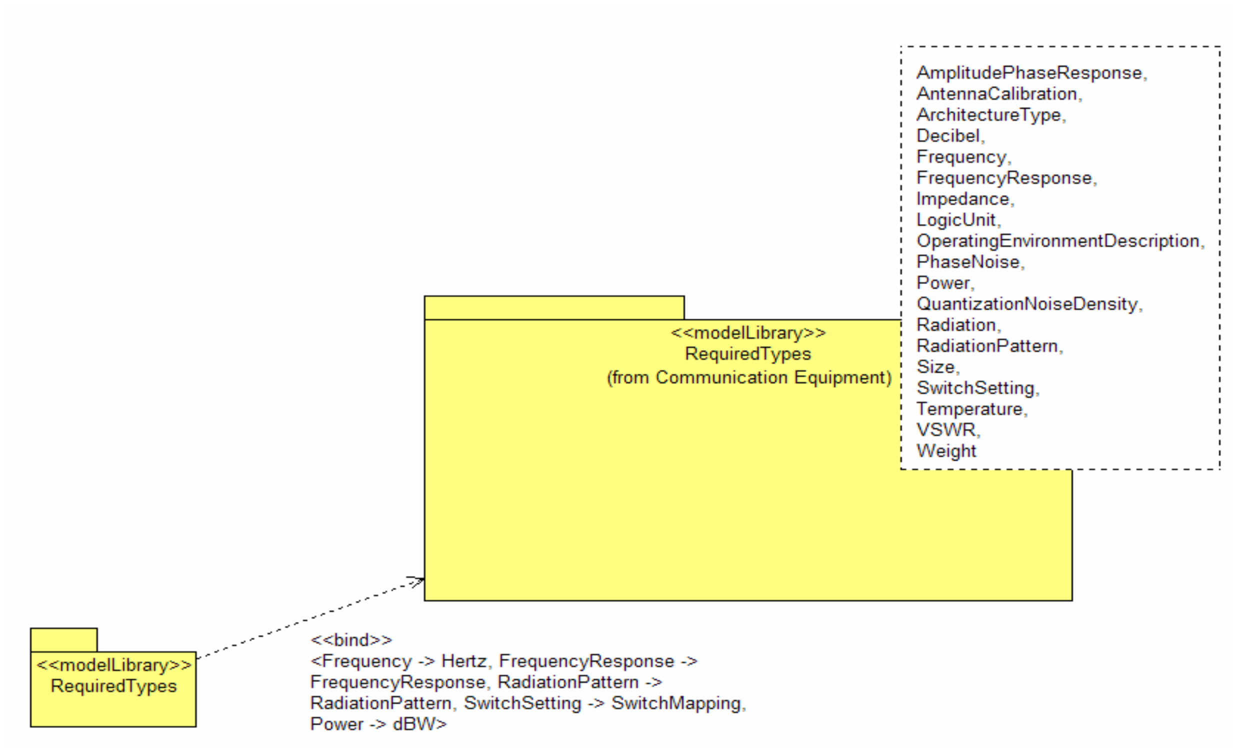


Figure 9-83 – RequiredTypes

.

9.5.2 Control

.

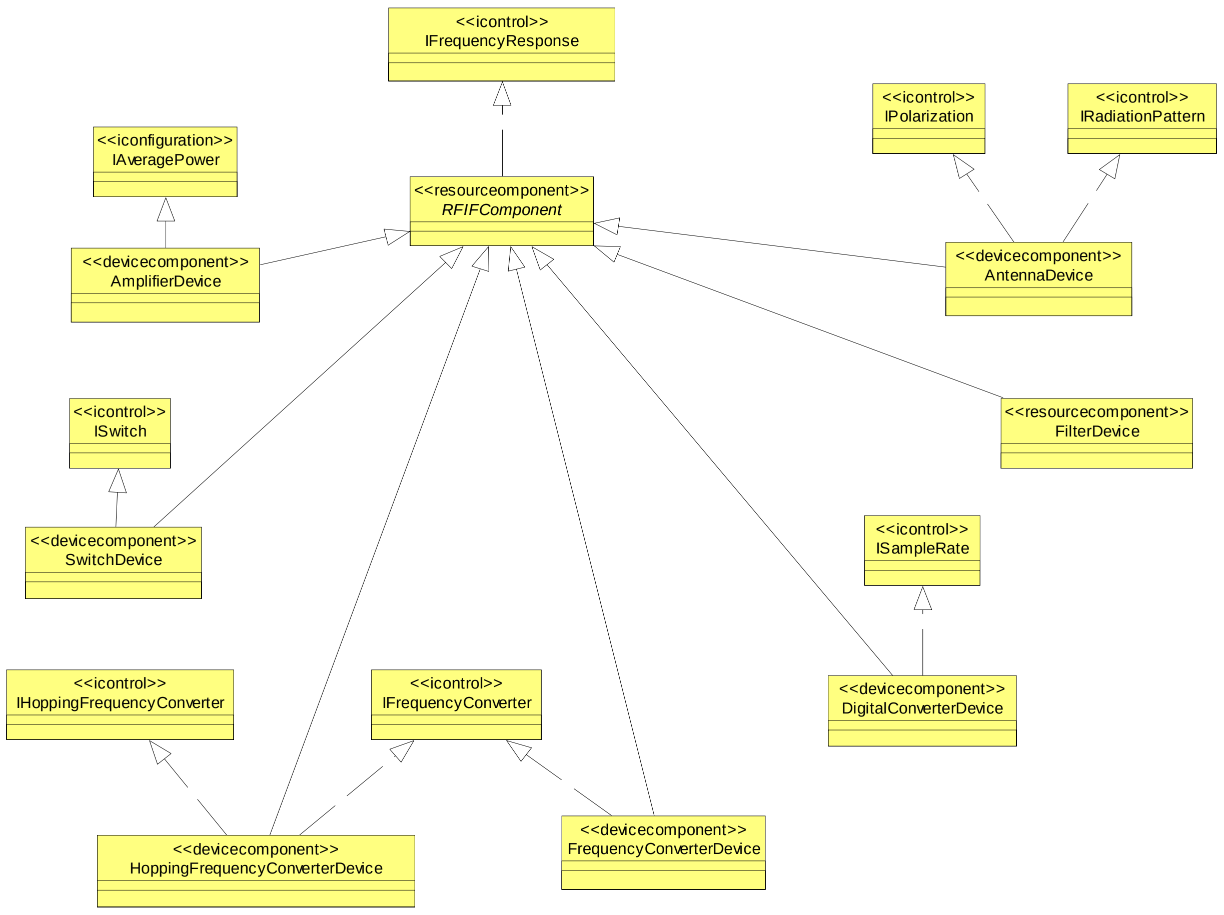


Figure 9-84 – RF/IF Facilities Overview

9.5.2.2.1 RFIFComponent

Description

The RFIFComponent component is an abstract component that realizes the IFrequencyResponse interface. All components in the RF/IF Facilities inherit from this component.

Note – Issue 7584 - Specify which interfaces are mandatory

Constraints

Note – Issue 7586

RFIFComponent shall provide one ControlPort and at least one DataControlPort or DataPort.

9.5.2.2.2 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-point frequency response could be the upper and lower 3 dB cut-off locations for a non-symmetric spectrum. 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.

Examples of components whose frequency response could be set with this API are pulse shaping filters and equalizers.

Attributes

- `<<configureproperty>>frequencyResponse: FrequencyResponsePoint [1..*]`
  The frequencyResponse attribute is the frequency response of the device. The frequency response specified is centered at 0 Hz.
- `<<configureproperty>>tunedFrequency: Hertz`
  The tunedFrequency attribute is the frequency at which the frequency response is centered.

Types and Exceptions

- `FrequencyResponsePoint  ( frequency: Hertz, amplitude: dB, phase: Degrees)`
  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).
- `Hertz`
  Hertz, a specialization of Double, denotes the number of cycles per second of a given signal.

Note – Issue 7895, fix types

- `Degrees`
  Degrees, a specializtion of Float, notes a degree value.
- `dB`
  dB, a specialization of Float, denotes the ratio between two voltage, current, or signal power levels.

9.5.2.2.3 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

- `<<configureproperty>>radiationPattern: RadiationPattern`

  The radiationPattern attribute represents the radiation pattern of the device.
- `<<configureproperty>>patternOrientation: PatternOrientation`

  The patternOrientation attribute is the actual pattern orientation, which is represented by an azimuth angle and an elevation angle. The antenna can be moved without having to change the radiation pattern.

Types and Exceptions

- `RadiationPattern ( azimuthPlane: RadiationPatternPoint [0..*], elevationPlane: RadiationPatternPoint [0..*])`

  Field intensity variation of an antenna as an angular function with respect to the azimuth and elevation axis.
- `RadiationPatternPoint  (gain: dB, angle: Degrees)`

  A single point in the radiation pattern is made of a gain value and an angle value.
- `PatternOrientation  (elevation: Degrees, azimuth: Degrees)`

  The pattern orientation is represented by an elevation angle which gives the vertical orientation and an azimuth angle which gives the horizontal orientation.

Note – Issue 7895, fix types

- `Degrees`

  Degrees, a specializtion of Float, notes a degree value.

9.5.2.2.4 IPolarization

Description

This interface is used to configure and / or control the polarization parameters of an antenna.

Attributes

- `<<configureproperty>>orientation: PolarizationKind`

  The orientation attribute represents the polarization of the antenna.

Note – Issue 7895, fix types

- `<<configureproperty>>ellipticity: Float`

  The ellipticity attribute is the 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.

Types and Exceptions

- `<<enumeration>>PolarizationKind (VERTICAL, HORIZONTAL, RIGHT_CIRCULAR_POLARIZE, LEFT_CIRCULAR_POLARIZE)`
  The orientation of the RF energy radiated from the device.

9.5.2.2.5 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

- `<<configureproperty>>nextInputFrequency: Hertz`
  The nextInputFrequency attribute is 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.
- `<<configureproperty>>nextOutputFrequency: Hertz`
  The nextOutputFrequency attribute is 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.
- `<<configureproperty>>currentInputFrequency: Hertz`
  The currentInputFrequency attribute is the frequency of the signal currently at the input of the device.
- `<<configureproperty>>currentOutputFrequency: Hertz`
  The currentOutputFrequency attribute is the frequency of the signal currently at the output of the device,

Types and Exceptions

- `Hertz`
  Hertz, a specialization of Double, denotes the number of cycles per second of a given signal.

9.5.2.2.6 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

- `<<configureproperty>>sampleRate: Hertz`
  The sampleRate attribute represents the number of samples the device takes per second.

Types and Exceptions

- `Hertz`

  Hertz, a specialization of Double, denotes the number of cycles per second of a given signal.

9.5.2.2.7 IAveragePower

Description

Note – Issue 7656

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 Automatic Gain Control (AGC) loop. Note that it is assumed that all other devices are average power neutral (i.e. they have a gain of 0 dB.)

Attributes

Note – Issue 7895 - Changed unit from dBW to Power

- `<<configureproperty>>averagePower: Power`

  The averagePower attribute represents the average power of the device.

9.5.2.2.8 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

Note – Issue 7895, fix types

- `<<configureproperty>>numberOfPorts: UShort`

  The numberOfPorts attribute represents the number of ports of the switch.
- `<<configureproperty>>switchSetting: SwitchMapping [0..*]`

  The switchSetting attribute represents the current configuration of the switch (i.e. which ports are connected together).

Types and Exceptions

Note – Issue 7895, fix types

- `SwitchMapping  (inputPortNumber: UShort, outputPortNumber: UShort)`

  A SwitchMapping is the association of an input port with an output port. Thus creating a connection inside the switch.

9.6 Radio Control Facilities

This section defines the facilities for radio control. Radio Control Facilities consist of facilities that are used to manage the control of a RadioSet such as audio alarms, Radio Set configuration, and Radio Set channel management. Only Radio Set channel facilities, in the section below, are defined for Radio Control Facilities.

9.6.1 Radio Set Facilities

This section defines the facilities for RadioSet channel management as depicted in Figure 9-85. The facilities defined for a RadioSet extends the component definitions as defined in the UML Profile for SWRadio::Infrastructure::Radio Management. The types of facilities offered by RadioSet are as follows:

1. Zeroize Control - provides the mechanism for zeroizing the RadioSet's classified or secure information.
2. Transmission Control - provides the mechanism for the controlling the transmission of the RadioSet's transmission or communication channel.
3. Communication Channel Control - provides the mechanism for managing a RadioSet's communication channel (unmanaged, managed, secure, and managed secure).
4. ChannelFactory - provides the mechanism for creating a communication channel.
5. RadioSet Control - provides the mechanism for managing a RadioSet (unmanaged, managed, secure, and managed secure).
6. Waveform Instantiation - provides the mechanism of instantiation a waveform on a channel.

Note – Issues 7959 and 7985, updated figure

9.6.1 Radio Set Facilities

Note – Issue 7895, fix types

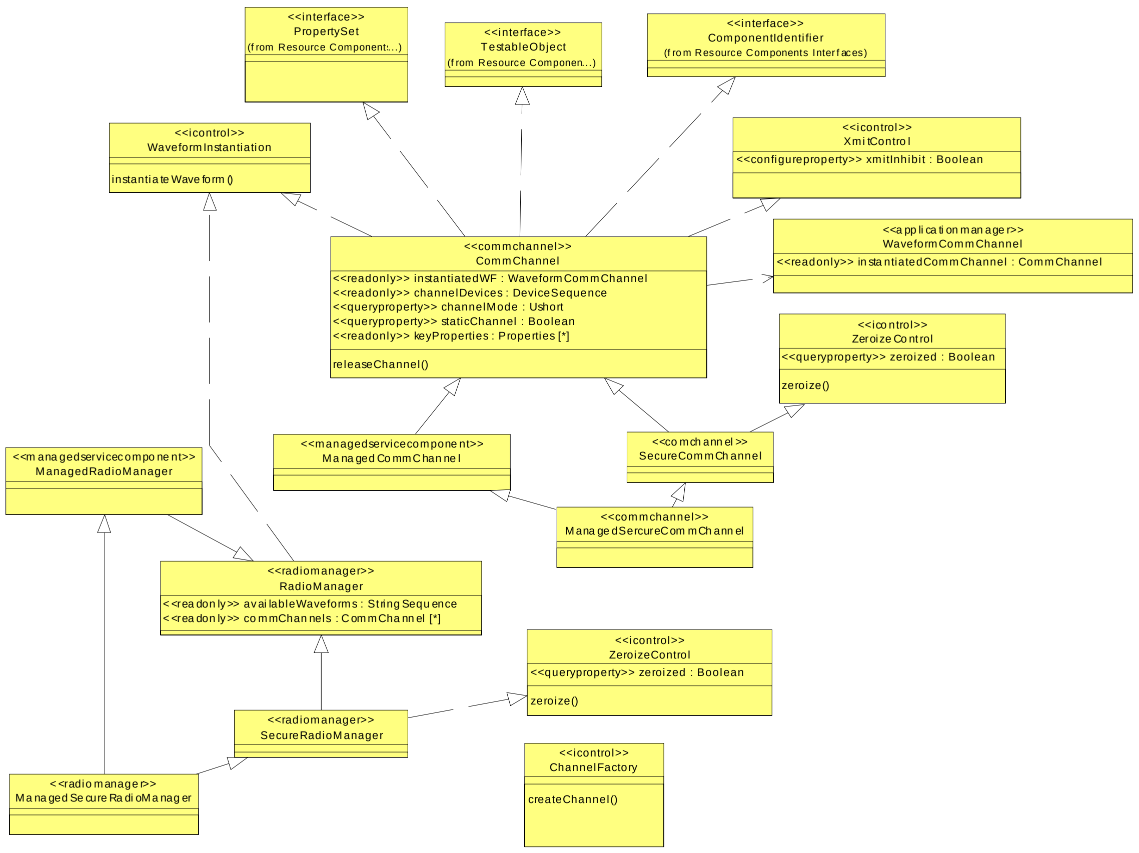

<<interface>> PropertySet (from Resource Component...)
<<interface>> TestableObject (from Resource Componen...)
<<interface>> ComponentIdentifier (from Resource Components Interfaces)
<<icontrol>> WaveformInstantiation
instantiateWaveform()
<<icontrol>> XmitControl
<<configureproperty>> xmitInhibit : Boolean
<<applicationmanager>> WaveformCommChannel
<<readonly>> instantiatedCommChannel : CommChannel
<<commchannel>> CommChannel
<<readonly>> instantiatedWF : WaveformCommChannel
<<readonly>> channelDevices : DeviceSequence
<<queryproperty>> channelMode : Ushort
<<queryproperty>> staticChannel : Boolean
<<readonly>> keyProperties : Properties [*]
releaseChannel()
<<icontrol>> ZeroizeControl
<<queryproperty>> zeroized : Boolean
zeroize()
<<managedservicecomponent>> ManagedRadioManager
<<managedservicecomponent>> ManagedCommChannel
<<comchannel>> SecureCommChannel
<<commchannel>> ManagedSercureCommChannel
<<radiomanager>> RadioManager
<<readonly>> availableWaveforms : StringSequence
<<readonly>> commChannels : CommChannel [*]
<<icontrol>> ZeroizeControl
<<queryproperty>> zeroized : Boolean
zeroize()
<<radiomanager>> SecureRadioManager
<<icontrol>> ChannelFactory
createChannel()
<<radiomanager>> ManagedSecureRadioManager

Figure 9-85 – Radio Set Facilities Overview

9.6.1.1 CommChannel

Description

The <<commchannel>> CommChannel component takes on the definition as described in the UML Profile for SWRadio::Infrastructure::Radio Management in addition to the interfaces realized by this component and provides additional attributes and operations.

Attributes

Note – Issue 7985 stereotype name

- `<<readonlyonly>>channelDevices : DeviceSequence`

  The channelDevices attribute contains the devices associated with this CommChannel. The devices could vary depending on the type of channel (static or not) and if instantiated.

Note – Issue 7895 - fix types

- `<<queryproperty>>channelMode: Ushort`
  The channelMode attribute indicates the capability of the channel. The values for channelMode are:
  1 means FULL_DUPLEX
  2 means RECEPTION_ONLY (half duplex)
  3 means XMIT_ONLY (half duplex)
- `<<readonly>>keyProperties: Properties`
  The keyProperties attribute contains information about each key associated with the channel.
- `<<readonly>>instantiatedWF: WaveformCommChannel`
  The instantiatedWF attribute contains the deployed waveform application associated with the instantiated channel. The instantiatedWF is a nil reference when the channel is not instantiated.
- `<<queryproperty>>staticChannel: Boolean`
  The staticChannel attribute indicates if the channel is static. A static channel means the channelDevices does not change and the communication path from baseband I/O to antenna is completely defined.

Operations

- `releaseChannel(): {raises = (releaseError)}`
  The releaseChannel operation provides the mechanism of uninstantiating the channel. The releaseChannel operation shall remove the deployed waveform as specified in the instantiatedWF attribute from the channel. The releaseChannel operation shall destroy the deployed waveform as specified in the instantiatedWF attribute. The releaseChannel operation shall raise the ReleaseError exception when the channel cannot be successfully released due to internal processing error(s).

Types and Exceptions

- `<<exception>>ReleaseError`
  The ReleaseError exception, specialization of SystemException, is raised when the releaseChannel operation is unsuccessfully due to internal processing errors. The error number indicates an ErrorNumberType value (e.g., E2BIG, ENAMETOOLONG, ENFILE, ENODEV, ENOENT, ENOEXEC, ENOMEM, ENOTDIR, ENXIO, EPERM). The message is component-dependent, providing additional information describing the reason for the error.

9.6.1.2 ChannelFactory

Description

The ChannelFactory interface provides the mechanisms for creating a communication channel.

Operations

- `createChannel(in channelProperties: in Properties, return CommChannel): {raises = ( CreateError, InvalidChannelProperties}`

  The createChannel operation creates a channel based upon the input channel properties. parameter.

Types and Exceptions

- `<<exception>>CreateError`

  The CreateError exception, specialization of SystemException, is raised when the createChannel operation is unsuccessfully due to internal processing errors. The error number indicates an ErrorNumberType value (e.g., E2BIG, ENAMETOOLONG, ENFILE, ENODEV, ENOENT, ENOEXEC, ENOMEM, ENOTDIR, ENXIO, EPERM). The message is component-dependent, providing additional information describing the reason for the error.
- `<<exception>>InvalidChannelParameters (invalidProperties: Properties)`

  The InvalidChannelParameters exception is raised when the input channelParameters parameter is invalid.

Semantics

The instantiateChannel operation shall deploy the waveform as specified by the input waveformName parameter onto a CommChannel. The instantiateChannel operation shall return a CommChannel when the instantiateChannel operation successfully instantiated with the waveform application onto the channel. The instantiateChannel operation shall use the input wfParameters for the initial configuration of the deployed waveform. The instantiateChannel shall use the channelParameters for the initial setup of the instantiated CommChannel.

The instantiateChannel operation shall raise the UnknownWaveform exception when the input waveformName is not known. The instantiateChannel operation shall raise the InstantiateError exception when the CommChannel cannot be successfully instantiated due to internal processing error(s). The instantiateChannel operation shall raise the InvalidChannelProperties exception when the input channelParameters parameter is invalid. The InvalidChannelProperties invalidProperties identifies the properties that are invalid. The instantiateChannel operation shall raise the InvalidWFProperties exception when the input channelParameters parameter is invalid. The InvalidWFlProperties invalidProperties identifies the properties that are invalid.

9.6.1.3 ManagedCommChannel

Description

Note – Issue 7959 1st sentence

The <<managedservicecomponent>> ManagedCommChannel component takes on the definition as described in the UML Profile for SWRadio::Infrastructure::Radio Services in addition to the specialization of the CommChannel. The ManagedCommChannel provides the mechanism of a managed CommChannel with state behavior.

Semantics

The ManagedCommChannel's operational state is based upon the operational state of its communication channel's devices. The ManagedCommChannel's usage state is IDLE when the communication channel has not been instantiated with a waveform. The ManagedCommChannel's usage state becomes BUSY when a waveform is instantiated on the communication channel. The ManagedCommChannel's administrative state is SHUTTING_DOWN or LOCKED then the communication channel is unavailable for waveform instantiation. This administrative state of the communication channel's devices may also be affected upon ManagedCommChannel admin state changes. Some devices may be shareable across communication channels, which may not affect their admin states when communication channel admin state changes. While other devices are only associated with one communication channel, which will effect their admin states.

Whenever the adminState attribute changes, a StateChangeEventType (Infrastructure::Radio Management::Event Channels) event may be issued to an event channel. The StateChangeEventType event data shall be populated as follows when issued:

1. The producerId field is the identifier attribute of the RadioManager component.
2. The sourceId field is the identifier attribute of the CommChannel component.
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.

Whenever the operationalState attribute changes, a StateChangeEventType event may be issued to an event channel. The event data shall be populated as follows when issued:

1. The producerId field is the identifier attribute of the RadioManager component.
2. The sourceId field is the identifier attribute of the CommChannel component.
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.

Whenever the usageState attribute changes, a StateChangeEventType event may be issued to an event channel. The StateChangeEventType event data shall be populated as follows when issued:

1. The producerId field is the identifier attribute of the RadioManager.
2. The sourceId field is the identifier attribute of the CommChannel.
3. The stateChangeCategory field is USAGE_STATE_EVENT.

The stateChangeFrom and stateChangeTo fields reflect the usageState attribute value before and after the state change, respectively.

9.6.1.4 ManagedRadioManager

Description

Note – Issue 7959

The <<managedservicecomponent>> ManagedRadioManager component takes on the definition as described in the UML Profile for SWRadio::Infrastructure::Radio Services in addition to the specialization of the RadioManager. The ManagedRadioManager provides the mechanism for a managed RadioManager with state behavior.

Semantics

The ManagedRadioManager's operational state shall be based upon the operational state of its communication channels and devices. The ManagerRadioManager's usage state shall be IDLE when all of its communication channels are IDLE. The ManagedRadioManager's usage state becomes ACTIVE when any of its communication channel is not IDLE. The ManagedRadioManager's usage state shall be BUSY when all of its communication channels are not IDLE. The ManagedRadioManager's administrative state is SHUTTING_DOWN or LOCKED then its communication channels shall be unavailable for waveform instantiation.

Whenever the adminState attribute changes, a StateChangeEventType (Infrastructure::Radio Management::Event Channels) event may be issued to an event channel. The StateChangeEventType event data shall be populated as follows when issued:

1. The producerId field is the identifier attribute of the ManagedRadioManager component.
2. The sourceId field is the identifier attribute of the ManagedRadioManager component.
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.

Whenever the operationalState attribute changes, a StateChangeEventType event may be issued to an event channel. The event data shall be populated as follows when issued:

1. The producerId field is the identifier attribute of the ManagedRadioManager component.
2. The sourceId field is the identifier attribute of the ManagedRadioManager component.
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.

Whenever the usageState attribute changes, a StateChangeEventType event may be issued to an event channel. The StateChangeEventType event data shall be populated as follows when issued:

1. The producerId field is the identifier attribute of the ManagedRadioManager.
2. The sourceId field is the identifier attribute of the ManagedRadioManager.
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.

9.6.1.5 ManagedSecureCommChannel

Description

The <<commchannel>> ManagedSecureCommChannel component takes on the definition as described in the UML Profile for SWRadio::Infrastructure::Radio Management in addition to the specializations of the SecureCommChannel and ManagedCommChannel.

Semantics

This type of communication channel provides both managed and secure capability.

9.6.1.6 ManagedSecureRtadioManager

Description

The <<radiomanager>> ManagedSecureRadioManager component takes on the definition as described in the UML Profile for SWRadio::Infrastructure::Radio Management in addition to the specializations of the SecureRadioManager and ManagedRadioManager.

Semantics

This type of radio manager provides both managed and secure capability.

9.6.1.7 RadioManager

Description

The <<radiomanager>> RadioManager component takes on the definition as described in the UML Profile for SWRadio::Infrastructure::Radio Management in addition to the interfaces realized by this component and provides additional attributes and operations.

Attributes

- `<<readonly>>availableWaveforms: StringSequence`
  The availableWaveforms attribute shall contain the names of the installed waveforms in the RadioSet.
- `<<readonly>>commChannels: CommChannel [1..*]`
  The commChannels attribute shall contain the set of communication channels for a RadioSet that this RadioManager is managing.

Semantics

The RadioManager's instantiateChannel operation instantiates one of its CommChannels using the input parameters.

9.6.1.8 SecureRadioManager

Description

The <<radiomanager>> SecureRadioManager component takes on the definition as described in the UML Profile for SWRadio::Infrastructure::Radio Management in addition to the specializations of the RadioManager and the interfaces realized by this component. The SecureRadioManager provides the mechanism of managing a secure radio manager.

Semantics

The usage of a radio manager after it has been zeroized is unspecified.

9.6.1.9 SecureCommChannel

Description

The <<commchannel>> SecureCommChannel component takes on the definition as described in the UML Profile for SWRadio::Infrastructure::Radio Management in addition to the specializations of the CommChannel and the interfaces realized by this component. The SecureCommChannel provides the mechanism of managing a secure communication channel.

Semantics

The usage of a communication channel after it has been zeroized is unspecified.

9.6.1.10 WaveformCommChannel

Description

The <<applicationmanager>> WaveformCommChannel component takes on the definition as described in the UML Profile for SWRadio::Infrastructure::SWRadio Deployment::Application Deployment in addition to its attributes.

Attributes

- `<<readonly>>instantiatedCommChannel : CommChannel`
  The instantiatedCommChannel is the CommChannel associated with the deployed waveform application.

9.6.1.11 WaveformInstantiation

Description

The WaveformInstantiation interface provides the mechanisms for instantiation a waveform application onto a communication channel.

Operations

- `instantiateWaveform (in waveformName: String, in instanceWFName: String, in wfProperties : Properties, in channelProperties: in Properties, return WaveformCommChannel): {raises = (InstantiationError, InvalidChannelProperties, InvalidWFProperties, UnknownWaveform)}`
  The instantiateWavform operation deploys a waveform application onto a channel.

Types and Exceptions

- `<<exception>>InstantiationError`
  The InstantiationError exception, specialization of SystemException, is raised when the instantiateWaveform operation is unsuccessfully due to internal processing errors. The error number indicates an ErrorNumberType value (e.g., E2BIG, ENAMETOOLONG, ENFILE, ENODEV, ENOENT, ENOEXEC, ENOMEM, ENOTDIR, ENXIO, EPERM). The message is component-dependent, providing additional information describing the reason for the error.

- `<<exception>>InvalidChannelParameters (invalidProperties: Properties)`
  The InvalidChannelParameters exception is raised when the input channelParameters parameter is invalid.
- `<<exception>>InvalidWFParameters (invalidProperties: Properties)`
  The InvalidWFParameters exception is raised when the input wfParameters parameter is invalid.
- `<<exception>>UnknownWaveform`
  The UnknownWaveform exception indicates the waveform is not known.

Semantics

The instantiateWaveform operation shall deploy the waveform as specified by the input waveformName parameter onto a CommChannel. The instantiateChannel operation shall return a WaveformCommChannel when the instantiateWaveform operation successfully instantiated with the waveform application onto the channel. The instantiateWaveform operation shall use the input wfParameters for the initial configuration of the deployed waveform. The instantiateChannel shall use the channelParameters for the initial setup of the instantiated CommChannel.

The instantiateWaveform operation shall raise the UnknownWaveform exception when the input waveformName is not known. The instantiateWaveform operation shall raise the InstantiateError exception when the CommChannel cannot be successfully instantiated due to internal processing error(s). The instantiateWaveform operation shall raise the InvalidChannelProperties exception when the input channelParameters parameter is invalid. The InvalidChannelProperties invalidProperties identifies the properties that are invalid. The instantiateWaveform operation shall raise the InvalidWFProperties exception when the input channelParameters parameter is invalid. The InvalidWFProperties invalidProperties identifies the properties that are invalid.

9.6.1.12 XmitControl

Description

The XmitControl interface provides the mechanism to control a component's transmission such as transmission of radio frequencies.

Attributes

Note – Issue 7985

- `<<configureproperty>>xmitInhibit: Boolean`
  The xmitInhibit attribute is used to control and return the status of the transmission state of a component.

Semantics

The xmitInhibit attribute when a configuration value of “True” means the component shall inhibit transmission, otherwise the component can transmit.

9.6.1.13 ZeroizeControl

Description

The ZeroizeControl interface provides the mechanism to zeroize a component's environment or information.

Attributes

Note – Issue 7985

- `<<queryproperty>>zeroized: Boolean`

  The zeroize attribute is used to return the status of the zeroized state of a component. A True value indicates the component is zeroized, otherwise the component is not zeroized.

Operations

- `zeroize ()`

  The zeroize operation is used to command the component to zeroize its environment. The information that gets zeroized is component dependent.

10 Platform Specific Model (PSM)

Note – Issue 7845

The SWRadio PSM consists of CORBA and XML that are based upon the PIM and UML Profile for SWRadio. The PIM to PSM transformation rules are not universal rules for creating *any* PSM, but only used for the purpose of this specification. This section defines a non-normative reference PSM. Non-CORBA PSMs may also be fully compliant to this specification as a whole.

Note – Issue 7742 updated CORBA transformations.

The rule set for transforming UML packages, interfaces, 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 attributes with readonly and readwrite map to CORBA attributes in CORBA interfaces.
3. UML attributes with configureproperty, queryproperty and testproperty do not map to CORBA attributes in CORBA interfaces. Instead XML definitions are used that follow the Property types as defined in UML Profile for SWRadio::Application and Device Components::Properties section.
4. UML classes without operations that are not stereotyped and used for type definitions map to CORBAStruct stereotypes in the CORBA interfaces and modules. The parent classes do not get translated into CORBA types instead the parent class attributes are added to the subclass in the CORBA definition.
5. UML <<datatype>> map to CORBA basic types. Primitive types are mapped to CORBA primitive types and primitive sequence types are mapped to CORBA Typedef of primitive sequence types.
6. UML exceptions and exception extensions map to CORBA exceptions. There is no specializations of exceptions in CORBA so the (UML Profile for SWRadio::Application and Device Components::BaseTypes) SystemException definition does not appear in the generated SWRadio CORBA interfaces but all the specialization exceptions of SystemException are in the SWRadio CORBA interfaces with the same attributes as defined for SystemException.
7. UML attributes that have a cardinality of many [*] map to a CORBA Typedef of sequence types.
8. UML operations and <<optional>> operations map to operations in the SWRadio CORBA interfaces.
9. Transformations are only performed for concrete classes, not for template classes. Concrete classes that bind to template classes are used in the PSM.
10. For Interfaces that reference a component stereotype for a type, the "component" qualifier is removed from the name. For Example, FileManagerComponent would become FileManager as the type for the parameter or attribute.

The SWRadio CORBA PSM corresponds to:

1. PIM Facilities: The top most CORBA is called DfSWRadio which maps to the PIM Facilities package. The packages (e.g., Common Layer Facilities) directly beneath PIM Facilities map to CORBA modules but without facilities in their. In some case these packages have further CORBA

modules. This occurs when a package has more than one interface. The DfSWRadio maps to existing IDL definition used in industry, therefore the IDL does not follow all of the OMG CORBA guidelines (e.g., operation, attribute, and parameter names), in order to reduce impact on industry.

- Common Layer Facilities (Annex B)

Note – Issue 8201 Resolution (Common Radio Facilities now covers Lightweight Services and the Lightweight Log Service, instead of File Services which used to be in Annex C

- Common Radio Facilities (Section 3.3 CORBA Services Specifications)
- Data Link Layer Facilities (Annex D)
- Physical Layer Facilities (Annex E and F). Annex E contains the Physical layer interfaces and Annex F contains the Physical Layer properties that were identified as configquery or query in the interface.
    - I/O Facilities
    - RF Facilities
- Radio Control Facilities (Annex G)

Note – Issue 7742 Updated text and table below

---

2. The UML Profile for SWRADIO maps to a CORBA module named CF (Core Framework). The CF CORBA module is an existing IDL definition used in industry, therefore the CF IDL does not follow all of the OMG CORBA guidelines (e.g., operation, attribute, and parameter names), in order to reduce impact on industry. The CF CORBA IDL is depicted in Annex A and Annex C (File Services). The CF CORBA module is broken up into multiple files as shown in Table 1. The reason for the break-up into multiple files is a memory foot print size for the embedded radio environment. Specific

interfaces are only used and implemented on certain devices within the swradio. The interfaces used vary by the type of developers (waveform, device, radio management) for a swradio radio. These developers use different set of interfaces for the components they are developing.

Table 10-18 – Core Framework CORBA Module Overview

UML Profile for SWRadio Specification Sections		IDL File	CORBA Module
Application and Device Components	BaseTypes	CFCommonTypes.idl	CF
		CFBaseTypes.idl	CF
	BaseTypes - each primitive sequence type is mapped to its own file	CFPortTypes_BooleanSequence.idl CFPortTypes_CharSequence.idl CFPortTypes_ShortSequence.idl CFPortTypes_UshortSequence.idl CFPortTypes_LongSequence.idl CFPortTypes_UlongSequence.idl CFPortTypes_LongLongSequence.idl CFPortTypes_UlongLongSequence.idl CFPortTypes_FloatSequence.idl CFPortTypes_DoubleSequence.idl CFPortTypes_LongDoubleSequence.idl CFPortTypes_WcharSequence.idl CFPortTypes_WstringSequence.idl CFPortTypes.idl	PortTypes within CF
	Resource Components Interfaces except for ResourceFactory	CFResources.idl	CF
	Resource Components Interfaces - ResourceFactory only	CFResourceFactory. idl	CF
	Device Components Interfaces	CFDevices.idl	CF

Table 10-18 – Core Framework CORBA Module Overview

UML Profile for SWRadio Specification Sections			IDL File	CORBA Module
Infrastructure	Radio Management	Device Management Interfaces - Each interface maps to its own IDL file.	CFServiceRegistration.idl, CFDeviceManager.idl	CF
		RadioSet Management Interfaces - Each interface maps to its own IDL file	CFDomainEventChannels.idl, CFDomainInstallation.idl, CFDeviceManagerRegistration.idl, CFDomainManager.idl	CF
		Domain Event Channels	CF_SE_DomainEvent.idl CF_SE_StateEvent.idl	CF
	Radio Services	File Services Interfaces - Each interface maps to its own IDL file	CFFile.idl, CFFileSystem.idl, CFFileManager.idl	CF in Annex C
		Radio Services Interfaces	CFStateManagement.idl	CF
	SWRadio Deployment	Applications Deployment Interfaces	CFApplications.idl	CF

Note – Issue 7581 updated File Service name in table above, Issue 8291 properties PSM

Other non-CORBA PSM transforms (e.g., XML) are as follows:

1. The UML Profile for SWRadio::Application and Device Components::Properties maps to a SWRadio Properties XML definitions as specified in Annex I. Each property definition maps to an XML element definition. Abstract classes are not directly transformed into XML, instead their attributes are used for the concrete subclass XML definitions. Attributes with default values are created as XML attributes. All other attributes are created as XML elements. The name and integerId is a unique value for each property in a XML properties set. Only the properties attributes stated in the Properties section are used for the XML properties definition. Specific transformstions of the properties are as follows:
    - Primitive types are mapped to the corresponding enumeration literal in the SimpleType XML element
    - EnumerationProperty attributes map to the EnumerationLiteral XML element. The attribute name and value maps to the label and value xml elements.
    - ConfigureProperty and QueryProperty that are primitive types maps to the XML ConfigureQuerySimpleProperty XML element.
    - ConfigureProperty and QueryProperty that are primitive sequence types maps to the XML ConfigureQuerySimpleSeqProperty XML element.
    - ConfigureProperty and QueryProperty that are a StructProperty type maps to the XML ConfigureQueryStructProperty XML element.

- ConfigureProperty and QueryProperty that are a StructProperty sequence type maps to the XML StructSequenceProperty XML element.
- TestProperty maps to the XML TestProperty element
- CharacteristicProperty maps to XML CharacteristicProperty
- CapacityProperty mapes to XML CapacityProperty
- CharacteristicSelectionProperty maps to XML CharacteristicSelectionProperty
- CharacteristicSetProperty maps to XML CharacteristicSetProperty
- ExecutableProperty maps to XML ExecutableProperty

Note – Issue 7582, changed wording for communication equipment item 2

---

2. The UML Profile for SWRadio::Communication Equipment and UML Profile for SWRadio::Infrastructure::Communication Channel map to SWRadio Channel and Communication Equipment XML definitions as specified in Annex J. The mappings follow the transformation rules for components in item 1, above, and the following:

- Communication Equipment
    - Each CommEquipment stereotype or UML Device definition maps to the CommEquipment XML element definition. The CommEquipment name and stereotype names map to the name and stereotypeName elements of the CommEquipment XML element.
    - All properties of the CommEquipment map to the properties of the CommEquipment XML element as specified in item 1 (Properties) above.
    - All ports (AnalogInputPort, AnalogOutputPort, and DigitalPort map to the ports element of the CommEquipment XML element.
        - The properties of all communication equipment ports map to the properties of the Port XML element as specified in item 1 (Properties) above.
        - The Port name and stereotype name map to the name and stereotypeName elements of the Port XML element.
- Communication Channel
    - All Channel stereotypes map to the Channel XML element.
    - The properties of a Channel map to the properties element of the Channel XML element as specified in item 1 (Properties) above.
    - The Channel name and stereotype name map to the name and stereotypeName elements of the Channel XML element.
    - Associated Channels (LogicalPhysicalChannel, LogicalIOChannel, LogicalProcessingChannel, LogicalSecurityChannel) map to the subchannels XML element of the Channel XML element as references to their Channel XML element.
    - Associated CommEquipments map to the commEquipments element of the Channel XML element as references to their CommEquipment XML element.

- Channel Connections map to connections element of the Channel XML element. A CommEquipmentConnector maps to the CommEquipmentConnector XML element.

3. Operating System Profile (Annex H)

Note – Issue 8868

---

4. Descriptors

In industry there are two sets of XML definitions that could be used for the deployment of components with an SWRadio RadioSet or RadioSystem, which are the Document Type Definitions (DTDs) as described in Annex L and CCM Schema XML as described in the COBRA Components Model (CCM). The relationships of these XML elements to the SWRadio components are depicted in Table TBD below.

Table 10-1

SWRadio Component Type	**Descriptors PSM**	
	Document Type Definitions XML	**CCM Schema XML**
ApplicationManager ApplicationFactoryComponent	Software Assembly Descriptor, Software Package Descriptor, Software Component Descriptor, Properties Descriptor	ComponentPackageDescription, ComponentInterfaceDescription, ComponentAssemblyDescription, MonolithicImplementationDescription, and SWRadio Properties XML
DeviceManagerComponent	Device Configuration Descriptor	
DomainManagerComponent	Domain Configuration Descriptor	
SWRadioComponent ServiceComponent	Software Assembly Descriptor, Software Package Descriptor, Software Component Descriptor, Properties Descriptor	ComponentPackageDescription, ComponentInterfaceDescription, ComponentAssemblyDescription, MonolithicImplementationDescription, and SWRadio Properties XML

Annex A Core Framework CORBA IDL (non-normative)

Note – Issue 7845 - Section changed to non-normative

A.1 Base Types Interfaces

A.1.1 CF Common Types Interface

```
//File: CFCommonTypes.idl
```

A.1.1 CF Common Types Interface

```
#ifndef __CFOMMONTYPES_DEFINED
#define __CFCOMMONTYPES_DEFINED

#pragma prefix "omg.org"

module CF {

   typedef sequence <octet> OctetSequence;
   typedef sequence <string> StringSequence;

   struct DataType {
      string id;
         any value;
   };

   typedef sequence <DataType> Properties;
   typedef DataType PropertyValue;

   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_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
   };
```

Note – Issue 7895, added TimeType

```
   struct TimeType {
      unsigned long seconds;
      unsigned long nanoseconds;
   };

   exception InvalidFileName {
      ErrorNumberType errorNumber;
      string msg;
   };

};

#endif
```

A.1.2 CF Base Types Interface

```
//File: CFBaseTypes.idl

#ifndef __CFBASETYPES_DEFINED
#define __CFBASETYPES_DEFINED

#pragma prefix "omg.org"

module CF {

   exception InvalidObjectReference {
      string msg;
   };

   exception InvalidProfile {};

   typedef Object Service;
};

#endif
```

A.2 CF Resource Interfaces

```
//File: CFResources.idl

#ifndef __CFRESOURCES_DEFINED
#define __CFRESOURCES_DEFINED

#include "CFBaseTypes.idl"
#include "CFCommonTypes.idl"

#pragma prefix "omg.org"

module CF {

   interface ComponentIdentifier {
      readonly attribute string identifier;
   };

   exception UnknownProperties {
      Properties invalidProperties;
   };

   interface ControllableComponent {

      exception StartError {
         ErrorNumberType errorNumber;
         string msg;
      };

      exception StopError {
         ErrorNumberType errorNumber;
         string msg;
      };

      readonly attribute boolean started;

      void start ()
         raises (StartError);

      void stop ()
         raises (StopError);

   };

   interface LifeCycle {
      exception InitializeError {
         StringSequence errorMessages;
      };

      exception ReleaseError {
         StringSequence errorMessages;
      };

      void initialize ()
         raises (InitializeError);

      void releaseObject ()
         raises (ReleaseError);

   };

   interface PortConnector {

      exception InvalidPort {
         string msg;
         unsigned short errorCode;
      };

      exception OccupiedPort {};

      void connectPort (
         in string requiredPortName,
         in Object connection,
         in string connectionId
         )
         raises (InvalidPort,OccupiedPort);
```

Note – Issue 7580 Resolution

```
      void disconnectPort (
         in string requiredPortName,
         in string connectionId
         )
         raises (InvalidPort);

   };
```

Note – Issue 7579 Resolution

```
   interface PortSupplier {
      struct PortType {
         string name;
         Object objectRef;
      };

      typedef sequence <PortType> PortSequence;

      exception UnknownPorts {
         StringSequence invalidPorts;
      };

      void getProvidedPorts (
         inout PortSequence ports
         )
         raises (UnknownPorts);

   };

   interface PropertySet {

      exception InvalidConfiguration {
         Properties invalidProperties;
         string msg;
      };

      exception PartialConfiguration {
         StringSequence reasons;
         Properties invalidProperties;
      };

      void configure (
         in Properties configProperties
         )
         raises (InvalidConfiguration,PartialConfiguration);

      void query (
         inout Properties configProperties
         )
         raises (UnknownProperties);

   };

   interface TestableObject {

      exception UnknownTest {};

      void runTest (
         in string testid,
         inout Properties testValues
         )
         raises (UnknownTest,UnknownProperties);
   };

   interface Resource : LifeCycle, TestableObject, PropertySet,
      PortSupplier, ControllableComponent,
      PortConnector, ComponentIdentifier {};

};

#endif
```

A.3 CF ResourceFactory Interfaces

```
//File: CFResourceFactory.idl

#ifndef __CFRESOURCEFACTORY_DEFINED
#define __CFRESOURCEFACTORY_DEFINED

#include "CFResources.idl"
#include "CFCommonTypes.idl"

#pragma prefix "omg.org"

module CF {

   interface ResourceFactory : ComponentIdentifier {

      exception InvalidResourceId {};

      exception ShutdownFailure {
         string msg;
      };

      exception CreateResourceFailure {
         ErrorNumberType errorNumber;
         string msg;
      };

      Resource createResource (
         in string resourceId,
         in Properties qualifiers
         )
         raises (CreateResourceFailure);

      void releaseResource (
         in string resourceId
         )
         raises (InvalidResourceId);
      void shutdown ()
         raises (ShutdownFailure);

   };

};

#endif
```

A.4 CF Devices Interfaces

Note – Issue 7581

```
   //File: CFDevices.idl

   #ifndef __CFDEVICES_DEFINED
   #define __CFDEVICES_DEFINED

   #include "CFResources.idl
```

Note – Issue 7581

```
   #include "CFFileSystem.idl"
   #include "CFStateManagement.idl"
   #include "CFBaseTypes.idl"
   #include "CFCommonTypes.idl"

   #pragma prefix "omg.org"

   module CF {
```

Note – Issue 8842

```
      interface DeviceComposition;
      interface Device;

      typedef sequence <Device> DeviceSequence;

      interface DeviceComposition {

         readonly attribute DeviceSequence compositeParts;

         void addDevice (
            in Device associatedDevice
            )
            raises (InvalidObjectReference);
```

```
   void removeDevice (
      in Device associatedDevice
      )
      raises (InvalidObjectReference);

};

interface Device : Resource, StateManagement {

   exception InvalidState {
      string msg;
   };

   exception InvalidCapacity {
      string msg;
      Properties capacities;
   };

   readonly attribute string softwareProfile;
   readonly attribute string label;
   readonly attribute DeviceComposition compositeDevice;

   boolean allocateCapacity (
      in Properties capacities
      )
      raises (InvalidCapacity,InvalidState);

   void deallocateCapacity (
      in Properties capacities
      )
      raises (InvalidCapacity,InvalidState);

};

interface LoadableDevice : Device {

   enum LoadType {
      SHARED_LIBRARY,
      EXECUTABLE,
      KERNEL_MODULE,
      DRIVER
   };

   exception InvalidLoadKind {};

   exception LoadFail {
      ErrorNumberType errorNumber;
      string msg;
   };
```

Note – Issue 7581

```
      void load (
         in FileSystem fs,
         in string fileName,
         in LoadType loadKind
         )
         raises (InvalidState,InvalidLoadKind,InvalidFileName,LoadFail);

      void unload (
         in string fileName
         )
         raises (InvalidState,InvalidFileName);

   };

   interface ExecutableDevice : LoadableDevice {

      exception InvalidProcess {
         ErrorNumberType errorNumber;
         string msg;
      };

      exception InvalidFunction {};
```

Note – Issue 8949, change unsigned long to long

```
      typedef long ProcessID_Type;

      exception InvalidParameters {
         Properties invalidParms;
      };

      exception InvalidOptions {
         Properties invalidOpts;
      };

      const string STACK_SIZE = "STACK_SIZE";
      const string PRIORITY_ID = "PRIORITY";

      exception ExecuteFail {
         ErrorNumberType errorNumber;
         string msg;
      };
      const string THREAD_CREATE_REQUEST = "CREATE_THREAD";
      const string RUNTIME_OPTIONS = "RUNTIME_OPTIONS";
      const string RUNTIME_REQUEST = "RUNTIME_REQUEST";

      void terminate (
         in ProcessID_Type processId
         )
         raises (InvalidProcess,InvalidState);

      ProcessID_Type execute (
         in string name,
         in Properties options,
         in Properties parameters
         )
         raises (InvalidState,InvalidFunction,InvalidParameters,
            InvalidOptions,InvalidFileName,ExecuteFail);

   };

};

#endif
```

A.5 CF DeviceManager Interfaces

A.5.1 CF Service Registration Interface

```
//File: CFServiceRegistration.idl

#ifndef __CFSERVICEREGISTRATION_DEFINED
#define __CFSERVICEREGISTRATION_DEFINED

#include "CFBaseTypes.idl"

#pragma prefix "omg.org"

module CF {

   interface ServiceRegistration {

      void registerService (
         in Service registeringService,
         in string name
         )
         raises (InvalidObjectReference);

      void unregisterService (
         in Service registeredService,
         in string name
         )
         raises (InvalidObjectReference);
   };

};

#endif
```

A.5.2 CF DeviceManager Interface

```
   //Source file: CFDeviceManager.idl

   #ifndef __CFDEVICEMANAGER_DEFINED
   #define __CFDEVICEMANAGER_DEFINED

   #include "CFBaseTypes.idl"
   #include "CFCommonTypes.idl"
   #include "CFResources.idl"
```

Note – Issue7581

```
   #include "CFFileSystem.idl"

   #pragma prefix "omg.org"

   module CF {

      interface DeviceManager : PropertySet, PortConnector,
         ComponentIdentifier, PortSupplier {

         struct ServiceType {
            Service serviceObject;
            string serviceName;
         };

         typedef sequence <ServiceType> ServiceSequence;

         readonly attribute string deviceConfigurationProfile;
```

Note – Issue 7581

```
         readonly attribute FileSystem fileSys;
         readonly attribute string label;
         readonly attribute ServiceSequence registeredServices;

         void shutdown ();
```

```
        string getComponentImplementationId (
           in string componentInstantiationId
           );
```

Note – Issue 7905 Resolution - Deleted portExists operation

```
     };

  };

  #endif
```

A.6 CF DomainManager Interfaces

A.6.1 CF Domain Event Channels Interface

```
  //Source file: CFDomainEventChannels.idl

  #ifndef __CFDOMAINEVENTCHANNELS_DEFINED
  #define __CFDOMAINEVENTCHANNELS_DEFINED

  #include "CFBaseTypes.idl"

  #pragma prefix "omg.org"

  module CF {

     interface DomainEventChannels {

        exception AlreadyConnected {};
        exception InvalidEventChannelName {};
        exception NotConnected {};

        void registerWithEventChannel (
           in Object registeringObject,
           in string registeringId,
           in string eventChannelName
           )
           raises (InvalidObjectReference,
                   InvalidEventChannelName,AlreadyConnected);

        void unregisterFromEventChannel (
           in string unregisteringId,
           in string eventChannelName
           )
           raises (InvalidEventChannelName,NotConnected);

     };
```

```
};

#endif
```

A.6.2 CF Domain Installation Interface

```
//File: CFDomainInstallation.idl

#ifndef __CFDOMAININSTALLATION_DEFINED
#define __CFDOMAININSTALLATION_DEFINED

#include "CFCommonTypes.idl"
#include "CFBaseTypes.idl"

#pragma prefix "omg.org"

module CF {

   interface DomainInstallation {

      exception ApplicationInstallationError {
         ErrorNumberType errorNumber;
         string msg;
      };

      exception InvalidIdentifier {};

      exception ApplicationUninstallationError {
         ErrorNumberType errorNumber;
         string msg;
      };

      void installApplication (
         in string profileFileName
         )
         raises (InvalidProfile,InvalidFileName,
                 ApplicationInstallationError);

      void uninstallApplication (
         in string applicationId
         )
         raises (InvalidIdentifier,ApplicationUninstallationError);

   };

};

#endif
```

A.6.3 CF Device Manager Registration Interface

```
//File: CFDeviceManagerRegistration.idl

#ifndef __CFDEVICEMANAGERREGISTRATION_DEFINED
#define __CFDEVICEMANAGERREGISTRATION_DEFINED

#include "CFDeviceManager.idl"
#include "CFBaseTypes.idl"

#pragma prefix "omg.org"

module CF {

   interface DeviceManagerRegistration {

      exception DeviceManagerNotRegistered {};

      exception RegisterError {
         ErrorNumberType errorNumber;
         string msg;
      };

      exception UnregisterError {
         ErrorNumberType errorNumber;
         string msg;
      };

      void registerDeviceManager (
         in DeviceManager deviceMgr
         )
         raises (InvalidObjectReference,InvalidProfile,RegisterError);

      void unregisterDeviceManager (
         in DeviceManager deviceMgr
         )
         raises (InvalidObjectReference,UnregisterError);

      void registerService (
         in Object registeringService,
         in DeviceManager registeredDeviceMgr,
         in string name
         )
         raises (InvalidObjectReference,InvalidProfile,
                 DeviceManagerNotRegistered,RegisterError);
```

```
        void unregisterService (
           in Object unregisteringService,
           in string name
           )
           raises (InvalidObjectReference,UnregisterError);

     };

  };

  #endif
```

A.6.4 CF DomainManager Interface

```
  //File: CFDomainManager.idl

  #ifndef __CFDOMAINMANAGER_DEFINED
  #define __CFDOMAINMANAGER_DEFINED

  #include "CFDeviceManager.idl"
```

Note – Issue 7581

```
  #include "CFFileManager.idl"
  #include "CFResources.idl"
  #include "CFApplications.idl"

  #pragma prefix "omg.org"

  module CF {

     interface DomainManager : PropertySet, PortSupplier {

        typedef sequence <Application> ApplicationSequence;
        typedef sequence <ApplicationFactory> ApplicationFactorySequence;
        typedef sequence <DeviceManager> DeviceManagerSequence;

        readonly attribute DeviceManagerSequence deviceManagers;
        readonly attribute ApplicationSequence applications;
        readonly attribute ApplicationFactorySequence applicationFactories;
```

Note – Issue 7581

```
        readonly attribute FileManager fileMgr;
        readonly attribute string domainManagerProfile;
```

Note – Issue 7905 Resolution - Deleted portExists operation

```
      };

   };

   #endif
```

A.7 CF Application Interfaces

```
   //File: CFApplications.idl

   #ifndef __CFAPPLICATIONS_DEFINED
   #define __CFAPPLICATIONS_DEFINED

   #include "CFResources.idl"
   #include "CFCommonTypes.idl"

   #pragma prefix "omg.org"

   module CF {

      interface Application;

      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 : ComponentIdentifier {

         exception CreateApplicationRequestError {
            DeviceAssignmentSequence invalidAssignments;
         };

         exception CreateApplicationError {
            ErrorNumberType errorNumber;
            string msg;
         };

         exception InvalidInitConfiguration {
            Properties invalidProperties;
         };

         readonly attribute string name;
         readonly attribute string softwareProfile;

         Application create (
            in string name,
            in Properties initConfiguration,
            in DeviceAssignmentSequence deviceAssignments
            )
            raises (CreateApplicationError,CreateApplicationRequestError,
                    InvalidInitConfiguration);

      };

   };

   #endif
```

A.8 CF StateManagement Interface

```
//File: CFStateManagement.idl

#ifndef __CFSTATEMANAGEMENT_DEFINED
#define __CFSTATEMANAGEMENT_DEFINED

#pragma prefix "omg.org"
```

```
module CF {

   interface StateManagement {

      enum AdminType {
         SHUTTING_DOWN,
         UNLOCKED,
         LOCKED
      };

      enum OperationalType {
         ENABLED,
         DISABLED
      };

      enum UsageType {
         IDLE,
         ACTIVE,
         BUSY
      };

      enum AdminRequestSupportedType {
         ALL,
         NOT_IMPLEMENTED,
         LOCK_REQUEST,
         SHUTDOWN_REQUEST,
         UNLOCK_REQUEST
      };

      struct StatesType {
         AdminType adminState;
         OperationalType operationalState;
         UsageType usageState;
      };

      exception UnsupportedRequest {};

      enum AdminRequestType {
         LOCK,
         SHUTDOWN,
         UNLOCK
      };

      readonly attribute UsageType usageState;
      readonly attribute AdminType adminState;
      readonly attribute OperationalType operationalState;
      readonly attribute StatesType states;
      readonly attribute AdminRequestSupportedType
                         adminStateRequestSupportedCharacterisitic;
      void setAdminState (
         in AdminRequestType adminRequest
         )
         raises (UnsupportedRequest);

   };

};

#endif
```

A.9 CF Port Types

A.9.1 Boolean Sequence Port Type

```
//File: CFPortTypes_BooleanSequence.idl

#ifndef __PORTTYPES_BOOLEANSEQUENCE_DEFINED
#define __PORTTYPES_BOOLEANSEQUENCE_DEFINED

#pragma prefix "omg.org"

module CF {

   module PortTypes {

      typedef sequence <boolean> BooleanSequence;

   };

};

#endif
```

A.9.2 Char Sequence Port Type

```
//File: CFPortTypes_CharSequence.idl

#ifndef __PORTTYPES_CHARSEQUENCE_DEFINED
#define __PORTTYPES_CHARSEQUENCE_DEFINED

#pragma prefix "omg.org"

module CF {

   module PortTypes {
      typedef sequence <char> CharSequence;

   };

};

#endif
```

A.9.3 Short Sequence Port Type

```
//File: CFPortTypes_ShortSequence.idl

#ifndef __PORTTYPES_SHORTSEQUENCE_DEFINED
#define __PORTTYPES_SHORTSEQUENCE_DEFINED

#pragma prefix "omg.org"

module CF {

   module PortTypes {

      typedef sequence <short> ShortSequence;

   };

};

#endif
```

A.9.4 Ushort Sequence Port Type

```
//File: CFPortTypes_UshortSequence.idl

#ifndef __PORTTYPES_USHORTSEQUENCE_DEFINED
#define __PORTTYPES_USHORTSEQUENCE_DEFINED

#pragma prefix "omg.org"

module CF {

   module PortTypes {

      typedef sequence <unsigned short> UshortSequence;

   };

};
```

```
#endif
```

A.9.5 Long Sequence Port Type

```
//File: CFPortTypes_LongSequence.idl

#ifndef __PORTTYPES_LONGSEQUENCE_DEFINED
#define __PORTTYPES_LONGSEQUENCE_DEFINED

#pragma prefix "omg.org"

module CF {

   module PortTypes {

      typedef sequence <long> LongSequence;

   };

};

#endif
```

A.9.6 Ulong Sequence Port Type

```
//File: CFPortTypes_UlongSequence.idl

#ifndef __PORTTYPES_ULONGSEQUENCE_DEFINED
#define __PORTTYPES_ULONGSEQUENCE_DEFINED

#pragma prefix "omg.org"

module CF {

   module PortTypes {

      typedef sequence <unsigned long> UlongSequence;

   };

};

#endif
```

A.9.7 LongLong Sequence Port Type

```
//File: CFPortTypes_LongLongSequence.idl

#ifndef __PORTTYPES_LONGLONGSEQUENCE_DEFINED
#define __PORTTYPES_LONGLONGSEQUENCE_DEFINED

#pragma prefix "omg.org"

module CF {

   module PortTypes {

      typedef sequence <long long> LongLongSequence;

   };

};

#endif
```

A.9.8 UlongLong Sequence Port Type

```
//File: CFPortTypes_UlongLongSequence.idl

#ifndef __PORTTYPES_ULONGLONGSEQUENCE_DEFINED
#define __PORTTYPES_ULONGLONGSEQUENCE_DEFINED

#pragma prefix "omg.org"

module CF {

   module PortTypes {

      typedef sequence <unsigned long long> UlongLongSequence;

   };

};

#endif
```

A.9.9 Float Sequence Port Type

```
//File: CF_PortTypes_FloatSequence.idl
```

```
#ifndef __PORTTYPES_FLOATSEQUENCE_DEFINED
#define __PORTTYPES_FLOATSEQUENCE_DEFINED

#pragma prefix "omg.org"

module CF {

   module PortTypes {

      typedef sequence <float> FloatSequence;

   };

};

#endif
```

A.9.10 Double Sequence Port Type

```
//File: CFPortTypes_DoubleSequence.idl

#ifndef __PORTTYPES_DOUBLESEQUENCE_DEFINED
#define __PORTTYPES_DOUBLESEQUENCE_DEFINED

#pragma prefix "omg.org"

module CF {

   module PortTypes {

      typedef sequence <double> DoubleSequence;

   };

};

#endif
```

A.9.11 LongDouble Sequence Port Type

```
//File: CFPortTypes_LongDoubleSequence.idl

#ifndef __PORTTYPES_LONGDOUBLESEQUENCE_DEFINED
#define __PORTTYPES_LONGDOUBLESEQUENCE_DEFINED

#pragma prefix "omg.org"
```

```
module CF {

   module PortTypes {

      typedef sequence <long double> LongDoubleSequence;

   };

};

#endif
```

A.9.12 Wchar Sequence Port Type

```
//File: CFPortTypes_WcharSequence.idl

#ifndef __PORTTYPES_WCHARSEQUENCE_DEFINED
#define __PORTTYPES_WCHARSEQUENCE_DEFINED

#pragma prefix "omg.org"

module CF {

   module PortTypes {

      typedef sequence <wchar> WcharSequence;

   };

};

#endif
```

A.9.13 Wstring Sequence Port Type

```
//File: CFPortTypes_WstringSequence.idl

#ifndef __PORTTYPES_WSTRINGSEQUENCE_DEFINED
#define __PORTTYPES_WSTRINGSEQUENCE_DEFINED

#pragma prefix "omg.org"

module CF {

   module PortTypes {

      typedef sequence <wstring> WstringSequence;
   };

};

#endif
```

Note – Issue 7587

A.9.14 CF Port Types CORBA Module

```
   //File: CFPortTypes.idl

   #ifndef __CFPORTTYPES_DEFINED
   #define __CFPORTTYPES_DEFINED

   #include "CFPortTypes_UlongLongSequence.idl"
   #include "CFPortTypes_LongDoubleSequence.idl"
   #include "CFPortTypes_BooleanSequence.idl"
   #include "CFPortTypes_UlongSequence.idl"
   #include "CFPortTypes_LongLongSequence.idl"
   #include "CFPortTypes_CharSequence.idl"
   #include "CFPortTypes_UshortSequence.idl"
   #include "CFPortTypes_LongSequence.idl"
   #include "CFPortTypes_DoubleSequence.idl"
   #include "CFPortTypes_ShortSequence.idl"
   #include "CFPortTypes_WcharSequence.idl"
   #include "CFPortTypes_FloatSequence.idl"
   #include "CFPortTypes_WstringSequence.idl"

   #endif
```

A.10 CF Event Types

A.10.1 Domain Event

```
   //File: CF_SE_DomainEvent.idl

   #ifndef __SE_DOMAINEVENT_DEFINED
   #define __SE_DOMAINEVENT_DEFINED

   #pragma prefix "omg.org"

   #include "CF_SE_StateEvent.idl"

   module CF {
      module StandardEvent {

         struct DomainManagementObjectRemovedEventType {
            string producerId;
            string sourceId;
            string sourceName;
            SourceCategoryType sourceCategory;
         };

         struct DomainManagementObjectAddedEventType {
            string producerId;
            string sourceId;
            string sourceName;
            SourceCategoryType sourceCategory;
            Object sourceReference;
         };

      };

   };

   #endif
```

A.10.2 State Event

```
   //File: CF_SE_StateEvent.idl

   #ifndef __SE_STATEEVENT_DEFINED
   #define __SE_STATEEVENT_DEFINED

   #pragma prefix "omg.org"

   module CF {

      module StandardEvent {

         enum StateChangeCategoryType {
            ADMINISTRATIVE_STATE_EVENT,
            OPERATIONAL_STATE_EVENT,
            USAGE_STATE_EVENT
         };

         enum StateChangeType {
            LOCKED,
            UNLOCKED,
            SHUTTING_DOWN,
            ENABLED,
            DISABLED,
            IDLE,
            ACTIVE,
            BUSY
         };

         typedef unsigned short SourceCategoryType;

         struct StateChangeEventType {
            string producerId;
            string sourceId;
            StateChangeCategoryType stateChangeCategory;
            StateChangeType stateChangeFrom;
            StateChangeType stateChangeTo;
         };

         const SourceCategoryType APPLICATION = 1;
         const SourceCategoryType APPLICATION_FACTORY = 2;
         const SourceCategoryType COMM_CHANNEL = 3;
         const SourceCategoryType DEVICE = 4;
         const SourceCategoryType DEVICE_MANAGER = 5;
         const SourceCategoryType DOMAIN_MANAGER = 6;
         const SourceCategoryType SERVICE = 7;

      };

   };

   #endif
```

Note – Issue 7587

A.11 Core Framework CORBA Module

```
   //File: CF.idl



   #ifndef __CF_DEFINED
   #define __CF_DEFINED

   #include "CFPortTypes.idl"
   #include "CFDevices.idl"
   #include "CFResourceFactory.idl"
   #include "CFServiceRegistration.idl"
   #include "CF_SE_DomainEvent.idl"
```

```
#include "CFDomainEventChannels.idl"
#include "CFDomainManager.idl"
#include "CFDeviceManagerRegistration.idl"
#include "CFDomainInstallation.idl"


#endif
```

A.11 Core Framework CORBA Module

Annex B Common Layer Facilities CORBA IDL (non-normative)

Note – Issue 7845 - Section changed to non-normative

B.1 Common Layer Basic Types

```
//File: DfSWRadioCommonLayerBasicTypes.idl

#ifndef __DFSWRADIOCOMMONLAYERBASICTYPES_DEFINED
#define __DFSWRADIOCOMMONLAYERBASICTYPES_DEFINED

#include "CFCommonTypes.idl"

#pragma prefix "omg.org"

module DfSWRadio {

   module CommonLayer {

      typedef CF::OctetSequence AddressType;
```

Note – Issue 7895 - moved TimeType to A.1.1

```
      struct SduSizeType {
         unsigned long maxSduSize;
         unsigned long minSduSize;
      };

   };

};

#endif
```

B.2 Error Control Interfaces

B.2.1 Error Control Management Interface

```
//File: DfSWRadioErrorControlManagement.idl

#ifndef __DFSWRADIOERRORCONTROLMANAGEMENT_DEFINED
#define __DFSWRADIOERRORCONTROLMANAGEMENT_DEFINED

#pragma prefix "omg.org"

module DfSWRadio {

   module CommonLayer {

      module ErrorControl {

         struct ErrorControlParamsType {
            boolean errorControl;
            boolean slidingWindowARQ;
            boolean ARQStopWait;
            boolean forwardErrorCorrection;
         };
```

Note – Issue 7878 Resolution (Renamed “ErrorControl” interface to “Error_Control”)

```
         interface Error_Control {
            attribute ErrorControlParamsType errorControlParams;
            void estimateSequenceNumber ();
            void checkSequenceNumber ();
            void requestRetransmit ();
            void reportReceptionError ();
            void checkFrameError ();
            void forwardErrorCorrection ();
         };

      };

   };

};

#endif
```

B.2.2 Signal Interface

```
//File: DfSWRadioErrorControlSignal.idl
```

```
#ifndef __DFSWRADIOERRORCONTROLSIGNAL_DEFINED
#define __DFSWRADIOERRORCONTROLSIGNAL_DEFINED

#pragma prefix "omg.org"

#include "CFCommonTypes.idl"

module DfSWRadio {

   module CommonLayer {

      module ErrorControl {

         interface Signal {

            oneway void signalStatus (
               in CF::Properties status
               );

         };

      };

   };

};

#endif
```

Note – Issue 7587

B.2.3 DfSWRadio Error Control CORBA Module

```
//Source file: DfSWRadioErrorControl.idl

#ifndef __DFSWRADIOERRORCONTROL_DEFINED
#define __DFSWRADIOERRORCONTROL_DEFINED

#include "DfSWRadioErrorControlManagement.idl"
#include "DfSWRadioErrorControlSignal.idl"

#endif
```

B.3 Flow Control Interfaces

B.3.1 Flow Control Management Interface

```
//File: DfSWRadioFlowControlManagement.idl
```

B.3.1 Flow Control Management Interface

```
#ifndef __DFSWRADIOFLOWCONTROLMANAGEMENT_DEFINED
#define __DFSWRADIOFLOWCONTROLMANAGEMENT_DEFINED

#pragma prefix "omg.org"

module DfSWRadio {

   module CommonLayer {

      module FlowControl {

         interface FlowControlManagement {

            attribute boolean priorityHandling;
            attribute boolean flowControlSignaling;
            attribute double dataRate;
            attribute boolean emptySignalling;

            void negotiateFlowControl ();
            void tearDownFlowControl ();

         };

         interface PriorityFlowControl : FlowControlManagement {

            readonly attribute unsigned short numPriorityQueues;

            octet createPriorityQueue (
               in long priority,
               in long queueSize,
               in long highWaterMarkThreshold,
               in long lowWaterMarkThreshold
               );

            void destroyPriorityQueue (
               in octet priorityQueueID
               );

            octet createWindowedPriorityQueue (
               in long priority,
               in long queueSize,
               in long highWaterMarkThreshold,
               in long lowWaterMarkThreshold
               );

         };

      };

   };
};

#endif
```

B.3.2 Flow Control Signaling Interface

```
//File: DfSWRadioFlowControlSignaling.idl

#ifndef __DFSWRADIOFLOWCONTROLSIGNALING_DEFINED
#define __DFSWRADIOFLOWCONTROLSIGNALING_DEFINED

#pragma prefix "omg.org"

module DfSWRadio {

   module CommonLayer {

      module FlowControl {

         interface FlowControlSignalling {

            oneway void signalCongestion (
               in octet priorityQueueID
               );

            oneway void signalHighWatermark (
               in octet priorityQueueID
               );

            oneway void signalLowWatermark (
               in octet priorityQueueID
               );

            oneway void signalEmpty (
               in octet priorityQueueID
               );

            oneway void signalACK (
               in octet priorityQueueID
               );

            oneway void signalNAK (
               in octet priorityQueueID
               );

         };

      };
   };

};

#endif
```

Note – Issue 7587

B.3.3 DfSWRadio Flow Control CORBA Module

```
   //File: DfSWRadioFlowControl.idl

   #ifndef __DFSWRADIOFLOWCONTROL_DEFINED
   #define __DFSWRADIOFLOWCONTROL_DEFINED

   #include "DfSWRadioFlowControlManagement.idl"
   #include "DfSWRadioFlowControlSignaling.idl"

   #endif
```

B.4 Measurement Interfaces

B.4.1 Measurement Types

```
   //File: DfSWRadioMeasurementTypes.idl

   #ifndef __DFSWRADIOMEASUREMENTTYPES_DEFINED
   #define __DFSWRADIOMEASUREMENTTYPES_DEFINED
```

Note – Issue 8980 - Remove “#include "DfSWRadioCommonLayerBasicTypes.idl" statement

```
   #include "CFCommonTypes.idl"

   #pragma prefix "omg.org"

   module DfSWRadio {

      module CommonLayer {

         module Measurement {
```

Note – Issue 7878 Resolution (Renamed “Measurement” type to “MeasurementType”)

```
            struct MeasurementType {
               string sourceId;
               string pointId;
```

Note – Issue 7895, move TimeType to CF Common Types

```
            CF::TimeType timeStamp;
            CF::Properties data;
         };

      };

   };

};

#endif
```

B.4.2 Measurement Management Interfaces

```
//File: DfSWRadioMeasurementManagement.idl

#ifndef __DFSWRADIOMEASUREMANAGEMENT_DEFINED
#define __DFSWRADIOMEASUREMANAGEMENT_DEFINED
```

Note – Issue 7895 - moved TimeType CFCommonTypes.idl

Note – Issue 8980 - Remove “#include "DfSWRadioCommonLayerBasicTypes.idl" statement

```
#include "CFCommonTypes.idl"

#include "DfSWRadioMeasurementPoint.idl"
#include "DfSWRadioMeasurementStorage.idl"


#pragma prefix "omg.org"

module DfSWRadio {

   module CommonLayer {

      module Measurement {

         typedef sequence <MeasurementPoint> MeasurementPointSequence;
         typedef sequence <MeasurementStorage> MeasurementStorageSequence;

         interface MeasurementPlan {
```

```
        attribute string name;
        readonly attribute boolean activated;
```

Note – Issue 7895 - moved TimeType CFCommonTypes.idl

```
        attribute CF::TimeType deferred;

        MeasurementStorageSequence listStorages ();

        MeasurementStorage createStorage (
           in string fileName
           );

        void setStorage (
           in MeasurementStorage storage
           );

        MeasurementPointSequence listPoints ();

        void addPoint (
           in MeasurementPoint point
           );

        void removePoint (
           in string pointId
           );

        void removeStorage (
           in string storageId
           );

     };

     typedef sequence <MeasurementPlan> MeasurementPlanSequence;

     interface MeasurementPlanManager {

        readonly attribute boolean activated;
        attribute string planId;
```

Note – Issue 7895 - moved TimeType CFCommonTypes.idl

```
        attribute CF::TimeType startTime;

        MeasurementPlan createPlan (
           in string name
           );
```

```
            MeasurementPlanSequence listPlans ();

            void start ();
            void suspend ();
            void stop ();

         };

      };

   };

};

#endif
```

B.4.3 Measurement Point Interface

```
//File: DfSWRadioMeasurementPoint.idl

#ifndef __DFSWRADIOMEASUREMENTPOINT_DEFINED
#define __DFSWRADIOMEASUREMENTPOINT_DEFINED
```

Note – Issue 8980 - Replace “#include "DfSWRadioCommonLayerBasicTypes.idl" statement with “#include "CFCommonTypes.idl" statement

```
#include "CFCommonTypes.idl"
#include "DfSWRadioMeasurementStorage.idl"

#pragma prefix "omg.org"

module DfSWRadio {

   module CommonLayer {

      module Measurement {

         interface MeasurementPoint {

            readonly attribute string identifier;
```

Note – Issue 7895 - moved TimeType CFCommonTypes.idl

```
                attribute CF::TimeType delay;
                attribute MeasurementStorage storage;
                readonly attribute string dataType;
                readonly attribute boolean activated;

                void activate ();
                void deactivate ();

            };

        };

    };

};

#endif
```

B.4.4 Measurement Recorder Interface

```
//File: DfSWRadioMeasurementRecorder.idl

#ifndef __DFSWRADIOMEASUREMENTRECORDER_DEFINED
#define __DFSWRADIOMEASUREMENTRECORDER_DEFINED

#include "DfSWRadioMeasurementTypes.idl"

#pragma prefix "omg.org"

module DfSWRadio {

   module CommonLayer {

      module Measurement {

         interface MeasurementRecorder {
```

Note – Issue 7878 Resolution (Renamed “Measurement” type to “MeasurementType”)

```
               oneway void record (
                  in MeasurementType in_measurement
                  );

         };

      };

   };
```

```
};

#endif
```

B.4.5 Measurement Storage Interface

```
//File: DfSWRadioMeasurementStorage.idl

#ifndef __DFSWRADIOMEASUREMENTSTORAGE_DEFINED
#define __DFSWRADIOMEASUREMENTSTORAGE_DEFINED

#include "CFCommonTypes.idl"

#include "DfSWRadioMeasurementTypes.idl"

#pragma prefix "omg.org"

module DfSWRadio {

   module CommonLayer {

      module Measurement {

         enum StoragePolicyType {
            ONESHOT,
            CIRCULAR
         };
```

Note – Issue 7878 Resolution (Renamed “Measurement” type to “MeasurementType”)

```
         typedef sequence <MeasurementType> MeasurementSequence;

         interface MeasurementStorage {

            attribute string fileName;
            attribute StoragePolicyType storagePolicy;
            attribute unsigned long maxSize;

            MeasurementSequence query (
               in CF::Properties queryProperties
               );

            void clear ();

            void truncate (
               in long size
               );
            void remove ();

         };

      };

   };

};

#endif
```

Note – Issue 7587

B.4.6 DfSWRadio Measurement CORBA Module

```
   //File: DfSWRadioMeasurement.idl

   #ifndef __DFSWRADIOMEASUREMENT_DEFINED
   #define __DFSWRADIOMEASUREMENT_DEFINED

   #include "DfSWRadioMeasurementManagement.idl"
   #include "DfSWRadioMeasurementRecorder.idl"

   #endif
```

B.5 PDU Interfaces

```
   //File: DfSWRadioPDU.idl

   #ifndef __DFSWRADIOPDU_DEFINED
   #define __DFSWRADIOPDU_DEFINED
```

Note – Issue 8296

```
   #include "DfSWRadioCommonLayerBasicTypes.idl"
   #include "CFCommonTypes.idl"
   #include "DfSWRadioFlowControl.idl"

   #pragma prefix "omg.org"

   module DfSWRadio {

      module CommonLayer {

         module PDUFacilities {
            struct ControlHeaderType {
               DfSWRadio::CommonLayer::AddressType sourceAddress;
               DfSWRadio::CommonLayer::AddressType destinationAddress;
               DfSWRadio::CommonLayer::SduSizeType sduSize;
               long priority;
               long sequenceNumber;
            };

            interface BasePdu {

               attribute DfSWRadio::CommonLayer::SduSizeType sduSize;

            };

            interface ConcretePdu : BasePdu {

               oneway void pushPDU (
                  in ControlHeaderType control,
                  in CF::OctetSequence sdu
                  );

            };

            interface ConcreteDataPdu : BasePdu {

               oneway void pushPDU (
                  in CF::OctetSequence sdu
                  );

            };

         };

      };

   };

   #endif
```

B.6 Quality of Service Interface

```
   //File: DfSWRadioQoSManagement.idl

   #ifndef __DFSWRADIOQOSMANAGEMENT_DEFINED
   #define __DFSWRADIOQOSMANAGEMENT_DEFINED

   #pragma prefix "omg.org"

   module DfSWRadio {
      module CommonLayer {

         module QosManagement {

            interface QualityOfService {

               void transmitQoSParameters ();
               void negotiateQoSParameters ();

            };

         };

   };

   #endif
```

B.7 Stream Interface

```
   //File: DfSWRadioStreamControl.idl

   #ifndef __DFSWRADIOSTREAMCONTROL_DEFINED
   #define __DFSWRADIOSTREAMCONTROL_DEFINED

   #include "DfSWRadioCommonLayerBasicTypes.idl"

   #pragma prefix "omg.org"

   module DfSWRadio {

      module CommonLayer {

         module StreamControl {

            interface Stream {

               octet establishStream (
                   in AddressType sourceAddress,
                   in AddressType destinationAddress,
                   in long priority
                  );

               void releaseStream (
                   in octet streamID
                  );

               void localSetup ();

            };
```

```
        };

    };

};

#endif
```

Note – Issue 7587

B.8 DfSWRadio Common Layer Module

```
//File: DfSWRadioCommonLayer.idl

#ifndef __DFSWRADIOCOMMONLAYER_DEFINED
#define __DFSWRADIOCOMMONLAYER_DEFINED

#include "DfSWRadioErrorControl.idl"
#include "DfSWRadioMeasurement.idl"
#include "DfSWRadioPDU.idl"
#include "DfSWRadioQoSManagement.idl"
#include "DfSWRadioStreamControl.idl"
#include "DfSWRadioFlowControl.idl"

#endif
```

B.8 DfSWRadio Common Layer Module

Annex C Common Radio Facilities CORBA IDL (non-normative)

Note – Issue 7845 - Section changed to non-normative

Note – Issue 7581 CORBA module name changed to CF

C.1 CF File Services Interfaces

C.1.1 CF File Interface

```
//Source file: CFFile.idl
#ifndef __CFFILE_DEFINED
#define __CFFILE_DEFINED
#include "CFCommonTypes.idl"
#ifdef _PRE_3_0_COMPILER_
#pragma prefix "omg.org"
#endif
module CF {

   exception FileException {
      ErrorNumberType errorNumber;
      string msg;
   };
   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);
   };
};
#endif
```

C.1.2 FileSystem

```
//Source file: CFFileSystem.idl


#ifndef __CFFILESYSTEM_DEFINED
#define __CFFILESYSTEM_DEFINED
#include "CFFile.idl"
#ifdef _PRE_3_0_COMPILER_
#pragma prefix "omg.org"
#endif

module CF {
   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);
```

```
      };
   };
   #endif
```

C.1.3 FileManager

```
   //Source file: CFFileManager.idl

   #ifndef __CFFILEMANAGER_DEFINED
   #define __CFFILEMANAGER_DEFINED

   #include "CFFileSystem.idl"

   #ifdef _PRE_3_0_COMPILER_

   #pragma prefix "omg.org"

   #endif


   module CF {

      interface FileManager : FileSystem {
         struct MountType {
            FileSystem fs;
            string mountPoint;
         };

         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
```

C.2 DF SWRadio Common Radio

C.2.1 Managed Component Statuses Interface

```
   //File: DfSWRadioManagedComponentStatuses.idl

   #ifndef __DFSWRADIOMANAGEDCOMPONENTSTATUSES_DEFINED
   #define __DFSWRADIOMANAGEDCOMPONENTSTATUSES_DEFINED

   #pragma prefix "omg.org"

   module DfSWRadio {

      module CommonRadio {

         interface DfSwrManagedComponentStatuses {
            const unsigned short ALARM_UNDER_REPAIR = 2;
            const unsigned short ALARM_CRITICAL = 4;
            const unsigned short ALARM_MAJOR = 8;
            const unsigned short ALARM_MINOR = 16;
            const unsigned short ALARM_OUTSTANDING = 32;
            const unsigned short PROCEDURAL_INITIALIZATION = 2;
            const unsigned short PROCEDURAL_NOT_INITIALIZED = 4;
            const unsigned short PROCEDURAL_INITIALIZING = 8;
            const unsigned short PROCEDURAL_REPORTING = 16;
            const unsigned short PROCEDURAl_TERMINATING = 32;
            const unsigned short AVAILABILITY_IN_TEST = 2;
            const unsigned short AVAILABILITY_FAILED = 4;
            const unsigned short AVAILABILITY_POWER_OFF = 8;
            const unsigned short AVAILABILITY_OFFLINE = 16;
            const unsigned short AVAILABILITY_OFFDUTY = 32;
            const unsigned short AVAILABILITY_DEPENDENCY = 64;
            const unsigned short AVAILABILITY_DEGRADED = 128;
            const unsigned short AVAILABILITY_NOT_INSTALLED = 256;
            const unsigned short AVAILABILITY_LOG_FULL = 512;
            const unsigned short CONTROL_SUBJECT_TO_TEST = 2;
            const unsigned short CONTROL_PART_OF_SERVICES_LOCKED = 4;
            const unsigned short CONTROL_RESERVED_FOR_TEST = 8;
            const unsigned short CONTROL_SUSPENDED = 16;
            const unsigned short STANDBY_HOT = 2;
            const unsigned short STANDBY_COLD = 4;
            const unsigned short STANDBY_PROVIDING_SERVICE = 8;
         };

      };

   };

   #endif
```

Annex D Data Link Layer Facilities CORBA IDL (non-normative)

Note – Issue 7845 - Section changed to non-normative

D.1 Data Link Layer Interfaces

D.1.1 Data Link Layer Types

```
//File: DfSWRadioDataLinkLayerTypes.idl

#ifndef __DFSWRADIODATALINKLAYERTYPES_DEFINED
#define __DFSWRADIODATALINKLAYERTYPES_DEFINED

#include "DfSWRadioCommonLayerBasicTypes.idl"
#include "CFCommonTypes.idl"
#pragma prefix "omg.org"

module DfSWRadio {

   module DataLinkLayer {

      enum LinkServiceType {
         CONNECTION,
         CONNECTIONLESS,
         ACKCONNECTIONLESS
      };

      struct SAPAddressType {
         unsigned long sap;
         CF::OctetSequence address;
      };
```

```
        struct ConnectionIDType {
           DfSWRadio::CommonLayer::AddressType sourceAddress;
           DfSWRadio::CommonLayer::AddressType destinationAddress;
           long priority;
           SAPAddressType sapAddress;
           LinkServiceType linkService;
        };

     };

  };

  #endif
```

D.1.2 Data Link Layer Ack Connectionless Interfaces

```
  //File: DfSWRadioDataLinkLayerAckConnectionless.idl

  #ifndef __DFSWRADIODATALINKLAYERACKCONNECTIONLESS_DEFINED
  #define __DFSWRADIODATALINKLAYERACKCONNECTIONLESS_DEFINED

  #include "DfSWRadioCommonLayerBasicTypes.idl"
  #include "DfSWRadioFlowControlManagement.idl"
  #include "DfSWRadioPDU.idl"

  #pragma prefix "omg.org"

  module DfSWRadio {

     module DataLinkLayer {

        module LinkAckConnectionless {

           enum PacketIndicatorType {
              PI_ONEWAY,
              PI_TWOWAY
           };

           struct IndicatorHeaderType {
              PacketIndicatorType packetIndicator;
              boolean useAckServiceInMAC;
              DfSWRadio::CommonLayer::AddressType sourceAddress;
              DfSWRadio::CommonLayer::AddressType destinationAddress;
              long priority;
              DfSWRadio::CommonLayer::SduSizeType sduSize;
              long sequenceNumber;
           };
```

```
struct ReplyHeaderType {
   unsigned long correlationID;
   DfSWRadio::CommonLayer::AddressType sourceAddress;
   DfSWRadio::CommonLayer::AddressType destinationAddress;
   long priority;
   DfSWRadio::CommonLayer::SduSizeType sduSize;
   long sequenceNumber;
};

struct RequestHeaderType {
   PacketIndicatorType packetIndicator;
   unsigned long correlationID;
   boolean useAckServiceInMAC;
   DfSWRadio::CommonLayer::AddressType sourceAddress;
   DfSWRadio::CommonLayer::AddressType destinationAddress;
   long priority;
   DfSWRadio::CommonLayer::SduSizeType sduSize;
   long sequenceNumber;
};

interface AckConnectionlessLink {

   void ackReception (
      in octet sequenceNumber
   );

   void nakreception (
      in octet sequenceNumber
   );

};
```

Note – Issue 8296

```
interface AckIndicatorPdu :
    DfSWRadio::CommonLayer::FlowControl::PriorityFlowControl,
    CommonLayer::PDUFacilities::BasePdu {

   oneway void pushPDU (
      in octet priority,
      in IndicatorHeaderType control,
      in CF::OctetSequence sdu
   );

};

interface AckReplyPdu :
    DfSWRadio::CommonLayer::FlowControl::PriorityFlowControl,
    CommonLayer::PDU::BasePdu {
```

```
               oneway void pushPDU (
                  in octet priority,
                  in ReplyHeaderType control,
                  in CF::OctetSequence sdu
               );

            };
```

Note – Issue 8296

```
            interface AckRequestPdu :

                DfSWRadio::CommonLayer::FlowControl::PriorityFlowControl,
                DfSWRadio::CommonLayer::PDUFacilities::BasePdu {

               oneway void pushPDU (
                  in octet priority,
                  in RequestHeaderType control,
                  in CF::OctetSequence sdu
               );

            };

         };

      };

   };

   #endif
```

D.1.3 Data Link Layer Connection Interface

```
   //File: DfSWRadioDataLinkLayerConnection.idl

   #ifndef __DFSWRADIODATALINKLAYERCONNECTION_DEFINED
   #define __DFSWRADIODATALINKLAYERCONNECTION_DEFINED

   #include "DfSWRadioDataLinkLayerTypes.idl"

   #pragma prefix "omg.org"

   module DfSWRadio {

      module DataLinkLayer {

         typedef sequence <ConnectionIDType> ConnectionIdSequence;
```

```
        interface ConnectionLink {
```

Note – Issue 8296

```
            /* BEGIN UPDATE: Change parameters to AddressType, return
   ConnectionIDType
               NOTE: Update UML
            */

            ConnectionIDType establishStream (
               in DfSWRadio::CommonLayer::AddressType sourceAddress,
               in DfSWRadio::CommonLayer::AddressType destinationAddress
               );

            void startStream (
               in ConnectionIDType streamID
               );

            void stopStream (
               in ConnectionIDType streamID
               );

            void releaseStream (
               in ConnectionIDType streamID
               );
```

Note – Issues 7787, 8296

```
   /* BEGIN UPDATE: Change return type to 'ConnectionIDType'
               NOTE: Update UML
               QUESTION: Should parameter be 'Sequence' or 'Type'?
            */

            ConnectionIDType muxStreams (
               in ConnectionIdSequence streamIDs
               );
            /* END UPDATE: Issue 7787 */

   /* BEGIN UPDATE: Change return type to 'ConnectionIDType'
               NOTE: Update UML
               QUESTION: Should parameter be 'Sequence' or 'Type'?
            */
            ConnectionIDType demuxStream (
               in ConnectionIDType streamID
               );
            /* END UPDATE: Issue 7787 */

        };
```

```
      };

   };

   #endif
```

D.1.4 Data Link Layer Connectionless Interfaces

```
   //File: DfSWRadioDataLinkLayerConnectionless.idl

   #ifndef __DFSWRADIODATALINKLAYERCONNECTIONLESS_DEFINED
   #define __DFSWRADIODATALINKLAYERCONNECTIONLESS_DEFINED

   #include "DfSWRadioPDU.idl"
   #include "DfSWRadioFlowControlManagement.idl"
   #include "DfSWRadioCommonLayerBasicTypes.idl"
   #include "CFCommonTypes.idl"

   #pragma prefix "omg.org"

   module DfSWRadio {

      module DataLinkLayer {

         module LinkConnectionless {

            struct IndicatorHeaderType {
               boolean isGroupAddress;
               DfSWRadio::CommonLayer::AddressType sourceAddress;
               DfSWRadio::CommonLayer::AddressType destinationAddress;
               long priority;
               DfSWRadio::CommonLayer::SduSizeType sduSize;
               long sequenceNumber;
            };
```

Note – Issue 8296

```
            typedef DfSWRadio::CommonLayer::PDUFacilities::ControlHeaderType
                RequestHeaderType;

            interface IndicatorPdu :
                DfSWRadio::CommonLayer::FlowControl::PriorityFlowControl,
                DfSWRadio::CommonLayer::PDUFacilities::BasePdu {

               oneway void pushPDU (
                  in octet priority,
                  in IndicatorHeaderType control,
                  in CF::OctetSequence sdu
                  );
            };
```

Note – Issue 8296

```
            interface RequestPdu :
                DfSWRadio::CommonLayer::FlowControl::PriorityFlowControl,
                DfSWRadio::CommonLayer::PDUFacilities::BasePdu {

               oneway void pushPDU (
                  in octet priority,
                  in RequestHeaderType control,
                  in CF::OctetSequence sdu
                  );

            };

         };

      };

   };

   #endif
```

D.1.5 Data Link Layer Local Management

```
   //File: DfSWRadioDataLinkLayerLocalManagement.idl

   #ifndef __DFSWRADIODATALINKLAYERLOCALMANAGEMENT_DEFINED
   #define __DFSWRADIODATALINKLAYERLOCALMANAGEMENT_DEFINED

   #include "DfSWRadioDataLinkLayerTypes.idl"

   #pragma prefix "omg.org"

   module DfSWRadio {

      module DataLinkLayer {

         struct BindRequestType {
            SAPAddressType sapAddress;
            unsigned long maxConnectionInd;
            LinkServiceType linkService;
            boolean isListenStream;
            boolean autoXID;
            boolean autoTest;
         };
```

```
struct BindResponseType {
   SAPAddressType sapAddress;
   unsigned long maxConnectionInd;
   boolean autoXID;
   boolean autoTest;
};

enum PromiscuousModeType {
   PHYSICAL,
   SAP,
   MULTI
};

enum ServiceErrorType {
   ERROR_INVALID_STATE,
   ERROR_UNSUPPORTED,
   ERROR_BAD_ADDRESS,
   ERROR_BAD_CORRELATION,
   ERROR_NOT_ENABLED,
   ERROR_TOO_MANY,
   ERROR_NO_ACCESS,
   ERROR_BOUND,
   ERROR_NO_AUTO,
   ERROR_NO_XIDAUTO,
   ERROR_NO_TESTAUTO,
   ERROR_BAD_DATA,
   ERROR_NO_ADDRESS,
   ERROR_BAD_SAP,
   ERROR_BAD_QOS_PARAMETERS,
   ERROR_UNDELIVERABLE
};

interface LocalLinkManagement {

   exception SystemError {
      unsigned long errNo;
   };

   exception ServiceUsageError {
      ServiceErrorType qualifier;
   };

   exception InvalidPort {};

   attribute DfSWRadio::CommonLayer::SduSizeType sduSize;

   void getInfo (
      in ConnectionIDType connectionID
      )
      raises (InvalidPort,SystemError);
```

```
            BindResponseType bindStream (
               in ConnectionIDType connectionID,
               in BindRequestType bindRequest
               )
               raises (InvalidPort,ServiceUsageError,SystemError);
```

Note – Issue 8296, 7725

```
            BindResponseType unbindStream (
               in ConnectionIDType connectionID
            ) raises (InvalidPort,ServiceUsageError,SystemError);


            BindResponseType bindSubsequentStream(
               in ConnectionIDType connectionID,
               in BindRequestType bindRequest
               )
               raises (InvalidPort,ServiceUsageError,SystemError);
```

Note – Issue 8296

```
            BindResponseType unbindSubsequentStream (
               in ConnectionIDType connectionID,
               )
               raises (InvalidPort,ServiceUsageError,SystemError);

            void enableMulticast (
               in ConnectionIDType connectionID
               );

            void disableMulticast (
               in ConnectionIDType connectionID
               );

            void enablePromiscuousMode (
               in ConnectionIDType connectionID,
               in PromiscuousModeType promiscouosMode
               );

            void disablePromiscuousMode (
               in ConnectionIDType connectionID
               );

         };

      };

   };
```

```
#endif
```

D.2 MAC Interfaces

```
//File: DfSWRadioDataLinkLayerMAC.idl

#ifndef __DFSWRADIODATALINKLAYERMAC_DEFINED
#define __DFSWRADIODATALINKLAYERMAC_DEFINED

#include "CFCommonTypes.idl"
#include "DfSWRadioCommonLayerBasicTypes.idl"

#pragma prefix "omg.org"

module DfSWRadio {

   module DataLinkLayer {

      module MACControl {

         struct MediumAccessControlHeaderType {
            DfSWRadio::CommonLayer::AddressType receiverAddress;
            DfSWRadio::CommonLayer::AddressType transmitterAddress;
            CF::OctetSequence CRC;
            long frameType;
            long frameSubType;
            boolean moreFlag;
            boolean retryFlag;
            CF::OctetSequence powerManagementCommands;
            CF::OctetSequence privacyKey;
         };

         interface MediumAccessControl {

            boolean determineMediumAccessParameters ();

            boolean activateChannel (
                in long presetNum
               );

         };

         interface MacPdu {

            oneway void pushPDU (
               in MediumAccessControlHeaderType control,
               in CF::OctetSequence sdu
               );
         };
      };
   };
};
#endif
```

Note – Issue 7587

D.3 DfSWRadio Data Link Layer Module

```
//File: DfSWRadioDataLinkLayer.idl

#ifndef __DFSWRADIODATALINKLAYER_DEFINED
#define __DFSWRADIODATALINKLAYER_DEFINED

#include "DfSWRadioDataLinkLayerAckConnectionless.idl"
#include "DfSWRadioDataLinkLayerConnection.idl"
#include "DfSWRadioDataLinkLayerConnectionless.idl"
#include "DfSWRadioDataLinkLayerLocalManagement.idl"
#include "DfSWRadioDataLinkLayerMAC.idl"

#endif
```

D.3 DfSWRadio Data Link Layer Module

Annex E Physical Layer CORBA IDL (non-normative)

Note – Issue 7845 - Section changed to non-normative

E.1 Physical Layer Input/Output Interfaces

Note – Issue 7868

```
//Source file: DfSWRadioPhysicalLayer.idl

#ifndef __DFSWRADIOPHYSICALLAYER_DEFINED

#define __DFSWRADIOPHYSICALLAYER_DEFINED

#ifdef _PRE_3_0_COMPILER_

#pragma prefix "omg.org"

#endif

module DfSWRadio {

   module PhysicalLayer {

      interface IOSignals {

         oneway void signalRTS (

            in boolean rts

            );

      };

      module SerialIO {

         interface SerialIOSignals : IOSignals {

         };

         interface SerialIOControl

            void enableRTS_CTS (

               in boolean enable
```

```
                  );



               void setCTS (
                  in boolean cts
                  );
            };
         };
         module AudioIO {
            interface PTTSignals : IOSignals {
            };
         };
      };
   };
   #endif
```

Annex F Physical Layer Properties (non-normative)

Note – Issue 7845 - Section changed to non-normative

Note – Issue 7583 replace section with I/O XML Properties

F.1 I/O XML Properties

F.1.1 Audio XML Properties

```
<?xml version="1.0" encoding="UTF-8"?>

<!--Sample XML file generated by XMLSpy v2005 sp2 U
(http://www.altova.com)-->

<SWRadio:Properties xmlns:SWRadio="http://schema.omg.org/SWRadio"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schema.omg.org/SWRadio

D:\\SWRadio\Properties.xsd">

<SWRadio:ConfigureQuerySimpleProperty>

<description>Width of frequency band.</description>

   <label>Band Width</label>

   <name>bandWidth</name>

   <integerId>30</integerId>

   <type>ulong</type>

   <value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<label>Delat Group Delay</label>

<name>deltaGroupDelay</name>
```

```
<integerId>31</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<label>Gain Controller Dynamic</label>

<name>gainControllerDynamic</name>

<integerId>32</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<label>Gain Controller Step</label>

<name>gainControllerStep</name>

<integerId>33</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>High bound sampling frequency in order to satisfy the Shannon
sampling criterion.</description>

<label>High Bound Frequency</label>

<name>highBoundFrequency</name>

<integerId>34</integerId>

<type>ushort</type>

<value></value>
```

```
</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Defines the high bound rejection limit in low frequencies to
avoid continuous component (pass band).</description>

<label>High Bound Frequency Pass Band</label>

<name>highBoundFrequencyPB</name>

<integerId>35</integerId>

<type>ushort</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>High bound of rejection gain.</description>

<label>High Bound Rejection Gain</label>

<name>highBoundRejectionGain</name>

<integerId>36</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>High bound of rejection slope.</description>

<label>High Bound Rejection Slope</label>

<name>highBoundRejectionSlope</name>

<integerId>4</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>
```

```
<SWRadio:ConfigureQuerySimpleProperty>

<description>High bound of transition band.</description>

<label>High Bound Transition Band</label>

<name>highBoundTransitionBand</name>

<integerId>37</integerId>

<type>ulong</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>capability of the gain.</description>

<label>Level Adjustment Dynamic</label>

<name>levelAdjustmentDynamic</name>

<integerId>38</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>granularity of the gain.</description>

<label>Level Adjustment Step</label>

<name>levelAdjustmentStep</name>

<integerId>39</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>
```

```
<description>Low bound sampling frequency in order to satisfy the Shannon
sampling criterion.</description>

<label>Low Bound Frequency</label>

<name>lowBoundFrequency</name>

<integerId>40</integerId>

<type>ushort</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Defines the low bound rejection limit in low frequencies to
avoid continuous component (pass band).</description>

<label>Low Bound Frequency Pass Band</label>

<name>lowBoundPB</name>

<integerId>41</integerId>

<type>ushort</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Low bound of rejection gain.</description>

<label>Low Bound Rejection Gain</label>

<name>lowBoundRejectionGain </name>

<integerId>42</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Low bound of rejection slope.</description>
```

```
<label>Low Bound Rejection Slope</label>

<name>lowBoundRejectionSlope</name>

<integerId>43</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<label>Low Bound Rejection Slope</label>

<name>lowBoundRejectionSlope</name>

<integerId>44</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Low bound of transition band.</description>

<label>Low Bound Transition Band</label>

<name>lowBoundTransitionBand </name>

<integerId>45</integerId>

<type>ulong</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Maximum allowed latency.</description>

<label>Max Latency</label>

<name>maxLatency</name>

<integerId>46</integerId>
```

```
<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Defines maximum bound of nominal level.</description>

<label>Max Nominal Level</label>

<name>maxNominalLevel</name>

<integerId>47</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Defines minimal bound of nominal level.</description>

<label>Min Nominal Level</label>

<name>minNominalLevel</name>

<integerId>48</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Defines the instruction for output analog signal nominal
level.</description>

<label>Nominal Level</label>

<name>NominalLevel</name>

<integerId>49</integerId>

<type>long</type>
```

```
<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Defines the level of noise (assumed white) present in audio
frequency samples as inputting inside (resp. being output from) Audio.
Expressed in dBFS/Hz. Possible spurious are integrated in this
value.</description>

<label>NoiseFloor</label>

<name>noiseFloor</name>

<integerId>50</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Defines the level of quantification noise present in digital
samples as inputting inside (resp. being output from) ADC. Expressed in
dBFS.</description>

<label>Quantification Noise Floor</label>

<name>QuantificationNoiseFloor</name>

<integerId>51</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<label>Ripple</label>

<name>ripple</name>

<integerId>52</integerId>

<type>long</type>
```

```
<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<label>Quantification Noise Floor</label>

<name>QuantificationNoiseFloor</name>

<integerId>53</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Defines the sampling frequency of the audio frequency
signal.</description>

<label>Sampling Frequency</label>

<name>SamplingFrequency</name>

<integerId>54</integerId>

<type>ushort</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Avoid gain saturation (in dBfs).</description>

<label>Saturation Merge</label>

<name>saturationMerge</name>

<integerId>55</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>
```

```
<SWRadio:ConfigureQuerySimpleProperty>

<description>Expresses the expected variations of signal magnitude around
the nominal level.</description>

<label>Signal Dynamicr</label>

<name>SignalDynamic</name>

<integerId>56</integerId>

<type>long</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:CharacteristicProperty>

   <capabilityModel>eq</capabilityModel>

   <locallyManaged>false</locallyManaged>

   <description>Defines the type of device.</description>

   <label>Device Type</label>

   <name>DeviceType</name>

   <type>string</type>

   <value>AudioDevice</value>

</SWRadio:CharacteristicProperty>

<SWRadio:CharacteristicProperty>

   <capabilityModel>"eq"</capabilityModel>

   <locallyManaged>false</locallyManaged>

   <description>Defines if the device is on red (unencrypted boundary) or
black side (encrypted boundary) of an encryption boundary ( Black/Encrypted
= 0, Red/Unencrypted = 1).</description>

   <name>location</name>

   <type>ushort</type>

   <enumerations>
```

```
            <enumerationLiteral>

               <label>Black/Encrypted</label>

               <value>0</value>

            </enumerationLiteral><enumerationLiteral>

               <label>Red/Unencrypted</label>

               <value>1</value>

            </enumerationLiteral>

         </enumerations>

      </SWRadio:CharacteristicProperty>

      <SWRadio:CapacityProperty>

         <capabilityModel>"counter"</capabilityModel>

         <locallyManaged>true</locallyManaged>

         <description>Specifies the number of audio ports for a
      device.</description>

         <label>Ports Capacity</label>

         <name>portsCapacity</name>

         <type>ushort</type>

         <value>1</value>

      </SWRadio:CapacityProperty>

      </SWRadio:Properties>
```

F.1.2 Serial XML Properties

```
      <?xml version="1.0" encoding="UTF-8"?>

      <!--Sample XML file generated by XMLSpy v2005 sp2 U
      (http://www.altova.com)-->

      <SWRadio:Properties xmlns:SWRadio="http://schema.omg.org/SWRadio"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://schema.omg.org/SWRadio

      D:\\SWRadio\Properties.xsd">
```

```
<SWRadio:ConfigureQuerySimpleProperty>

<description>(Asynchronous protocol only) Number of bits in character (5,
6, 7, or 8).</description>

   <label>Character Width</label>

   <name>characterWidth</name>

   <integerId>13</integerId>

   <type>ushort</type>

   <value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty isReadOnly="true">

<description>Indicates the CTS status.</description>

<label>CTS Status</label>

<name>ctsStatus</name>

<integerId>1</integerId>

<type>boolean</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Controls whether flow Control signals should be generated.
True means Xon and False means Xoff.</description>

<label>Flow Control Xon Xoff</label>

<name>flowControlXonXoff</name>

<integerId>2</integerId>

<type>boolean</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>
```

```
<description>To enable/disable use of RTS/CTS hardware signals used for
flow control.</description>

<label>Hardware Control</label>

<name>hardwareFlowControl</name>

<integerId>11</integerId>

<type>boolean</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty isReadOnly="true">

<description>Maximum size of payload for the pushPDU() method in
ConcreteDataPDU interface.</description>

<label>Max Payload Size</label>

<name>maxPayloadSize</name>

<integerId>4</integerId>

<type>ushort</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty isReadOnly="true">

<description>Minimum size of payload for the pushPDU() method in
ConcreteDataPDU interface.</description>

<label>Min Payload Size</label>

<name>minPayloadSize</name>

<integerId>3</integerId>

<type>ushort</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>
```

```
<description>Asynchronous protocol only) Number of start bits (0 or
1).</description>

<label>Number of Start Bits</label>

<name>numberStartBits</name>

<integerId>15</integerId>

<type>ushort</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>(Asynchronous protocol only) Number of stop bits (1 or
2).</description>

<label>Number of Stop Bits</label>

<name>numberStopBits</name>

<integerId>16</integerId>

<type>ushort</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Optional, used only for receive flow control.  IDLE time that
Serial I/O waits before data received through the serial port must be
forwarded to the component connected to the DataOutPort. IDLE time in number
of not received characters unit.</description>

<label>On Threshold</label>

<name>onThreshold</name>

<integerId>14</integerId>

<type>ulong</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>
```

```
<SWRadio:ConfigureQuerySimpleProperty>

<description>ype of parity checking (Even = 0, odd = 1).</description>

<label>Parity Checking</label>

<name>parityChecking</name>

<integerId>12</integerId>

<type>ushort</type><enumerations>

   <enumerationLiteral>

      <label>Even</label>

      <value>0</value>

   </enumerationLiteral><enumerationLiteral>

      <label>Odd</label>

      <value>1</value>

   </enumerationLiteral>

</enumerations>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Sets asynchronous serial data protocol (Asynchronous=0 and
Synchronous = 1).</description>

<label>Protocol</label>

<name>protocol</name>

<integerId>7</integerId>

<type>ushort</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>
```

```
<description>Baud rate for Receive data</description>

<label>Receive Baud Rate</label>

<name>receiveBaudRate</name>

<integerId>8</integerId>

<type>ulong</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Size of packets to buffer before any data is written to device
caller.</description>

<label>Receive Buffer Size</label>

<name>receiveBufferSize</name>

<integerId>17</integerId>

<type>ulong</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Clock source for Receive data: internal Receive baud rate
generator, external clock line, and Transmit clock source, respectively.
Predefined values for coding scheme are 0=Internal Receive  and 1=External
clock.</description>

<label>Receive Clock Source</label>

<name>receiveClockSource</name>

<integerId>10</integerId>

<type>ushort</type>

<enumerations>

   <enumerationLiteral>

      <label>Internal Receive</label>
```

```
        <value>0</value>

    </enumerationLiteral><enumerationLiteral>

        <label>External clock</label>

        <value>1</value>

    </enumerationLiteral>

</enumerations>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Sets the encoding method for Transmission of serial data to
NRZ, NRZI Mark, FM0, Manchester, and Differential Manchester, respectively.
Predefined values for coding scheme are 0=NRZ, 1=NRZI Mark, 2=FM0,
3=Manchester, and 4=Differential Manchester, respectively.</description>

<label>Receive Encoding</label>

<name>receiveEncoding</name>

<integerId>9</integerId>

<type>ushort</type>

<enumerations>

    <enumerationLiteral>

        <label>"NRZ"</label>

        <value>0</value>

    </enumerationLiteral>

    <enumerationLiteral>

        <label>"NRZI Mark"</label>

        <value>1</value>

    </enumerationLiteral>

    <enumerationLiteral>
```

```
            <label>"FM0"</label>
            <value>2</value>
         </enumerationLiteral>
         <enumerationLiteral>
            <label>"Manchester"</label>
            <value>3</value>
         </enumerationLiteral>
         <enumerationLiteral>
            <label>"Differential Manchester"</label>
            <value>4</value>
         </enumerationLiteral>
      </enumerations>
      <value></value>
      </SWRadio:ConfigureQuerySimpleProperty>
      <SWRadio:ConfigureQuerySimpleProperty isReadOnly="true">
      <description>Retrieves the RTS/CTS mode.</description>
      <label>RTS CTS Mode</label>
      <name>rts_cts_mode</name>
      <integerId>5</integerId>
      <type>boolean</type>
      <value></value>
      </SWRadio:ConfigureQuerySimpleProperty>
      <SWRadio:ConfigureQuerySimpleProperty>
      <description>Baud rate for transmit data.</description>
      <label>Transmit Baud Rate</label>
      <name>transmitBaudRate</name>
```

```
<integerId>19</integerId>

<type>ulong</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Clock source for Transmission of data: internal Transmit baud
rate generator, external clock line, Receive clock source, and clock
recovery, respectively. Predefined values for coding scheme are 0=Internal
Receive  and 1=External clock.</description>

<label>Transmit Clock Source</label>

<name>transmitClockSource</name>

<integerId>20</integerId>

<type>ushort</type>

<enumerations>

   <enumerationLiteral>

      <label>Internal Receive</label>

      <value>0</value>

   </enumerationLiteral><enumerationLiteral>

      <label>External clock</label>

      <value>1</value>

   </enumerationLiteral>

</enumerations>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Sets the encoding method for Transmission of serial data to
NRZ, NRZI Mark, FM0, Manchester, and Differential Manchester, respectively.
Predefined values for coding scheme are 0=NRZ, 1=NRZI Mark, 2=FM0,
3=Manchester, and 4=Differential Manchester, respectively.</description>
```

```
<label>Transmit Encoding</label>

<name>transmitEncoding</name>

<integerId>18</integerId>

<type>ushort</type>

<enumerations>

   <enumerationLiteral>

      <label>"NRZ"</label>

      <value>0</value>

   </enumerationLiteral>

   <enumerationLiteral>

      <label>"NRZI Mark"</label>

      <value>1</value>

   </enumerationLiteral>

   <enumerationLiteral>

      <label>"FM0"</label>

      <value>2</value>

   </enumerationLiteral>

   <enumerationLiteral>

      <label>"Manchester"</label>

      <value>3</value>

   </enumerationLiteral>

   <enumerationLiteral>

      <label>"Differential Manchester"</label>

      <value>4</value>

   </enumerationLiteral>

</enumerations>
```

```
<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:ConfigureQuerySimpleProperty>

<description>Set if on-going transmission..</description>

<label>Transmit Active</label>

<name>txActive</name>

<integerId>21</integerId>

<type>boolean</type>

<value></value>

</SWRadio:ConfigureQuerySimpleProperty>

<SWRadio:CharacteristicProperty>

   <capabilityModel>"eq"</capabilityModel>

   <locallyManaged>false</locallyManaged>

   <label>Device Type</label>

   <name>DeviceType</name>

   <type>string</type>

   <value>SerialDevice</value>

</SWRadio:CharacteristicProperty>

<SWRadio:CharacteristicProperty>

   <capabilityModel>"eq"</capabilityModel>

   <locallyManaged>false</locallyManaged>

   <name>location</name>

   <type>ushort</type>

   <enumerations>

      <enumerationLiteral>

         <label>Black/Encrypted</label>
```

```
            <value>0</value>
         </enumerationLiteral><enumerationLiteral>
            <label>Red/Unencrypted</label>
            <value>1</value>
         </enumerationLiteral>
      </enumerations>
   </SWRadio:CharacteristicProperty>
   <SWRadio:CapacityProperty>
      <capabilityModel>"counter"</capabilityModel>
      <locallyManaged>true</locallyManaged>
      <label>Ports Capacity</label>
      <name>portsCapacity</name>
      <type>ushort</type>
      <value>1</value>
   </SWRadio:CapacityProperty>
   </SWRadio:Properties>
```

Annex G Radio Control Facilities CORBA IDL (non-normative)

Note – Issue 7845 - Section changed to non-normative

G.1 Radio Set Management Interfaces

```
//File: DfSWRadioControlRadioSetManagement.idl

#ifndef __DFSWRADIOCONTROLRADIOSETMANAGEMENT_DEFINED
#define __DFSWRADIOCONTROLRADIOSETMANAGEMENT_DEFINED

#include "CFCommonTypes.idl"
#include "CFApplications.idl"
#include "CFResources.idl"
#include "CFDevices.idl"
#include "CFStateManagement.idl"
#include "CFDomainManager.idl"

#pragma prefix "omg.org"

module DfSWRadio {

   module RadioControl  {

      module RadioSetManagement {

         interface CommChannel;

         interface ZeroizeControl {
            void zeroize ();
         };

         interface ChannelFactory {
            CommChannel createChannel (
               in CF::Properties channelProperties
            );
         };

         interface WaveformCommChannel : CF::Application {
            attribute CommChannel instantiatedCommChannel;
         };
```

```
            interface WaveformInstantiation {
               WaveformCommChannel instantiateWaveform (
                  in string waveformName,
                  in string instanceWFName,
                  in CF::Properties wfConfigProperties,
                  in CF::Properties channelConfigProperties
               );
            };

            interface CommChannel : CF::PropertySet, CF::TestableObject,
               CF::ComponentIdentifier, WaveformInstantiation {

               readonly attribute WaveformCommChannel instantiatedWF;
               readonly attribute CF::DeviceSequence channelDevices;
               readonly attribute CF::Properties keyProperties;

               void releaseChannel ();

            };

            interface ManagedCommChannel : CF::StateManagement, CommChannel{};

            interface SecureCommChannel : CommChannel, ZeroizeControl {};

            interface ManagedSercureCommChannel : ManagedCommChannel,
                SecureCommChannel, ZeroizeControl {};

            interface RadioManager: CF::DomainManager, WaveformInstantiation {

               typedef sequence <CommChannel> CommChannelSequence;
               attribute CommChannelSequence commChannels;
               attribute CF::StringSequence availableWaveforms;

            };

            interface ManagedRadioManager:CF::StateManagement, RadioManager{};

            interface SecureRadioManager : RadioManager, ZeroizeControl {};

            interface ManagedSecureRadioManager : SecureRadioManager,
                ManagedRadioManager {

            };

         };

      };

   };

   #endif
```

G.1 Radio Set Management Interfaces

Annex H Operating System Profiles (non-normative)

Note – Issue 7845 - annex non-normative

H.1 SCA Application Environment Profile

H.1.1 Scope

Note – Issue 8934

This appendix defines the application environment profiles for the SWRadio, based on Standardized Application Environment Profile - POSIX® Realtime Application Support (AEP), IEEE Std 1003.13-1998.

The appendix includes two specific profiles, which are characterized as follows:

1. The application environment profile (AEP), which is the SWRadio wide profile. The AEP is the preferred profile for the SWRadio and its utilization is encouraged for all processing environments.
2. The lightweight application environment profile (LwAEP). LwAEP is more constrained than the AEP and is targeted towards environments with limited computing support. Examples of embedded processors include Digital Signal Processors (DSPs), processor cores within Field Programmable Gate Arrays (FPGA's) and micro-controllers. Mandatory application of LwAEP shall apply only to DSPs. Use of LwAEP in an FPGA-based processor core is encouraged but not required."

H.1.2 Standards

Note – Issue 8934

The following standards are required in whole or in part by the SWRadio AEPs.

Table H-19 – Required Standards

Standard	AEP	LwAEP
C Standard (ISO/IEC 9899:1990	PRT	PRT
POSIX.1 (ISO/IEC 9945 -1):1996	PRT	PRT
POSIX.1b (ISO/IEC 9945 -1):1996	PRT	PRT
POSIX.1c (ISO/IEC 9945 -1):1996	PRT	PRT
POSIX.5b (IEEE 1003.5 - 1992)	OPT	OPT

Note – Issue 8836 - Table above, changed references POSIX 1.x references from 1997 to 1996

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.

H.1.3 Constraints

Note – Issue 8838 - Shall ==> Must

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 must 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.

Note – Issue 8934

Key considerations in selection of functions for the embedded processor are as follows:

- Of late, DSP development environments include operating systems that offer a rich and scaleable feature set - pre-emptive multitasking, installable interrupt handlers and inter-process communications.
- Current DSP technology does not employ Memory Management Units (MMU's).
- Different DSP environments sometimes offer extensions or services that target specific market segments - optimizations for video processing, power savings features and kernel support for real-time debugging.
- Current embedded state-of-the-art does not exploit loadable modules. Entire FPGA and DSP images containing infrastructure and application software are loaded simultaneously as part of waveform instantiation.

Ultimately the presence of a full-featured RTOS in the embedded processor is a relatively new practice and yet one that is recognized as offering life cycle software cost benefit. The state-of-the-art will continue to advance and this Appendix shall not disallow the migration of new design paradigms as they become matured and practiced.

H.1.3.1 POSIX.1.

Note – Issue 7586, Issue 8934

Table H-20 – POSIX.1 Option Requirements

Option	AEP	LwAEP
{NGROUPS_MAX}	-	-
{_POSIX_CHOWN_RESTRICTED}	NRQ	NRQ
{_POSIX_JOB_CONTROL}	NRQ	NRQ
{_POSIX_NO_TRUNC}	PRI	NRQ
{_POSIX_SAVED_IDS}	NRQ	NRQ
{_POSIX_VDISABLE}	NRQ	NRQ

Note – Issue 7586, Issue 8934

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.

Embedded processor C/C++ run-time libraries typically do not support stdio.h or iostream.h

H.1.3.1.1 Single Process Function Behavior

The functions in Table H-21 shall behave as described in the referenced clauses.

Table H-21 – POSIX_SINGLE_PROCESS Functions

Function	Reference in POSIX.1	AEP	LwAEP
sysconf ( )	4.8.1	NRQ	NRQ
uname( )	4.4.1	NRQ	NRQ
time()	4.5.1	MAN	NRQ

Note – Issue 8934 table above

NRQ - Not required for this profile.

MAN - Mandatory for this profile.

H.1.3.1.2 Multi-Process Function Behavior

The functions listed in Table H-22 shall behave as described in the referenced clauses.

Table H-22 – POSIX_MULTI_PROCESS Functions

Function	Reference in POSIX.1	AEP	LwAEP
execl ( )	3.1.2	NRQ	NRQ
execv ( )	3.1.2	NRQ	NRQ
execle ( )	3.1.2	NRQ	NRQ
execve ( )	3.1.2	NRQ	NRQ
execlp ( )	3.1.2	NRQ	NRQ
execvp ( )	3.1.2	NRQ	NRQ
_exit ( )	3.2.2	NRQ	NRQ
fork( )	3.1.1	NRQ	NRQ
getenv ( )	4.6.1	NRQ	NRQ
getpid ( )	4.1.1	NRQ	NRQ
getppid ( )	4.1.1	NRQ	NRQ
pthread_atfork()	3.1.3	NRQ	NRQ
sleep ( )	3.4.3	NRQ	NRQ
times ( )	4.5.2	NRQ	NRQ
wait( )	3.2.1	NRQ	NRQ
waitpid ( )	3.2.1	NRQ	NRQ
assert ( )	8.1, 8.2, 8.3	NRQ	NRQ
exit ( )	8.1, 8.2, 8.3	NRQ	NRQ
setlocale ( )	8.1, 8.2, 8.3	MAN	NRQ

Note – Issue 8830 - pthread_atfork() added to the table above

MAN - Mandatory for this profile.

NRQ - Not Required for this profile

Note – Issue 8934 modifed table above and added text below

.setlocale() is a part of the AEP but not LwAEP because embedded processors do not support streaming character oriented output.

H.1.3.1.3 Job Control Function Behavior

The functions listed in Table H-23 shall behave as described in the referenced clauses.

Table H-23 – POSIX_JOB_CONTROL Functions

Function*	Reference in POSIX.1	AEP	LwAEP
setpgid()	4.3.3	NRQ	NRQ
tcgetpgrp()	7.2.3	NRQ	NRQ
tcsetpgrp()	7.2.4	NRQ	NRQ
*	7.1.1.4	NRQ	NRQ

Note – Issue 8934 modified table above

NOTE:

NRQ - Not required for this profile.

- Further functionality is also defined here.

H.1.3.1.4 Signals Function Behavior

Note – Issue 8934 added new 1st paragraph and modified table below

Operating systems on embedded processors typically support neither signaling nor exception handling. POSIX does not define behaviors associated with divide by zero or overflow / underflow. Signaling methods introduced as part of POSIX.1c are more consistent with the multi-threaded, single process model of the DSP environment

The functions listed in Table H-24 shall behave as described in the referenced clauses, except for the following constraints:

(1) An application strictly conforming to the 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 the AEP shall not call the kill() function with a negative argument because these profiles do not require process group functionality.

Table H-24 – POSIX_SIGNALS Functions

Function	Reference in POSIX.1	AEP	LwAEP
alarm()*	3.4.1	NRQ	NRQ
kill()	3.3.2	MAN	NRQ
pause()	3.4.2	MAN	NRQ
sigaction()	3.3.4	MAN	NRQ
sigaddset()	3.3.3	MAN	NRQ
sigdelset()	3.3.3	MAN	NRQ
sigemptyset()	3.3.3	MAN	NRQ

Table H-24 – POSIX_SIGNALS Functions

Function	Reference in POSIX.1	AEP	LwAEP
sigfillset()	3.2.3	MAN	NRQ
sigismember()	3.3.3	MAN	NRQ
sigpending()	3.3.6	MAN	NRQ
sigprocmask()	3.3.5	MAN	NRQ
sigsupend()	3.3.7	MAN	NRQ
abort()	8.1,8.2,8.3	MAN	MAN
siglongjmp()	8.1,8.2,8.3	NRQ	NRQ
sigsetjmp()	8.1,8.2,8.3	NRQ	NRQ

NOTE:

MAN - Mandatory for this profile.

NRQ - Not Required for this profile.

*Functionality provided through the POSIX timers.

abort() is used to support assert() which is widely supported.

.

H.1.3.1.5 User Group Function Behavior

The functions listed in Table H-25 shall behave as described in the referenced clauses.

Table H-25 – POSIX_USER_GROUPS Functions

Function	Reference in POSIX.1	AEP	LwAEP
getegid()	4.2.1	NRQ	NRQ
geteuid()	4.2.1	NRQ	NRQ
getgid()	4.2.1	NRQ	NRQ
getgroups()	4.2.3	NRQ	NRQ
getlogin()	4.2.4	NRQ	NRQ
getpgrp()	4.3.1	NRQ	NRQ
getuid()	4.2.1	NRQ	NRQ
setuid()	4.2.2	NRQ	NRQ
setsid()	4.3.2	NRQ	NRQ
setgid()	4.2.2	NRQ	NRQ

Note – Issue 8934 modified table above

NOTE:

NRQ - Not required for this profile.

H.1.3.1.6 File System Function Behavior

The functions listed in Table H-26 shall behave as described in the referenced clauses.

Table H-26 – POSIX_FILE_SYSTEM Functions

Function	Reference in POSIX.1	AEP	LwAEP
access()	5.6.3	MAN	NRQ
chdir()	5.2.1	MAN	NRQ
closedir()	5.1.2	MAN	NRQ
creat()	5.3.2	MAN	NRQ
fpathconf()	5.7.1	MAN	NRQ
fstat()	5.6.2	MAN	NRQ
getcwd()	5.2.2	MAN	NRQ
link()	5.3.4	MAN	NRQ
mkdir()	5.4.1	MAN	NRQ
opendir()	5.1.2	MAN	NRQ
pathconf()	5.7.1	MAN	NRQ
readdir()	5.1.2	MAN	NRQ
rename()	5.5.3	MAN	NRQ
rewinddir()	5.1.2	MAN	NRQ
rmdir()	5.5.2	MAN	NRQ
stat()	5.6.2	MAN	NRQ
unlink()	5.5.1	MAN	NRQ
utime()	5.6.6	MAN	NRQ
remove()	8.1, 8.2, 8.3	MAN	NRQ
rename()	8.1, 8.2, 8.3	MAN	NRQ
tmpfile()	8.1, 8.2, 8.3	MAN	NRQ
tmpnam()	8.1, 8.2, 8.3	MAN	NRQ

Note – Issue 8934 modified table aboveand footnote below

NOTE:

MAN - Mandatory for this profile.

POSIX file system not generally supported in embedded operating systems.

H.1.3.1.7 File Attributes Function Behavior

The functions listed in Table H-27 shall behave as described in the referenced clauses, except for the following constraint:

Note – Issue 8833 - replaced SS--IIRRWWXXUU with S_IRWXU

(1) An application strictly conforming to the AEP shall be guaranteed that the file mode creation mask for any object created by any process is S_IRWXU; that is, the object shall be fully accessible to the creator.

Table H-27 – POSIX_FILE_ATTRIBUTES Functions

Function	Reference in POSIX.1	AEP	LwAEP
chmod()	5.6.4	NRQ	NRQ
chown()	5.6.5	NRQ	NRQ
umask()	5.3.3	NRQ	NRQ

Note – Issue 8934 modified table above

NOTE:

NRQ - Not required for this profile.

POSIX file system not generally supported in embedded operating systems.

H.1.3.1.8 File and Directory Management Function Behavior

The functions listed in Table H-28 shall behave as described in the referenced clauses.

Table H-28 – POSIX_FD_MGMT Functions

Function	Reference in POSIX.1	AEP	LwAEP
dup()	6.2.1	NRQ	NRQ
dup2()	6.2.1	NRQ	NRQ
fcntl()	6.5.2	NRQ	NRQ
lseek()	6.5.3	MAN	NRQ
fseek()	8.1, 8.2, 8.3	MAN	NRQ
ftell()	8.1, 8.2, 8.3	MAN	NRQ
rewind()	8.1, 8.2, 8.3	MAN	NRQ

Note – Issue 8934 modified table below and adde footnote below

NOTE:

NRQ - Not required for this profile.

MAN - Mandatory for this profile.

POSIX file system not generally supported in embedded operating systems.

H.1.3.1.9 Device I/O Function Behavior

The functions listed in Table H-29 shall behave as described in the referenced clauses.

Table H-29 – POSIX_DEVICE_IO Functions

Function	Reference in POSIX.1	AEP	LwAEP
close()	6.3.1	MAN	NRQ
open()	5.3.1	MAN	NRQ
read()	6.4.1	MAN	NRQ
write()	6.4.2	MAN	NRQ
clearerr()	8.1, 8.2, 8.3	MAN	NRQ
fclose()	8.1, 8.2, 8.3	MAN	NRQ
fdopen()	8.1, 8.2, 8.3	MAN	NRQ
feof()	8.1, 8.2, 8.3	MAN	NRQ
ferror()	8.1, 8.2, 8.3	MAN	NRQ
fflush()	8.1, 8.2, 8.3	MAN	NRQ
fgetc()	8.1, 8.2, 8.3	MAN	NRQ
fileno()	8.1, 8.2, 8.3	MAN	NRQ
fgets()	8.1, 8.2, 8.3	MAN	NRQ
fopen()	8.1, 8.2, 8.3	MAN	NRQ
fprintf()	8.1, 8.2, 8.3	MAN	NRQ
fputc()	8.1, 8.2, 8.3	MAN	NRQ
fputs()	8.1, 8.2, 8.3	MAN	NRQ
fread()	8.1, 8.2, 8.3	MAN	NRQ
freopen()	8.1, 8.2, 8.3	MAN	NRQ
fscanf()	8.1, 8.2, 8.3	MAN	NRQ
fwrite()	8.1, 8.2, 8.3	MAN	NRQ
getc()	8.1, 8.2, 8.3	MAN	NRQ
getchar()	8.1, 8.2, 8.3	MAN	NRQ
gets()	8.1, 8.2, 8.3	MAN	NRQ
perror()	8.1, 8.2, 8.3	MAN	NRQ
printf()	8.1, 8.2, 8.3	MAN	NRQ
putc()	8.1, 8.2, 8.3	MAN	NRQ

Table H-29 – POSIX_DEVICE_IO Functions

Function	Reference in POSIX.1	AEP	LwAEP
putchar()	8.1, 8.2, 8.3	MAN	NRQ
puts()	8.1, 8.2, 8.3	MAN	NRQ
scanf()	8.1, 8.2, 8.3	MAN	NRQ
setbuf()	8.1, 8.2, 8.3	MAN	NRQ
sprintf()	8.1, 8.2, 8.3	MAN	NRQ
sscanf()	8.1, 8.2, 8.3	MAN	NRQ
ungetc()	8.1, 8.2, 8.3	MAN	NRQ

Note – Issue 8934 modified table above and added footnote below

NOTE:

MAN - Mandatory for this profile.

POSIX streams not generally supported in embedded operating systems.

H.1.3.1.10 Device-Specific Function Behavior

The functions listed in Table H-30 shall behave as described in the referenced clauses.

Table H-30 – POSIX_DEVICE_SPECIFIC Functions

Function	Reference in POSIX.1	AEP	LwAEP
cfgetispeed()	7.1.3	NRQ	NRQ
cfgetospeed()	7.1.3	NRQ	NRQ
cfsetispeed()	7.1.3	NRQ	NRQ
cfsetospeed()	7.1.3	NRQ	NRQ
ctermid()	4.7.1	NRQ	NRQ
isatty()	4.7.2	NRQ	NRQ
tcdrain()	7.2.2	NRQ	NRQ
tcflush()	7.2.2	NRQ	NRQ
tcflow()	7.2.2	NRQ	NRQ
tcgetattr()	7.2.1	NRQ	NRQ
tcsendbreak()	7.2.2	NRQ	NRQ
tcsetattr()	7.2.1	NRQ	NRQ
ttyname()	4.7.2	NRQ	NRQ

Note – Issue 8934 modified table above

NOTE:

NRQ - Not required for this profile

H.1.3.1.11System Database Function Behavior

The functions listed in Table H-31 shall behave as described in the referenced clauses.

Table H-31 – POSIX_SYSTEM_DATABASE Functions

Function	Reference in POSIX.1	AEP	LwAEP
getgrgid()	9.2.1	NRQ	NRQ
getgrnam()	9.2.1	NRQ	NRQ
getpwnam()	9.2.2	NRQ	NRQ
getpwuid()	9.2.2	NRQ	NRQ

Note – Issue 8934 modified table above

NOTE:

NRQ - Not required for this profile.

H.1.3.1.12Pipe Function Behavior

The function listed in Table H-32 shall behave as described in the referenced clause.

Table H-32 – POSIX_PIPE_Function

Function	Reference in POSIX.1	AEP	LwAEP
pipe()	6.1.1	NRQ	NRQ

Note – Issue 8934 modified table above

NOTE:

NRQ - Not required for this profile.

H.1.3.1.13FIFO Function Behavior

The functions listed in Table H-33 shall behave as described in the referenced clauses.

Table H-33 – POSIX_FIFO Function

Function	Reference in POSIX.1	AEP	LwAEP
mkfifo()	5.4.2	NRQ	NRQ

Note – Issue 8934 modified table above

NOTE:

NRQ - Not required for this profile.

H.1.3.1.14C Language-Specific Services Behavior

The functions listed in Table H-34, Table H-35, Table H-36 Table H-37, Table H-38, and Table H-39 shall behave as described in the referenced clauses.

Note – Issue 8934 added text below and mofiied tables below

LwAEP requires only a small subset of C Language specific functionality. There are many reasons for this consideration, the most of which is recognition of the fact that many DSPs are fixed point and support for a POSIX floating point math library is burdensome and unnecessary

Table H-34 – POSIX_C_LANG_SUPPORT Character Handling Functions

Function	Reference in the C Standard	AEP	LwAEP
isalnum()	4.3.1.1	MAN	NRQ
isalpha()	4.3.1.2	MAN	MAN
iscntrl()	4.3.1.3	MAN	NRQ
isdigit()	4.3.1.4	MAN	MAN
isgraph()	4.3.1.5	MAN	NRQ
islower()	4.3.1.6	MAN	NRQ
isprint()	4.3.1.7	MAN	MAN
ispunct()	4.3.1.8	MAN	NRQ
isspace()	4.3.1.9	MAN	NRQ
isupper()	4.3.1.10	MAN	NRQ
isxdigit()	4.3.1.11	MAN	MAN
tolower()	4.3.2.1	MAN	MAN
toupper()	4.3.2.2	MAN	MAN

NOTE:

MAN - Mandatory for this profile.

NRQ - Not required for this profile.

Table H-35 – POSIX_C_LANG_SUPPORT Mathematical Functions

Function	Reference in the C Standard	AEP	LwAEP
acos()	4.5.2.1	MAN	NRQ
asin()	4.5.2.2	MAN	NRQ
atan()	4.5.2.3	MAN	NRQ
atan2()	4.5.2.4	MAN	NRQ
ceil()	4.5.6.1	MAN	NRQ
cos()	4.5.2.5	MAN	NRQ
cosh()	4.5.3.1	MAN	NRQ
exp()	4.5.4.1	MAN	NRQ
fabs()	4.5.6.2	MAN	NRQ
floor()	4.5.6.3	MAN	NRQ
fmod()	4.5.6.4	MAN	NRQ
frexp()	4.5.4.2	MAN	NRQ
ldexp()	4.5.4.3	MAN	NRQ
log()	4.5.4.4	MAN	NRQ
log10()	4.5.4.5	MAN	NRQ
modf()	4.5.4.6	MAN	NRQ
pow()	4.5.5.1	MAN	NRQ
sin()	4.5.2.6	MAN	NRQ
sinh()	4.5.3.2	MAN	NRQ
sqrt()	4.5.5.2	MAN	NRQ
tan()	4.5.2.7	MAN	NRQ
tanh()	4.5.3.3	MAN	NRQ

NOTE:

NRQ - Not required for this profile.

MAN - Mandatory for this profile

Table H-36 – POSIX_C_LANG_SUPPORT Non-Local Jump Functions

Function	Reference in the C Standard	AEP	LwAEP
longjmp()	4.6.2.1	MAN	NRQ
setjmp()	4.6.1.1	MAN	NRQ

NOTE:

H.1.3 Constraints

NRQ - Not required for this profile.

MAN - Mandatory for this profile.

A form of context switch used to support a non-local exit.

Table H-37 – POSIX_C_LANG_SUPPORT General Functions

Function	Reference in the C Standard	AEP	LwAEP
abs()	4.10.6.1	MAN	MAN
atof()	4.10.1.1	MAN	NRQ
atoi()	4.10.1.2	MAN	MAN
atol()	4.10.1.3	MAN	MAN
bsearch()	4.10.5.1	MAN	NRQ
calloc()	4.10.3.1	MAN	MAN
free()	4.10.3.2	MAN	MAN
malloc()	4.10.3.3	MAN	MAN
qsort()	4.10.5.2	MAN	NRQ
rand()	4.10.2.1	MAN	MAN
realloc()	4.10.3.4	MAN	MAN
srand()	4.10.2.2	MAN	MAN

NOTE:

MAN - Mandatory for this profile.

NRQ - Not required for this profile

Support for dynamic memory allocation is essential to re-entrant object-oriented design

Table H-38 – POSIX_C_LANG_SUPPORT String Handling Functions

Function	Reference in the C Standard	AEP	LwAEP
strcat()	4.11.3.1	MAN	NRQ
strchr()	4.11.5.2	MAN	NRQ
strcmp()	4.11.4.2	MAN	NRQ
strcpy()	4.11.2.3	MAN	NRQ
strcspn()	4.11.5.3	MAN	NRQ
strlen()	4.11.6.3	MAN	NRQ
strncpy()	4.11.2.4	MAN	NRQ

Table H-38 – POSIX_C_LANG_SUPPORT String Handling Functions

Function	Reference in the C Standard	AEP	LwAEP
strncat()	4.11.3.2	MAN	NRQ
strncmp()	4.11.4.4	MAN	MAN
strpbkr()	4.11.5.4	MAN	NRQ
strrchr()	4.11.5.5	MAN	NRQ
strspn()	4.11.5.6	MAN	NRQ
strstr()	4.11.5.7	MAN	NRQ
strtok()	4.11.5.8	MAN	NRQ

NOTE:

MAN - Mandatory for this profile.

NRQ - Mandatory for this profile.

Table H-39 – POSIX_C_LANG_SUPPORT Data and Time Functions

Function	Reference in the C Standard	AEP	LwAEP
asctime()	4.12.3.1	MAN	NRQ
ctime()	4.12.3.2	MAN	NRQ
gmtime()	4.12.3.3	MAN	NRQ
localtime()	4.12.3.4	MAN	NRQ
mktime()	4.12.2.3	MAN	NRQ
strftime()	4.12.3.5	MAN	NRQ
time()	4.12.2.4	MAN	NRQ
tzset()	4.12.2.4	NRQ	NRQ

NOTE:

MAN - Mandatory for this profile.

NRQ - Mandatory for this profile.

H.1.3.2 POSIX.1b

Note – Issue 8838

The options, limits, and any other constraints on POSIX.1b shall be provided as described in Table H-40.

Note – Issue 8934 modified table and table footnotes below

H.1.3 Constraints

Table H-40 – POSIX.1b Option Requirements

Option	AEP	LwAEP
{_POSIX_ASYNCHRONOUS_IO}	MAN	NRQ
{_POSIX_MAPPED_FILES}	NRQ	NRQ
{_POSIX_MEMLOCK}	MAN	NRQ
{_POSIX_MEMLOCK_RANGE}	MAN	NRQ
{_POSIX_MEMORY_PROTECTION}	NRQ	NRQ
{_POSIX_MESSAGE_PASSING}	MAN	NRQ
{_POSIX_PRIORITIZED_IO}	NRQ	NRQ
{_POSIX_PRIORITY_SCHEDULING}	NRQ	NRQ
{_POSIX_REALTIME_SIGNALS}	MAN	NRQ
{_POSIX_SEMAPHORES}	MAN	PRT
{_POSIX_SHARED_MEMORY_OBJECTS}	NRQ	NRQ
{_POSIX_SYNCHRONIZED_IO}	PRT*	NRQ
{_POSIX_TIMERS}	MAN	PRT
{_POSIX_FSYNC}	PRT**	NRQ

NOTE:

NRQ - Not required for this profile.

MAN - Mandatory for this profile.

PRT - Partial, only the subset or options or Units of Functionality called out in B.3.2

* fdatasync not required

** fsync not required Heavy weight processes are typically not supported in embedded operating systems. The mandatory POSIX.1b options can be implemented without the use of heavy weight signaling.

Note – Issue 8934 added new subsections

H.1.3.2.1 POSIX Semaphores

The functions listed in Table H-41 shall behave as described in the referenced clauses.

Table H-41 – POSIX.1b Semaphore Requirements

Option	Reference in POSIX.1b	AEP	LwAEP
sem_init()	11.2.1	MAN	MAN
sem_close()	11.2.4	MAN	NRQ
sem_destroy()	11.2.2	MAN	MAN
sem_getvalue()	11.2.8	MAN	MAN
sem_open()	11.2.3	MAN	NRQ
sem_post()	11.2.7	MAN	MAN
sem_unlink()	11.2.5	MAN	NRQ
sem_wait()	11.2.6	MAN	MAN
sem_trywait()	11.2.6	MAN	MAN

NOTE:

MAN - Mandatory for this profile

NRQ - Not required for this profile

H.1.3.2.2 POSIX Timers

The functions listed in Table H-42 shall behave as described in the referenced clauses.

Table H-42 – POSIX.1b Timer Requirementss

Option	Reference in POSIX.1b	AEP	LwAEP
clock_getres()	14.2.1	MAN	MAN
clock_gettime()	14.2.1	MAN	MAN
clock_settime()	14.2.1	MAN	MAN
timer_create()	14.2.2	MAN	MAN
timer_delete()	14.2.3	MAN	MAN
timer_gettime()	14.2.4	MAN	MAN
timer_settime()	14.2.4	MAN	MAN
nanosleep()	14.2.5	MAN	MAN
timer_getoverrun()	14.2.4	MAN	NRQ

NOTE:

MAN - Mandatory for this profile

NRQ - Not required for this profile

H.1.3.3 POSIX.1c

Note – Issue 8838

Table H-43, Table H-44, Table H-45, Table H-46, Table H-47 and Table H-48 contain the required options, limits, and any other constraints on POSIX.1c. The options, limits and any other constraints on POSIC.1c as described in Table H-41 shall be provided.

Note – Issue 8934 modified table below

Table H-43 – POSIX.1c Option Requirements

Option	AEP	LwAEP
{_POSIX_THREADS}	PRT	PRT
{_POSIX_THREAD_ATTR_STACKADDR}	MAN	NRQ
{_POSIX_THREAD_ATTR_STACKSIZE}	MAN	MAN
{_POSIX_THREAD_PRIO_INHERIT}	MAN	NRQ
{_POSIX_THREAD_PRIO_PROTECT}	MAN	NRQ
{_POSIX_THREAD_PRIORITY_SCHEDULING}	MAN	PRT
{_POSIX_THREAD_PROCESS_SHARED}	NRQ	NRQ
{_POSIX_THREAD_SAFE_FUNCTIONS}	PRT	PRT

NOTE:

NRQ - Not required for this profile.

MAN - Mandatory for this profile.

Note – Issue 8835 - Changed section reference to H 1.3.3

PRT - Partial, only the subset of units of functionality called out in H.1.3.3

H.1.3.3.1 Re-entrant User Group Function Behavior

The function listed in Tables H-44 and H-45 shall behave as described in the referenced clause.

Note – Issue 8934 modified tables below.

Table H-44 – POSIX_USER_GROUPS_R Function

Function	Reference in POSIX.1c	AEP	LwAEP
getlogin_r()	4.2.4	NRQ	NRQ

NOTE:

NRQ - Not required for this profile.

Note – Issue 8834 - Added Re-entrant device Specific Function Behavior section

H.1.3.3.2 Re-entrant Device Specific Function Behavior

The function listed in Table H-45shall behave as described in the referenced clause.

.

Table H-45 – POSIX_DEVICE_SPECIFIC_R Function

Function	Reference in POSIX.1c	AEP	LwAEP
ttyname_r()	4.7.4	NRQ	NRQ

NOTE:

NRQ - Not required for this profile.

H.1.3.3.3 File Locking Function Behavior

The functions listed in Table H-46 shall behave as described in the referenced clauses.

Note – Issue 8934 Modified table below.

Table H-46 – POSIX_FILE_LOCKING Functions

Function	Reference in POSIX.1c	AEP	LwAEP
getc_unlocked()	8.2.7	NRQ	NRQ
getchar_unlocked()	8.2.7	NRQ	NRQ
flockfile()	8.2.6	NRQ	NRQ
ftrylockfile()	8.2.6	NRQ	NRQ
funlockfile()	8.2.6	NRQ	NRQ
putc_unlocked()	8.2.7	NRQ	NRQ
putchar_unlocked()	8.2.7	NRQ	NRQ

NOTE:

NRQ - Not required for this profile.

H.1.3.3.4 Re-entrant C Language Support Function Behavior

The functions listed in Table H-47 shall behave as described in the referenced clauses.

Note – Issue 8934 Modified table below

Table H-47 – POSIX_C_LANG_SUPPORT_R Functions

Function	Reference in POSIX.1c	AEP	LwAEP
asctime_r()	8.3.5	MAN	NRQ
ctime_r()	8.3.6	MAN	NRQ
gmtime_r()	8.3.7	MAN	NRQ
localtime_r()	8.3.8	MAN	NRQ
rand_r()	8.3.9	MAN	MAN
readdir_r()	5.1.2	MAN	NRQ
strtok_r()	8.3.4	MAN	NRQ

Note – Issue 8832 - readdir_r() added to the table above

NOTE:

NRQ - Not required for this profile

MAN - Mandatory for this profile.

H.1.3.3.5 Re-entrant System Database Function Behavior

The functions listed in Table H-48 shall behave as described in the referenced clauses.

Note – Issue 8934 modified table below

Table H-48 – POSIX_SYSTEM_DATABASE_R Functions

Function	Reference in POSIX.1c	AEP	LwAEP
getgrgid_r()	9.2.1	NRQ	NRQ
getgrnam_r()	9.2.1	NRQ	NRQ
getpwnam_r()	9.2.2	NRQ	NRQ
getwuid_r()	9.2.2	NRQ	NRQ

NOTE:

NRQ - Not required for this profile.

Note – Issue 8934 added new POSIX Threads section

H.1.3.3.6 POSIX Threads

The functions listed in Table H-49 shall behave as described in the referenced clauses.

Table H-49 – POSIX.1c Thread Requirements

Function	Reference in POSIX.1c	AEP	LwAEP
pthread_atfork()	3.1.3	MAN	NRQ
pthread_attr_xxx()	16.2.1	MAN	PRT
pthread_cancel()	18.2.1	MAN	MAN
pthread_cleanup_xxx()	18.2.3	MAN	NRQ
pthread_cond_xxx()	11.4	MAN	NRQ
pthread_condattr_xxx()	11.4.1	MAN	NRQ
pthread_create()	16.2.2	MAN	MAN
pthread_detach()	16.2.4	MAN	NRQ
pthread_equal()	16.2.7	MAN	MAN
pthread_exit()	16.2.5	MAN	MAN
pthread_getschedparam()	13.5.2	MAN	MAN
pthread_getspecific()	17.1.2	MAN	NRQ
pthread_join()	16.2.3	MAN	MAN
pthread_key_xxx()	17.1	MAN	NRQ
pthread_kill()	3.3.10	MAN	NRQ
pthread_mutex_xxx()	11.3	MAN	NRQ
pthread_mutexattr_xxx()	11.3.1	MAN	NRQ
pthread_once()	16.2.8	MAN	NRQ
pthread_self()	16.2.6	MAN	MAN
pthread_setcancelstate()	18.2.2	MAN	NRQ
pthread_setcaceltype()	18.2.2	MAN	NRQ
pthread_setschedparam()	13.5.2	MAN	MAN
pthread_setspecific()	17.1.2	MAN	NRQ
pthread_sigmask()	3.3.5	MAN	NRQ
pthread_testcancel()	18.2.2	MAN	NRQ

NOTE:

MAN - Mandatory for this profile.

NRQ - Not required for this profile.

PRT - Partial, only the following subset functionality is requred: pthread_attr_getschedparam();pthread_attr_getstacksize();pthread_attr_init();pthread_attr_setschedparam(); pthread_attr_setstacksize(). And to implement these mandatory stack and schedule functions, it is necessary to adequately define the unsigned integer type size_t and the struct sched_param.

H.1.3 Constraints

Annex I SWRadio Properties XML (non-normative)

Note – Issue 7845 - Section changed to non-normative, Issue 7582 properties definition modified to accommodate comm cahnnel and comm equipement xml

I.1 SWRadio Properties XML

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SWRadio="http://schema.omg.org/SWRadio"
targetNamespace="http://schema.omg.org/SWRadio">

   <xsd:complexType name="ConfigureQuerySimpleProperty">

      <xsd:sequence>

         <xsd:element name="stepSize" type="xsd:string" minOccurs="0"/>

         <xsd:element name="description" type="xsd:string" minOccurs="0"/>

         <xsd:element name="label" type="xsd:string" minOccurs="0"/>

         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="integerId" type="xsd:long" minOccurs="0"/>

         <xsd:element name="maxLatency" type="SWRadio:TimeType"
minOccurs="0"/>

         <xsd:element name="range" type="SWRadio:Range" minOccurs="0"/>

         <xsd:element name="units" type="xsd:string" minOccurs="0"/>

         <xsd:element name="type" type="SWRadio:SimpleType"/>

         <xsd:element name="enumerations" type="SWRadio:Enumerations"
minOccurs="0" maxOccurs="unbounded"/>

         <xsd:element name="value" type="xsd:string" minOccurs="0"/>

      </xsd:sequence>

      <xsd:attribute name="isReadOnly" type="xsd:boolean" use="optional"
default="false"/>

   </xsd:complexType>
   <xsd:element name="ConfigureQuerySimpleProperty"
type="SWRadio:ConfigureQuerySimpleProperty"/>

   <xsd:complexType name="EnumerationLiteral">

      <xsd:sequence>

         <xsd:element name="label" type="xsd:string"/>

         <xsd:element name="value" type="xsd:string"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="EnumerationLiteral"
type="SWRadio:EnumerationLiteral"/>

   <xsd:complexType name="StructProperty">

      <xsd:sequence>

         <xsd:element name="description" type="xsd:string" minOccurs="0"/>

         <xsd:element name="label" type="xsd:string" minOccurs="0"/>

         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="integerId" type="xsd:long" minOccurs="0"/>

         <xsd:element name="maxLatency" type="SWRadio:TimeType"
minOccurs="0"/>

         <xsd:element name="range" type="SWRadio:Range" minOccurs="0"/>

         <xsd:element name="units" type="xsd:string" minOccurs="0"/>

         <xsd:element name="simple" type="SWRadio:SimpleProperty"
minOccurs="0" maxOccurs="unbounded"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="StructProperty" type="SWRadio:StructProperty"/>

   <xsd:complexType name="StructSequenceProperty">

      <xsd:sequence>

         <xsd:element name="stepSize" type="xsd:string" minOccurs="0"/>
         <xsd:element name="description" type="xsd:string" minOccurs="0"/>

         <xsd:element name="label" type="xsd:string" minOccurs="0"/>

         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="integerId" type="xsd:long" minOccurs="0"/>

         <xsd:element name="maxLatency" type="SWRadio:TimeType"
minOccurs="0"/>

         <xsd:element name="range" type="SWRadio:Range" minOccurs="0"/>

         <xsd:element name="units" type="xsd:string" minOccurs="0"/>

         <xsd:element name="structValues" type="SWRadio:StructProperty"
maxOccurs="unbounded"/>

      </xsd:sequence>

      <xsd:attribute name="isReadOnly" type="xsd:boolean" use="optional"
default="false"/>

   </xsd:complexType>

   <xsd:element name="StructSequenceProperty"
type="SWRadio:StructSequenceProperty"/>

   <xsd:complexType name="TestProperty">

      <xsd:sequence>

         <xsd:element name="description" type="xsd:string" minOccurs="0"/>

         <xsd:element name="label" type="xsd:string" minOccurs="0"/>

         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="integerId" type="xsd:long" minOccurs="0"/>

         <xsd:element name="maxLatency" type="SWRadio:TimeType"
minOccurs="0"/>

         <xsd:element name="range" type="SWRadio:Range" minOccurs="0"/>

         <xsd:element name="units" type="xsd:string" minOccurs="0"/>

         <xsd:element name="inputValue" type="SWRadio:SimpleProperty"
minOccurs="0" maxOccurs="unbounded"/>
```

```
         <xsd:element name="resultValue" type="SWRadio:SimpleProperty"
maxOccurs="unbounded"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="TestProperty" type="SWRadio:TestProperty"/>

   <xsd:complexType name="ExecutableProperty">

      <xsd:sequence>

         <xsd:element name="description" type="xsd:string" minOccurs="0"/>

         <xsd:element name="label" type="xsd:string" minOccurs="0"/>

         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="integerId" type="xsd:long" minOccurs="0"/>

         <xsd:element name="maxLatency" type="SWRadio:TimeType"
minOccurs="0"/>

         <xsd:element name="range" type="SWRadio:Range" minOccurs="0"/>

         <xsd:element name="units" type="xsd:string" minOccurs="0"/>

         <xsd:element name="type" type="SWRadio:SimpleType"/>

         <xsd:element name="enumerations" type="SWRadio:Enumerations"
minOccurs="0" maxOccurs="unbounded"/>

         <xsd:element name="value" type="xsd:string" minOccurs="0"/>

         <xsd:element name="queryable" type="xsd:string"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="ExecutableProperty"
type="SWRadio:ExecutableProperty"/>

   <xsd:complexType name="SimpleProperty">

      <xsd:sequence>

         <xsd:element name="description" type="xsd:string" minOccurs="0"/>

         <xsd:element name="label" type="xsd:string" minOccurs="0"/>
         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="integerId" type="xsd:long" minOccurs="0"/>

         <xsd:element name="maxLatency" type="SWRadio:TimeType"
minOccurs="0"/>

         <xsd:element name="range" type="SWRadio:Range" minOccurs="0"/>

         <xsd:element name="units" type="xsd:string" minOccurs="0"/>

         <xsd:element name="type" type="SWRadio:SimpleType"/>

         <xsd:element name="enumerations" type="SWRadio:Enumerations"
minOccurs="0" maxOccurs="unbounded"/>

         <xsd:element name="value" type="xsd:string"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="SimpleProperty" type="SWRadio:SimpleProperty"/>

   <xsd:complexType name="CapacityProperty">

      <xsd:sequence>

         <xsd:element name="capabilityModel" type="xsd:string"
minOccurs="0"/>

         <xsd:element name="locallyManaged" type="xsd:boolean"
minOccurs="0"/>

         <xsd:element name="description" type="xsd:string" minOccurs="0"/>

         <xsd:element name="label" type="xsd:string" minOccurs="0"/>

         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="integerId" type="xsd:long" minOccurs="0"/>

         <xsd:element name="maxLatency" type="SWRadio:TimeType"
minOccurs="0"/>

         <xsd:element name="range" type="SWRadio:Range" minOccurs="0"/>

         <xsd:element name="units" type="xsd:string" minOccurs="0"/>

         <xsd:element name="type" type="SWRadio:SimpleType"/>
         <xsd:element name="enumerations" type="SWRadio:Enumerations"
minOccurs="0" maxOccurs="unbounded"/>

         <xsd:element name="value" type="xsd:string" minOccurs="0"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="CapacityProperty" type="SWRadio:CapacityProperty"/>

   <xsd:complexType name="CharacteristicProperty">

      <xsd:sequence>

         <xsd:element name="capabilityModel" type="xsd:string"
minOccurs="0"/>

         <xsd:element name="locallyManaged" type="xsd:boolean"
minOccurs="0"/>

         <xsd:element name="description" type="xsd:string" minOccurs="0"/>

         <xsd:element name="label" type="xsd:string" minOccurs="0"/>

         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="integerId" type="xsd:long" minOccurs="0"/>

         <xsd:element name="maxLatency" type="SWRadio:TimeType"
minOccurs="0"/>

         <xsd:element name="range" type="SWRadio:Range" minOccurs="0"/>

         <xsd:element name="units" type="xsd:string" minOccurs="0"/>

         <xsd:element name="type" type="SWRadio:SimpleType"/>

         <xsd:element name="enumerations" type="SWRadio:Enumerations"
minOccurs="0" maxOccurs="unbounded"/>

         <xsd:element name="value" type="xsd:string" minOccurs="0"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="CharacteristicProperty"
type="SWRadio:CharacteristicProperty"/>

   <xsd:complexType name="ConfigureQuerySimpleSeqProperty">
```

```
      <xsd:sequence>

         <xsd:element name="stepSize" type="xsd:string" minOccurs="0"/>

         <xsd:element name="description" type="xsd:string" minOccurs="0"/>

         <xsd:element name="label" type="xsd:string" minOccurs="0"/>

         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="integerId" type="xsd:long" minOccurs="0"/>

         <xsd:element name="maxLatency" type="SWRadio:TimeType"
minOccurs="0"/>

         <xsd:element name="range" type="SWRadio:Range" minOccurs="0"/>

         <xsd:element name="units" type="xsd:string" minOccurs="0"/>

         <xsd:element name="type" type="SWRadio:SimpleType"/>

         <xsd:element name="enumerations" type="SWRadio:Enumerations"
minOccurs="0" maxOccurs="unbounded"/>

         <xsd:element name="value" type="xsd:string" minOccurs="0"
maxOccurs="unbounded"/>

      </xsd:sequence>

      <xsd:attribute name="isReadOnly" type="xsd:boolean" use="optional"
default="false"/>

   </xsd:complexType>

   <xsd:element name="ConfigureQuerySimpleSeqProperty"
type="SWRadio:ConfigureQuerySimpleSeqProperty"/>

   <xsd:complexType name="CharacteristicSelectionProperty">

      <xsd:sequence>

         <xsd:element name="capabilityModel" type="xsd:string"
minOccurs="0"/>

         <xsd:element name="locallyManaged" type="xsd:boolean"
minOccurs="0"/>

         <xsd:element name="description" type="xsd:string" minOccurs="0"/>

         <xsd:element name="label" type="xsd:string" minOccurs="0"/>
         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="integerId" type="xsd:long" minOccurs="0"/>

         <xsd:element name="maxLatency" type="SWRadio:TimeType"
minOccurs="0"/>

         <xsd:element name="range" type="SWRadio:Range" minOccurs="0"/>

         <xsd:element name="units" type="xsd:string" minOccurs="0"/>

         <xsd:element name="type" type="SWRadio:SimpleType"/>

         <xsd:element name="enumerations" type="SWRadio:Enumerations"
minOccurs="0" maxOccurs="unbounded"/>

         <xsd:element name="value" type="xsd:string"
maxOccurs="unbounded"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="CharacteristicSelectionProperty"
type="SWRadio:CharacteristicSelectionProperty"/>

   <xsd:complexType name="CharacteristicSetProperty">

      <xsd:sequence>

         <xsd:element name="capabilityModel" type="xsd:string"
minOccurs="0"/>

         <xsd:element name="locallyManaged" type="xsd:boolean"
minOccurs="0"/>

         <xsd:element name="description" type="xsd:string" minOccurs="0"/>

         <xsd:element name="label" type="xsd:string" minOccurs="0"/>

         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="integerId" type="xsd:long" minOccurs="0"/>

         <xsd:element name="maxLatency" type="SWRadio:TimeType"
minOccurs="0"/>

         <xsd:element name="range" type="SWRadio:Range" minOccurs="0"/>

         <xsd:element name="units" type="xsd:string" minOccurs="0"/>
```

```
            <xsd:element name="characterisitics" type="SWRadio:StructProperty"
maxOccurs="unbounded"/>

         </xsd:sequence>

      </xsd:complexType>

      <xsd:element name="CharacteristicSetProperty"
type="SWRadio:CharacteristicSetProperty"/>

      <xsd:complexType name="Range">

         <xsd:sequence>

            <xsd:element name="min" type="xsd:string"/>

            <xsd:element name="max" type="xsd:string"/>

         </xsd:sequence>

      </xsd:complexType>

      <xsd:element name="Range" type="SWRadio:Range"/>

      <xsd:complexType name="TimeType">

         <xsd:sequence>

            <xsd:element name="seconds" type="xsd:unsignedLong"/>

            <xsd:element name="nanoseconds" type="xsd:unsignedLong"/>

         </xsd:sequence>

      </xsd:complexType>

      <xsd:element name="TimeType" type="SWRadio:TimeType"/>

      <xsd:complexType name="ConfigureQueryStructProperty">

         <xsd:sequence>

            <xsd:element name="description" type="xsd:string" minOccurs="0"/>

            <xsd:element name="label" type="xsd:string" minOccurs="0"/>

            <xsd:element name="name" type="xsd:string"/>

            <xsd:element name="integerId" type="xsd:long" minOccurs="0"/>
            <xsd:element name="maxLatency" type="SWRadio:TimeType"
   minOccurs="0"/>

            <xsd:element name="range" type="SWRadio:Range" minOccurs="0"/>

            <xsd:element name="units" type="xsd:string" minOccurs="0"/>

            <xsd:element name="simple" type="SWRadio:SimpleProperty"
   minOccurs="0" maxOccurs="unbounded"/>

            <xsd:element name="stepSize" type="xsd:string" minOccurs="0"/>

         </xsd:sequence>

         <xsd:attribute name="isReadOnly" type="xsd:boolean" use="optional"
   default="false"/>

      </xsd:complexType>

      <xsd:element name="ConfigureQueryStructProperty"
   type="SWRadio:ConfigureQueryStructProperty"/>

      <xsd:complexType name="Enumerations">

         <xsd:sequence>

            <xsd:element name="enumerationLiteral"
   type="SWRadio:EnumerationLiteral" maxOccurs="unbounded"/>

         </xsd:sequence>

      </xsd:complexType>

      <xsd:element name="Enumerations" type="SWRadio:Enumerations"/>

      <xsd:simpleType name="SimpleType">

         <xsd:restriction base="xsd:string">

            <xsd:enumeration value="boolean"/>

            <xsd:enumeration value="char"/>

            <xsd:enumeration value="double"/>

            <xsd:enumeration value="float"/>

            <xsd:enumeration value="short"/>

            <xsd:enumeration value="long"/>
            <xsd:enumeration value="longlong"/>

            <xsd:enumeration value="objref"/>

            <xsd:enumeration value="octet"/>

            <xsd:enumeration value="string"/>

            <xsd:enumeration value="ulong"/>

            <xsd:enumeration value="ulonglong"/>

            <xsd:enumeration value="ushort"/>

            <xsd:enumeration value="wchar"/>

            <xsd:enumeration value="wstring"/>

            <xsd:enumeration value="longdouble"/>

         </xsd:restriction>

      </xsd:simpleType>

      <xsd:complexType name="Properties">

         <xsd:choice minOccurs="0" maxOccurs="unbounded">

            <xsd:element ref="SWRadio:ConfigureQuerySimpleProperty"/>

            <xsd:element ref="SWRadio:TestProperty"/>

            <xsd:element ref="SWRadio:ExecutableProperty"/>

            <xsd:element ref="SWRadio:CapacityProperty"/>

            <xsd:element ref="SWRadio:CharacteristicProperty"/>

            <xsd:element ref="SWRadio:ConfigureQuerySimpleSeqProperty"/>

            <xsd:element ref="SWRadio:CharacteristicSelectionProperty"/>

            <xsd:element ref="SWRadio:CharacteristicSetProperty"/>

            <xsd:element ref="SWRadio:ConfigureQueryStructProperty"/>

         </xsd:choice>

      </xsd:complexType>

      <xsd:element name="Properties" type="SWRadio:Properties"/>
```

```
   </xsd:schema>
```

Annex J Communication Channel XML (non-normative)

Note – Issue 7582, added XML for comm channel and equipment

J.1 CommChannel XML

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SWRadio="http://schema.omg.org/SWRadio"
targetNamespace="http://schema.omg.org/SWRadio">

   <xsd:include schemaLocation="D:\\SWRadio\Properties.xsd"/>

   <xsd:complexType name="CommEquipment">

      <xsd:sequence>

         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="stereotypeName" type="xsd:string"/>

         <xsd:element name="properties" type="SWRadio:Properties"/>

         <xsd:element name="ports" type="SWRadio:Ports"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="CommEquipment" type="SWRadio:CommEquipment"/>

   <xsd:complexType name="Port">

      <xsd:sequence>

         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="stereotypeName" type="xsd:string"/>

         <xsd:element name="properties" type="SWRadio:Properties"
maxOccurs="unbounded"/>

      </xsd:sequence>

   </xsd:complexType>
   <xsd:element name="Port" type="SWRadio:Port"/>

   <xsd:complexType name="CommEquipmentConnector">

      <xsd:sequence>

         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="sinkPortName" type="xsd:string"/>

         <xsd:element name="sinkCommEquipmentName" type="xsd:string"/>

         <xsd:element name="sourcePortName" type="xsd:string"/>

         <xsd:element name="sourceCommEquipmentName" type="xsd:string"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="CommEquipmentConnector"
type="SWRadio:CommEquipmentConnector"/>

   <xsd:complexType name="Channel">

      <xsd:sequence>

         <xsd:element name="name" type="xsd:string"/>

         <xsd:element name="stereotypeName" type="xsd:string"/>

         <xsd:element name="properties" type="SWRadio:Properties"/>

         <xsd:element name="subchannels" type="SWRadio:References"
minOccurs="0"/>

         <xsd:element name="commEquipments" type="SWRadio:References"
minOccurs="0"/>

         <xsd:element name="connections" type="SWRadio:Connections"
minOccurs="0"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="Channel" type="SWRadio:Channel"/>

   <xsd:complexType name="Ports">

      <xsd:sequence>
```

```
         <xsd:element name="port" type="SWRadio:Port"
maxOccurs="unbounded"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="Ports" type="SWRadio:Ports"/>

   <xsd:complexType name="References">

      <xsd:sequence>

         <xsd:element name="nameRef" type="xsd:string"
maxOccurs="unbounded"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="References" type="SWRadio:References"/>

   <xsd:complexType name="Connections">

      <xsd:sequence>

         <xsd:element name="connection"
type="SWRadio:CommEquipmentConnector" maxOccurs="unbounded"/>

      </xsd:sequence>

   </xsd:complexType>

   <xsd:element name="Connections" type="SWRadio:Connections"/>

   <xsd:complexType name="CommChannel">

      <xsd:choice minOccurs="0" maxOccurs="unbounded">

         <xsd:element ref="SWRadio:CommEquipment"/>

         <xsd:element ref="SWRadio:Channel"/>

      </xsd:choice>

   </xsd:complexType>

   <xsd:element name="CommChannel" type="SWRadio:CommChannel"/>

</xsd:schema>
```

J.1 CommChannel XML

Annex K SWRadio CORBA IDL (non-normative)

K.1 Domain Facility Software Radio Module

```
//File: DfSWRadio.idl

#ifndef __DFSWRADIO_DEFINED
#define __DFSWRADIO_DEFINED

#include "DfSWRadioManagedComponentStatuses.idl"
#include "DfSWRadioCommonLayer.idl"
#include "DfSWRadioPhysicalLayer.idl"
#include "DfSWRadioDataLinkLayer.idl"
#include "DfSWRadioControlRadioSetManagement.idl"

#endif
```

Annex L SWRadio Document Type Definitions (non-normative)

The SWRadio specification provides architectural specifications for the deployment of communications software into a Software Definable Radio (SDR) device. The intent of the SDR device is to provide a re-configurable platform, which can host software components written by various vendors to support user functional services. The SWRadio specification requires portable software components to provide common information called a domain profile. The intent of this appendix is to clearly define to the component developers the requirements of information and format for the delivery of this information. The radio management functions use the component deployment information expressed in the Domain Profile. The information is used to start, initialize, and maintain the applications that are installed into the SWRadio-compliant system.

This specification defines the XML Document Type Definition (DTD) set for use in deploying SWRadio components. The complete DTD set is contained in Attachment 1 to this Appendix.

L.1 Deployment Overview.

The hardware devices and software components that make up an SWRadio Radio Set or Radio System are described by a set of XML descriptor files that are collectively referred to as a Radio Domain Profile. Descriptor files are Software Package, Device Package, Properties, Software Component, Software Assembly, Device Configuration, and DomainManagerComponent Configuration. A Software Profile is either a Software Assembly Descriptor (for applications) or a Software Package Descriptor (for all other software components and hardware devices). These descriptor files describe the identity, capabilities, properties, and inter-dependencies of the hardware devices and software components that make up the system. All of the descriptive data about a system is expressed in the XML vocabulary. This document includes a UML diagram of each complex XML element defined. That is, a UML diagram is provided for each element that makes use of more than one type of XML element as a part of its definition. The UML diagram precedes the XML definition that it represents.

Figure L-86 depicts the relationships between the descriptor files that are used to describe a system's hardware and software assets. The XML vocabulary within each of these files describes a distinct aspect of the hardware and software assets.

A Software Assembly Descriptor file describes how multiple components of an assembly, i.e., an application, are deployed and interconnected. A Software Assembly Descriptor file is associated with one or more Software Package Descriptor files. Each component of the Software Assembly Descriptor is described in a Software Package Descriptor file. Information about the interfaces that a component publishes and/or consumes is contained in a Software Component Descriptor file. Each Software Component Descriptor file is associated with a Software Package Descriptor file that describes one or more implementations of the software component. Software properties are described in a Properties Descriptor File that may be applicable to all implementations of the component, i.e., associated at the Software Package Descriptor level or applicable to a single implementation of the component.

Two types of files, a Device Package Descriptor, and a Device Configuration Descriptor, describe hardware devices and are known collectively as a Device Profile. The hardware device is described by the Device Package Descriptor. The logical device is described by the Software Package Descriptor. The Device Configuration Descriptor contains the associations between hardware devices and logical devices. A Device Package Descriptor file identifies the class of the device. Property files, associated with Device Package Descriptors, contain information about the properties of the hardware device being deployed such as serial number, processor type, and allocation capacities.

A Device Configuration Descriptor file describes the components that are initially started up on the device and how to find the DomainManagerComponent. Each component of the Device Configuration Descriptor is described in a Software Package Descriptor file.

The DomainManagerComponent Configuration Descriptor file contains a reference to the DomainManagerComponent Software Package Descriptor file.

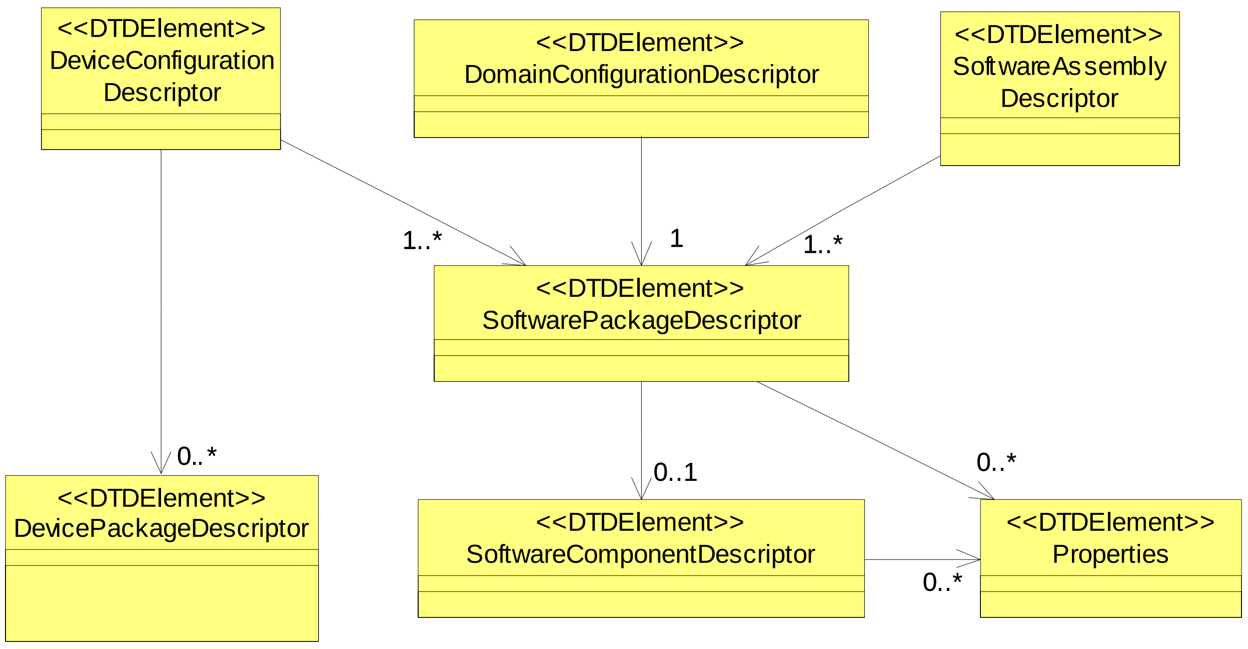


Figure L-86 – Relationships Between SWRadio DTD XML File Types

L.2 Software Package Descriptor.

The SWRadio Software Package Descriptor is used at deployment time to load an SWRadio compliant component and its various implementations. The information contained in the Software Package Descriptor will provide the basis for the Radio Management function to manage the component within the SWRadio architecture.

The software package descriptor may contain various implementations of any given component. Within the specification of a software package descriptor several other files are referenced including a component level *propertyfile* and a software component *descriptor* file. Within any given implementation there may be additional *propertyfiles*.

L.2.1 Software Package.

The *softpkg* element (see Figure L-87) indicates a Software Package Descriptor (SPD) definition. The *softpkg* id uniquely identifies the package. The version attribute specifies the version of the component. The name attribute is a user-friendly label for the *softpkg* element. The name attribute is supplied when the id is not user-friendly such as a DCE UUID. The DCE UUID format starts with the characters "DCE:" and is followed by the printable form of the UUID, a colon, and a decimal minor version number, for example: "DCE:700dc518-0110-11ce-ac8f-0800090b5d3e:1". The decimal minor version number is optional. The type attribute indicates whether or not the component implementation is SWRadio compliant (e.g., “swradio_compliant”, “swradio_non_compliant”). All files referenced by a Software Package are located in the same directory as the SPD file or a directory that is relative to the directory where the SPD file is located. The type attribute is used to establish the level of compliance with the SWRadio spec.

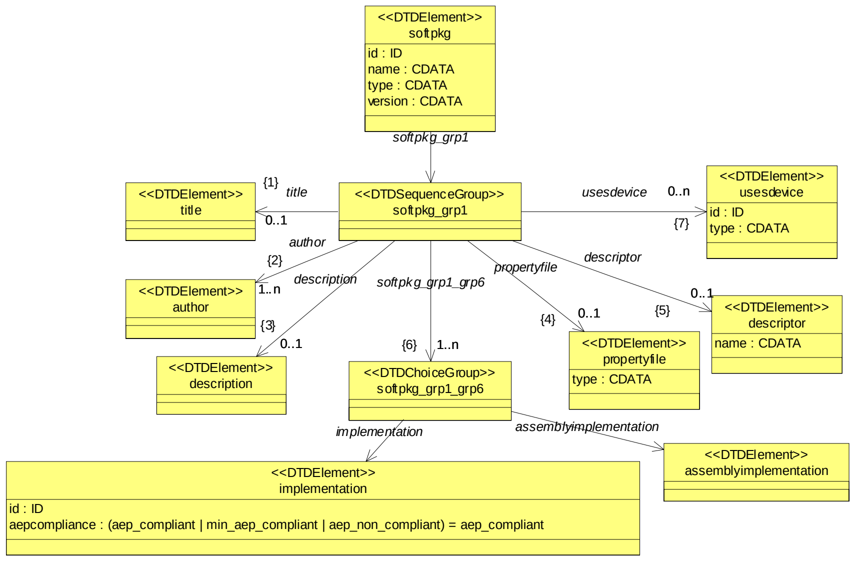


Figure L-87 – *softpkg* Element Relationships

The set of properties for a Software Package come from the union of these properties sources using the following precedence order:

3. SPD Implementation Properties - indicates the implementation values for a properties that are specific to one and only one one implementation.
4. SPD level properties - indicates the implementation value for a property that is true for all implementations unless over-ridden at the implementation element level.
5. SCD level properties - property definitions for all implementations

Any duplicate properties having the same ID are ignored. Duplicated properties must be the same property type, only the value can be over-ridden. The SPD-level and implementation-level properties only state what the values are for a component implementation specified in SPD. These property values are not used by ApplicationFactoryComponent for initial configuration of the deployed component and ExecuableProperty(s) for the deployed component main program. ExecutableProperty(s) are usually only defined at the SDP-level and implementation level. These properties are used for the ExecutableDeviceComponent execute operation options parameter.

```
<!ELEMENT softpkg

      ( title?

      , author+

      , description?

      , propertyfile?

      , descriptor?

      , (implementation | assemblyimplementation)+

      , usesdevice*

)>

!ATTLIST softpkg

   id    ID    #REQUIRED

   name  CDATA #REQUIRED

   type CDATA #IMPLIED

   version CDATA #IMPLIED >
```

L.2.1.1 *propertyfile*.

The *propertyfile* element is used to indicate the local filename of the Property Descriptor file associated with the Software Package. The intent of the *propertyfile* will be to provide the definition of *properties* elements common to all component implementations being deployed in accordance with the Software Package (*softpkg*).Property Descriptor files may also contain *properties* elements that are used in definition of command and control id value pairs used by the SWRadio PropertySet configure() and query() operations. The format of the *properties* element is described in the Properties Descriptor (Section Properties Descriptor.).

```
<!ELEMENT propertyfile

(localfile

)>

<!ATTLIST propertyfile

type CDATA #IMPLIED>
```

L.2.1 Software Package.

L.2.1.1.1 *localfile.*

The *localfile* element is used to reference a file in the same directory as the SPD file or a directory that is relative to the directory where the SPD file is located. When the name attribute is a simple name, the file exists in the same directory as the SPD file. A relative directory indication begins either with "../" meaning parent directory and "./" means current directory in the name attribute. Multiple "../" and directory names can follow the initial "../" in the name attribute. All name attributes must have a simple name at the end of the file name.

```
<!ELEMENT localfile EMPTY>

<!ATTLIST localfile

name CDATA #REQUIRED>
```

L.2.1.2 *title*.

The *title* element is used for indicating a title for the software component being installed in accordance with the *softpkg* element.

```
<!ELEMENT title (#PCDATA)>
```

L.2.1.3 *author.*

The *author* element (see Figure L-88) will be used to indicate the name of the person, the company, and the web page of the developer producing the component being installed into the system.

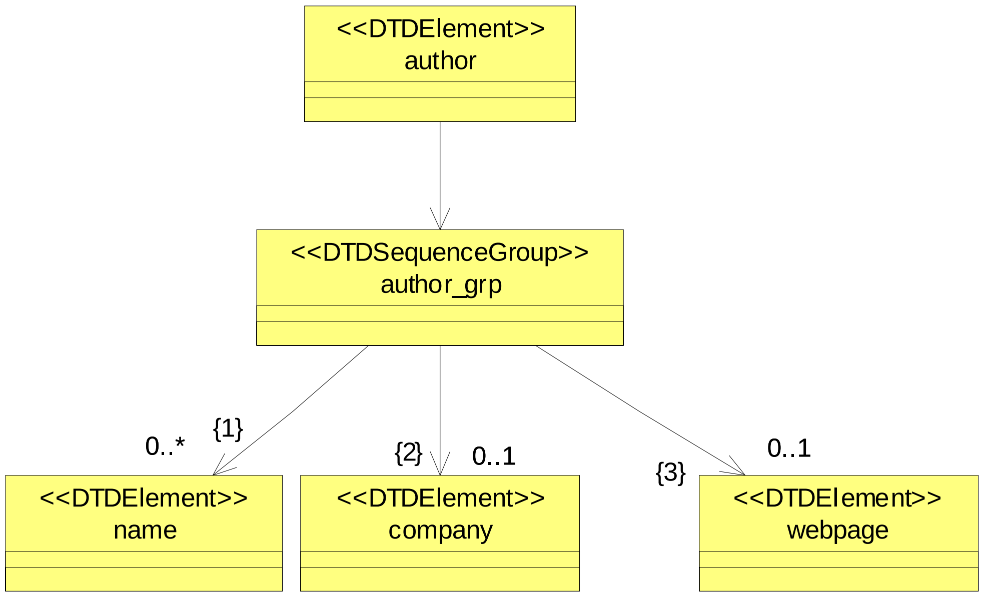


Figure L-88 – *author* Element Relationships

```
<!ELEMENT author

      ( name*

      , company?

      , webpage?

)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT webpage (#PCDATA)>
```

L.2.1.4 *description.*

The *description* element will be used to describe any pertinent information about the software component being delivered to the system.

```
<!ELEMENT description (#PCDATA)>
```

L.2.1.5 *descriptor.*

The *descriptor* element points to the local filename of the Software Component Descriptor (SCD) file used to document the interface information for the component being delivered to the system. In the case of an SCA Component, the SCD will contain information about three aspects of the component (the component type, message ports, and IDL interfaces). The SCD file is optional, since some SCA components are non-CORBA components, like digital signal processor (DSP) “c” code (see section on software component descriptor file, section Software Component Descriptor.).

```
<!ELEMENT descriptor

      (localfile

)>

<!ATTLIST descriptor

      name  CDATA #IMPLIED>
```

L.2.1.6 *implementation.*

The *implementation* element (see Figure L-89) contains descriptive information about the particular implementation template for a software component contained in the *softpkg* element. The *implementation* element is intended to allow multiple component templates to be delivered to the system in one Software Package. Each *implementation* element is intended to allow the same component to support different types of processors, operating systems, etc. The *implementation* element will also allow definition of implementation-dependent properties for use in CF *Device*, CF *Application*, or CF *Resource* creation. The *implementation* element’s id attribute uniquely identifies a specific implementation of the component. The compiler, *programminglanguage*, *humanlanguage*, os, processor, and *runtime* elements are optional dependency elements. The aepcompliance attribute is used to establish the level of compliance with the SWRadio Application Environment Profiles (AEPs) (refer to Annex H). The aepcompliance indicates if an implementation is compliant with an AEPs and to which one.

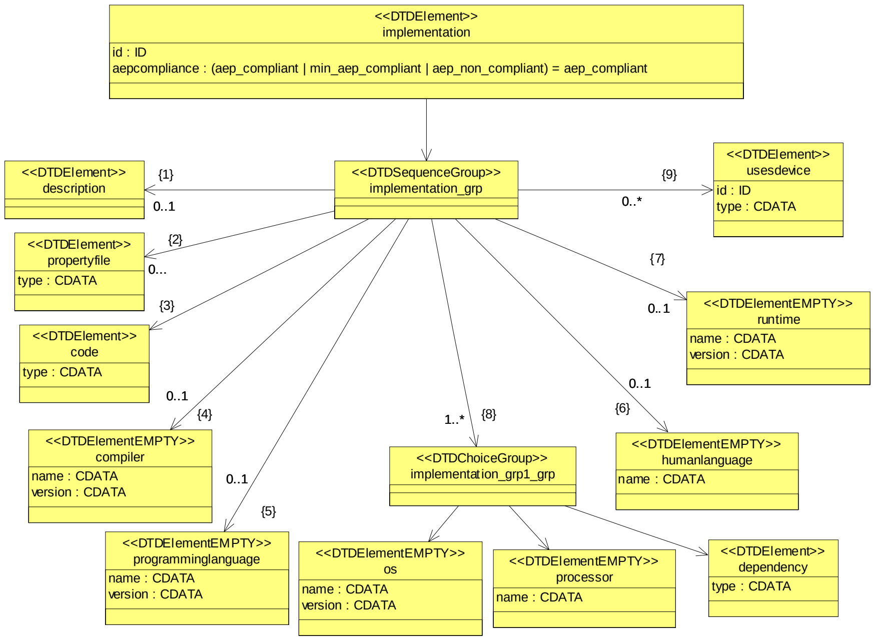


Figure L-89 – *implementation* Element Relationships

```
<!ELEMENT  implementation

   ( description?

   , propertyfile?

   , code

   , compiler?

   , programminglanguage?

   , humanlanguage?

   , runtime?

   , ( os

     | processor

     | dependency
```

```
        )+

   , usesdevice*

      )>

   <!ATTLIST implementation

         id ID #REQUIRED

   aepcompliance (aep_compliant | lwaep_compliant |aep_non_compliant)
   “aep_compliant”>
```

L.2.1.6.1 *propertyfile*.

The *propertyfile* element is used to indicate the local filename of the Property Descriptor file associated with this *implementation* element. Although the specification does not restrict the specific use of the Property Descriptor file based on context, it is intended within the *implementation* element to provide component implementation specific *properties* elements for use in command and control id value pair settings to the ResourceComponent configure() and query() operations. See the description of the *properties* element format in the Properties Descriptor, section L.4.

```
   <!ELEMENT propertyfile

         (localfile

         )>

   <!ATTLIST propertyfile

         type CDATA #IMPLIED>

   <!ELEMENT localfile EMPTY>

   <!ATTLIST localfile

         name CDATA #REQUIRED>
```

L.2.1.6.2 *description.*

The *description* element will be used to describe any pertinent information about the software component implementation that the software developer wishes to document within the software package profile.

```
   <!ELEMENT description (#PCDATA)>
```

L.2.1.6.3 *code.*

The *code* element (see Figure L-90) will be used to indicate the local filename of the code that is described by the *softpk*g element, for a specific implementation of the software component. The stack size and priority are options parameters used by the ExecutableDeviceComponent *execute*() operation. Data types for the values of these options are unsigned long. The *type* attribute for the *code* element will also indicate the type of file being delivered to the system. The *entrypoint* element provides the means for providing the name of the entry point of the component being delivered. The valid values for the type attribute are: “Executable”, “KernelModule”, “SharedLibrary”, and “Driver.”

The meaning of the code type attribute:

1. Executable means to use LoadableDeviceComponent *load* and ExecutableDeviceComponent *execute*. This is a “main” process.
2. Driver and Kernel Module means load only.
3. SharedLibrary means dynamic linking.

- Without a code *entrypoint* element means load only.
- With a code *entrypoint* element means load and ExecutableDeviceComponent *execute*.

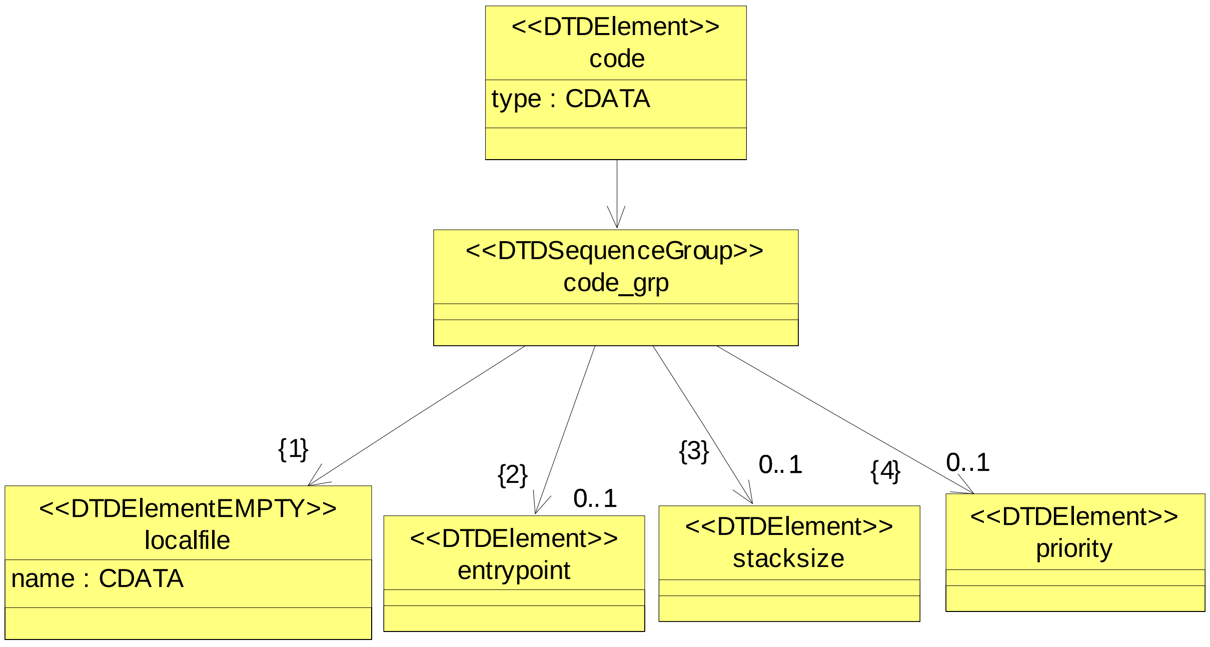


Figure L-90 – *code* Element Relationships

L.2.1 Software Package.

```
<!ELEMENT code
      ( localfile
      , entrypoint?
, stacksize?
      , priority?
      )>
<!ATTLIST code
      type CDATA #IMPLIED>


<!ELEMENT localfile EMPTY>
<!ATTLIST localfile
   name CDATA #REQUIRED>
<!ELEMENT entrypoint (#PCDATA)>
<!ELEMENT stacksize (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
```

L.2.1.6.4 compiler.

The *compiler* element will be used to indicate the compiler used to build the software component being described by the *softpkg* element. The required *name* attribute will specify the name of the compiler used, and the *version* attribute will contain the compiler version.

```
<!ELEMENT compiler EMPTY>
<!ATTLIST compiler
      name  CDATA #REQUIRED
      version CDATA #IMPLIED>
```

L.2.1.6.5 *programminglanguage.*

The *programminglanguage* element will be used to indicate the type of programming language used to build the component implementation. The required *name* attribute will specify a language such as “c”, “c++”, or “java”.

```
<!ELEMENT programminglanguage EMPTY>

<!ATTLIST programminglanguage

      name  CDATA #REQUIRED

      version CDATA #IMPLIED>
```

L.2.1.6.6 *humanlanguage.*

The *humanlanguage* element will be used to indicate the human language for which the software component was developed.

```
<!ELEMENT humanlanguage EMPTY>

<!ATTLIST humanlanguage

      name CDATA #REQUIRED>
```

L.2.1.6.7 *os.*

The *os* element will be used to indicate the *operating system* on which the software component is capable of operating. The required *name* attribute will indicate the name of the operating system and the *version* attribute will contain the operating system. The *os* attributes will be defined in a DeviceComponent’s property file as an allocation property (SWRadio ServiceProperty) of string type and with names os_name and *os*_version and with an *action* element value other than “external”. The *os* element is automatically interpreted as a dependency and compared against DeviceComponent’s allocation properties (SWRadio ServiceProperty) with names of *os*_name and *os*_version. Legal *os*_name attribute values are listed in Attachment 2 to this appendix.

```
<!ELEMENT os EMPTY>

<!ATTLIST os

      name  CDATA #REQUIRED

      version CDATA #IMPLIED>
```

L.2.1.6.8 *processor.*

The *processor* element will be used to indicate the *processor* and/or *processor family* on which this software component will operate. The processor name attribute will be defined in a DeviceComponent’s property file as an allocation property (SWRadio ServiceProperty) of string type and with a name of processor_name and with an *action* element value other than “external”. The *processor* element is automatically interpreted as a dependency and compared against a DeviceComponent’s allocation property with a name of processor_name. Legal processor_name attribute values are listed in Attachment 2 to this appendix.

```
<!ELEMENT processor EMPTY>

<!ATTLIST processor

      name CDATA #REQUIRED>
```

L.2.1.6.9 *dependency.*

The *dependency* element (see Figure L-91) is used to indicate the dependent relationships between the components being delivered and other components and devices, in a SWRadio compliant system. The *softpkgref* element is used to specify a Software Package file that must be resident within the system for the component, described by this *softpkg* element, to load without errors. The propertyref or propertyvaluesref references a specific ServiceProperty (allocation property), by ServiceProperty identifier, and provide the value that will be used by a ManagedServiceComponent CapabilityModel. The DomainManagerComponent will use these dependency definitions to assure that ServiceComponent(s) that are necessary for proper operation of the implementation are present and available. The type attribute is descriptive information indicating the type of dependency.

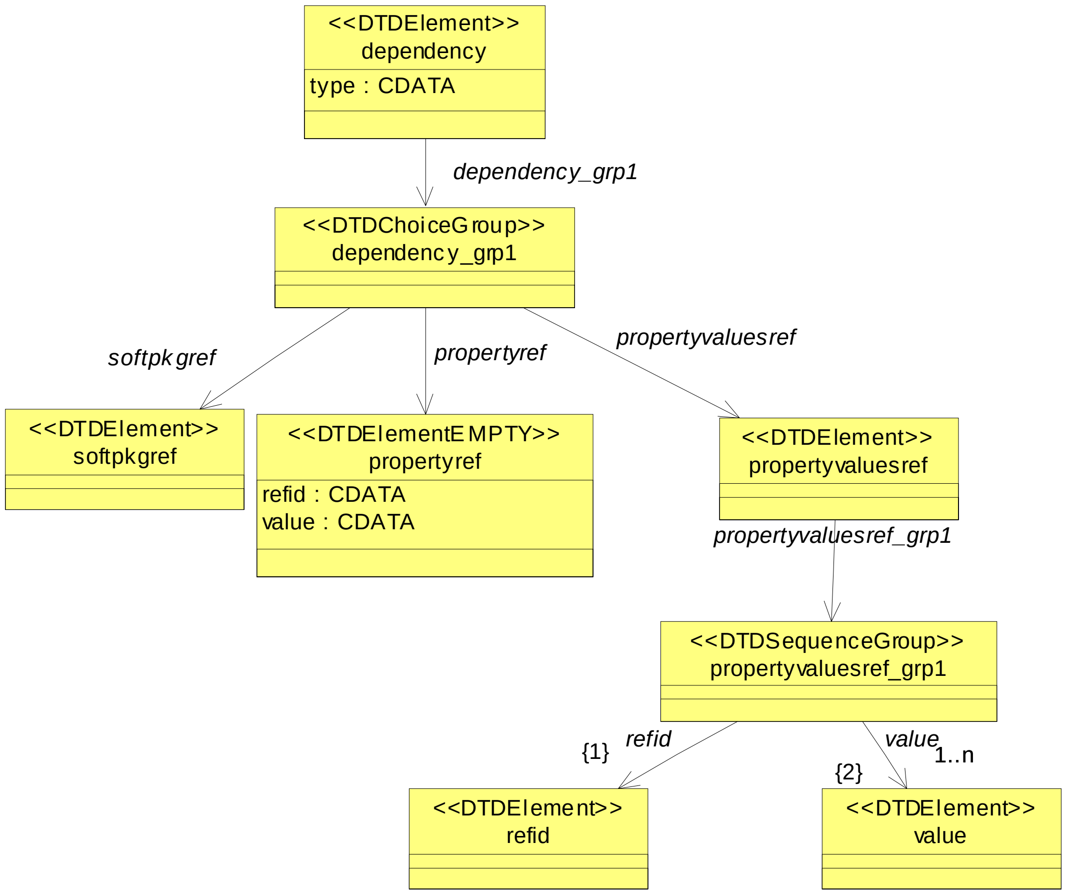


Figure L-91 – *dependency* Element Relationships

```
<!ELEMENT dependency
    ( softpkgref
      | propertyref
      | propertyvaluesref
 )>
 <!ATTLIST dependency
  type   CDATA   #REQUIRED>
```

L.2.1.6.10 *softpkgref*.

The *softpkgref* element (see Figure L-92) refers to a *softpkg* element contained in another Software Package Descriptor file and indicates a file-load dependency on that file. The other file is referenced by the *localfile* element. An optional *implref* element refers to a particular implementation-unique identifier, within the Software Package Descriptor of the other file.

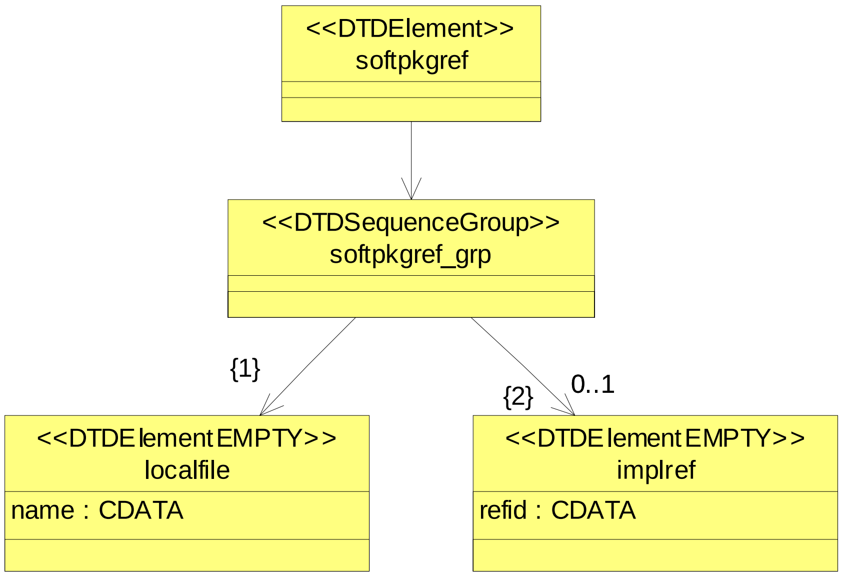


Figure L-92 – *softpkgref* Element Relationships

```
<!ELEMENT softpkgref

      ( localfile

      , implref?

      )>

<!ELEMENT implref EMPTY>

<!ATTLIST implref

   refid CDATA #REQUIRED>
```

L.2.1.6.11*propertyref*.

The *propertyref* element is used to indicate a reference (refid attribute*)* to a ServiceProperty (allocation property), defined in some ServiceComponent’s property file, and the requested value (value attribute) for the Service-Property (allocation property). This deployment requirement is used by the ApplicationFactoryComponent to find the right ServiceComponent or DeviceComponent that can met the requirements as specified by the value attribute.

```
<!ELEMENT propertyref EMPTY>

<!ATTLIST propertyref

      refid CDATA #REQUIRED

      value CDATA #REQUIRED>
```

L.2.1.6.12 runtime.

The *runtime* element specifies a runtime required by a component implementation. An example of the runtime is a Java VM.

```
<!ELEMENT runtime EMPTY>

<!ATTLIST runtime

      name  CDATA#REQUIRED>

      version CDATA #IMPLIED>
```

L.2.1.6.13*propertyvaluesref*.

The *propertyvaluesref* element is used to indicate a reference (refid element*)* to a ServiceProperty (allocation property), defined in some ServiceComponent’s property file, and the requested values (value element) for the ServiceProperty (allocation property). This deployment requirement is used by the ApplicationFactoryComponent to find the right ServiceComponent or DeviceComponent that can met the requirements as specified by the value element.

```
 <!ELEMENT propertyvaluesref

      (refid

       ,value+)>

 <!ELEMENT refid (#PCDATA)>
```

```
    <!ELEMENT value (#PCDATA)>
```

L.2.1.7 *usesdevice*.

The *usesdevice* element describes any "uses" relationships this component has with a ServiceComponent in the system. The *propertyref* or propertyvaluesref element references ServiceProperty(s) (e.g., allocation properties), which indicate the ServiceComponent to be used (characteristics), and/or the capacity(s) needed from the ServiceComponent.

```
      <!ELEMENT  usesdevice
      ( (propertyref
         | propertyvaluesref)+
         )>
      <!ATTLIST usesdevice
            id     ID          #REQUIRED
            type   CDATA       #REQUIRED>
```

L.2.1.7.1 *propertyref*.

SeeL.2.1.7.1 for a definition of the *propertyref* element.

L.2.1.7.2 *propertyvaluesref*.

See L.2.1.6.13 for a definition of the *propertyvaluesref* element.

L.2.1.8 assemblyimplementation

The assemblyimplementation element references a Software Assembly Descriptor (SAD) file for implementation of a component.

```
   <!ELEMENT assemblyimplementation

      (localfile

)>
```

L.3 Device Package Descriptor.

The SWRadio Device Package Descriptor (DPD) is the part of a Device Profile that contains hardware device Registration attributes, which are typically used by a Human Computer Interface application to display information about the device(s) resident in an SWRadio-compliant radio system. DPD information is intended to provide hardware configuration and revision information to a radio operator or to radio maintenance personnel. A DPD may be used to describe a single hardware element residing in a radio or it may be used to describe the complete hardware structure of a radio. In either case, the description of the hardware structure should be consistent with hardware partitioning as described in UML Profile for SWRadio for communicaiton equipment and communicaiton channel.

L.3.1 Device Package.

The *devicepkg* element (see Figure L-93) is the root element of the DPD. The *devicepkg* id attribute uniquely identifies the package. The version attribute specifies the version of the *devicepkg*. The format of the version string is numerical major and minor version numbers separated by commas (e.g., "1,0,0,0"). The name attribute is a user-friendly label for the *devicepkg*.

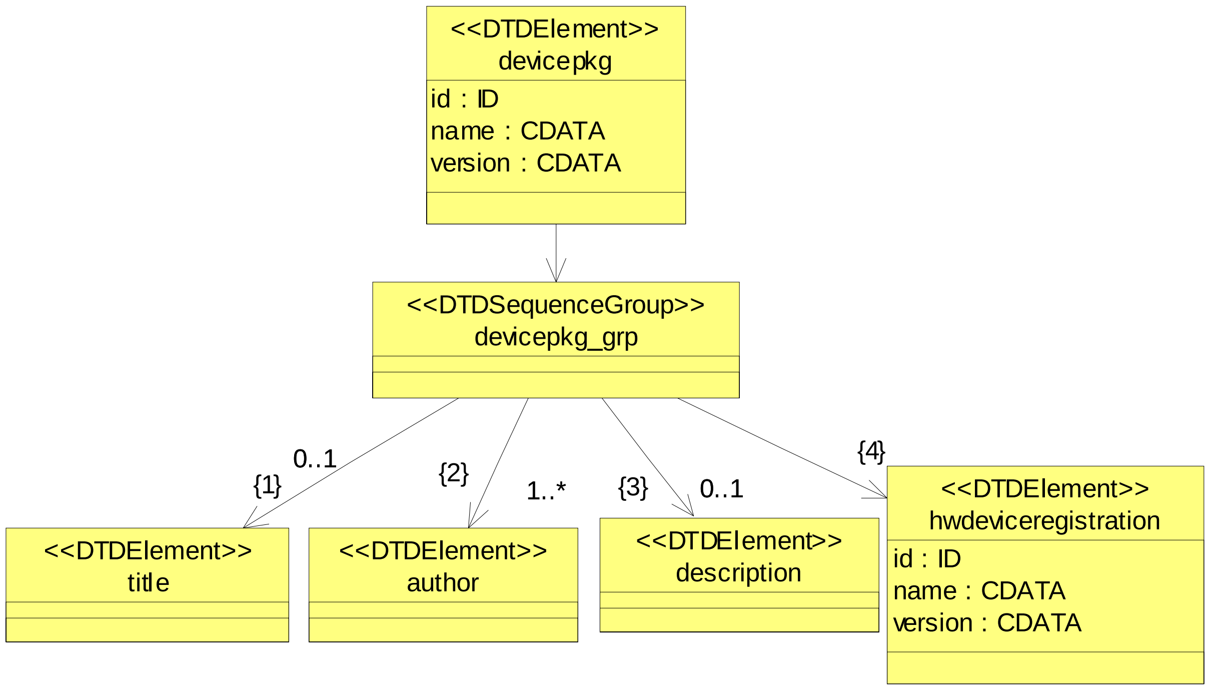


Figure L-93 – *devicepkg* Element Relationships

```
<!ELEMENT devicepkg
```

L.3.1 Device Package.

```
( title?

, author+

, description?

, hwdeviceregistration

)>

<!ATTLIST devicepkg

id    ID #REQUIRED

name  CDATA #REQUIRED

version CDATA #IMPLIED>
```

L.3.1.1 *title*.

The *title* element is used for indicating a title for the hardware device being described by *devicepkg*.

```
<!ELEMENT title (#PCDATA)>
```

L.3.1.2 *author*.

See L.2.1.3 for a definition of the *author* element.

L.3.1.3 *description*.

The *description* element is used to describe any pertinent information about the device implementation that the hardware developer wishes to document within the Device Package.

```
<!ELEMENT description (#PCDATA)>
```

L.3.1.4 *hwdeviceregistration*.

The *hwdeviceregistration* element (see Figure L-94) provides device-specific information for a hardware device. The *hwdeviceregistration* id attribute uniquely identifies the device. The version attribute specifies the version of the *hwdeviceregistration* element. The format of the version string is numerical major and minor version numbers separated by commas (e.g., "1,0,0,0"). The name attribute is a user-friendlylabel for the hardware device being registered. The name attribute is supplied when the id is not user-friendly such as a DCE UUID. At a minimum, the *hwdeviceregistration* element must include a description, the manufacturer, the model number and the device's hardware class(es).

L.3.1 Device Package.

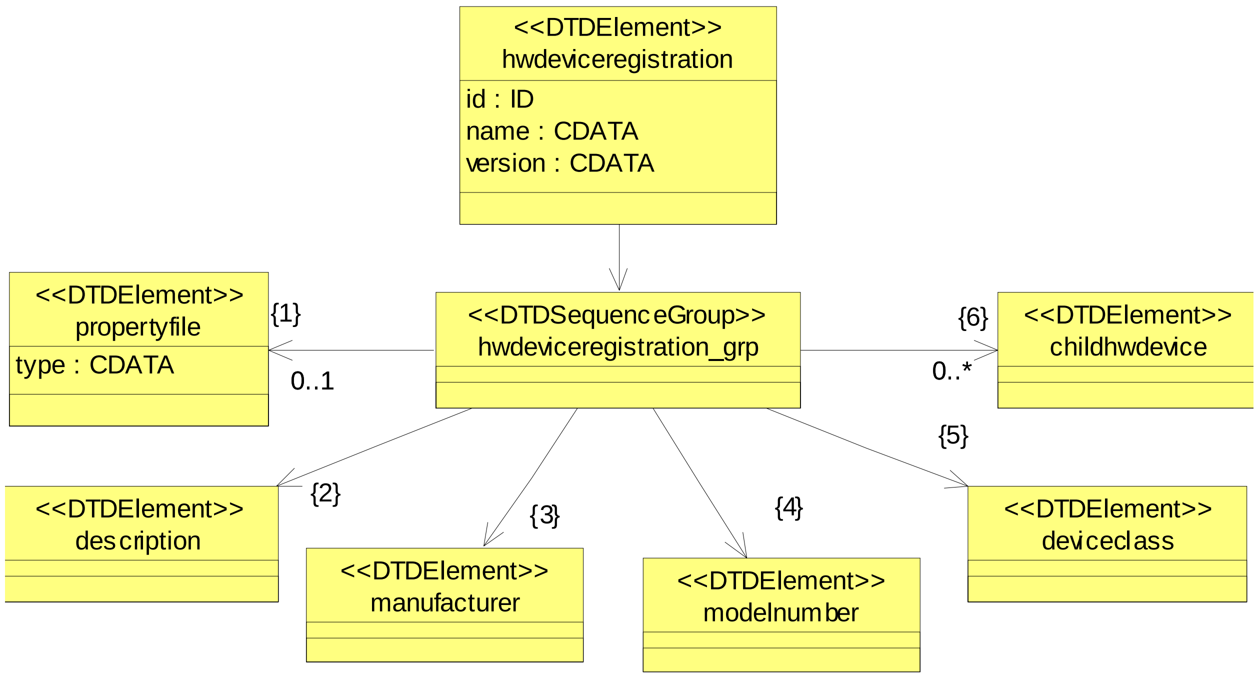


Figure L-94 – *hwdeviceregistration* Element Relationships

```
<!ELEMENT hwdeviceregistration

( propertyfile?

, description

, manufacturer

, modelnumber

, deviceclass

, childhwdevice*

)>

<!ATTLIST hwdeviceregistration

id         ID             #REQUIRED
```

```
   name      CDATA          #REQUIRED

   version   CDATA          #IMPLIED>
```

L.3.1.4.1 ***propertyfile*****.**

The *propertyfile* element is used to indicate the local filename of the property file associated with the *hwdevice-registration* element. The format of a property file is described in the Properties Descriptor (Section D.4).

The intent of the property file is to provide the definition of *properties* elements for the hardware device being deployed and described in the Device Package (*devicepkg*) or *hwdeviceregistration* element.

```
   <!ELEMENT propertyfile

         ( localfile

         )>

      <!ATTLIST propertyfile

         type CDATA #IMPLIED>

         <!ELEMENT localfile EMPTY>

      <!ATTLIST localfile

         name CDATA REQUIRED>
```

L.3.1.4.2 ***description*****.**

See L.2.1.4 for definition of the *description* element.

L.3.1.4.3 ***manufacturer.***

The *manufacturer* element is used to convey the name of manufacturer of the device being installed.

```
   <!ELEMENT manufacturer (#PCDATA)>
```

L.3.1.4.4 ***modelnumber.***

The *modelnumber* element is used to indicate the manufacture's model number, for the device being installed.

```
   <!ELEMENT modelnumber (#PCDATA)>
```

L.3.1.4.5 *deviceclass*.

The *deviceclass* element is used to identify one or more hardware classes that make up the device being installed (as defined in UML Profile for SWRadio communicaiton equipment).

```
      <!ELEMENT deviceclass

         ( class+

         )>

      <!ELEMENT class (#PCDATA)>
```

L.3.1.4.6 *childhwdevice*.

The *childhwdevice* element (see Figure L-95) indicates additional device-specific information for hardware devices that make up the root or parent hardware device registration. An example of *childhwdevice* would be a radio's RF module that has receiver and exciter functions within it. In this case, a CF *Device* representing the RF module itself would be a parent *Device* with its DPD, and the receiver and exciter are child devices to the module. The parent / child relationship indicates that when the RF module is removed from the system, the receiver and exciter devices are also removed.

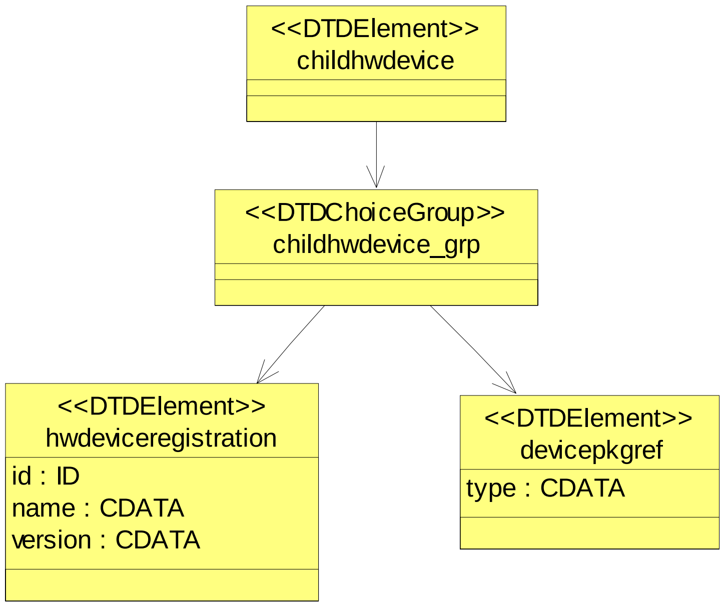


Figure L-95 – *childhwdevice* Element Relationships

```
<!ELEMENT childhwdevice

   ( hwdeviceregistration

   | devicepkgref

   )>
```

L.3.1.4.7 *hwdeviceregistration*.

The *hwdeviceregistration* element provides device-specific information for the child hardware device. See L.3.1.4 for definition of the *hwdeviceregistration* element.

L.3.1.4.8 *devicepkgref*.

The *devicepkgref* element is used to indicate the local filename of a Device Package Descriptor file pointed to by Device Package Descriptor (e.g., a devicepkg within a devicepkg).

```
      <!ELEMENT devicepkgref

   ( localfile
   )>
      <!ATTLIST devicepkgref
         type CDATA #IMPLIED>
```

L.4 Properties Descriptor.

The Properties Descriptor file details component and device attribute settings. For purposes of the SWRadio Property Descriptor files will contain *simple*, *simplesequence*, *test*, *struct* or *structsequence* elements. These elements will be used to describe attributes of a component that will be used for dependency checking. These elements will also be used for SWRadio component values used by a ResourceComponent's *configure*(), *query*(), and *runTest*() operations.

L.4.1 *properties*.

The *properties* element (see Figure L-96) is used to describe property attributes that will be used in the *configure*( ) and *query*( ) operations for SWRadio ResourceComponents and for definition of attributes used for dependency checking. The *properties* element can also used in the TestableObject *runTest*() operation to configure tests and provide test results.

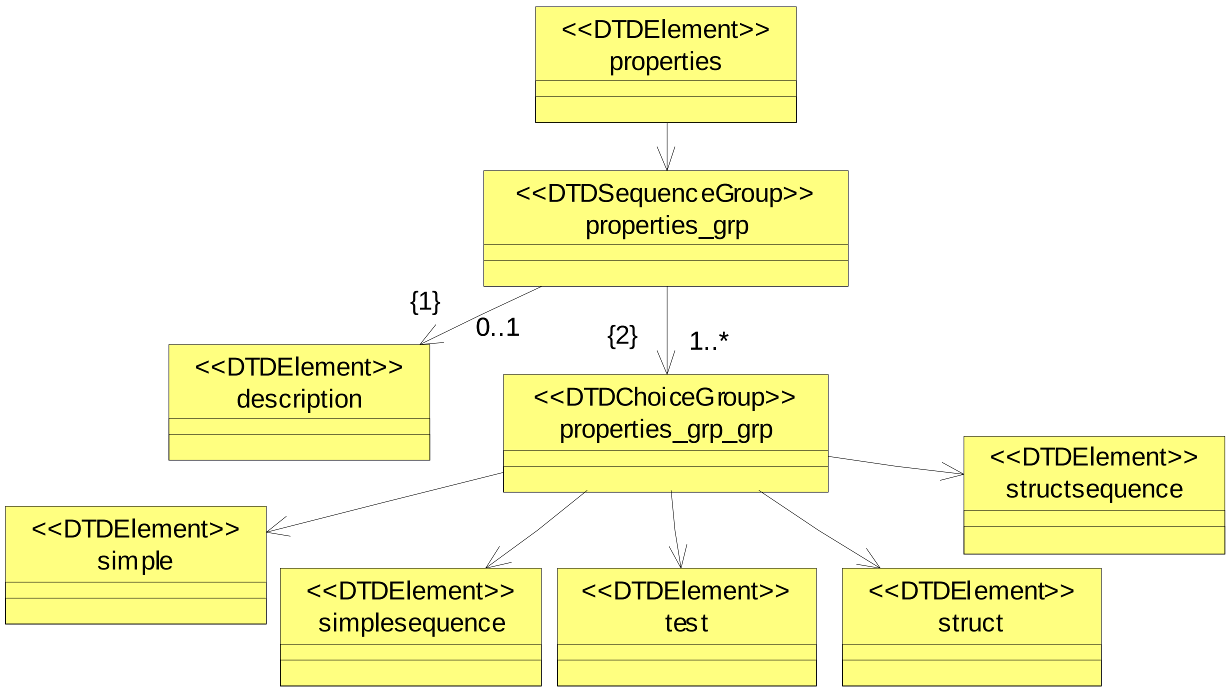

<<DTDElement>> properties
<<DTDSequenceGroup>> properties_grp
{1}
0..1
{2}
1..*
<<DTDElement>> description
<<DTDChoiceGroup>> properties_grp_grp
<<DTDElement>> structsequence
<<DTDElement>> simple
<<DTDElement>> simplesequence
<<DTDElement>> test
<<DTDElement>> struct

Figure L-96 – *properties* Element Relationships

```
<!ELEMENT properties

( description?

   , ( simple
   | simplesequence

   | test

   | struct

   | structsequence

   )+

)>
```

L.4.1.1 *simple.*

The *simple* element (see Figure L-97 and table L-50) provides for the definition of a property which includes a unique id, type, name and mode attributes of the property that will be used in the PropertySet *configure*() and *query*() operations, or for indication of component capabilities. The *simple* element is specifically designed to support id-value pair definitions. A *simple* property id attribute corresponds to the id of the id-value pair. The value and range of a simple property correspond to the value of the id-value pair. The optional *enumerations* element allows for the definition of a label-to-value for a particular property. The mode attribute defines whether the *properties* element is "readonly", "writeonly" or "readwrite". The id attribute is an identifier for the *simple* property element. The id attribute for all other *simple* property elements can be any valid XML ID type. The mode attribute is only meaningful when the type of the *kind* element is "configure". The integerID attribute is used to specify a integer identifier for a simple property, which when used has precedence over the ID attribute.

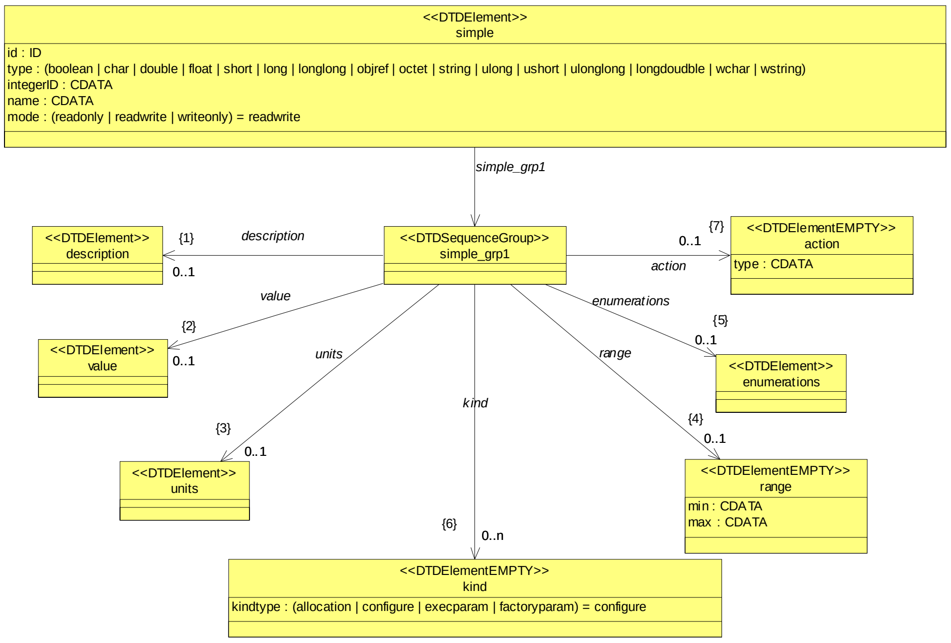


Figure L-97 – *simple* Element Relationships

```
<!ELEMENT simple
      ( description?
      , value?
      , units?
      , range?
      , enumerations?
      , kind*
      , action?
      )>
<!ATTLIST simple
      id ID             #REQUIRED
```

L.4.1 properties.

```
        type  ( boolean | char  | double | float

           | short | long | longlong | objref | octet

           | string | ulong |ushort | ulonglong |longdoudble | wchar | wstring
   )  #REQUIRED

        integerID CDATA  #IMPLIED

        name CDATA     #IMPLIED

        mode( readonly | readwrite | writeonly)  “readwrite”>
```

Simple Properties	**Id**	**Type**	**integerID**	**Name**	**Mode**	**Value**	**Units**	**Range**	**Enum**	**Kind**	**Action**
Configure & Query	+	+	int	*	RW (default)	+	*	*	*	configure	N/A
Configure Only	+	+	int	*	WO	+	*	*	*	configure	N/A
Query Only	+	+	int	*	RO	---	*	*	*	configure	N/A
ServiceComponent’s ServiceProperty (LocallyManaged=false)	+	+	int	*	---	+	*	*	*	allocation	Eq, ne gt, lt, ge, le
ServiceComponent’s ServiceProperty (LocallyManaged=true)	+	+	int	*	---	+	*	*	*	allocation	External(default)

Simple Properties	Id	Type	integerID	Name	Mode	Value	Units	Range	Enum	Kind	Action
ExecuteProperty	+	+	int	*	---	+	*	*	*	execparm	N/A
ResourceFactoryComponent’s ConfigureProperty	+	+	int	*	---	+	*	*	*	factoryparm	N/A
TestProperty’s InputValueProperty or ResultValueProperty	+	+	int	*	---	+	*	*	*	test	N/A

```
Legend:
+     :     Is required and may contain any value, For ID only certain
            characters can be used. Value must conform to type.
*     :     Is optional and may contain any value
int   :     value is integer characters
N/A   :     Not Applicable, will be ignored if used
```

Table L-50 – Simple Elements & Attributes Summary

L.4.1.1.1 *description.*

The *description* element is used to provide a description of the *properties* element that is being defined.

```
<!ELEMENT description (#PCDATA)>
```

L.4.1.1.2 *value.*

The *value* element is used to provide a value setting to the *properties* element.

```
<!ELEMENT value (#PCDATA)>
```

L.4.1.1.3 *units.*

The *units* element describes the intended practical data representation to be used for the *properties* element.

```
<!ELEMENT units (#PCDATA)>
```

L.4.1.1.4 *range.*

The *range* element describes the specific *min* and *max* values that are legal for the *simple* element. The intent of the *range* element is to provide a means to perform range validation. This element is not used by the CF *ApplicationFactory* or CF *Application* implementations.

```
<!ELEMENT range EMPTY

<!ATTLIST range

   min CDATA #REQUIRED

   max CDATA #REQUIRED>
```

L.4.1.1.5 *enumerations.*

The *enumerations* element is used to specify one or more *enumeration* elements.

```
<!ELEMENT enumerations

      ( enumeration+

      )>
```

The *enumeration* element is used to associate a value attribute with a label attribute.. Enumerations are legal for various integer type *properties* elements. An Enumeration value is assigned to a property that implements the CORBA long type. Enumeration values are implied; if not specified by a developer, the initial implied value is 0 and subsequent values are incremented by 1.

```
<!ELEMENT enumeration EMPTY>

<!ATTLIST enumeration

   label CDATA #REQUIRED

   value CDATA #IMPLIED>
```

L.4.1.1.6 *kind.*

The *kind* element's kindtype attribute is used to specify the kind of property. The types of kindtype attributes are:

1. configure, which is used in the *configure*() and *query* () operations of the PropertySet interface. The ApplicationFactoryComponent and DeviceManagerComponent will use the configure kind of properties to build the *Properties* input parameter to the *configure* () operation that is invoked on the Component(s) during application creation. When the mode is readonly, only the *query*() behavior is supported. When the mode is writeonly, only the *configure()* behavior is supported. When the mode is readwrite, both *configure()* and *query()* are supported.
2. test, which is used in the runTest() operation in the TestableObject interface. The test kind of properties will be used as the testValues parameter to the runTest() operation.
3. allocation, which is used in the *allocateCapacity*() and *deallocateCapacity*() operations of the Device interface. The ApplicationFactoryComponent and DeviceManagerComponent will use the allocation kind of properties to build the *capacities* inout parameter to the *allocateCapacity*() operation that is invoked on the *Device*Component(s) during application creation when the simple property action element is external. The ApplicationFactoryComponent and DeviceManagerComponent (not DeviceComponent) manages an Allocation property when the action value is not external. Allocation properties that are external can also be queried using the PropertySet *query()* operation.
4. execparam,. which is used in the *execute* operations of the Device interface. The ApplicationFactoryComponent and DeviceManagerComponent will use the execparam kind of properties to build the Properties input parameter to the *execute*() operation that is invoked on the ExecutableDevicecomponent(s) during component and/or application creation. Only simple elements can be used as execparam types.
5. factoryparam, are properties that are only for the *createResource*() operation of the ResourceFactory interface. The ApplicationFactoryComponent will use the factoryparam type of properties to build the *Properties* input parameter to the *createResource*() operation.

A property can have multiple *kind* elements and the default kindtype is configure.

```
<!ELEMENT kind EMPTY>

<!ATTLIST kind

      kindtype( allocation | configure | test |

       execparam| factoryparam) “configure”>
```

L.4.1.1.7 *action.*

The *action* element is used to define the type of comparison used to compare an SPD property value to a DeviceComponent property value, during the process of checking SPD dependencies. The type attribute, of the *action* element, will determine the type of comparison to be made (e.g., equal, not equal, greater than, etc.). When the action's type is not external then the ApplicationFactoryComponent and DeviceManagerComponent performs the action comparison, not the DeviceComponent. The default value for type is external when not specified.

When the action is “external” then the DeviceComponent is locally managing the allocation propery (e.g., ServiceProperty). The ApplicationFactoryComponent cannot managed these properties, instead it must use the alllocateCapacity operation on a compatible DeviceComponents. For non-external action types, the allocateCapacity operation is not called on a DeviceComponent.

In principle, the *action* element defines the operation executed during the comparison of the allocation property value, provided by an SPD *dependency* element, to the associated allocation property value of a DeviceComponent. The allocation property is on the left side of the action and the dependency value is on the right side of the action. This process allows for the allocation of appropriate objects within the system based on their attributes, as defined by their dependent relationships.

For example, if a *DeviceComponent’s* properties file defines a DeviceKind allocation property whose *action* element is set to "equal", then at the time of dependency checking a valid DeviceKind property is checked for equality. If a software component implementation is dependent on a DeviceKind property with its value set to "NarrowBand", then the component's SPD dependency *propertyref* element will reference the id of the DeviceKind allocation property with a value of "NarrowBand". At the time of dependency checking, the ApplicationFactoryComponent will check DeviceComponent(s) whose *properties kind* element is set to “allocation” and property id is DeviceKind for equality against a "NarrowBand" value.

```
<!ELEMENT action EMPTY>

<ATTLIST action

      type  CDATA #REQUIRED>
```

L.4.1.1.8 *simplesequence*.

The *simplesequence* element (see Figure L-98) is used to specify a list of *properties* with the same characteristics (e.g., type, range, units, etc.). The *simplesequence* element definition is similar to the *simple* element definition except that it has a list of values instead of one value. The *simplesequence* element maps to the basic primitive sequence types and CF PortTypes CORBA modules, defined in UML Profile for SWRadio, based upon the type attribute.

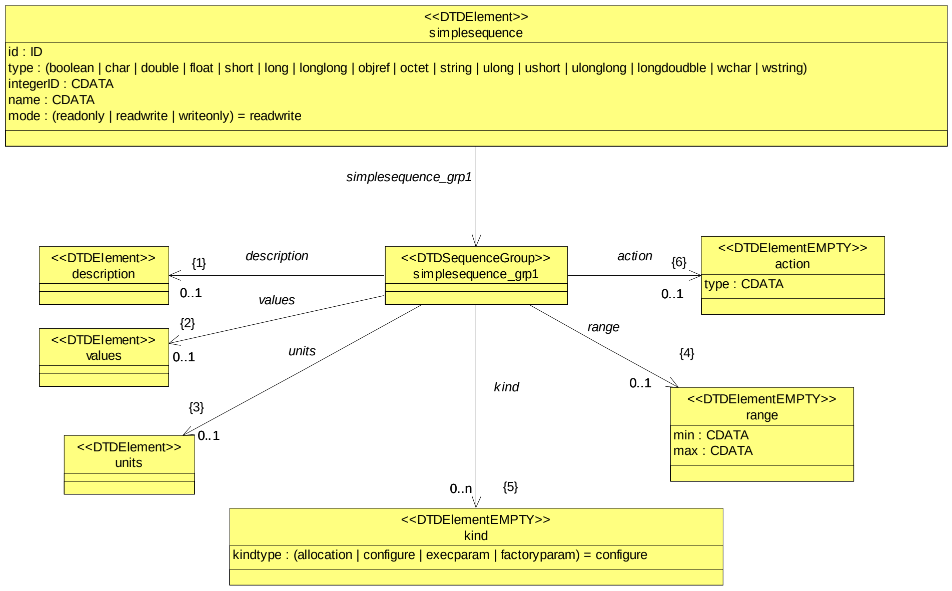


Figure L-98 – *simplesequence* Element Relationships

```
<!ELEMENT simplesequence
        ( description?
        , values?
        , units?
        , range?
        , kind*
        , action?
        )>
<!ATTLIST simplesequence
        id ID                   #REQUIRED
        type( boolean | char  | double | float
           | short | long | longlong | objref | octet
```

```
         | string | ulong |ushort | ulonglong |longdoudble | wchar | wstring
)     #REQUIRED

         integerID CDATA      #IMPLIED

         name    CDATA        #IMPLIED

mode(readonly | readwrite | writeonly) “readwrite”>

<!ELEMENT values

      (value+

)>
```

L.4.1.2 *test.*

The *test* element (see Figure L-99) is used to specify a list of test properties for executing the TestableObject *runTest*() operation to perform a component specific test. This definition contains *inputvalue* and *resultvalue* elements and it has a testid attribute for grouping test properties to a specific test. *Inputvalues* are used to configure the test to be performed (e.g., frequency and RF power output level). When the test has completed, *resultvalues* contain the results of the testing (e.g., Pass or a fault code/message).

L.4.1 properties.

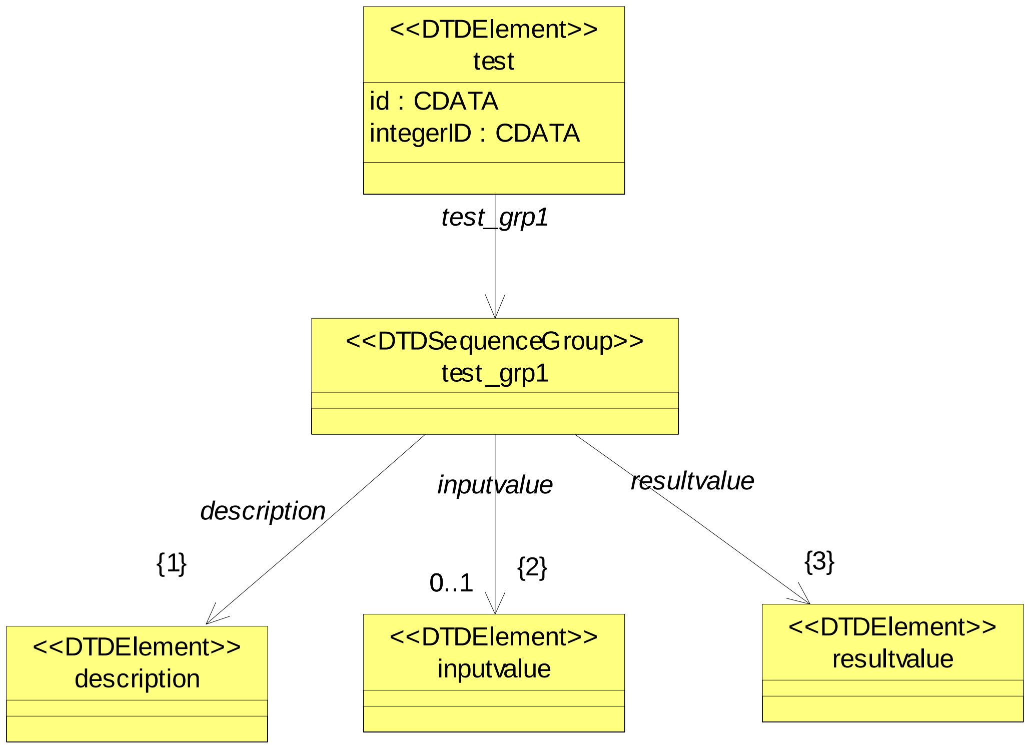


Figure L-99 – *test* Element Relationships

```
<!ELEMENT test

      ( description

      , inputvalue?

      , resultvalue

      )>

<!ATTLIST test

      id CDATA #REQUIRED

      integerID CDATA      #IMPLIED

>
```

L.4.1 properties.

L.4.1.2.1 *inputvalue*.

The *inputvalue* element is used to provide test configuration properties. The Simple properties it contains must be of kind "test".

```
<!ELEMENT inputvalue

      ( simple+

   )>
```

L.4.1.2.2 *resultvalue*.

The *resultvalue* element is used to provide test result properties. The Simple properties it contains must be of kind "test".

```
<!ELEMENT resultvalue

      ( simple+

)>
```

L.4.1.3 *struct*.

The *struct* element (see Figure L-100) is used to group properties with different characteristics (i.e., similar to a structure or record entry). Each item in the *struct* element can be a different simple type (e.g., short, long, etc.). The *struct* element corresponds to the Properties type where each *struct* item (ID, value) corresponds to a *properties* element list item. The *properties* element list size is based on the number of *struct* items.

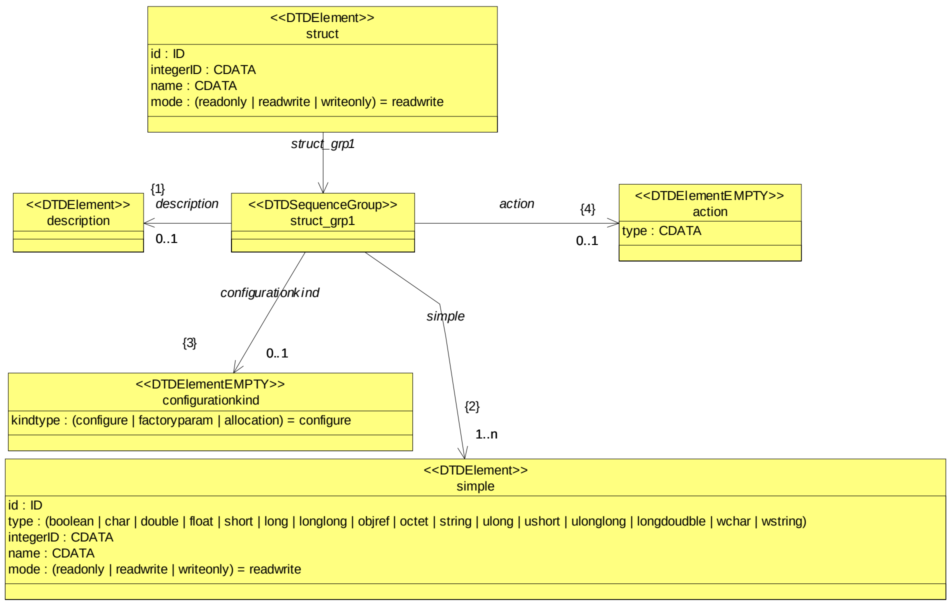

<<DTDElement>>
struct
id : ID
integerID : CDATA
name : CDATA
mode : (readonly | readwrite | writeonly) = readwrite
struct_grp1
<<DTDElement>>
description
{1}
description
0..1
<<DTDSequenceGroup>>
struct_grp1
action
{4}
0..1
<<DTDElementEMPTY>>
action
type : CDATA
configurationkind
simple
{3}
0..1
<<DTDElementEMPTY>>
configurationkind
kindtype : (configure | factoryparam | allocation) = configure
{2}
1..n
<<DTDElement>>
simple
id : ID
type : (boolean | char | double | float | short | long | longlong | objref | octet | string | ulong | ushort | ulonglong | longdoudble | wchar | wstring)
integerID : CDATA
name : CDATA
mode : (readonly | readwrite | writeonly) = readwrite

Figure L-100 – *struct* Element Relationships

```
<!ELEMENT struct

      ( description?

      , simple+

      , configurationkind?

      , action?

)>

<!ATTLIST struct

      id       ID      #REQUIRED

      integerID CDATA      #IMPLIED

      name CDATA   #IMPLIED

mode(readonly | readwrite | writeonly)"readwrite">
```

L.4.1.3.1 *configurationkind.*

The *configurationkind* element's kindtype attribute is used to specify the kind of property. The kindtypes are:

1. configure, which is used in the *configure*() and *query*() operations of the SWRadio Resource interface. The ApplicationFactoryComponent and DeviceManagerComponent will use the configure kind of properties to build the Properties input parameter to the *configure*() operation that is invoked on the ResourceComponent(s) during application creation. When the mode is readonly, only the *query* behavior is supported. When the mode is writeonly, only the *configure* behavior is supported. When the mode is readwrite, both *configure* and *query* are supported.

2. factoryparam, which is used in the *createResource* operations of the *ResourceFactory* interface. The ApplicationFactoryComponent will use the factoryparam kind of properties to build the Properties input parameter to the *createResource*() operation. A property can have multiple *configurationkind* elements and their default kindtype is "configure".

3. allocation, which is used in the *allocateCapacity*() and *deallocateCapacity*() operations of the Device interface. The ApplicationFactoryComponent and DeviceManagerComponent will use the allocation kind of properties to build the *capacities* inout parameter to the *allocateCapacity*() operation that is invoked on the *Device*Component(s) during application creation when the simple property action element is external. The ApplicationFactoryComponent and DeviceManagerComponent (not DeviceComponent) manages an Allocation property when the action value is not external. Allocation properties that are external can also be queried using the PropertySet *query()* operation.

```
<!ELEMENT configurationkind EMPTY>

<!ATTLIST configurationkind

kindtype(configure | factoryparam | allocation) “configure”>
```

L.4.1.4 *structsequence.*

The *structsequence* element (see Figure L-101) is used to specify a list of properties with the same *struct* characteristics. The *structsequence* element maps to a *properties* element having the Properties type. Each item in the Properties type will be the same *struct* definition as referenced by the structrefid attribute.

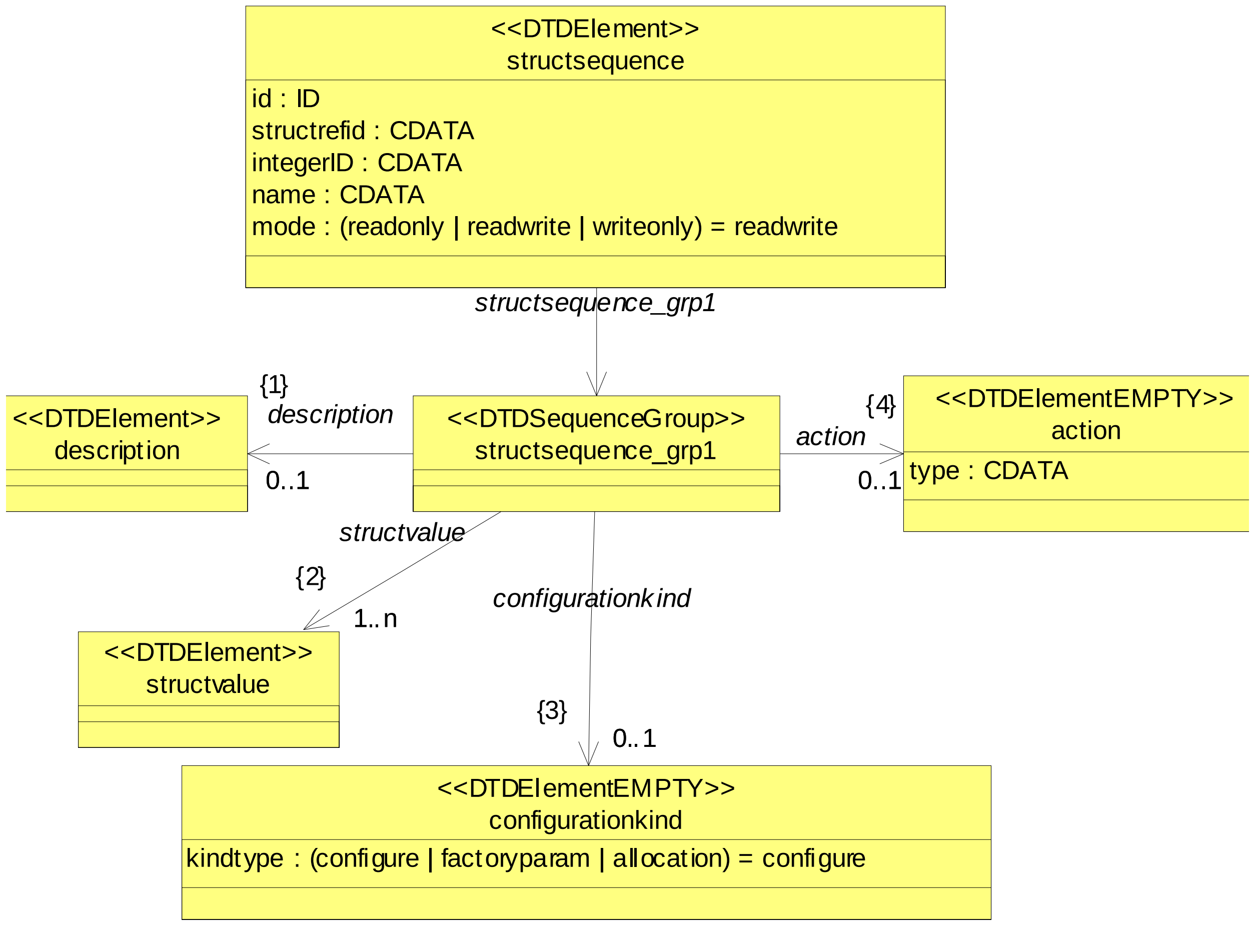


Figure L-101 – *structsequence* Element Relationships

```
<!ELEMENT structsequence

      ( description?

      , structvalue+

      , configurationkind?

      , action?

      )>
```

L.4.1 properties.

```
<!ATTLIST structsequence
      id ID #REQUIRED
      structrefid CDATA #REQUIRED
      integerID CDATA      #IMPLIED
      name CDATA #IMPLIED
mode  (readonly | readwrite | writeonly) "readwrite">

<!ELEMENT structvalue
      (simpleref+
)>
<!ELEMENT simpleref EMPTY>
<!ATTLIST simpleref
      refid CDATA #REQUIRED
      value CDATA #REQUIRED>
```

L.5 Software Component Descriptor.

The Software Component Descriptor (SCD) defines elements necessary for describing the ports, interfaces, and properties for a component definition.

L.5.1 *softwarecomponent.*

The *softwarecomponent* element (see Figure L-102) is the root element of the software component descriptor file. For use within the SWRadio the sub-elements that are supported include:

- *corbaversion* – indicates which version of CORBA the component is developed for.
- *componentrepid* – is the repository id of the component
- *componenttype* – identifies the type of software component object
- *componentfeatures* – provides the supported message ports for the component
- *interface* – describes the component unique id and name for supported interfaces.

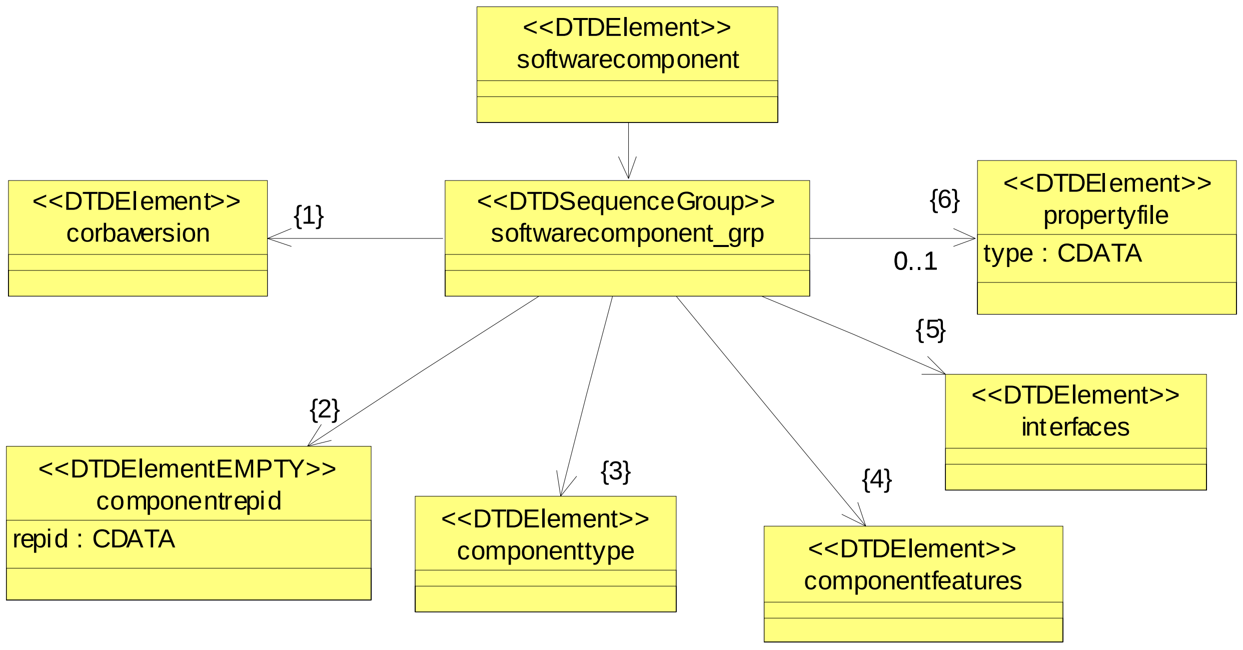


Figure L-102 – *softwarecomponent* Element Relationships

```
<!ELEMENT softwarecomponent
```

```
        ( corbaversion
        , componentrepid
        , componenttype
        , componentfeatures
        , interfaces
        , propertyfile?
        )>
```

L.5.1.1 *corbaversion.*

The *corbaversion* element is intended to indicate the version of CORBA that the delivered component supports.

```
<!ELEMENT corbaversion (#PCDATA)>
```

L.5.1.2 *componentrepid.*

The *componentrepid* uniquely identifies the interface that the component is implementing. The *componentrepid* may be referred to by the *componentfeatures* element. The *componentrepid* is derived from interfaces such as the *Resource*, *Device*, or *ResourceFactory.*

```
<!ELEMENT componentrepid EMPTY>
<!ATTLIST componentrepid
      repid CDATA #REQUIRED>
```

L.5.1.3 *componenttype*.

The *componenttype* describes properties of the component. For SWRadio components, the component types include elements such as service, resource, device, resourcefactory, domainmanager, log, filesystem, filemanager, devicemanager, namingservice and eventservice.

```
<!ELEMENT componenttype (#PCDATA)>
```

L.5.1.4 *componentfeatures.*

The *componentfeatures* element (see Figure L-103) is used to describe a component with respect to the components that it inherits from, the interfaces the component supports, and its provides and uses *ports*. The component interface is usually Resource, ResourceFactory, or service interface such as Device, LoadableDevice, and ExecutableDevice. If a component extends the Resource or Device interfaces then all the inherited interfaces (e.g., Resource) are depicted as supportsinterface elements.

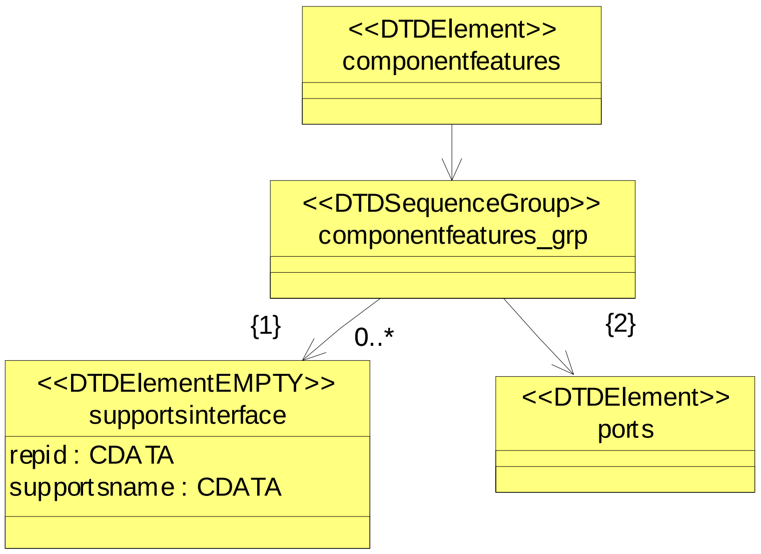


Figure L-103 – *componentfeatures* Element Relationships

```
<!ELEMENT componentfeatures

      ( supportsinterface*

, ports

      )>
```

L.5.1.4.1 *supportsinterface.*

The *supportsinterface* element is used to identify an IDL interface that the component supports. These interfaces are distinct interfaces that were inherited by the component's specific interface. One can widen the component's interface to be a *supportsinterface*. The repid is used to refer to the *interface* element (see section L.5.1.5).

```
<!ELEMENT supportsinterface EMPTY>
```

```
<!ATTLIST supportsinterface

      repid    CDATA #REQUIRED

      supportsname CDATA #REQUIRED>
```

L.5.1.4.2 *ports.*

The *ports* element (see Figure L-104) describes what interfaces a component provides and uses (or requires). The *provides* elements are interfaces that are not part of a component's interface but are independent interfaces known as facets (in CORBA Components terminology) (i.e. a *provides* port at the end of a path, like I/O Device or Modem Device). The *uses* element desribes the interfaces needed by a component. These uses ports are connected to a *provides* or *supportinterfaces* interface. Any number of *uses* and *provides* elements can be given in any order. Each *ports* element has a name and references an interface by repid (see section L.5.1.5). The port names are used in the Software Assembly Descriptor to connect ports together. A *ports* element also has an optional *porttype* element that allows for identification of port classification. Values for *porttype* include "data", "control", "responses", and "test". If a *porttype* is not given then "control" is assumed.

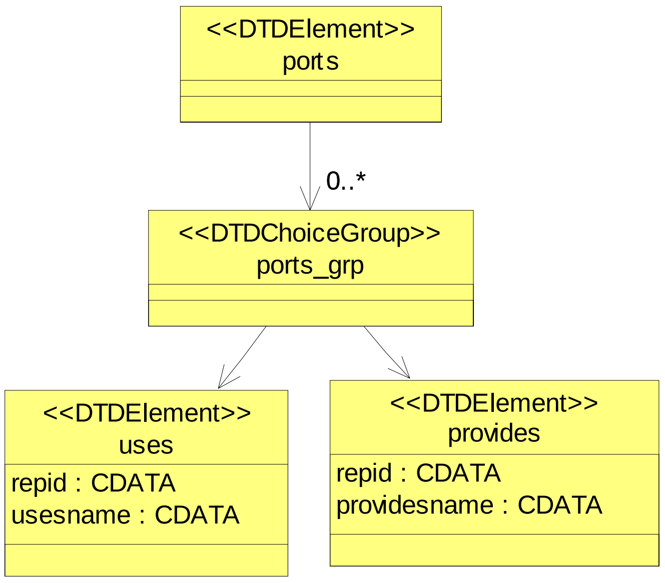


Figure L-104 – *ports* Element Relationships

```
<!ELEMENT ports

      ( provides

      | uses
      )*>


<!ELEMENT provides

      ( porttype*

)>

<!ATTLIST provides

      repid    CDATA #REQUIRED

      providesname CDATA #REQUIRED>


<!ELEMENT uses

( porttype*

)>

<!ATTLIST uses

   repid CDATA #REQUIRED

   usesname CDATA #REQUIRED>


<!ELEMENT porttype EMPTY>

<!ATTLIST porttype

type ( data | control | responses | test ) #REQUIRED>
```

L.5.1.5 *interfaces*.

The *interfaces* element is made up of one to many *interface* elements.

```
<!ELEMENT interfaces

      ( interface+

)>
```

The *interface* element describes an interface that the component, either directly or through inheritance, provides, uses, or supports. The name attribute is the character-based non-qualified name of the interface. The repid attribute is the unique repository id of the interface, which has formats specified in the CORBA specification. The repid is also used to reference an *interface* element elsewhere in the SCD, for example from the *inheritsinterface* element.

```
<!ELEMENT interface

      ( inheritsinterface*

)>

<!ATTLIST interface

   repid CDATA #REQUIRED

   name CDATA #REQUIRED>

<!ELEMENT inheritsinterface EMPTY>

<!ATTLIST inheritsinterface

      repid CDATA #REQUIRED
```

L.5.1.6 *propertyfile*.

Refer to section L.2.1.1 *propertyfile* for definition of *propertyfile*. The properties defined at the SCD are the definition of properties supported by all implementations, and be managed by the PropertySet interface as described in the UML Profile for SWRadio.

L.6 Software Assembly Descriptor.

This section describes the XML elements of the Software Assembly Descriptor (SAD) XML file; the *softwareassembly* element (see Figure L-105). The SAD is based on the CORBA Components Specification Component Assembly Descriptor. The intent of the software assembly is to provide the means of describing the assembled functional application and the interconnection characteristics of the SWRadio components within that application. The component assembly provides four basic types of application information for Radio Management. The first is partitioning information that indicates special requirements for collocation of components, the second is the assembly controller for the software assembly, the third is connection information for the various components that make up the application assembly, and the fourth is the visible ports for the application assembly.

The installation of an application into the system involves the installation of a SAD file. The SAD file references component's SPD files to obtain deployment information for these components.The *softwareassembly* element's id attribute uniquely identifies the assembly. The *softwareassembly* element's name attribute is the user-friendly name for the ApplicationFactoryComponent name attribute. The name attribute is supplied when the id is not user-friendly such as a DCE UUID. The softwareassembly element's version attribute is the version of the application.

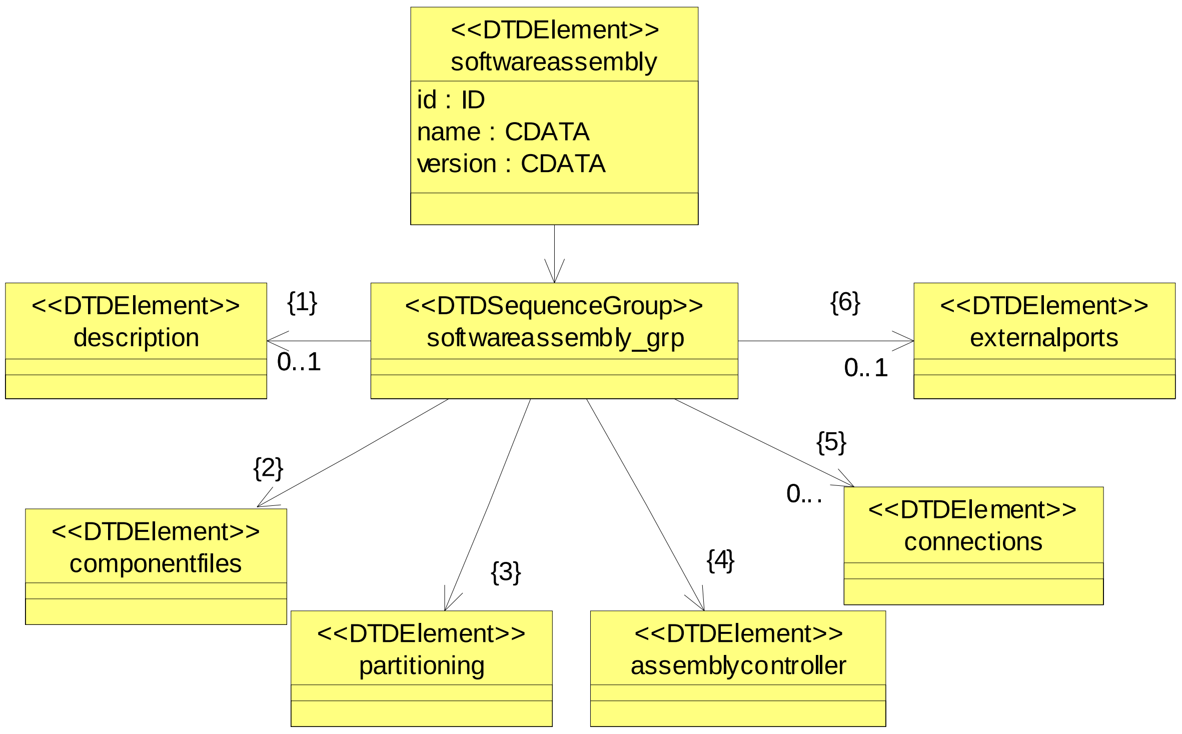


Figure L-105 – *softwareassembly* Element Relationships

```
<!ELEMENT softwareassembly

      ( description?

      , componentfiles

      , partitioning

      , assemblycontroller

      , connections?

      , externalports?

      )>

<!ATTLIST softwareassembly

      id ID #REQUIRED

      name CDATA #IMPLIED

      version CDATA #IMPLIED>
```

L.6.1 *description.*

The *description* element of the component assembly may be used to describe any information the developer would like to indicate about the assembly.

```
<!ELEMENT description (#PCDATA)>
```

L.6.2 *componentfiles.*

The *componentfiles* element is used to indicate that an assembly is made up of 1..n component files. The *componentfile* element contains a reference to a local file, which is a Software Package Descriptor file (see section L.2).

```
<!ELEMENT componentfiles

      ( componentfile+

      )>
```

L.6.2.1 *componentfile.*

The *componentfile* element is a reference to a local file. See section L.2.1.1.1 for the definition of the *localfile* element. The type attribute is “Software Package Descriptor.

```
<!ELEMENT componentfile

      ( localfile

      )>

<!ATTLIST componentfile

      id ID #REQUIRED

      type CDATA #IMPLIED>
```

L.6.3 *partitioning.*

A component *partitioning* element (see Figure L-106) specifies a deployment pattern of components and their components-to-hosts relationships. A component instantiation is captured inside a *componentplacement* element. The *hostcollocation* element allows the components to be placed on a common device. When the *component-placement* is by itself and not inside a *hostcollocation*, it then has no collocation constraints.

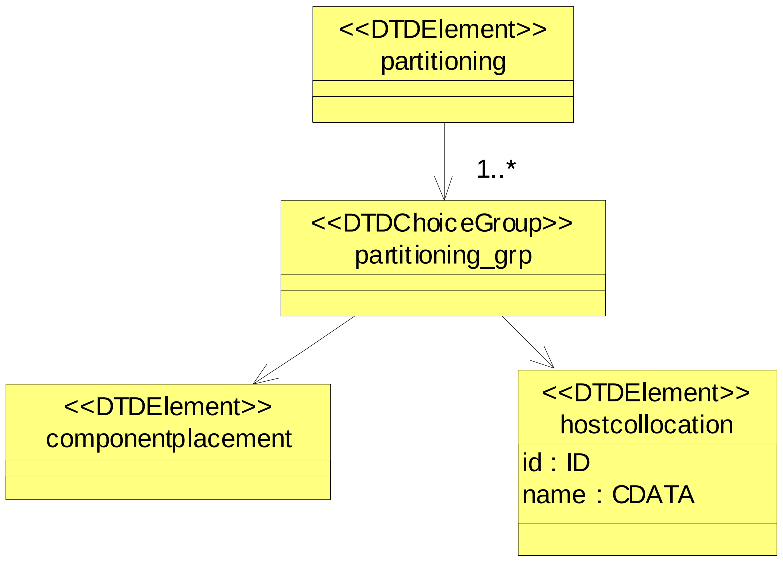


Figure L-106 – *partitioning* Element Relationships

L.6.3 partitioning.

```
<!ELEMENT partitioning

( componentplacement

       |  hostcollocation

       )+>
```

L.6.3.1 *componentplacement.*

The *componentplacement* element (see Figure L-107) defines a particular deployment of a component. The component can be deployed either directly or by using a ResourceFactoryComponent.

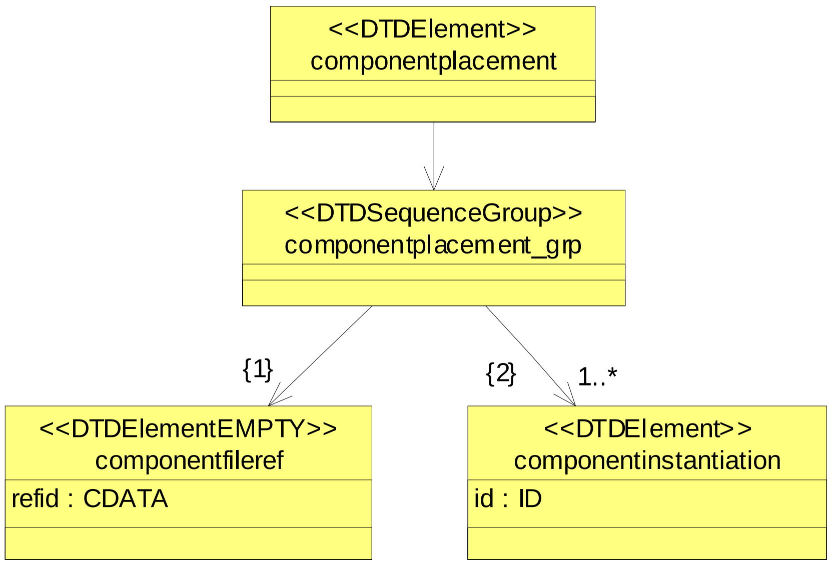


Figure L-107 – *componentplacement* Element Relationships

```
<!ELEMENT componentplacement

       ( componentfileref

       , componentinstantiation+

       )>
```

L.6.3.1.1 *componentfileref.*

The *componentfileref* element is used to reference a particular Software PackageDescriptor file. The *componentfileref* element's refid attribute corresponds to the *componentfile* element's id attribute.

```
<!ELEMENT componentfileref  EMPTY>

<!ATTLIST componentfileref

      refid CDATA #REQUIRED>
```

L.6.3.1.2 *componentinstantiation.*

The *componentinstantiation* element (see Figure L-108) is intended to describe a particular instantiation of a component relative to a *componentplacement* element. The *componentinstantiation's* id attribute uniquely identifies the component. The *componentinstantiation* element's id may be referenced by the *usesport* and *providesport* elements within the SAD file. It is the component name for the instantiation not the application name.

The optional *componentproperties* element (see Figure L-109) is a list of configure, factoryparam, and/or execparam properties values that are used in creating the component or for the initial configuration of the component. The "configure" or "factoryparm" kinds of property definitions as stated in the corresponding SCD. The "execparm" kind of property definitions as stated in the corresponding SPD.

The following sources will be searched in the given precedence order for initial values for "configure" kind of properties, whose modes are "readwrite" or "writeonly":

1. The *componentproperties* element of the *componentinstantiation* element in SAD

The following sources will be searched in the given precedence order for initial values for the "execparam" kind of properties:

1. The componentproperties element of the componentinstantiation element in SAD or the *componentinstantiation* element's *findcomponent* element's *componentresourcefactoryref* element's *resourcefactoryproperties* element in the SAD

The following sources will be searched initial values for the "factoryparam" kind of properties in the given precedence order:

1. The *componentinstantiation* element's *findcomponent* element's *componentresourcefactoryref* element's *resourcefactoryproperties* element in the SAD

The optional *findcomponent* element (see Figure L-110) is used to obtain the object reference for the component instance. The two sources for obtaining an object reference are:

1. The *componentresourcefactoryref* element, which refers to a particular ResourceFactoryComponent *componentinstantiation* element found in the SAD, which is used to obtain a ResourceComponent instance for this *componentinstantiation* element. The refid attribute refers to a unique

componentinstantiation id attribute. The *componentresourcefactoryref* element contains an optional *resourcefactoryproperties* element (see Figure L-111), which specifies the properties "qualifiers", for the ResourceFactoryComponent *create* call.

2. The optional *findcomponent* element should be specified except when there is no object reference for the component instance (e.g., DSP code). The CORBA Naming Service, which is used to find the component's object reference. The name specified in the *namingservice* element is a partial name that is used by the ApplicationFactoryComponent to form the complete context name.

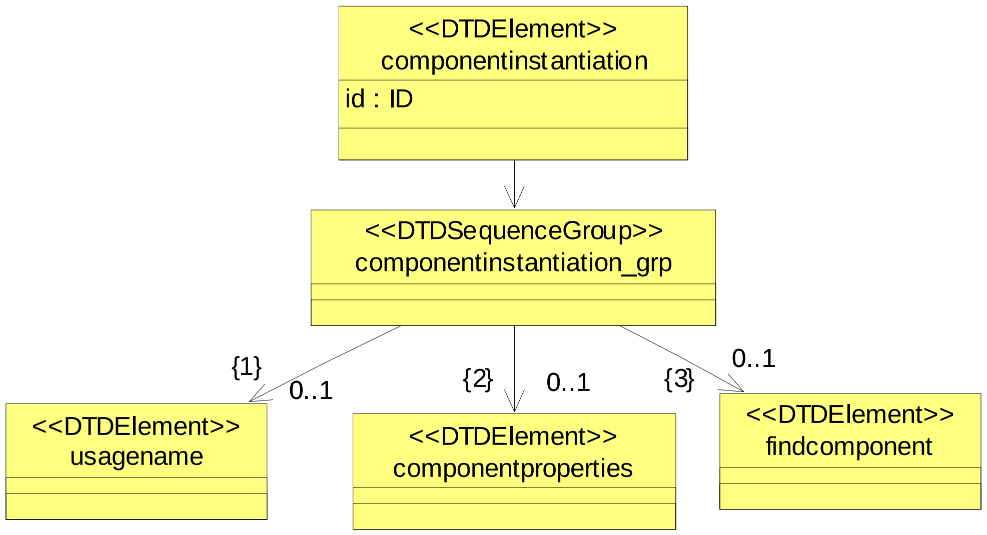


Figure L-108 – *componentinstantiation* Element Relationships

```
<!ELEMENT componentinstantiation

      ( usagename?

      , componentproperties?

      , findcomponent?

      )>

<!ATTLIST componentinstantiation

      id ID #REQUIRED>


<!ELEMENT usagename (#PCDATA)>
```

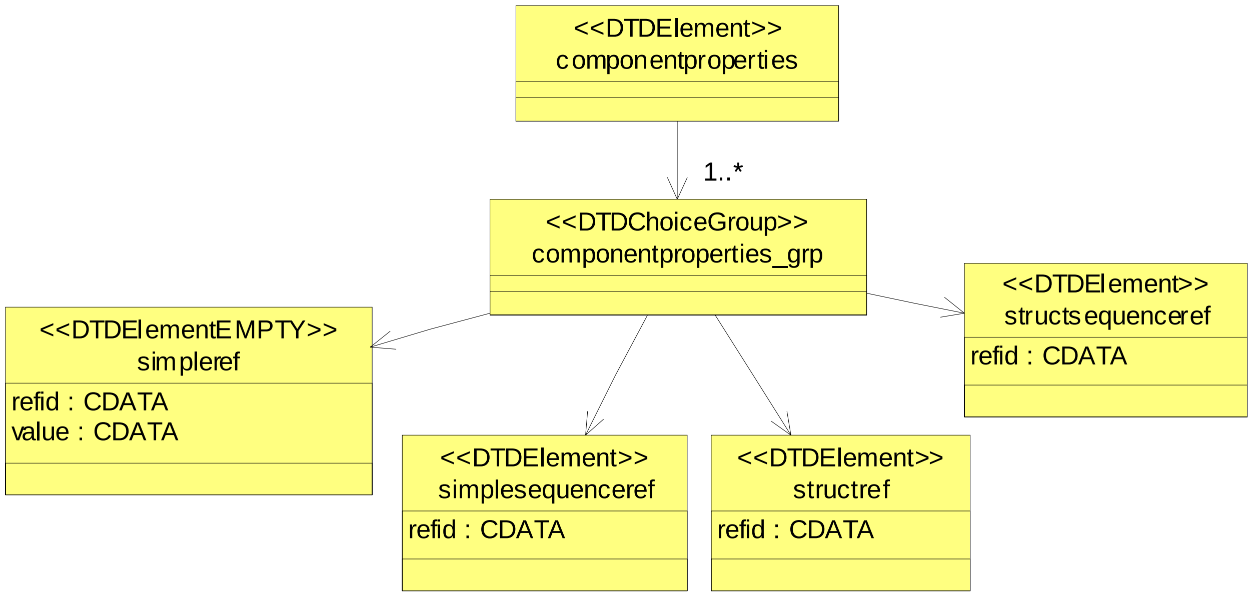


Figure L-109 – *componentproperties* Element Relationships

```
<!ELEMENT componentproperties
      ( simpleref
      | simplesequenceref
      | structref
      | structsequenceref
      )+ >
```

L.6.3 partitioning.

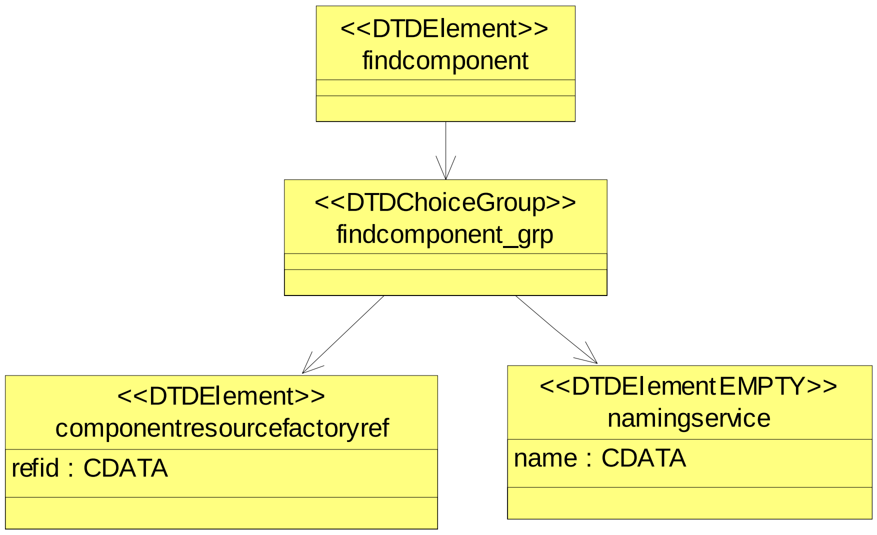


Figure L-110 – *findcomponent* Element Relationships

```
<!ELEMENT findcomponent

      ( componentresourcefactoryref

      | namingservice

      )>

<!ELEMENT componentresourcefactoryref

      ( resourcefactoryproperties?

      )>

<!ATTLIST componentresourcefactoryref

      refid CDATA #REQUIRED>
```

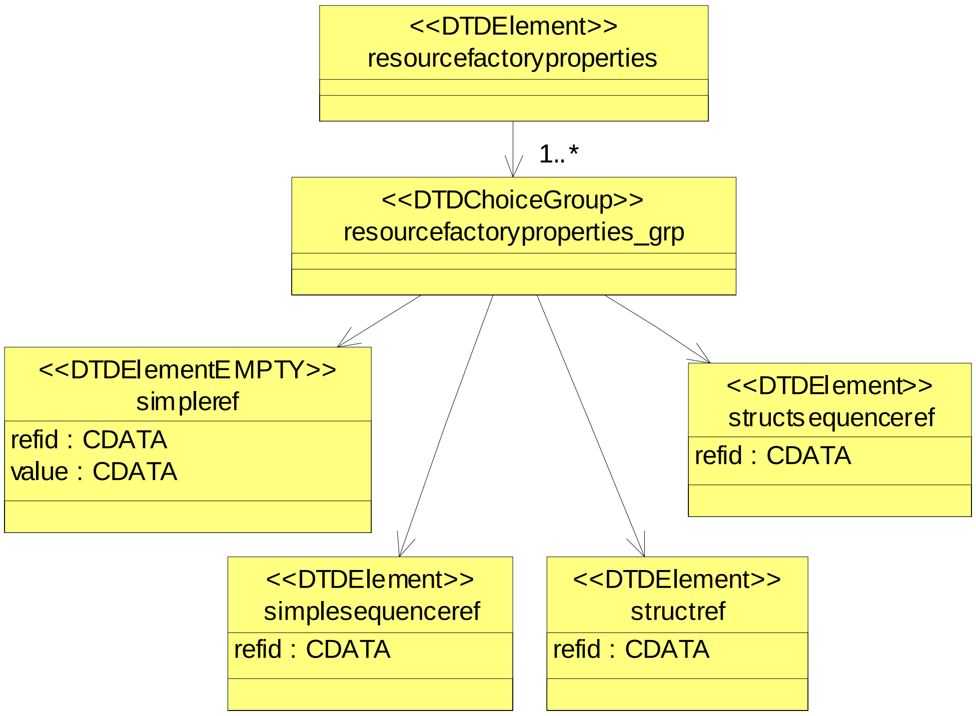


Figure L-111 – *resourcefactoryproperties* Element Relationships

```
<!ELEMENT resourcefactoryproperties

      ( simpleref

      | simplesequenceref

      | structref

      | structsequenceref

      )+ >

<!ELEMENT simpleref EMPTY>

<!ATTLIST simpleref

      refid CDATA #REQUIRED

      value CDATA #REQUIRED>
```

```
<!ELEMENT simplesequenceref

      (values

      )>

<!ATTLIST simplesequenceref

      refid CDATA #REQUIRED>

<!ELEMENT structref

         (simpleref+
         )>
   <!ATTLIST structref
         refid CDATA #REQUIRED>

   <!ELEMENT structsequenceref
         ( structvalue+
   )>
   <!ATTLIST structsequenceref
         refid CDATA #REQUIRED>

   <!ELEMENT structvalue
         (simpleref+
         )>


<!ELEMENT values

      ( value+

      )>

<!ELEMENT value (#PCDATA)>
```

L.6.3.2 *hostcollocation.*

The *hostcollocation* element specifies a group of component instances that are to be deployed together on a single host. For purposes of the SWRadio, the *componentplacement* element will be used to describe the 1...n components that will be collocated on the same host platform. Within the SWRadio specification, a host platform will be interpreted as a single device. The id and name attributes are optional but may be used to uniquely identify a set of collocated components within a SAD file.

```
<!ELEMENT hostcollocation
```

```
        ( componentplacement
        )+>
<!ATTLIST hostcollocation
        id ID #IMPLIED
        name CDATA #IMPLIED>
```

L.6.3.2.1 *componentplacement*.

See *componentplacement,* section L.6.3.1.

L.6.4 *assemblycontroller.*

The *assemblycontroller* element indicates the component that is the main ResourceComponent controller for the assembly. The ApplicationManager component delegates its *Resource configure*(), *query*(), *start(), stop()*, and *runTest()* operations to the *ResourceComponent's* Assembly Controller component.

```
<!ELEMENT assemblycontroller
        ( componentinstantiationref
        )>
```

L.6.5 *connections.*

The *connections* element is a child element of the *softwareassembly* element. The *connections* element is intended to provide the connection map between components in the assembly.

```
<!ELEMENT connections
( connectinterface*
)>
```

L.6.5 connections.

L.6.5.1 *connectinterface.*

The *connectinterface* element (see Figure L-112) is used when application components are being assembled to describe connections between their port interfaces. The *connectinterface* element consists of a *usesport* element and a *providesport, componentsupportedinterface, or findby* element. These elements are intended to connect two compatible components.

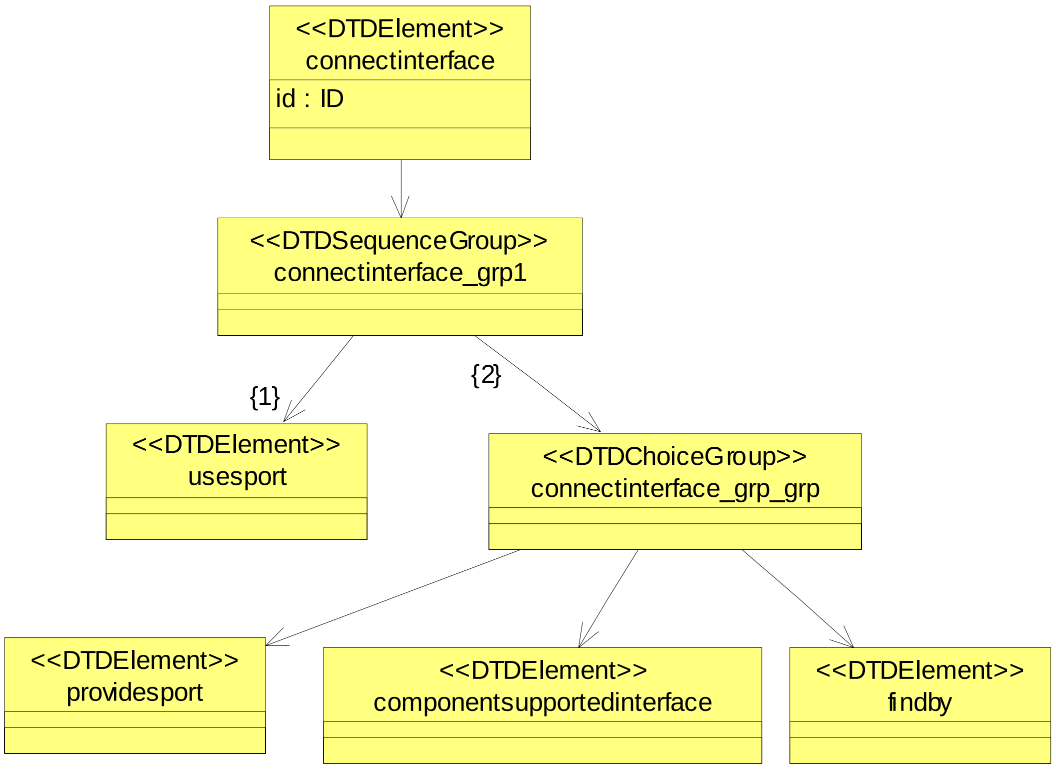


Figure L-112 – *connectinterface* Element Relationships

```
<!ELEMENT connectinterface

      ( usesport

      , ( providesport

         | componentsupportedinterface

              | findby

         )
```

```
        )>

    <!ATTLIST connectinterface

        id ID #IMPLIED>
```

L.6.5.1.1 *usesport.*

The *usesport* element (see Figure L-113) identifies, using the *usesidentifier* element, the component port that is using the provided interface from the *providesport* element. A *SWRadio* component may be referenced by one of four elements. One element is the *componentinstantiationref* that refers to the *componentinstantiation* id attribute (*see componentinstantiation*) within the assembly; the other elements are *findby, devicethatloadedthiscomponentref*, and *deviceusedbythiscomponentref.*

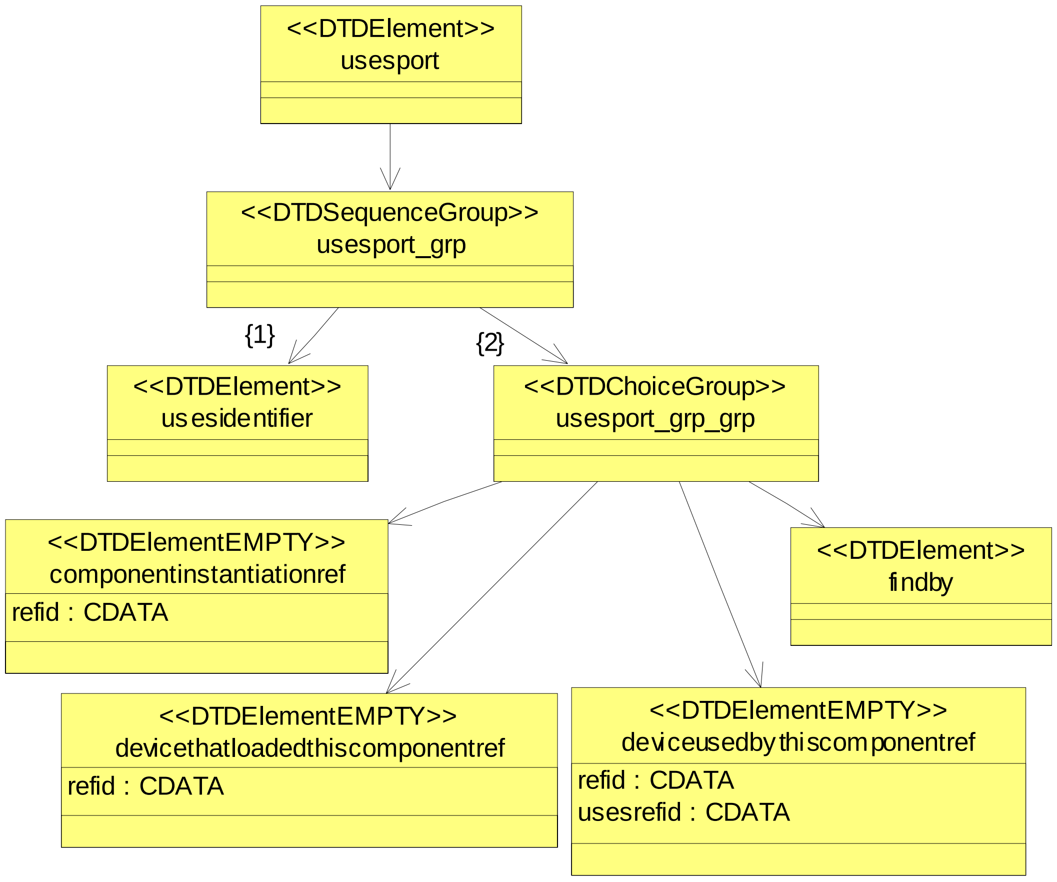


Figure L-113 – *usesport* Element Relationships

L.6.5 connections.

```
<!ELEMENT usesport
      ( usesidentifier
      ,  ( componentinstantiationref
         | devicethatloadedthiscomponentref
       | deviceusedbythiscomponentref
         | findby
         )
      )>
```

L.6.5.1.1.1*usesidentifier.*

The *usesidentifier* element identifies which “uses port” on the component is to participate in the connection relationship. This identifier will correspond with an id for one of the component ports specified in the Software Component Descriptor (see section L.5).

```
<!ELEMENT usesidentifier (#PCDATA)>
```

L.6.5.1.1.2*componentinstantiationref.*

The *componentinstantiationref* element refers to the id attribute of the *componentinstantiation* element within the Software Assembly Descriptor file. The refid attribute will correspond to the unique *componentinstantiation* id attribute.

```
<!ELEMENT componentinstantiationref EMPTY>
<!ATTLIST componentinstantiationref
      refid CDATA #REQUIRED>
```

L.6.5.1.1.3*findby.*

The *findby* element (see Figure L-114) is used to resolve a connection between two components. It tells the Domain Management function how to locate a component interface involved in a connection relationship. The *namingservice* element specifies a naming service name to search for the desired component interface.

The *domainfinder* element specifies an element within the domain that is known to the Domain Management function.

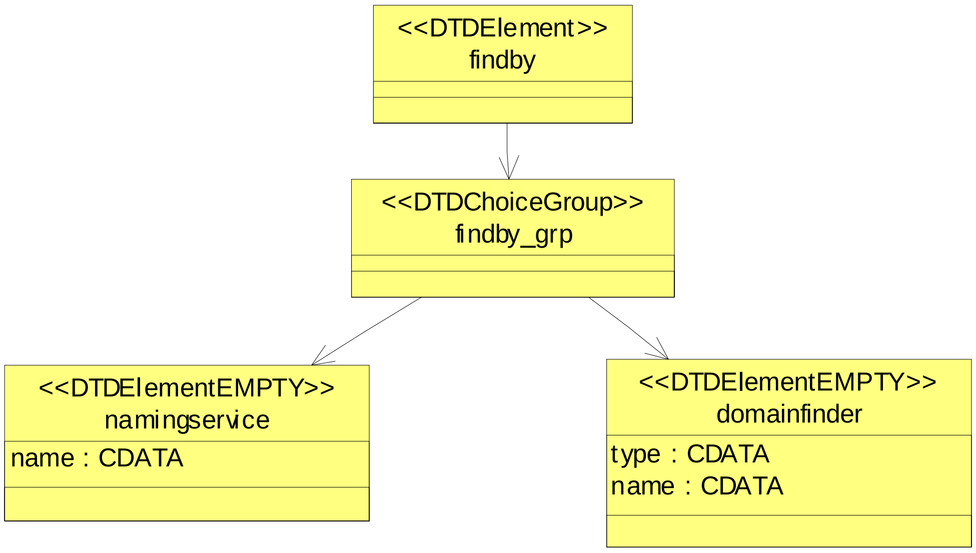


Figure L-114 – *findby* Element Relationships

```
<!ELEMENT findby

      ( namingservice

      | domainfinder

      )>
```

L.6.5.1.1.3.1namingservice.

The *namingservice* element is a child element of the *findby* element. The *namingservice* element is used to indicate to the ApplicationFactoryComponent the requirement to find a component interface. The ApplicationFactoryComponent will use the *name* attribute to search the CORBA Naming Service for the appropriate interface.

```
<!ELEMENT namingservice EMPTY
```

```
<!ATTLIST namingservice

      name CDATA #REQUIRED>
```

L.6.5.1.1.3.2*domainfinder.*

The *domainfinder* element is a child element of the *findby* element. The *domainfinder* element is used to indicate to the ApplicationFactoryComponent the necessary information to find an object reference that is of specific type and may also be known by an optional name within the domain. At a minimum the following valid type attribute values need to be supported “filemanager”, “log”, “eventchannel”, “namingservice”, “application”, and “service”. If a name attribute is not supplied, then the component reference returned is the *DomainManagerComponent’s FileManager* or Naming Service corresponding to the type attribute provided. If a name attribute is not supplied and the type attribute has a value of “application”, “service”, or “log” then a null reference is returned. The type attribute value of “eventchannel” is used to specify the event channel to be used in the OE’s CORBA Event Service for producing or consuming events. If the name attribute is not supplied and the type attribute has a value of “eventchannel” then the Incoming Domain Management event channel is used.

```
<!ELEMENT domainfinder EMPTY>

<!ATTLIST domainfinder

      type CDATA #REQUIRED

      name CDATA #IMPLIED>
```

L.6.5.1.1.4*devicethatloadedthiscomponentref.*

The *devicethatloadedthiscomponentref* element refers to a specific component found in the assembly, which is used to obtain the DeviceComponent that was used to load the referenced component from the ApplicationFactoryComponent. The DeviceComponent obtained is then associated with this component instance. This relationship is needed when a component (e.g., modem adapter) is pushing data and/or commands to a non-CORBA capable device such as modem.

```
<!ELEMENT devicethatloadedthiscomponentref EMPTY>

<!ATTLIST devicethatloadedthiscomponentref

      refidC DATA #REQUIRED>
```

L.6.5.1.1.5*deviceusedbythiscomponentref.*

The *deviceusedbythiscomponentref* element refers to a specific component, within the assembly, which is used to obtain the DeviceComponent that is being used by the specific component from the ApplicationFactoryComponent. This relationship is needed when a component is pushing or pulling data and/or commands to another component that exists in the system such as an audio device.

```
<!ELEMENT deviceusedbythiscomponentref EMPTY>

<!ATTLIST deviceusedbythiscomponentref

refid CDATA #REQUIRED

usesrefid CDATA #REQUIRED>
```

L.6.5.1.2 *providesport.*

The *providesport* element (see Figure L-115) identifies, using the *providesidentifier* element, the component port that is provided to the *usesport* interface within the *connectinterface* element. A SWRadio component may be referenced by one of four elements. One element is the *componentinstantiationref* that refers to the *componentinstantiation* id (*see componentinstantiation*) within the assembly; the other elements are *findby*, *devicethatloadedthiscomponentref*, and *deviceusedbythiscomponentref.* The *findby* element by itself is used when the object reference is not a ResourceComponent type.

L.6.5 connections.

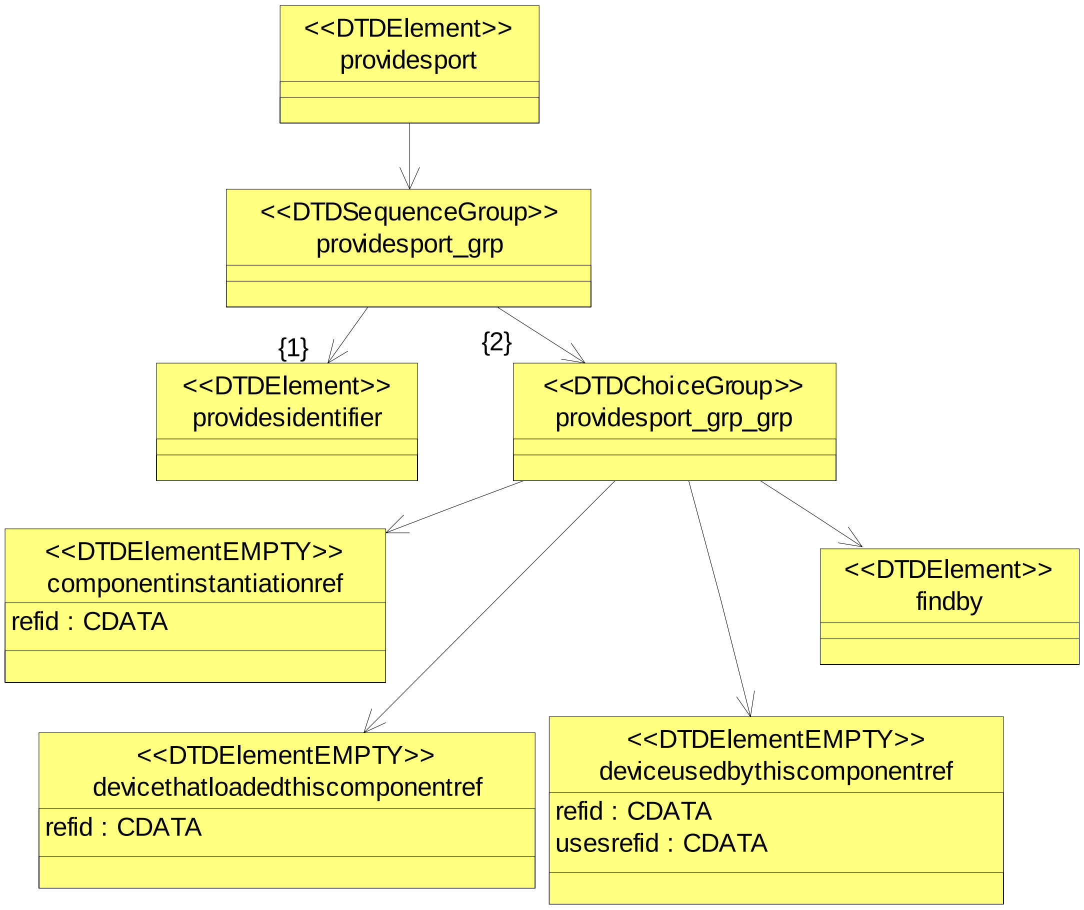


Figure L-115 – *providesport* Element Relationships

```
<!ELEMENT providesport
      ( providesidentifier
      , ( componentinstantiationref
        | devicethatloadedthiscomponentref
        | deviceusedbythiscomponentref
        | findby
        )
)>
```

L.6.5.1.2.1*providesidentifier.*

The *providesidentifier* element identifies which “provides port” on the component is to participate in the connection relationship. This identifier will correspond with a repid attribute for one of the component ports elements, specified in the Software Component Descriptor (see section L.5).

```
<!ELEMENT providesidentifier (#PCDATA)>
```

L.6.5.1.2.2*componentinstantiationref.*

See componentinstantiationref. for a description of the *componentinstantiationref* element.

L.6.5.1.2.3*findby.*

See section findby. for a description of the *findby* element. The *namingservice* element’s name attribute denotes a complete naming context.

L.6.5.1.2.4*devicethatloadedthiscomponentref.*

See section TBD for a description of the *devicethatloadedthiscomponentref* element.

L.6.5.1.2.5*deviceusedbythiscomponentref.*

See TBD for a description of the *deviceusedbythiscomponentref* element.

L.6.5.1.3 *componentsupportedinterface.*

The *componentsupportedinterface* element (see Figure L-116) specifies a component, which has a *supportsinterface* element, that can satisfy an interface connection to a port specified by the *usesport* element, within a *connectinterface* element. This component is identified by a *componentinstantiationref* or a *findby* element. The *componentinstantiationref* identifies a component within the assembly. The *findby* element points to an existing component that can be found within a Naming Service.

L.6.5 connections.

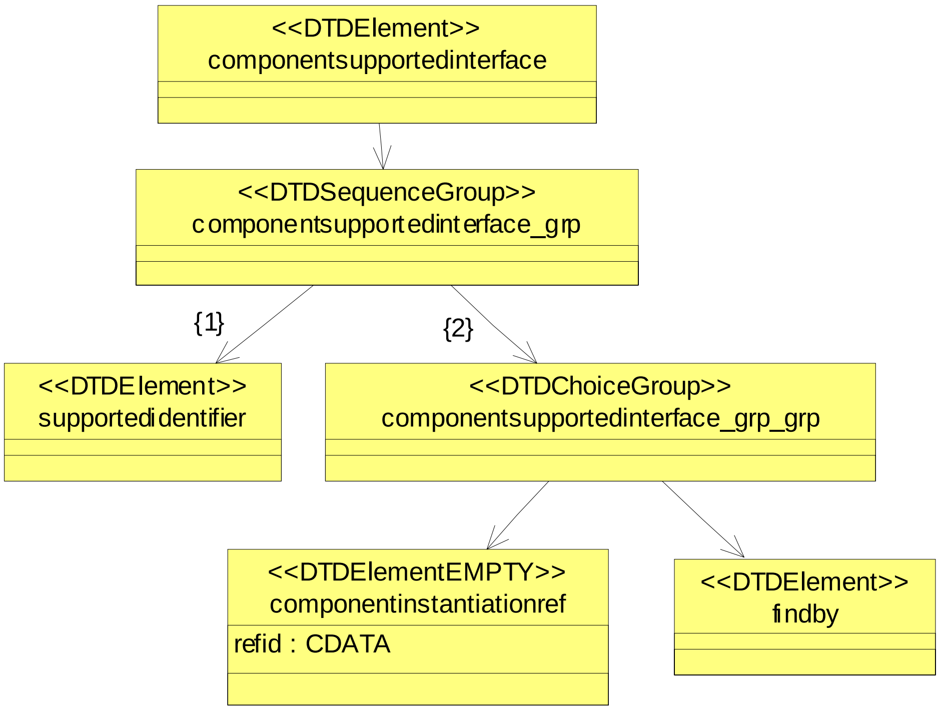


Figure L-116 – *componentsupportedinterface* Element Relationships

```
<!ELEMENT componentsupportedinterface

( supportedidentifier

, ( componentinstantiationref

     | findby

        )

      )>
```

L.6.5.1.3.1*supportedidentifier.*

The *supportedidentifier* element identifies which supported interface on the component is to participate in the connection relationship. This identifier will correspond with the repid attribute of one of the component's *supportsinterface* elements, specified in the Software Component Descriptor.

```
<!ELEMENT supportedidentifier (#PCDATA)>
```

L.6.5.1.3.2*componentinstantiationref.*

See section componentinstantiationref. for a description of the *componentinstantiationref* element.

L.6.5.1.3.3findby.

See section findby. for a description of the *findby* element.

L.6.6 *externalports.*

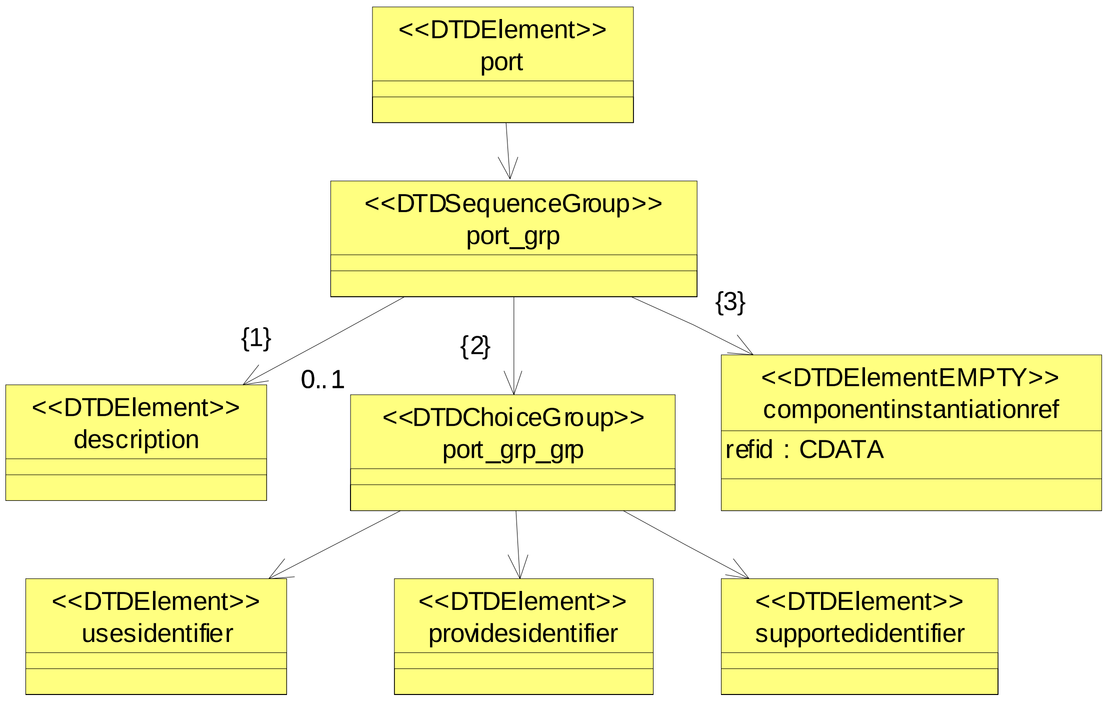
The optional *externalports* element is a child element of the *softwareassembly* element (see Figure L-117). The *externalports* element is used to identify the visible ports for the software assembly. The ApplicationManager *getport*() operation is used to access the assembly's visible ports.

```
<!ELEMENT externalports

( port+

)>
```

L.6.6 externalports.



<<DTDElement>> port
<<DTDSequenceGroup>> port_grp
{1}
0..1
{2}
{3}
<<DTDElement>> description
<<DTDChoiceGroup>> port_grp_grp
<<DTDElementEMPTY>> componentinstantiationref
refid : CDATA
<<DTDElement>> usesidentifier
<<DTDElement>> providesidentifier
<<DTDElement>> supportedidentifier

Figure L-117 – *port* Element Relationships

```
<!ELEMENT port

      ( description?

      , ( usesidentifier | providesidentifier |

          supportedidentifier)

      , componentinstantiationref

      )>

<!ELEMENT description (#PCDATA)>
```

L.7 Device Configuration Descriptor.

This section describes the XML elements of the Device Configuration Descriptor (DCD) XML file; the *device-configuration* element (see Figure L-118). The DCD is based on the SAD (e.g., componentfiles, partitioning, etc.) DTD. The intent of the DCD is to provide the means of describing the components that are initially started on the DeviceManagerComponent node, how to obtain the DomainManagerComponent object reference, connections of services to components (*ServiceComponent(*s), DeviceManagerComponent), and the characteristics (file system names, etc.) for a DeviceManagerComponent. The *componentfiles* and *partitioning* elements are optional; if not provided, that means no components are started up on the node, except for a DeviceManagerComponent. If the *partitioning* element is specified then a *componentfiles* element has to be specified also.

The *deviceconfiguration* element's id attribute is a unique identifier within the domain for the device configuration. The name attribute is the user-friendly name for the DeviceManagerComponent's label attribute. The name attribute is supplied when the id is not user-friendly such as a DCE UUID.

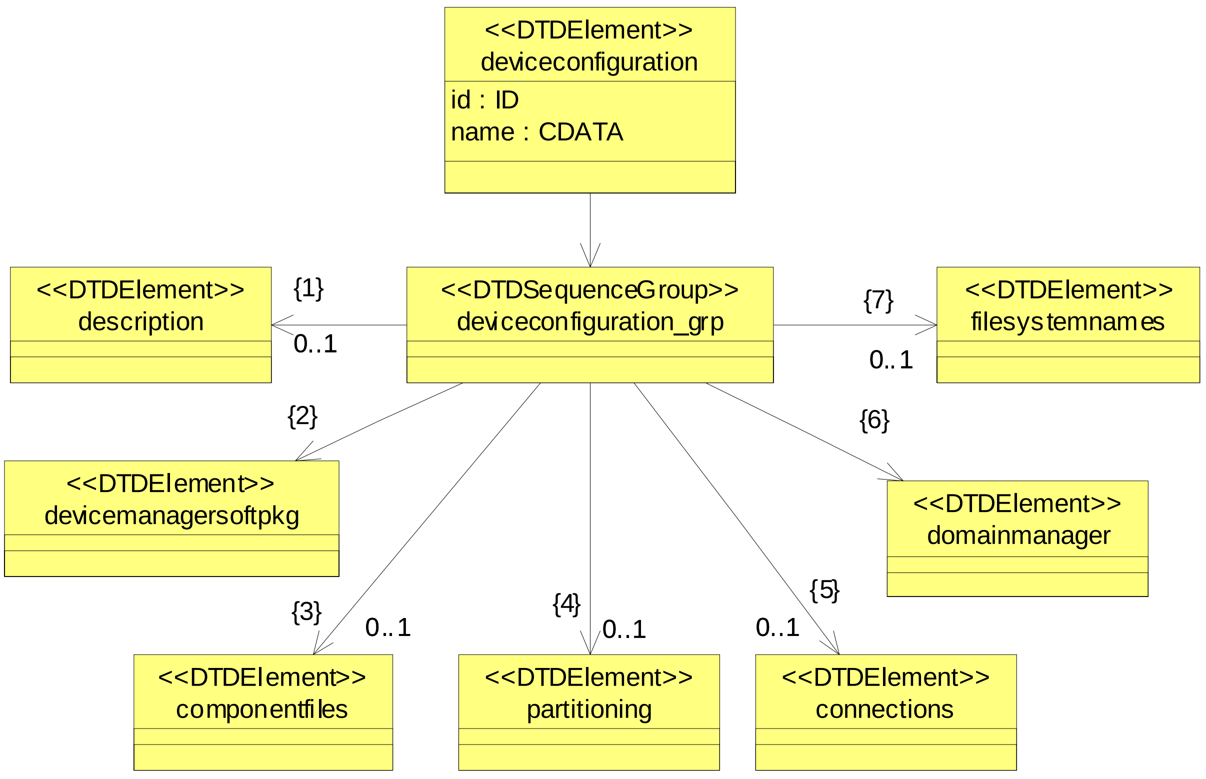


Figure L-118 – *deviceconfiguration* Element Relationships

```
<!ELEMENT deviceconfiguration
```

```
         ( description?

         , devicemanagersoftpkg

         , componentfiles?

         , partitioning?

         , connections?

         , domainmanager

         , filesystemnames?

         )>

   <!ATTLIST deviceconfiguration

         id ID #REQUIRED

         name CDATA #IMPLIED>
```

L.7.1 *description.*

The optional *description* element, of the *deviceconfiguration* element, may be used to provide information about the device configuration.

```
   <!ELEMENT description (#PCDATA)>
```

L.7.2 *devicemanagersoftpkg*.

The *devicemanagersoftpkg* element refers to the SPD for the DeviceManagerComponent that corresponds to this DCD. The SPD file is referenced by a *localfile* element. The referenced file can be used to describe the DeviceManagerComponent implementation and to specify the *usesports* for the services (Log*(s)*, etc.) used by the DeviceManagerComponent. See (section L.2.1.1.1) for description of the *localfile* element.

```
   <!ELEMENT devicemanagersoftpkg

   ( localfile

   )>
```

L.7.3 *componentfiles.*

The optional *componentfiles* element is used to reference deployment information for components that are started up on the device. The *componentfile* element references a Software Package Descriptor (SPD). The SPD, for example, can be used to describe ServiceComponents, DeviceManagerComponents, a *DomainManagerComponent*, a Naming Service, and File Services. See section L.6.2 for the definition of the *componentfiles* element.

L.7.4 *partitioning.*

The optional *partitioning* element consists of a set of *componentplacement* elements. A component instantiation is captured inside a *componentplacement* element.

```
<!ELEMENT partitioning

( componentplacement

      )*>
```

L.7.5 *componentplacement.*

The *componentplacement* element (see Figure L-119) is used to define a particular deployment of a component. The *componentfileref* element identifies the component to be deployed. The *componentinstantiation* element identifies the actual component created and its id attribute is a DCE UUID value with the format as specified in section Software Package.. Multiple components of the same kind can be created within the same *componentplacement* element.

The optional *deployondevice* element indicates the device on which the *componentinstantiation* element is deployed. The optional *compositepartofdevice* element indicates the device that the *componentinstantiation* element is aggregated with to form an aggregate relationship. When the component is a logical *Device*, the *devicepkgfile* element indicates the hardware device information for the logical *Device*.

L.7.5 componentplacement.

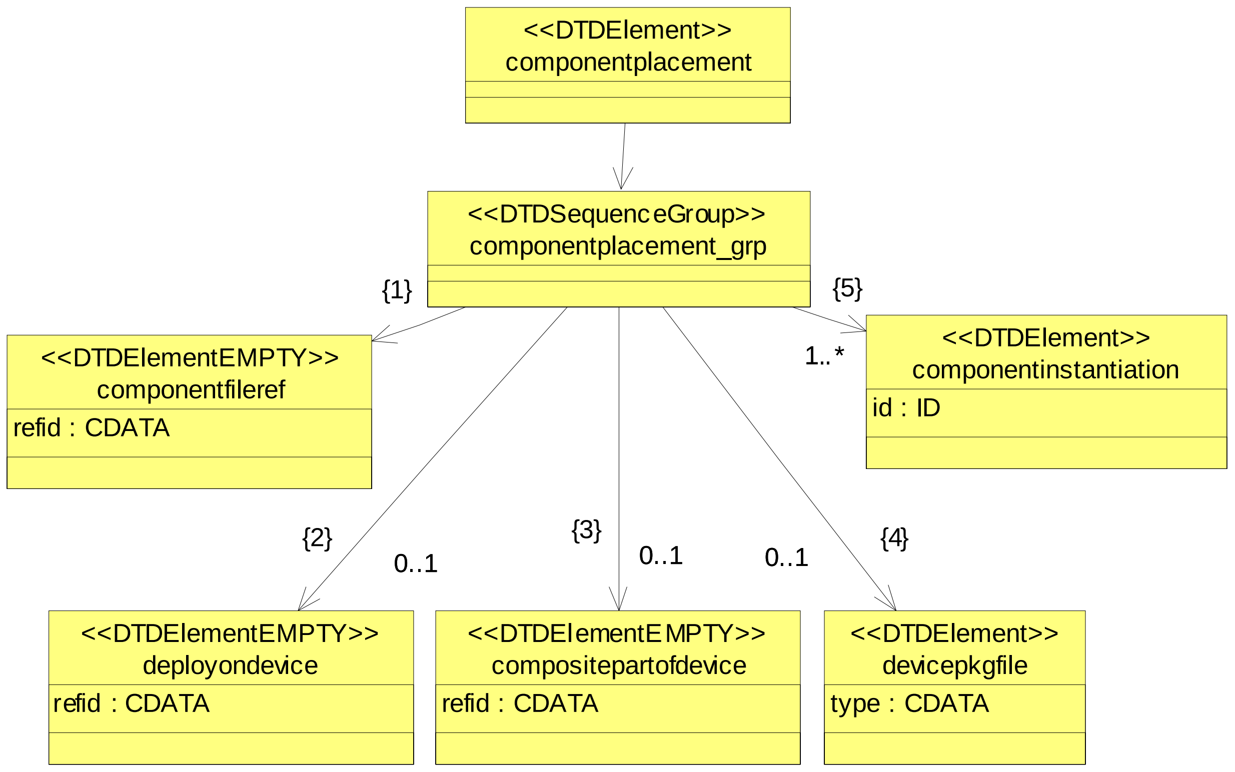


Figure L-119 – *componentplacement* Element Relationships

```
<!ELEMENT componentplacement

      ( componentfileref

      , deployondevice?

      , compositepartofdevice?

      , devicepkgfile?

      , componentinstantiation+

      )>
```

L.7.5.1 ***componentfileref.***

The *componentfileref* element is used to reference a *componentfile* element within the *componentfiles* element. The *componentfileref* element's refid attribute corresponds to a *componentfile* element's id attribute.

```
<!ELEMENT componentfileref  EMPTY>

<!ATTLIST componentfileref

      refid CDATA #REQUIRED>
```

L.7.5.1.1 ***deployondevice.***

The *deployondevice* element is used to reference a *componentinstantiation* element on which this *componentinstantiation* is deployed.

```
<!ELEMENT deployondevice  EMPTY>

<!ATTLIST deployondevice

      refid CDATA #REQUIRED>
```

L.7.5.1.2 ***devicepkgfile.***

The *devicepkgfile* element is used to refer to a device package file that contains the hardware device definition.

```
<!ELEMENT devicepkgfile

( localfile

)>

<!ATTLIST devicepkgfile

type CDATA #IMPLIED>
```

L.7.5.1.2.1***localfile.***

See L.2.1.1.1 for a definition of the *localfile* element.

L.7.5.1.3 *compositepartofdevice.*

The *compositepartofdevice* element is used when an aggregate relationship exists to reference the *componentin-stantiation* element that describes the whole *Device* for which this *Device's componentinstantiation* element de-scribes a part of the aggregate *Device*.

```
<!ELEMENT compositepartofdevice  EMPTY>

<!ATTLIST compositepartofdevice

      refid CDATA #REQUIRED>
```

L.7.5.1.4 *componentinstantiation.*

The *componentinstantiation* element (see Figure L-120) is intended to describe a particular instantiation of a component relative to a *componentplacement* element. The *componentinstantiation*'s id attribute is a DCE UUID that uniquely identifier the component. The id is a DCE UUID value as specified in section Software Package.. The *componentinstantiation* contains a *usagename* element that is intended for an applicable name for the com-ponent. The optional *componentproperties* element (see Figure L-121) is a list of property values that are used in configuring the component. D.6.3.1.2 defines the property list for the *componentinstantiation* element, which contains initial properties values. For a component service type (e.g,, Log), the *usagename* element needs to be unique for each service type.

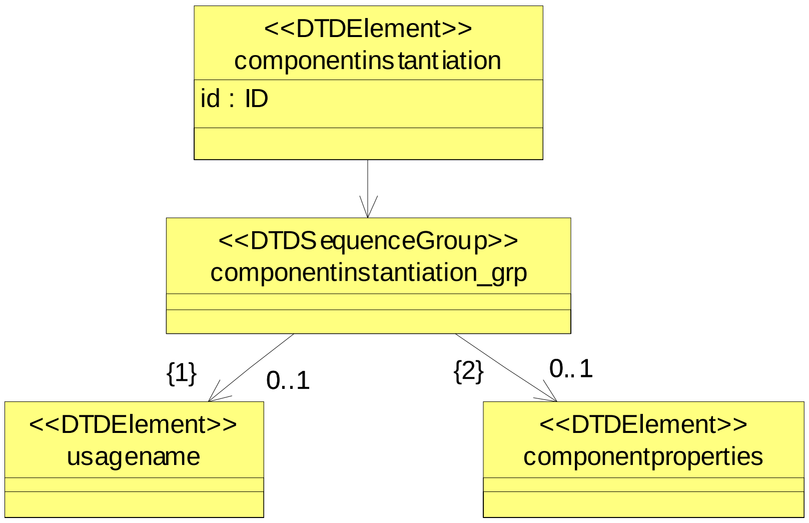


Figure L-120 – *componentinstantiation* Element Relationships

```
<!ELEMENT componentinstantiation
```

```
      ( usagename?
      ,componentproperties?
      )>
<!ATTLIST componentinstantiation
      id ID #REQUIRED>
<!ELEMENT usagename (#PCDATA)>
```

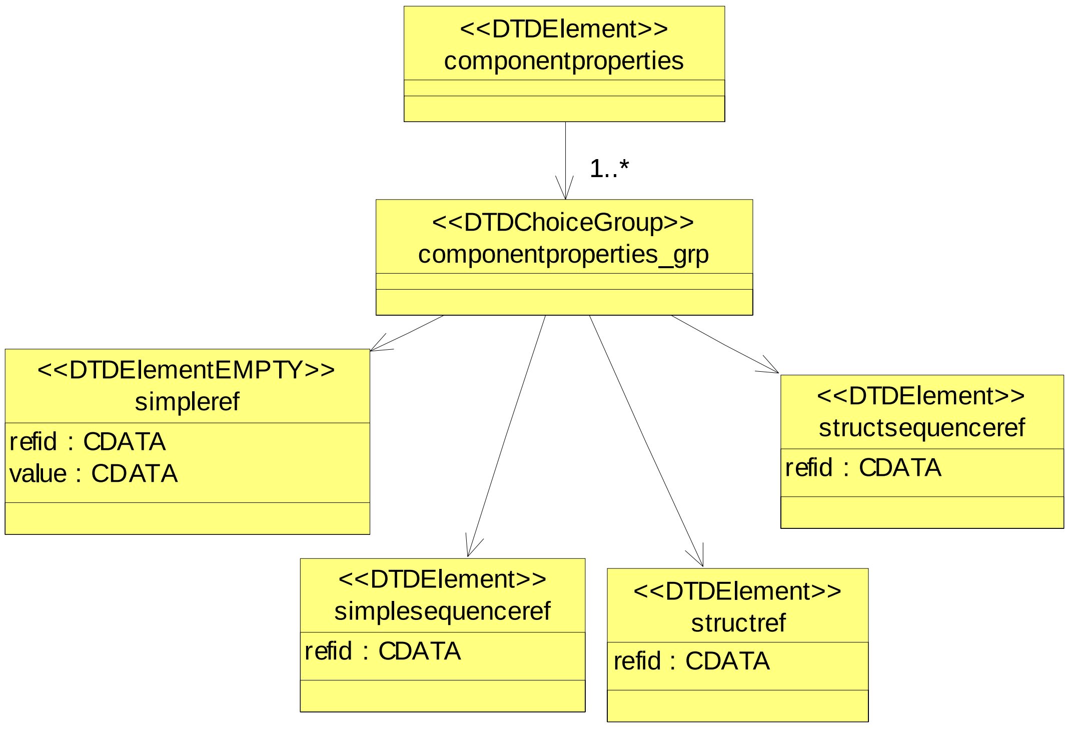


Figure L-121 – *componentproperties* Element Relationships

```
<!ELEMENT componentproperties
( simpleref
| simplesequenceref
| structref
| structsequenceref
)+ >

<!ELEMENT simpleref EMPTY>
<!ATTLIST simpleref
refid CDATA #REQUIRED
value CDATA #REQUIRED>

<!ELEMENT simplesequenceref
   ( values
   )>
<!ATTLIST simplesequenceref
   refid CDATA #REQUIRED>

<!ELEMENT structref
   ( simpleref+
   )>
<!ATTLIST structref
   refid CDATA #REQUIRED>

<!ELEMENT structsequenceref
   ( structvalue+
      )>
<!ATTLIST structsequenceref
      refid CDATA #REQUIRED>
```

```
<!ELEMENT structvalue

   ( simpleref+

   )>

<!ELEMENT values

   ( value+

   )>

<!ELEMENT value (#PCDATA)>
```

L.7.6 *connections*.

The *connections* element in the DCD is the same as the *connections* element in the SAD in section D.6.5. The *connections* element in the DCD is used to indicate the services (Log, etc…) instances that are used by the DeviceManagerComponent and ServiceComponent(s) in the DCD. The DomainManagerComponent will parse the connections element and make the connections when the DeviceManagerComponent registers with the DomainManagerComponent. To establish connections to a DeviceManagerComponent, the DCD's *deviceconfiguration* element's id attribute value is used for the SAD's *usesport* element's *componentinstantiationref* element's refid attribute value.

L.7.7 domainmanager

The *domainmanager* element (see Figure L-122) indicates how to obtain the DomainManagerComponent object reference. See sections L.6.5.1.1.3.1 for description of the namingservice.

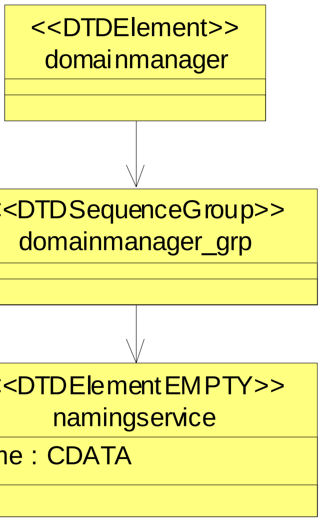


Figure L-122 – *domainmanager* Element Relationships

```
<!ELEMENT domainmanager

( namingservice)>)>

<!ELEMENT namingservice EMPTY>

<!ATTLIST namingservice

   name CDATA #REQUIRED>
```

L.7.8 *filesystemnames.*

The optional *filesystemnames* element indicates the mounted file system names for DeviceManagerComponent'*s FileManager.*

```
<!ELEMENT filesystemnames

      ( filesystemname+

      )>
```

```
<!ELEMENT filesystemname EMPTY>

<!ATTLIST filesystemname

      mountname CDATA #REQUIRED

      deviceid CDATA #REQUIRED>
```

L.8 *DomainManager* Configuration Descriptor.

This section describes the XML elements of the *DomainManagerComponent* Configuration Descriptor (DMD) XML file; the *domainmanagerconfiguration* element (see Figure L-123). The *domainmanagerconfiguration* element id attribute is a DCE UUID that uniquely identifies the *DomainManagerComponent*. The id is a DCE UUID value as specified in section Software Package..

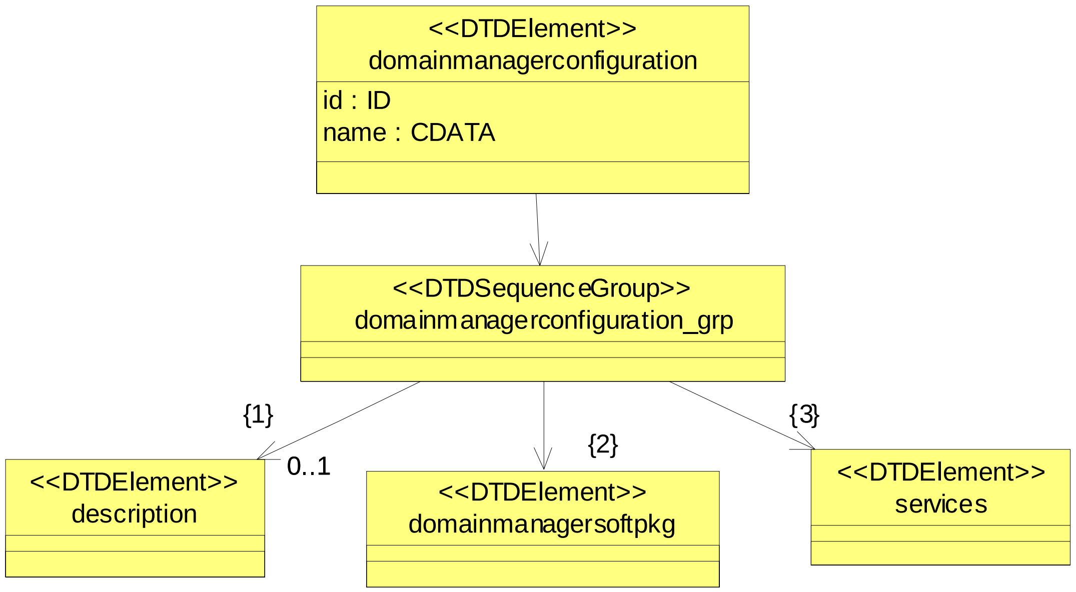


Figure L-123 – *domainmanagerconfiguration* Element Relationships

```
<!ELEMENT domainmanagerconfiguration

   ( description?

   , domainmanagersoftpkg

   , services?

   )>

<!ATTLIST domainmanagerconfiguration

   id ID #required

   name #CDATA #required>
```

L.8.1 *description*.

The optional *description* element of the DMD may be used to provide information about the configuration.

```
<!ELEMENT description (#PCDATA)>
```

L.8.2 *domainmanagersoftpkg*.

The *domainmanagersoftpkg* element refers to the SPD for the *DomainManagerComponent*. The SPD file is referenced by a *localfile* element. This SPD can be used to describe the *DomainManagerComponent* implementation and to specify the *usesports* for the services (Log*(s)*, etc…) used by the *DomainManagerComponent*. See section L.2.1.1.1 for description of the localfile element.

```
<!ELEMENT domainmanagersoftpkg

( localfile

) >
```

L.8.3 *services*.

The *services* element in the DMD is used by the *DomainManagerComponent* to determine which service (Log, etc.) instances to use; it makes use of the *service* element (see Figure L-124). See section L.6.5.1.1.3 for a description of the *findby* element. See section L.6.5.1.1.1 for a description of the *usesidentifier* element.

```
<!ELEMENT services
   ( service+
   ) >
```

L.8.3 services.

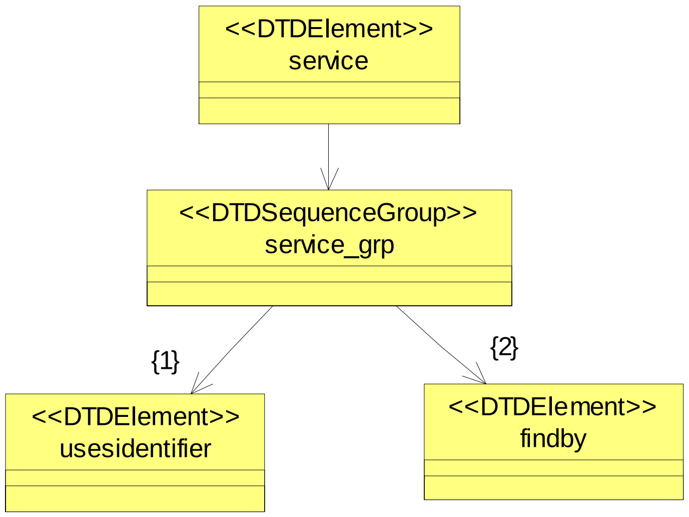


Figure L-124 – *service* Element Relationships

```
<!ELEMENT service
   ( usesidentifier
   , findby
   )>
```

L.9 Document Type Definitions.

L.9.1 Software Package Descriptor DTD

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT softpkg

      ( title?

   , author+

   , description?

   , propertyfile?

   , descriptor?

   , (implementation | assemblyimplementation)+

   , usesdevice*

   )>

<!ATTLIST softpkg

   id    ID #REQUIRED

   name  CDATA #REQUIRED

   type  CDATA #IMPLIED

   version CDATA #IMPLIED >


<!ELEMENT propertyfile

   (localfile

   )>

<!ATTLIST propertyfile

   type  CDATA #IMPLIED>


<!ELEMENT localfile EMPTY>
```

```
<!ATTLIST localfile
   name  CDATA #REQUIRED>
<!ELEMENT assemblyimplementation
   (localfile
   )>
<!ELEMENT title (#PCDATA)>

<!ELEMENT author
   ( name*
   , company?
   , webpage?
   )>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT webpage (#PCDATA)>

<!ELEMENT descriptor
   (localfile
   )>
<!ATTLIST descriptor
   name  CDATA #IMPLIED>

<!ELEMENT implementation
```

```
   ( description?
   , propertyfile?
   , code
   , compiler?
   , programminglanguage?
   , humanlanguage?
   , runtime?
   , ( os
     | processor
     | dependency
     )+
   , usesdevice*
   )>
<!ATTLIST implementation
   id    ID #REQUIRED
   aepcompliance (aep_compliant | min_aep_compliant | aep_non_compliant)
"aep_compliant">
<!ELEMENT description (#PCDATA)>
<!ELEMENT code
   ( localfile
   , entrypoint?
   , stacksize?
   , priority?
   )>
<!ATTLIST code
   type  CDATA #IMPLIED>
```

```
<!ELEMENT entrypoint (#PCDATA)>

<!ELEMENT stacksize (#PCDATA)>

<!ELEMENT priority (#PCDATA)>

<!ELEMENT compiler EMPTY>
<!ATTLIST compiler
   name  CDATA #REQUIRED
   version CDATA #IMPLIED>

<!ELEMENT programminglanguage EMPTY>
<!ATTLIST programminglanguage
   name  CDATA #REQUIRED
   version CDATA #IMPLIED>

<!ELEMENT humanlanguage EMPTY>
<!ATTLIST humanlanguage
   name  CDATA #REQUIRED>

<!ELEMENT os EMPTY>
<!ATTLIST os
   name  CDATA #REQUIRED
   versionCDATA #IMPLIED>
```

```
<!ELEMENT processor EMPTY>
<!ATTLIST processor
   name  CDATA #REQUIRED>


<!ELEMENT dependency
    ( softpkgref
      | propertyref
      | propertyvaluesref
 )>
 <!ATTLIST dependency
  type   CDATA   #REQUIRED>


<!ELEMENT softpkgref
         ( localfile
         , implref?
         )>


<!ELEMENT implref EMPTY>
<!ATTLIST implref
   refid CDATA #REQUIRED>


<!ELEMENT propertyref EMPTY>
<!ATTLIST propertyref
         refid CDATA #REQUIRED
         value CDATA #REQUIRED>
   <!ELEMENT propertyvaluesref

         (refid

          ,value+)>

    <!ELEMENT refid (#PCDATA)>

    <!ELEMENT value (#PCDATA)>



   <!ELEMENT runtime EMPTY>

   <!ATTLIST runtime

      name  CDATA #REQUIRED

      version CDATA #IMPLIED>



<!ELEMENT  usesdevice
      ( (propertyref
         | propertyvaluesref)+
         )>
      <!ATTLIST usesdevice
           id     ID          #REQUIRED
           type   CDATA       #REQUIRED>
```

L.9.2 Software Component Descriptor DTD

```
   <?xml version="1.0" encoding="UTF-8"?>

   <!ELEMENT softwarecomponent

      ( corbaversion

      , componentrepid

      , componenttype

      , componentfeatures

      , interfaces

      , propertyfile?

      )>
```

```
<!ELEMENT corbaversion (#PCDATA)>

<!ELEMENT componentrepid EMPTY>
<!ATTLIST componentrepid
   repid CDATA #REQUIRED>

<!ELEMENT componenttype (#PCDATA)>

<!ELEMENT componentfeatures
   ( supportsinterface*
   , ports
   )>

<!ELEMENT supportsinterface EMPTY>
<!ATTLIST supportsinterface
   repid    CDATA #REQUIRED
   supportsname CDATA #REQUIRED>

<!ELEMENT ports
   (provides
   | uses
   )*>

<!ELEMENT provides
   ( porttype*)>
```

```
<!ATTLIST provides
   repid    CDATA #REQUIRED
   providesname CDATA #REQUIRED>

<!ELEMENT uses
   ( porttype*)>
<!ATTLIST uses
   repid CDATA #REQUIRED
   usesnameCDATA #REQUIRED>

<!ELEMENT porttype EMPTY>
<!ATTLIST porttype
    type( data| control|
         responses| test ) #REQUIRED>

<!ELEMENT interfaces
   ( interface+
         )>

<!ELEMENT interface
   ( inheritsinterface*)>
<!ATTLIST interface
   repid CDATA #REQUIRED
   name  CDATA #REQUIRED>

<!ELEMENT inheritsinterface EMPTY>
```

```
<!ATTLIST inheritsinterface
   repid CDATA#REQUIRED>


<!ELEMENT propertyfile
   (localfile
   )>
<!ATTLIST propertyfile
   type  CDATA #IMPLIED>


<!ELEMENT localfile EMPTY>
<!ATTLIST localfile
   name  CDATA #REQUIRED>
```

L.9.3 Device Package Descriptor DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT devicepkg
         ( title?
         , author+
         , description?
         , hwdeviceregistration
         )>
<!ATTLIST devicepkg
         id ID #REQUIRED
         name CDATA #REQUIRED
         version CDATA #IMPLIED>


<!ELEMENT title (#PCDATA)>
```

L.9.3 Device Package Descriptor DTD

```
<!ELEMENT author

         ( name*

         , company?

         , webpage?

         )>


<!ELEMENT name (#PCDATA)>


<!ELEMENT company (#PCDATA)>


<!ELEMENT webpage (#PCDATA)>


<!ELEMENT description (#PCDATA)>


<!ELEMENT hwdeviceregistration

         ( propertyfile?

         , description

         , manufacturer

         , modelnumber

         , deviceclass

         , childhwdevice*

     )>

<!ATTLIST hwdeviceregistration

         id ID #REQUIRED

         name CDATA #REQUIRED
         version CDATA #IMPLIED>

<!ELEMENT propertyfile
   ( localfile
   )>
<!ATTLIST propertyfile
   type CDATA #IMPLIED>

<!ELEMENT localfile EMPTY>
<!ATTLIST localfile
         name CDATA #REQUIRED>

<!ELEMENT manufacturer (#PCDATA)>

<!ELEMENT modelnumber (#PCDATA)>

<!ELEMENT deviceclass
         (class+
         )>

<!ELEMENT class (#PCDATA)>

<!ELEMENT childhwdevice
         (hwdeviceregistration
         |devicepkgref
         )>
```

```
<!ELEMENT devicepkgref
         (localfile
         )>
<!ATTLIST devicepkgref
         type CDATA #IMPLIED>
```

L.9.4 Properties Descriptor DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT properties
      ( description?
      , ( simple
      | simplesequence
         | test
         | struct
         | structsequence
         )+
       )>

<!ELEMENT simple
   ( description?
   , value?
   , units?
   , range?
   , enumerations?
   , kind*
   , action?

   )>

<!ATTLIST simple

   id ID                   #REQUIRED

   type( boolean | char  | double | float

         | short | long | longlong | objref | octet

         | string | ulong |ushort | ulonglong |longdoudble | wchar | wstring
)     #REQUIRED

   integerID CDATA      #IMPLIED

   name CDATA           #IMPLIED

   mode( readonly| readwrite | writeonly)"readwrite">


<!ELEMENT description (#PCDATA)>


<!ELEMENT value (#PCDATA)>


<!ELEMENT units (#PCDATA)>


<!ELEMENT range EMPTY>

<!ATTLIST range

   min   CDATA #REQUIRED

   max   CDATA #REQUIRED>


<!ELEMENT enumerations

   ( enumeration+

   )>
```

L.9.4 Properties Descriptor DTD

```
<!ELEMENT enumeration EMPTY>
<!ATTLIST enumeration
   label CDATA #REQUIRED
   value CDATA #IMPLIED>

<!ELEMENT kind EMPTY>
<!ATTLIST kind
   kindtype(allocation | configure | execparam | factoryparam)
"configure">

<!ELEMENT action EMPTY>
<!ATTLIST action
   type CDATA #REQUIRED>

<!ELEMENT simplesequence
   ( description?
   , values?
   , units?
   , range?
   , kind*
   , action?
   )>
<!ATTLIST simplesequence
   id ID                #REQUIRED
   type  ( boolean | char  | double | float
          | short | long | longlong | objref | octet

          | string | ulong |ushort | ulonglong |longdoudble | wchar | wstring
)  #REQUIRED

   integerID CDATA      #IMPLIED

   name CDATA        #IMPLIED

   mode(readonly | readwrite | writeonly)"readwrite">


<!ELEMENT values

   ( value+

)>


<!ELEMENT test

         ( description

         , inputvalue?

   , resultvalue

   )>

<!ATTLIST test

   id    CDATA #REQUIRED

   integerID CDATA      #IMPLIED>


<!ELEMENT inputvalue

   ( simple+

   )>


<!ELEMENT resultvalue

   ( simple+
   )>


<!ELEMENT struct
   ( description?
   , simple+
   , configurationkind?
   , action?
   )>
<!ATTLIST struct
   id    ID #REQUIRED
   integerID CDATA      #IMPLIED
   name      CDATA #IMPLIED
         mode(readonly | readwrite | writeonly)  "readwrite">


<!ELEMENT configurationkind EMPTY>
<!ATTLIST configurationkind
   kindtype(configure | factoryparam | allocation)"configure">


<!ELEMENT structsequence
   ( description?
   , structvalue+
   , configurationkind?
   , action?
   )>
<!ATTLIST structsequence
   id       ID #REQUIRED
```

```
    structrefid CDATA #REQUIRED
    integerID CDATA      #IMPLIED
    name       CDATA #IMPLIED
    mode  (readonly | readwrite | writeonly)  "readwrite">

 <!ELEMENT structvalue
    (simpleref+
    )>

 <!ELEMENT simpleref EMPTY>
 <!ATTLIST simpleref
    refid CDATA #REQUIRED
    value CDATA #REQUIRED>
```

L.9.5 Software Assembly Descriptor DTD

```
 <?xml version="1.0" encoding="UTF-8"?>
 <!ELEMENT softwareassembly
    ( description?
    , componentfiles
    , partitioning
    , assemblycontroller
    , connections?
    , externalports?
    )>
 <!ATTLIST softwareassembly
```

```
      id    ID #REQUIRED
      name  CDATA #IMPLIED
      version CDATA #IMPLIED>

   <!ELEMENT description (#PCDATA)>
   <!ELEMENT componentfiles
      ( componentfile+
      )>

   <!ELEMENT componentfile
      ( localfile
      )>
   <!ATTLIST componentfile
      id    ID #REQUIRED
      type  CDATA #IMPLIED>

   <!ELEMENT localfile EMPTY>
   <!ATTLIST localfile
      name  CDATA #REQUIRED>

   <!ELEMENT partitioning
            ( componentplacement
      |  hostcollocation
      )+>

   <!ELEMENT componentplacement
      ( componentfileref
      , componentinstantiation+
      )>

   <!ELEMENT componentfileref  EMPTY>
   <!ATTLIST componentfileref
      refid CDATA #REQUIRED>

   <!ELEMENT componentinstantiation
      ( usagename?
      , componentproperties?
      , findcomponent?
      )>
   <!ATTLIST componentinstantiation
      id    ID #REQUIRED>

   <!ELEMENT usagename (#PCDATA)>

   <!ELEMENT componentproperties
      ( simpleref
      | simplesequenceref
      | structref
      | structsequenceref
      )+ >

   <!ELEMENT findcomponent
```

```
      ( componentresourcefactoryref
      | namingservice
      )>

   <!ELEMENT componentresourcefactoryref
      ( resourcefactoryproperties?
      )>
   <!ATTLIST componentresourcefactoryref
      refid CDATA #REQUIRED>

   <!ELEMENT resourcefactoryproperties
      ( simpleref
      | simplesequenceref
      | structref
      | structsequenceref
      )+ >

   <!ELEMENT simpleref EMPTY>
   <!ATTLIST simpleref
      refid CDATA #REQUIRED
      value CDATA #REQUIRED>

   <!ELEMENT simplesequenceref
      (values
      )>
   <!ATTLIST simplesequenceref
      refid CDATA #REQUIRED>


   <!ELEMENT structref
      (simpleref+
      )>
   <!ATTLIST structref
      refid CDATA #REQUIRED>


   <!ELEMENT structsequenceref
      ( structvalue+
            )>
   <!ATTLIST structsequenceref
      refid CDATA #REQUIRED>


   <!ELEMENT structvalue
      (simpleref+
      )>


   <!ELEMENT values
      ( value+
      )>


   <!ELEMENT value (#PCDATA)>


   <!ELEMENT hostcollocation
      ( componentplacement
      )+>
   <!ATTLIST hostcollocation
      id    ID #IMPLIED
      name  CDATA #IMPLIED>

   <!ELEMENT assemblycontroller
      ( componentinstantiationref
      )>

   <!ELEMENT connections
            ( connectinterface*
            )>

   <!ELEMENT connectinterface
      ( usesport
      , ( providesport
         | componentsupportedinterface
         | findby
        )
      )>
   <!ATTLIST connectinterface
      id    ID #IMPLIED>

   <!ELEMENT usesport
      ( usesidentifier
      , (componentinstantiationref
         | devicethatloadedthiscomponentref
         | deviceusedbythiscomponentref
         | findby
         )
      )>

   <!ELEMENT usesidentifier (#PCDATA)>
   <!ELEMENT componentinstantiationref EMPTY>
   <!ATTLIST componentinstantiationref
      refid CDATA #REQUIRED>
   <!ELEMENT findby
      ( namingservice
      | domainfinder
      )>
   <!ELEMENT namingservice EMPTY>
   <!ATTLIST namingservice
      name  CDATA #REQUIRED>
   <!ELEMENT domainfinder EMPTY>
   <!ATTLIST domainfinder
      type  CDATA #REQUIRED
      name  CDATA #IMPLIED>

   <!ELEMENT devicethatloadedthiscomponentref EMPTY>
   <!ATTLIST devicethatloadedthiscomponentref
      refid CDATA #REQUIRED>
```

```
<!ELEMENT deviceusedbythiscomponentref EMPTY>
<!ATTLIST deviceusedbythiscomponentref
   refid CDATA #REQUIRED
   usesrefid CDATA #REQUIRED>
<!ELEMENT providesport
   ( providesidentifier
   , ( componentinstantiationref
     | devicethatloadedthiscomponentref
     | deviceusedbythiscomponentref
     | findby
        )
   )>
<!ELEMENT providesidentifier (#PCDATA)>


<!ELEMENT componentsupportedinterface
   ( supportedidentifier
   , ( componentinstantiationref
     | findby
     )
   )>
<!ELEMENT supportedidentifier (#PCDATA)>
<!ELEMENT externalports
   (port+
   )>


<!ELEMENT port
```

```
   ( description?
   , (usesidentifier | providesidentifier | supportedidentifier)
   , componentinstantiationref
   )>
```

L.9.6 Device Configuration Descriptor DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT deviceconfiguration
   ( description?
   , devicemanagersoftpkg
   , componentfiles?
   , partitioning?
   , connections?
   , domainmanager
   , filesystemnames?
   )>
<!ATTLIST deviceconfiguration
   id    ID #REQUIRED
   name  CDATA #IMPLIED>

<!ELEMENT description (#PCDATA)>

<!ELEMENT devicemanagersoftpkg
   ( localfile
   )>

<!ELEMENT componentfiles
```

```
      ( componentfile+
      )>

   <!ELEMENT componentfile
      ( localfile
      )>
   <!ATTLIST componentfile
      id    ID #REQUIRED
      type  CDATA #IMPLIED>

   <!ELEMENT localfile EMPTY>
   <!ATTLIST localfile
      name  CDATA #REQUIRED>

   <!ELEMENT partitioning
            ( componentplacement
      )*>

   <!ELEMENT componentplacement
      ( componentfileref
      , deployondevice?
      , compositepartofdevice?
      , devicepkgfile?
      , componentinstantiation+
      )>
```

```
<!ELEMENT componentfileref  EMPTY>
<!ATTLIST componentfileref
   refid CDATA #REQUIRED>


<!ELEMENT deployondevice  EMPTY>
<!ATTLIST deployondevice
   refid CDATA #REQUIRED>



<!ELEMENT compositepartofdevice  EMPTY>
<!ATTLIST compositepartofdevice
   refid CDATA #REQUIRED>


<!ELEMENT devicepkgfile
   (localfile
   )>
<!ATTLIST devicepkgfile
   type  CDATA #IMPLIED>


<!ELEMENT componentinstantiation
   ( usagename?
    ,componentproperties?
   )>
<!ATTLIST componentinstantiation
   id ID #REQUIRED>
```

```
<!ELEMENT usagename (#PCDATA)>


<!ELEMENT componentproperties
   ( simpleref
   | simplesequenceref
   | structref
   | structsequenceref
   )+ >


<!ELEMENT simpleref EMPTY>
<!ATTLIST simpleref
   refid CDATA #REQUIRED
   value CDATA #REQUIRED>


<!ELEMENT simplesequenceref
   (values
   )>
<!ATTLIST simplesequenceref
   refid CDATA #REQUIRED>


<!ELEMENT structref
   (simpleref+
   )>
<!ATTLIST structref
   refid CDATA #REQUIRED>
```

```
<!ELEMENT structsequenceref
   ( structvalue+
         )>
<!ATTLIST structsequenceref
   refid CDATA #REQUIRED>

<!ELEMENT structvalue
   (simpleref+
   )>

<!ELEMENT values
   ( value+
   )>

<!ELEMENT value (#PCDATA)>

<!ELEMENT connections
      ( connectinterface*
      )>

<!ELEMENT connectinterface
   ( usesport
   , ( providesport
     | componentsupportedinterface
     | findby
     )
   )>
<!ATTLIST connectinterface
   id    ID #IMPLIED>

<!ELEMENT usesport
   ( usesidentifier
   , (componentinstantiationref
      | devicethatloadedthiscomponentref
      | deviceusedbythiscomponentref
      | findby
      )
   )>

<!ELEMENT usesidentifier (#PCDATA)>

<!ELEMENT componentinstantiationref EMPTY>
<!ATTLIST componentinstantiationref
   refid CDATA #REQUIRED>

<!ELEMENT devicethatloadedthiscomponentref EMPTY>
<!ATTLIST devicethatloadedthiscomponentref
         refid CDATA #REQUIRED>

<!ELEMENT deviceusedbythiscomponentref EMPTY>
<!ATTLIST deviceusedbythiscomponentref
         refid CDATA #REQUIRED
         usesrefid CDATA #REQUIRED>


<!ELEMENT providesport
   ( providesidentifier
   , ( componentinstantiationref
     | devicethatloadedthiscomponentref
     | deviceusedbythiscomponentref
     | findby
     )
   )>


<!ELEMENT providesidentifier (#PCDATA)>


<!ELEMENT componentsupportedinterface
   ( supportedidentifier
   , ( componentinstantiationref
     | findby
     )
   )>


<!ELEMENT supportedidentifier (#PCDATA)>


<!ELEMENT domainmanager
   ( namingservice )>


<!ELEMENT namingservice EMPTY>
```

```
<!ATTLIST namingservice
   name  CDATA #REQUIRED>


<!ELEMENT findby
   ( namingservice
   | domainfinder
   )>


<!ELEMENT domainfinder EMPTY>
<!ATTLIST domainfinder
   type  CDATA #REQUIRED
   name  CDATA #IMPLIED>


<!ELEMENT filesystemnames
   (filesystemname+
   )>


<!ELEMENT filesystemname EMPTY>
<!ATTLIST filesystemname
   mountname CDATA #REQUIRED
   deviceid CDATA #REQUIRED>
```

L.9.7 Domain Configuration Descriptor DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT domainmanagerconfiguration
    ( description?
    , domainmanagersoftpkg
    , services?
    )>
<!ATTLIST domainmanagerconfiguration
   id    ID #REQUIRED
   name  CDATA #REQUIRED>

<!ELEMENT description (#PCDATA)>

<!ELEMENT domainmanagersoftpkg
   ( localfile
   )>

<!ELEMENT localfile EMPTY>
<!ATTLIST localfile
     name CDATA #REQUIRED>

<!ELEMENT services
   ( service+
   )>

<!ELEMENT service
   ( usesidentifier
   , findby
   )>
```

```
<!ELEMENT usesidentifier (#PCDATA)>


<!ELEMENT findby
   ( namingservice
   | domainfinder
   )>


<!ELEMENT namingservice EMPTY>
<!ATTLIST namingservice
   name  CDATA #REQUIRED>


<!ELEMENT domainfinder EMPTY>
<!ATTLIST domainfinder
   type  CDATA #REQUIRED
   name  CDATA #IMPLIED>
```