TERMS, CONDITIONS & NOTICES

This document has been prepared by the work group of WiNnForum project SCA-2015-001 “Transceiver Next” to assist The Software Defined Radio Forum Inc. (or its successors or assigns, hereafter “the Forum”). It may be amended or withdrawn at a later time and it is not binding on any member of the Forum or of the Transceiver Next work group.

Contributors to this document that have submitted copyrighted materials (the Submission) to the Forum for use in this document retain copyright ownership of their original work, while at the same time granting the Forum a non-exclusive, irrevocable, worldwide, perpetual, royalty-free license under the Submitter’s copyrights in the Submission to reproduce, distribute, publish, display, perform, and create derivative works of the Submission based on that original work for the purpose of developing this document under the Forum's own copyright.

Permission is granted to the Forum’s participants to copy any portion of this document for legitimate purposes of the Forum. Copying for monetary gain or for other non-Forum related purposes is prohibited.

THIS DOCUMENT IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NON-INFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY USE OF THIS SPECIFICATION SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE FORUM, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER, DIRECTLY OR INDIRECTLY, ARISING FROM THE USE OF THIS DOCUMENT.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the specification set forth in this document, and to provide supporting documentation.

This document was developed following the Forum's policy on restricted or controlled information (Policy 009) to ensure that that the document can be shared openly with other member organizations around the world. Additional Information on this policy can be found here: http://www.wirelessinnovation.org/page/Policies_and_Procedures

Although this document contains no restricted or controlled information, the specific implementation of concepts contain herein may be controlled under the laws of the country of origin for that implementation. Readers are encouraged, therefore, to consult with a cognizant authority prior to any further development.

Wireless Innovation Forum™ and SDR Forum™ are trademarks of the Software Defined Radio Forum Inc.
# Table of Contents

TERMS, CONDITIONS & NOTICES .......................................................................................... i

Table of Contents .................................................................................................................. ii

List of Figures .......................................................................................................................... v

List of Tables ............................................................................................................................ vii

Contributors ............................................................................................................................ viii

Standard history ....................................................................................................................... viii

Transceiver Facility PIM Specification ...................................................................................... 1

1 Introduction ............................................................................................................................ 1

1.1 Specification approach ........................................................................................................ 1

1.1.1 Model Driven Architecture (MDA) .............................................................................. 1

1.1.2 Implementation feedback collection ............................................................................ 2

1.1.3 Conventions .................................................................................................................. 2

1.1.4 Document structure ..................................................................................................... 3

1.2 Transceiver concepts .......................................................................................................... 3

1.2.1 Channels ....................................................................................................................... 3

1.2.2 I/O signals .................................................................................................................... 4

1.2.3 Processing phases ......................................................................................................... 5

1.2.4 Transmission ................................................................................................................ 5

1.2.5 Reception ..................................................................................................................... 8

1.2.6 Inter-burst characterization ......................................................................................... 11

1.2.7 Transceiver time ......................................................................................................... 12

1.3 Transceiver API .................................................................................................................. 12

1.3.1 Services ....................................................................................................................... 12

1.3.2 Services groups .......................................................................................................... 12

1.3.3 Implementation of services ........................................................................................ 13

2 Services .................................................................................................................................. 15

2.1 Provide services .................................................................................................................. 15

2.2 Use services ..................................................................................................................... 15

2.3 States machines ............................................................................................................... 16

2.3.1 Channels ..................................................................................................................... 16

2.3.2 CreationControl ........................................................................................................... 21

2.3.3 RadioSilence ................................................................................................................. 24

2.3.4 Retuning ....................................................................................................................... 26

2.4 Services groups description ............................................................................................ 27

2.4.1 Transceiver::Management ......................................................................................... 27

2.4.2 Transceiver::BurstControl .......................................................................................... 28

2.4.3 Transceiver::BasebandSignal .................................................................................... 31

2.4.4 Transceiver::Tuning .................................................................................................... 33

2.4.5 Transceiver::Notifications .......................................................................................... 34

2.4.6 Transceiver::GainControl .......................................................................................... 36

2.4.7 Transceiver::TransceiverTime ..................................................................................... 37

2.4.8 Transceiver::Strobing .................................................................................................. 38

3 Service primitives and attributes ......................................................................................... 39

3.1 Service primitives .............................................................................................................. 39
3.1.1 Transceiver::Management::Reset.................................................................................. 39
3.1.2 Transceiver::Management::RadioSilence .................................................................. 40
3.1.3 Transceiver::BurstControl::DirectCreation ............................................................ 42
3.1.4 Transceiver::BurstControl::RelativeCreation ......................................................... 44
3.1.5 Transceiver::BurstControl::AbsoluteCreation ......................................................... 46
3.1.6 Transceiver::BurstControl::StrobedCreation ............................................................ 48
3.1.7 Transceiver::BurstControl::Termination .................................................................. 50
3.1.8 Transceiver::BasebandSignal::SamplesReception .................................................. 52
3.1.9 Transceiver::BasebandSignal::SamplesTransmission .............................................. 54
3.1.10 Transceiver::BasebandSignal::RxPacketsLengthControl ........................................ 56
3.1.11 Transceiver::Tuning::InitialTuning ........................................................................ 57
3.1.12 Transceiver::Tuning::Retuning .............................................................................. 60
3.1.13 Transceiver::Notifications::Events ....................................................................... 62
3.1.14 Transceiver::Notifications::Errors .......................................................... 64
3.1.15 Transceiver::GainControl::GainChanges ............................................................... 67
3.1.16 Transceiver::GainControl::GainLocking ................................................................. 68
3.1.17 Transceiver::TransceiverTime::TimeAccess ........................................................ 71
3.1.18 Transceiver::Strobing::ApplicationStrobe ............................................................ 73

3.2 Exceptions .................................................................................................................. 74
3.2.1 Specification ........................................................................................................... 74
3.2.2 Associated properties ............................................................................................. 76
3.2.3 Behavior requirements ............................................................................................ 76

3.3 Attributes ................................................................................................................. 77
3.3.1 Channels attributes ............................................................................................... 77
3.3.2 Processing attributes ............................................................................................. 78

3.4 Types .......................................................................................................................... 80
3.4.1 Base assumptions .................................................................................................... 80
3.4.2 BasebandPacket ..................................................................................................... 80
3.4.3 BlockLength ........................................................................................................... 80
3.4.4 BasebandSample .................................................................................................... 81
3.4.5 BurstNumber .......................................................................................................... 81
3.4.6 CarrierFreq ............................................................................................................ 81
3.4.7 Delay ...................................................................................................................... 81
3.4.8 Error ....................................................................................................................... 82
3.4.9 Event ...................................................................................................................... 82
3.4.10 Gain ...................................................................................................................... 82
3.4.11 IQ ......................................................................................................................... 83
3.4.12 MetaData .............................................................................................................. 83
3.4.13 PacketLength ........................................................................................................ 83
3.4.14 SampleNumber ..................................................................................................... 84
3.4.15 StrobeSource ........................................................................................................ 84
3.4.16 TimeSpec ............................................................................................................... 84
3.4.17 TuningPreset ........................................................................................................ 85

4 Properties ..................................................................................................................... 86
4.1 Introduction ............................................................................................................... 86
4.1.1 Properties ............................................................................................................. 86
4.1.2 Properties naming .................................................................................................... 86
4.1.3 Portability engineering support ............................................................................... 86
4.1.4 Profiles .................................................................................................................... 87
4.2 Structure .......................................................................................................................... 88
4.3 Behavior .......................................................................................................................... 90
4.4 Notifications .................................................................................................................. 91
4.5 Interface declaration ..................................................................................................... 93
4.6 Initialization .................................................................................................................. 93
4.7 Parameters validity ........................................................................................................ 94
4.8 Rapidity .......................................................................................................................... 95
4.9 Storage ........................................................................................................................... 97
4.10 Levels ............................................................................................................................. 97
4.11 Channelization ............................................................................................................. 97
4.12 Temporal accuracy ..................................................................................................... 100
4.13 Invocation lead time ................................................................................................... 100
4.14 Invocation delay ........................................................................................................... 101
4.15 Worst-case execution time (WCET) ........................................................................... 102
5 References ....................................................................................................................... 103
  5.1 Referenced documents ................................................................................................. 103
END OF THE DOCUMENT .................................................................................................... 104
List of Figures

Figure 1 Overview of Transceiver Facility 1
Figure 2 Principle of transmission processing phase 6
Figure 3 Transmit impulse response 7
Figure 4 Nominal and specific Tx bursts shapings 8
Figure 5 Principle of reception processing phase 9
Figure 6 Receive impulse response 10
Figure 7 Principle of inter-burst duration 11
Figure 8 Principle of inter-processing duration 11
Figure 9 Channels statechart 16
Figure 10 CreationControl statechart 22
Figure 11 RadioSilence statechart 25
Figure 12 Retuning statechart 26
Figure 13 Services of Management services group 27
Figure 14 Management::Reset interface 27
Figure 15 Management::RadioSilence interface 27
Figure 16 Services of BurstControl services group 28
Figure 17 BurstControl::DirectCreation interface 29
Figure 18 BurstControl::RelativeCreation interface 29
Figure 19 BurstControl::AbsoluteCreation interface 30
Figure 20 BurstControl::StrobedCreation interface 30
Figure 21 BurstControl::Termination interface 31
Figure 22 Services of BasebandSignal services group 31
Figure 23 BasebandSignal::SamplesReception interface 32
Figure 24 BasebandSignal::SamplesTransmission interface 32
Figure 25 BasebandSignal::RxPacketsLengthControl interface 33
Figure 26 Services of Tuning services group 33
Figure 27 Tuning::InitialTuning interface 34
Figure 28 Tuning::Retuning interface 34
Figure 29 Services of Notifications services group 34
Figure 30 Notifications::Events interface 35
Figure 31 Notifications::Errors interface 35
Figure 32 Services of GainControl services group 36
Figure 33 GainControl::GainChanges interface 36
Figure 34 GainControl::AGCActivation interface 37
Figure 35 Service of TransceiverTime services group 37
Figure 36 TransceiverTime::TimeAccess interface 38
Figure 37 Service of Strobing services group 38
Figure 38 Strobing::ApplicationStrobe interface 38
Figure 39 Principle of startRadioSilence() 40
Figure 40 Principle of stopRadioSilence() 41
Figure 41 Principle of startBurst() 42
Figure 42 Principle of scheduleRelativeBurst() 44
Figure 43 Principle of scheduleAbsoluteBurst() 46
Figure 44 Principle of scheduleStrobedBurst() 48
Figure 45  Principle of pushRxPacket() 52
Figure 46  Principle of pushTxPacket() 54
Figure 47  Principle of setTuning() 58
Figure 48  Principle of retune() 60
Figure 49  Principle of notifyEvent() 63
Figure 50  Principle of notifyError() 64
Figure 51  Principle of indicateGain() 67
Figure 52  Principle of lockGain() 69
Figure 53  Principle of unlockGain() 70
Figure 54  Principle of getCurrentTime() 71
Figure 55  Principle of getLastStartTime() 72
Figure 56  Specification of fields of channel masks 99
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Provide services of Transceiver API</td>
<td>15</td>
</tr>
<tr>
<td>Table 2</td>
<td>Use services of Transceiver API</td>
<td>15</td>
</tr>
<tr>
<td>Table 3</td>
<td>Specification of <code>startBurst()</code> parameters</td>
<td>43</td>
</tr>
<tr>
<td>Table 4</td>
<td>Specification of <code>scheduleRelativeBurst()</code> parameters</td>
<td>45</td>
</tr>
<tr>
<td>Table 5</td>
<td>Specification of <code>scheduleAbsoluteBurst()</code> parameters</td>
<td>47</td>
</tr>
<tr>
<td>Table 6</td>
<td>Specification of strobe sources</td>
<td>48</td>
</tr>
<tr>
<td>Table 7</td>
<td>Specification of <code>scheduleStrobedBurst()</code> parameters</td>
<td>49</td>
</tr>
<tr>
<td>Table 8</td>
<td>Specification of <code>setBlockLength()</code> parameters</td>
<td>50</td>
</tr>
<tr>
<td>Table 9</td>
<td>Specification of <code>pushRxPacket()</code> parameters</td>
<td>53</td>
</tr>
<tr>
<td>Table 10</td>
<td>Specification of <code>pushTxPacket()</code> parameters</td>
<td>55</td>
</tr>
<tr>
<td>Table 11</td>
<td>Specification of <code>setRxPacketsLength()</code> parameters</td>
<td>57</td>
</tr>
<tr>
<td>Table 12</td>
<td>Specification of <code>setTuning()</code> parameters</td>
<td>59</td>
</tr>
<tr>
<td>Table 13</td>
<td>Specification of <code>retune()</code> parameters</td>
<td>61</td>
</tr>
<tr>
<td>Table 14</td>
<td>Specification of events</td>
<td>63</td>
</tr>
<tr>
<td>Table 15</td>
<td>Specification of <code>notifyEvent()</code> parameters</td>
<td>63</td>
</tr>
<tr>
<td>Table 16</td>
<td>Specification of errors</td>
<td>65</td>
</tr>
<tr>
<td>Table 17</td>
<td>Specification of <code>notifyError()</code> parameters</td>
<td>66</td>
</tr>
<tr>
<td>Table 18</td>
<td>Specification of errors mitigation behaviors</td>
<td>67</td>
</tr>
<tr>
<td>Table 19</td>
<td>Specification of <code>indicateGain()</code> parameters</td>
<td>68</td>
</tr>
<tr>
<td>Table 20</td>
<td>Specification of <code>getCurrentTime()</code> parameters</td>
<td>71</td>
</tr>
<tr>
<td>Table 21</td>
<td>Specification of <code>getLastStartTime()</code> parameters</td>
<td>72</td>
</tr>
<tr>
<td>Table 22</td>
<td>Specification of general exceptions</td>
<td>74</td>
</tr>
<tr>
<td>Table 23</td>
<td>Specification of range exceptions</td>
<td>75</td>
</tr>
<tr>
<td>Table 24</td>
<td>Specification of MILT exceptions</td>
<td>76</td>
</tr>
<tr>
<td>Table 25</td>
<td>Structure properties</td>
<td>88</td>
</tr>
<tr>
<td>Table 26</td>
<td>Behavior properties</td>
<td>90</td>
</tr>
<tr>
<td>Table 27</td>
<td>Notification properties</td>
<td>91</td>
</tr>
<tr>
<td>Table 28</td>
<td>Interface declaration properties</td>
<td>93</td>
</tr>
<tr>
<td>Table 29</td>
<td>Initialization properties</td>
<td>93</td>
</tr>
<tr>
<td>Table 30</td>
<td>Parameters validity properties</td>
<td>94</td>
</tr>
<tr>
<td>Table 31</td>
<td>Rapidity properties</td>
<td>95</td>
</tr>
<tr>
<td>Table 32</td>
<td>Tuning conditions</td>
<td>96</td>
</tr>
<tr>
<td>Table 33</td>
<td>Duplex conditions</td>
<td>96</td>
</tr>
<tr>
<td>Table 34</td>
<td>Storage properties</td>
<td>97</td>
</tr>
<tr>
<td>Table 35</td>
<td>Level properties</td>
<td>97</td>
</tr>
<tr>
<td>Table 36</td>
<td>Channelization properties</td>
<td>98</td>
</tr>
<tr>
<td>Table 37</td>
<td>Temporal accuracy properties</td>
<td>100</td>
</tr>
<tr>
<td>Table 38</td>
<td>Invocation lead time properties</td>
<td>101</td>
</tr>
<tr>
<td>Table 39</td>
<td>Invocation delay properties</td>
<td>101</td>
</tr>
<tr>
<td>Table 40</td>
<td>WCET properties of provide operations</td>
<td>102</td>
</tr>
<tr>
<td>Table 41</td>
<td>WCET properties of use operations</td>
<td>102</td>
</tr>
</tbody>
</table>
Contributors

The following individuals and their organization of affiliation are credited as Contributors to development of the specification, for having been involved in the work group that developed the draft then approved by WInnForum member organizations:

- Marc Adrat, FKIE,
- Claude Bélisle, NordiaSoft,
- Eric Campbell, Harris Corporation,
- Jean-Philippe Delahaye, DGA,
- Antonio Di Rocco, Leonardo,
- David Hagood, Cobham,
- Frédéric Leroy, ENSTA,
- Sarah Miller, Rockwell-Collins,
- David Murotake, Hitachi Kokusai Electric,
- Eric Nicollet, Thales Communications & Security,
- Peter Troll, Rohde & Schwarz,
- Dmitri Zvernick, NordiaSoft.

Standard history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1.0.0</td>
<td>28 January 2009</td>
<td>Initial release.</td>
</tr>
<tr>
<td>V2.0.0</td>
<td>9 November 2017</td>
<td>Release of the PIM specification for V2.0.0. Major upgrade of V1.0.0.</td>
</tr>
<tr>
<td>V2.1.0</td>
<td>20 January 2022</td>
<td>Addition of the Native C++ PSM specification, SCA PSM specification and FPGA PSM specification. The PIM specification content is identical to V2.0.0.</td>
</tr>
</tbody>
</table>
Transceiver Facility PIM Specification

1 Introduction

This document WINNF-TS-0008-V2.1.0 is the PIM specification (Platform-Independent Model) of WInnForum transceiver facility V2.1.0.

The transceiver facility V2.1.0 is also composed of the following detached appendices:

- WINNF-TS-0008-App01-V2.1.0 Transceiver Facility Native C++ PSM specification,
- WINNF-TS-0008-App02-V2.1.0 Transceiver Facility SCA PSM specification,
- WINNF-TS-0008-App03-V2.1.0 Transceiver Facility FPGA PSM specification,
- WINNF-TS-0008-App04-V2.1.0 Transceiver Facility Absolute Time Use Case.

V2.1.0 complements V2.0.0 with three PSM specifications.

The Transceiver Facility standardizes a service-oriented Transceiver Application Programming Interface (Transceiver API) and associated Transceiver Properties, in support of portability of radio applications and hospitality of reconfigurable transceiver implementations.

The transceiver is the processing stage situated between the antenna and the radio physical layer baseband processing. Its I/O signals are the baseband signal and the radio signal (see section 1.2.2), as depicted in following figure:

![Figure 1 Overview of Transceiver Facility](image)

1.1 Specification approach

1.1.1 Model Driven Architecture (MDA)

The Transceiver Facility structure is inspired by application of the Object Management Group (OMG) Model Driven Architecture (MDA) approach (see [Ref1]) to the technical domain of physical layer engineering of software-defined radio (SDR) systems.

The Transceiver Facility is composed of a core specification, denoted as the Platform-Independent Model (PIM) specification (this document) and appendices.

The core specification answers to the definition of a PIM provided by [Ref1]: “A PIM exhibits a sufficient degree of independence so as to enable its mapping to one or more platforms. This is commonly achieved by defining a set of services in a way that abstracts out technical details. Other models then specify a realization of these services in a platform specific manner.”.
Appendices are Platform-Specific Model (PSM) specifications specified for a number of programming paradigms supporting implementation of the PIM software interfaces.

The PSM specifications answer to the definition of a PSM provided by [Ref1]: “A PSM combines the specifications in the PIM with the details required to stipulate how a system uses a particular type of platform. If the PSM does not include all of the details necessary to produce an implementation of that platform it is considered abstract (meaning that it relies on other explicit or implicit models which do contain the necessary details).”.

When no standard PSM specification is applicable, a non-standard PSM has to be formally specified through a specification structured like standard PSMs.

1.1.2 Implementation feedback collection

Users of a core specification and standard PSM specifications are invited to submit implementation feedback to the WInnForum for consideration in perspective improvement of the Transceiver Facility.

Users of a non-standard PSM specification are invited to submit the non-standard specification as an input document to the WInnForum to be considered for future inclusion in the Transceiver Facility.

1.1.3 Conventions

The PIM specification refers itself as “the specification” in the remaining of the document.

A normative clause of the specification is a particular sentence that can be:

- A definition: defines a general concept, contains “is/are defined as”; name of the defined concept is formatted in italics,
- A declaration: specifies a formal concept (e.g. a state, an interface, an error), contains “is/are specified as/by”; name of the declared concept is formatted according to its nature,
- A requirement: specifies a condition to be respected by a transceiver, contains “shall”.

The term "unspecified" indicates an aspect that is not specified by the specification, more specific aspects being left to user’s decisions.
1.1.4 Document structure

The PIM specification is structured as follows:

- Section 1, Introduction: defines essential aspects, provides an overview of the specified services groups,
- Section 2, Services: specifies states machines, API services groups, provide and use services,
- Section 3, Service Primitives and Attributes: specifies API primitives, exceptions, attributes and types,
- Section 4, Properties: specifies properties characterizing transceiver instances,
- Section 5, PSM specifications: specifies rules pertaining to derived PSM specifications.

1.2 Transceiver concepts

A **transceiver** is defined as a subsystem of a radio platform that transforms, when it transmits, baseband signal(s) into radio signal(s) and, when it receives, radio signal(s) into baseband signal(s).

A **transceiver instance** is defined as one particular implementation of a transceiver.

One or several transceiver instances can be available on a radio platform and one or several transceiver instances can be used by a radio application.

The remainder of the specification is applicable to any particular transceiver instance, assumed fully independent of any other transceiver instance eventually available on a given radio platform.

1.2.1 Channels

1.2.1.1 Tx channels

A **transmit channel** (**Tx channel**) is defined as an elementary part of a transceiver instance that transforms, when it transmits, one baseband signal (see section 1.2.2.1) into one radio signal (see section 1.2.2.2).

A transmission is defined as a phase during which a Tx channel continuously transmits.

*Up-conversion* is defined as the signal processing performed by a Tx channel during a transmission.

A transceiver instance can have zero to several Tx channels. All Tx channels of a specific transceiver instance are controlled simultaneously by the radio application and operate synchronously.

**TX_CHANNELS** (see section 4.2) specifies the number of Tx channels of a transceiver instance.

1.2.1.2 Rx channels

A **receive channel** (**Rx channel**) is defined as an elementary part of a transceiver instance that transforms, when it receives, one radio signal into one baseband signal.
A *reception* is defined as a phase during which an *Rx channel* continuously receives.

*Down-conversion* is defined as the signal processing performed by an *Rx channel* during a *reception*.

A *transceiver instance* can have zero to several *Rx channels*. All *Rx channels* of a specific *transceiver instance* are controlled simultaneously by the *radio application* and operate synchronously.

**RX_CHANNELS** (see section 4.2) specifies the number of *Rx channels* of a *transceiver instance*.

### 1.2.1.3 Transceiver categories

A *simplex transceiver* is defined as a *transceiver* with transmit or receive capability, but not both. A *simplex transceiver* has one or many *Tx channels*, or one or many *Rx channels*.

A *duplex transceiver* is defined as a *transceiver* with one or many *Tx channels* and one or many *Rx channels*.

A *full-duplex transceiver* is defined as a *duplex transceiver* which *transmission* and *reception* phases are fully independent and can occur simultaneously.

A *half-duplex transceiver* is defined as a *duplex transceiver* with no simultaneous *transmission* and *reception* phases, due to sharing of critical processing resources between its *Tx channels* and *Rx channels*.

**DUPLEX** (see section 4.2) specifies if a *duplex transceiver* is *half-duplex* or *full-duplex*.

### 1.2.2 I/O signals

#### 1.2.2.1 Baseband signal

A *baseband signal* ($s_{BB}$) is defined as the complex digital signal exchanged between a *radio application* and *Tx channels* or *Rx channels*.

The *baseband sampling frequency* ($F_s^{BB}$) is defined as the sampling frequency of a *baseband signal*.

A *baseband sample* ($s_{BB}[n]$) is defined as a complex sample of the *baseband signal*, with $s_{BB}[n] = I + i \cdot Q$, where $i = \sqrt{-1}$.

The in-phase component ($I$) of a *baseband sample* is defined as its real part.

The quadrature component ($Q$) of a *baseband sample* is defined as its imaginary part.

$s_{BB}$ is defined as the Fourier transform of $s_{BB}$.

$L_{BB}$ is defined as the level of the *baseband signal* expressed in decibels relative to full scale (dBFS) for the applied numerical representation.

The *full-scale* (FS) of the numerical representation of the *baseband signal* is specified as, depending on value of **IQ_TYPE** (see section 4.5):
\[ L_{BB} \text{ shall be computed according to } L_{BB} = 10 \log_{10} \left( \frac{\sum_{i=0}^{N-1} |S_{BB}[n_0+i]|^2}{FS^2} \right). \]

1.2.2.2 Radio signal

The radio signal \( s_{RF} \) is defined as the analogue voltage signal at the output of Tx channel, during a Transmission, or at the input of Rx channel, during a Reception.

Radio signal is typically taken at the antenna connector, but can be defined elsewhere depending on usage context.

The carrier frequency \( f_c \) is defined as the radio frequency around which the radio signal spectrum is positioned.

Note: the carrier frequency is the center frequency of the Tx signal measured spectrum when the baseband signal is symmetrical. It is not always the case, e.g. in the case of single side band modulations.

\( \hat{s}_{RF} \) is defined as the Fourier transform of \( s_{RF} \).

\( L_{RF} \) is defined as the level of the radio signal expressed in decibels relative to one milliwatt (dBm).

1.2.3 Processing phases

A processing phase is defined as a continuous period of time during which Rx channels or Tx channels perform a signal processing transformation.

The activation time of a processing phase is defined as the time at which the processing phase starts.

The termination time of a processing phase is defined as the time at which the processing phase stops.

A baseband block is defined as the baseband signal exchanged between a radio application and one Rx channel or one Tx channel during a processing phase.

The sample number of a baseband sample is defined as its position within a baseband block, starting at 1 for the first sample.

1.2.4 Transmission

A transmission is defined as the processing phase of Tx channels.
The following figure illustrates the principle of a transmission:

![Figure 2 Principle of transmission processing phase]

1.2.4.1 Boundary signals

A transmit forwarded block (Tx forwarded block) is defined as a the baseband block sent by a radio application to one Tx channel during a transmission.

A transmit packet (Tx packet) is defined as a one elementary set of baseband samples successively sent by a radio application to one Tx channel for transfer of a Tx forwarded block.

A transmit processed block (Tx processed block) is defined as a the part of the Tx forwarded block up-converted by one Tx channel during a transmission.

Correct operation of Tx channels requires that the level of baseband signal is within a particular range.

The upper bound of this range generally corresponds to the maximum level maintaining Tx channels linearity. The lower bound of this range generally corresponds to the level required for the baseband signal to be able to drive the Tx channels processing.

A transmit burst (Tx burst) is defined as the radio signal sent by one Tx channel to the antenna during a transmission.

The core of a Tx burst is defined as the part of the Tx burst without its ramp-up and ramp-down.

1.2.4.2 Start and stop times

The start time of a Tx burst is defined as the start time of its core.

The start time of a Tx burst generally happens up-conversion latency after activation time.

The stop time of a Tx burst is defined as the stop time of its core.

The stop time of a Tx burst generally happens up-conversion latency before termination time.
1.2.4.3 Transmit transfer function

An ideal up-conversion generates a radio signal which spectrum is the zero-centered spectrum of the baseband signal translated around the carrier frequency, with application of an ideal low-pass filter of bandwidth $B$ to select the spectrum portion of interest.

An ideal up-conversion obeys to the following equation:

$$\dot{s}_{RF}(f + f_c) = \alpha.\text{rect}(f/B).\dot{s}_{BB}(f), \quad f \in [-F_{BB}^s / 2; +F_{BB}^s / 2]$$  \hspace{1cm} \text{Eq. 1},

where:

- rect( ) is the rectangular function,
- $\alpha$ is a real coefficient reflecting the up-conversion gain.

The transmit transfer function ($H_{TX}$) is defined as the transfer function nearing the ideal low-pass filter of the ideal up-conversion that is implemented by a Tx channel.

CHANNEL_MASK (see section 4.10) specify the frequency domain mask into which $H_{TX}$ fits.

The actual up-conversion performed by a Tx channel obeys to the up-conversion formula:

$$\dot{s}_{RF}(f + f_c) = H_{TX}(f).\dot{s}_{BB}(f), \quad f \in [-F_{BB}^s / 2; +F_{BB}^s / 2]$$  \hspace{1cm} \text{Eq. 2}.

The transmit impulse response ($h_{TX}$) is defined as the non-causal equivalent impulse response corresponding to up-conversion, symmetrical around the y-axis, with up-conversion latency equal to the half of its domain:

One has:

$$s_{RF}(t) = \sum_{k=0}^{L-1} \left[ (\Re(s_{BB}[k]).\cos(2\pi f_c t) - \Im(s_{BB}[k]).\sin(2\pi f_c t)).h_{TX}(t - t_s - k/F_{BB}^s) \right],$$  \hspace{1cm} \text{Eq. 3},

where:

- $L$ denotes the transmit block length,
- $\Re(\ )$ and $\Im(\ )$ denote the real and imaginary part of a complex number,
- $t_s$ denotes the start time.

Further technical information is available in technical literature, e.g. [Ref2] and [Ref3].
1.2.4.4 Transmit gain

The transmit gain ($G_{TX}$) of a transmission is specified as $G_{TX} = L_{RF} - L_{BB}$.

1.2.4.5 Tx shaping

Nominal shaping is defined as the case where the ramp-up and ramp-down parts of the Tx burst are the ramp-up and ramp-down of up-conversion.

Ad-hoc shaping is defined as the case where the ramp-up or ramp-down parts of the Tx burst modifies the ramp-up and ramp-down of up-conversion.

Ad-hoc shaping is unspecified, and has to be specified according to the radio application needs. **TX_SHAPING** (see section 4.2) specifies if the shaping is nominal or specific:

![Nominal and specific Tx bursts shapings](image)

**Figure 4** Nominal and specific Tx bursts shapings

1.2.5 Reception

A reception is defined as the processing phase of Rx channels.
The following figure illustrates the principle of a reception:

![Figure 5 Principle of reception processing phase](image)

1.2.5.1 Boundary signals

A receive burst (Rx burst) is defined as the radio signal sent by the antenna to one Rx channel during a reception.

Correct operation of Rx channels requires that the level of radio signal is within a particular range. The upper bound of this range generally corresponds to the protection of Rx channels against high level signals. The lower bound of this range generally corresponds to the expected sensitivity.

A receive block (Rx block) is defined as the baseband block sent by one Rx channel to a radio application during a reception.

A receive packet (Rx packet) is defined as a one elementary set of baseband samples successively sent by one Rx channel to a radio application for transfer of an Rx block.

Correct operation of a receiving radio application requires that the level of baseband signal is within a particular range.

The upper bound of this range generally corresponds to the level maximum level allowed to avoid saturation of the radio application processing. The lower bound of this range generally corresponds to the level under which the quantization noise impacts the reception performance.

1.2.5.2 Start and stop times

The start time of an Rx burst is defined as the time when the Rx burst starts.

The start time of an Rx burst is equal to its activation time.

The stop time of an Rx burst is defined as the time when the Rx burst stops.
The stop time of an Rx burst generally happens two times down-conversion latency before termination time, in order for the down-conversion processing chain to be fully flushed.

1.2.5.3 Receive transfer function

An ideal down-conversion generates a baseband signal which zero-centered spectrum is obtained from a perfect transposition of the radio signal spectrum considered around the carrier frequency, with application of an ideal low-pass filter of bandwidth $B$ to select the spectrum portion of interest.

An ideal down-conversion obeys to the following equation:

$$\hat{s}_{BB}(f) = \alpha \cdot \text{rect}(f/B) \cdot \hat{s}_{RF}(f - f_c), \ f \in [-F_{BB}/2; +F_{BB}/2]$$  \hspace{1cm} \text{Eq. 4},

where:

- \text{rect( )} is the rectangular function,
- $\alpha$ is a real coefficient reflecting the down-conversion gain.

The receive transfer function ($H_{RX}$) is defined as the transfer function nearing the ideal low-pass filter of the ideal down-conversion that is implemented by an Rx channel.

\text{CHANNEL\_MASK} (see section 4.10) specify the frequency domain mask into which $H_{RX}$ fits.

The actual down-conversion performed by an Rx channel obeys to the down-conversion formula:

$$\hat{s}_{BB}(f) = H_{RX}(f) \cdot \hat{s}_{RF}(f - f_c), \ f \in [-F_{BB}/2; +F_{BB}/2]$$  \hspace{1cm} \text{Eq. 5}.

The receive impulse response ($h_{RX}$) is defined as the non-causal equivalent impulse response corresponding to down-conversion, symmetrical around the y-axis, with down-conversion latency equal to the half of its domain:
One has:

\[
S_{BB}[k] = \left( s_{RF}(t) + i \cdot \hat{s}_{RF}(t) \right) \cdot e^{-2\pi if_{c}t} \ast h_{Tx}(t), t = t_{s} + k/F_{2}^{BB}, k \in [0; L - 1] \quad \text{Eq. 6,}
\]

where:
- \( \hat{s}_{RF}(t) \) denotes the Hilbert transform of \( s_{RF}(t) \),
- \( \ast \) denotes the convolution product operator,
- \( t_{s} \) denotes the start time,
- \( L \) denotes the receive block length.

Further technical information is available in technical literature, e.g. [Ref2] and [Ref3].

1.2.5.4 Receive gain

The receive gain \( (G_{RX}) \) of a reception is specified as \( G_{RX} = L_{BB} - L_{RF} \).

1.2.6 Inter-burst characterization

The inter-burst duration is defined as the duration of the period occurring between two consecutive core bursts.

INTER-BURST (see section 4.8) specifies the minimum value possibly taken by inter-burst duration.

The inter-processing duration is defined as the duration of the period occurring between two consecutive processing phases, as illustrated in the following figure:
INTER-PROCESSING (see section 4.8) specifies the minimum value possibly taken by inter-processing duration.

Inter-burst duration and inter-processing duration is at least equal to the tuning duration between the two bursts.

In addition to tuning duration, inter-burst duration comprises the duration of the bursts ramp-up and ramp-down, while inter-processing duration does not.

1.2.7 Transceiver time

Transceiver time is defined as the monotonic time implemented by a transceiver instance, used to exchange time specification of events related to operation of the transceiver.

Transceiver time is essentially used in the case absolute burst creation (see section 2.4.2.3), and values of transceiver time can be accessed by radio applications using a dedicated service (see section 2.4.7).

1.3 Transceiver API

The Transceiver API is defined as the service-oriented Application Programming Interface (API) of the specification.

1.3.1 Services

A service of the Transceiver API is defined as a capability of a transceiver instance that exchanges messages with a radio application in compliance with one attached software interface and the specified behavior.

A service interface is defined as the particular Interface Description Language (IDL) software interface attached to a service.

A service and its service interface have the same name.

A provide service is defined as a service whose service interface is used by a radio application and provided by a transceiver instance.

A use service is defined as a service whose service interface is used by a transceiver instance and provided by a radio application.

1.3.2 Services groups

A services group of the Transceiver API is defined as a set of provide services and use services sharing a common purpose.

The module of a service group is defined as the IDL module of the interfaces of the services of the services group.

A services group and its module have the same name.
The following services groups are specified:

- **Management**: general control,
- **BurstControl**: creation and termination of bursts,
- **BasebandSignal**: packet-based exchange of baseband blocks,
- **Tuning**: control of the tuning parameters,
- **Notifications**: notification of events and errors to the radio application,
- **GainControl**: automated gain control,
- **TransceiverTime**: access to transceiver time,
- **Strobing**: trigger of strobes for creation of strobed bursts.

### 1.3.3 Implementation of services

An active instance of a service is defined as a running implementation of the service that is connected to the radio application in conformance with the service interface.

#### 1.3.3.1 Access capabilities

The transceiver instances access capability is defined as the capability for the radio application software to access, before the CONFIGURED state is reached, to the transceiver instances it uses.

The active services access capability is defined as the capability for the radio application software to access, before the CONFIGURED state is reached, to the active services instances of the transceiver instances it uses.

The solution for transceiver instance access and active services access has to be specified by the applied PSM specification.

#### 1.3.3.2 Tx channels services

**SamplesTransmission** (see section 2.4.2.5) is the service enabling Tx forwarded block exchange.

A transceiver instance shall have one active instance of **SamplesTransmission** per Tx channel.

This implies **TX_CHANNELS** instances of the service are implemented.

**TX_SERVICES** (see section 4.2) specifies, if **TX_CHANNELS** > 0, the set of services attached to Tx channels.

A transceiver instance shall have, for each service attached to Tx channels, one active instance of the service that jointly applies to all Tx channels.

#### 1.3.3.3 Rx channels services

**SamplesReception** (see section 2.4.2.5) is the service enabling Rx block exchange.

A transceiver instance shall have one active instance of **SamplesReception** per Rx channel.
This implies $\text{RX\_CHANNELS}$ instances of the service are implemented.

RX\_SERVICES (see section 4.2) specifies, if $\text{RX\_CHANNELS} > 0$, the set of services attached to Rx channels.

A transceiver instance shall have, for each service attached to Rx channels, one active instance of the service that jointly applies to all Rx channels.
2 Services

2.1 Provide services

The following table lists the *provide services* of the API (used by a *radio application* and provided by a *transceiver instance*, see section 1.3.1):

<table>
<thead>
<tr>
<th>Services groups / Modules</th>
<th>Services / Interfaces</th>
<th>Primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>::Management::Reset</td>
<td>reset()</td>
</tr>
<tr>
<td></td>
<td>::Management::RadioSilence</td>
<td>startRadioSilence()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>stopRadioSilence()</td>
</tr>
<tr>
<td>BurstControl</td>
<td>::BurstControl::DirectCreation</td>
<td>startBurst()</td>
</tr>
<tr>
<td></td>
<td>::BurstControl::RelativeCreation</td>
<td>scheduleRelativeBurst()</td>
</tr>
<tr>
<td></td>
<td>::BurstControl::AbsoluteCreation</td>
<td>scheduleAbsoluteBurst()</td>
</tr>
<tr>
<td></td>
<td>::BurstControl::StrobedCreation</td>
<td>scheduleStrobedBurst()</td>
</tr>
<tr>
<td></td>
<td>::BurstControl::Termination</td>
<td>setBlockLength()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>stopBurst()</td>
</tr>
<tr>
<td>BasebandSignal</td>
<td>::BasebandSignal::SamplesTransmission</td>
<td>pushTxPacket()</td>
</tr>
<tr>
<td></td>
<td>::BasebandSignal::RxPacketsLengthControl</td>
<td>setRxPacketsLength()</td>
</tr>
<tr>
<td>Tuning</td>
<td>::Tuning::InitialTuning</td>
<td>setTuning()</td>
</tr>
<tr>
<td></td>
<td>::Tuning::Retuning</td>
<td>retune()</td>
</tr>
<tr>
<td>GainControl</td>
<td>::GainControl::GainLocking</td>
<td>lockGain()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unlockGain()</td>
</tr>
<tr>
<td>TransceiverTime</td>
<td>::TransceiverTime::TimeAccess</td>
<td>getCurrentTime()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getLastStartTime()</td>
</tr>
<tr>
<td>Strobing</td>
<td>::Strobing::ApplicationStrobe</td>
<td>triggerStrobe()</td>
</tr>
</tbody>
</table>

Table 1 Provide services of Transceiver API

2.2 Use services

The following table lists the *use services* of the API (provided by a *radio application* and used by a *transceiver instance*, see section 1.3.1):

<table>
<thead>
<tr>
<th>Services groups</th>
<th>Service / Interface</th>
<th>Primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>BasebandSignal</td>
<td>::BasebandSignal::SamplesReception</td>
<td>pushRxPacket()</td>
</tr>
<tr>
<td>Notifications</td>
<td>::Notifications::Events</td>
<td>notifyEvent()</td>
</tr>
<tr>
<td></td>
<td>::Notifications::Errors</td>
<td>notifyError()</td>
</tr>
<tr>
<td>GainControl</td>
<td>::GainControl::GainChanges</td>
<td>indicateGain()</td>
</tr>
</tbody>
</table>

Table 2 Use services of Transceiver API
2.3 States machines

The state machines specified in this section and their associated statecharts aim to comply with the OMG Unified Modeling Language v2.5, as specified in [Ref4].

All specified transitions are instantaneous.

Errors and exceptions handling are not modeled by the specified state machines.

2.3.1 Channels

 Channels is specified as the main state machine followed by channels of a transceiver instance.

An instance of Channels is simultaneously followed by all Tx channels of a transceiver instance.

An instance of Channels is simultaneously followed by all Rx channels of a transceiver instance.

The instances of Channels in a half-duplex transceiver are not independent: if channels are in TUNING or PROCESSING state, the other channels cannot be in one of those two states.

The following figure is the statechart of Channels state machine:

![Channels statechart](image)

Figure 9 Channels statechart

2.3.1.1 States

2.3.1.1.1 CONFIGURED

 CONFIGURED is specified as the main state of Channels during which channels of a transceiver instance are configured according to the needs of a supported radio application.
CONFIGURED is reached by the channels of a transceiver instance when they:

- Comply with the values of properties specified for the supported radio application (see section 4),
- Have attributes set to their initial values (see section 3.3),
- Can interact with the radio application according to interfaces of active services.

CONFIGURED decomposes into OPERATING and RESETTING sub-states.

Its entry transition brings to the OPERATING sub-state.

How CONFIGURED is reached has to be specified by the applied PSM specification.

2.3.1.1.2 OPERATING

OPERATING is specified as the sub-state of CONFIGURED during which channels are operational.

OPERATING decomposes into IDLE, TUNING and PROCESSING sub-states.

Its entry transition brings to the IDLE sub-state.

2.3.1.1.3 IDLE

IDLE is specified as the sub-state of OPERATING during which channels are inactive.

2.3.1.1.4 TUNING

TUNING is specified as the sub-state of OPERATING during which channels are tuned in accordance with the applicable tuning parameters set, as defined by CreationControl (see section 2.3.2).

Note: the concept of tuning of the specification is larger than only changing value of carrier frequency. It can imply modification of tuning preset and gain.

A channel shall, during the TUNING state, set the value of applicableTuningPreset attribute according to value of requestedTuningPreset:

- If equal to UndefinedTuningPreset: keep the value of applicableTuningPreset used for the previous burst,
- If not equal to UndefinedTuningPreset: apply requestedTuningPreset as the new value of applicableTuningPreset.

A channel shall, during TUNING state, set the value of applicableCarrierFreq attribute depending on value of requestedCarrierFreq:

- If equal to UndefinedCarrierFreq: keep the value of applicableCarrierFreq at termination of the previous burst,
- If not equal to UndefinedCarrierFreq: apply requestedCarrierFreq as the new value of applicableCarrierFreq.
A Tx channel shall, during TUNING state, set the value of applicableGain attribute depending on value of requestedGain:

- If equal to UndefinedGain: keep the value of applicableGain at termination of the previous burst,  
- If not equal to UndefinedGain: apply requestedGain as the new value of applicableGain.

Usage of requestedGain by an Rx channel is unspecified.

TUNING_DURATION (see section 4.8) specifies the maximum duration of TUNING state (see section 2.3.4).

2.3.1.1.5 PROCESSING

PROCESSING is specified as the sub-state of OPERATING during which channels are in a processing phase (transmission for Tx channels, reception for Rx channels) (see section 1.2.3, 1.2.4 and 1.2.5).

Tx channels requirements

Tx channels shall, during PROCESSING state, initiate up-conversion:

- With first sample of Tx processed block equal to first sample of Tx forwarded block,  
- With a ramp-up signal generated in accordance with TX_SHAPING (see section 4.2).

Tx channels shall, during a PROCESSING state, increment value of sampleCount (see section 3.3.1.2) each time a new baseband sample of the Tx forwarded block enters in up-conversion.

The valid input level range of Tx channels is defined as the interval \([\text{TX_MIN_BASEBAND_LEVEL}; \text{TX_MAX_BASEBAND_LEVEL}]\) (see section 4.10).

Tx channels shall, during a PROCESSING state and so long as the baseband signal is within the valid input level range, perform up-conversion in conformance with the up-conversion formula (see section 1.2.4).

Tx channels shall exhibit, during PROCESSING state and so long as the baseband signal level is within the valid input level range, an actual gain that belongs to applicableGain ± GAIN_ACC (see section 4.10).

Automatic level control (ALC)

Automatic level control (ALC) is defined as the capability for a Tx channel to automatically adjust the actually applied transmit gain, during early phase of the transmission, in order to radiate a desired level of radio signal.

ALC (see section 4.3) specifies the nature of the applied ALC.

Tx channels shall, during PROCESSING state and if ALC is equal to noALC, implement no ALC.

Tx channels shall, during PROCESSING state and if ALC is equal to activeALC, implement ALC.

Further aspects of the implemented ALC are unspecified.
Adjustment in transmit gain realized by an active ALC can be indicated to the radio application using the GainControl service (see section 2.4.6).

**Rx channels requirements**

*Rx channels shall*, during **PROCESSING** state, initiate down-conversion:

- Without transferring ramp-up samples to the radio application,
- With first sample of the Rx block equal to the sample following the ramp-up samples.

The valid input level range of *Rx channels* is defined as the interval \([RX\_MIN\_ RADIO\_ LEVEL \; RX\_MAX\_ RADIO\_ LEVEL]\) (see section 4.10).

*Rx channels shall*, during a **PROCESSING** state and so long the radio signal is within the valid input level range, perform down-conversion in conformance with the down-conversion formula (see section 1.2.5).

*Rx channels shall*, during a **PROCESSING** state, increment value of `sampleCount` (see section 3.3.1.2) each time a new baseband sample generated by down-conversion is assigned to an Rx packet.

The valid output level range of *Rx channels* is defined as the interval \([RX\_MIN\_ BASEBAND\_ LEVEL \; RX\_MAX\_ BASEBAND\_ LEVEL]\) (see section 4.10).

*Rx channels shall*, during **PROCESSING** state and so long the radio signal is within the valid input level range, deliver an output baseband signal which level is within the valid output level range.

**Automatic gain control (AGC)**

Automatic gain control (AGC) is defined as the capability for a *Rx channel* to automatically change the receive gain in order to deliver a baseband signal which meets the specified level requirements.

AGC (see section 4.3) specifies the nature of the applied AGC.

*Rx channels shall*, during **PROCESSING** state and if AGC is equal to `noAGC`, implement no AGC.

*Rx channels shall*, during **PROCESSING** state and if AGC is equal to `earlyControl`, implement an AGC that sets the receive gain at beginning of the Rx burst, to a value that is then kept constant for the remainder of the burst.

`EARLY_AGC_DELAY` (see section 4.8) specifies the delay available after start time of a Rx burst for an earlyControl AGC to have set the receive gain.

*Rx channels shall*, during **PROCESSING** state and if AGC is equal to `permanentControl`, implement an AGC that remains active during the full Rx burst.

Further aspects of the implemented AGC are unspecified.

For *Rx channels* implementing AGC, changes in receive gain can be indicated to the radio application using the GainChanges service (see section 2.4.6).

For *Rx channels* implementing a permanent AGC, the AGC can be deactivated and reactivated using the AGCActivation service (see section 2.4.6).
Channelization requirements

Channels shall exhibit, during PROCESSING state and so long as input signal level is within the valid input level range, an actual transfer function that fits into the mask defined by fields of CHANNEL_MASK (see section 4.10).

Channels shall exhibit, during PROCESSING state and so long as the input signal level is within the valid input level range, an actual baseband sampling frequency \( F_{BB} \) that belongs to CHANNEL_MASK.basebandSamplingFreq ± SAMPLING_FREQ_ACC (see section 4.10).

Channels shall exhibit, during PROCESSING state and so long as the input signal level is within the valid input level range, an actual carrier frequency that belongs to applicableCarrierFreq ± CARRIER_FREQ_ACC (see section 4.10).

Termination requirements

The last processed sample of a burst is defined as the sample of the processed block with a sample number equal to applicableBurstLength.

Note: value of applicableBurstLength can be set by a creation operation (see section 2.4.2) or updated by setBlockLength() or stopBurst() (see section 3.1.7).

Tx channels shall, during PROCESSING state:

- Make the sample of Tx forwarded block with sample number equal to applicableBurstLength the last sample of the Tx processed block,
- Discard any sample of the Tx forwarded block after the last sample,
- Use null flushing baseband samples until ramp-down is completed.

Channels shall trigger a ProcessingStop transition once ramp-down is completed and, for Tx channels, once the Tx forwarded block has been ended by the radio application.

Rx channels shall, during PROCESSING state, terminate down-conversion without transferring ramp-down samples to the radio application.

2.3.1.1.6 RESETTING

RESETTING is specified as the sub-state of CONFIGURED during which channels reset.

RESETTING is completed by channels of a transceiver instance once:

- Attributes are set back to their initial values (see section 3.3),
- Any previously used storage is cleared: for creation operation (see sections 3.1.3, 3.1.4, 3.1.5 and 3.1.6), tuning parameters set (see section 3.1.11) or baseband samples of Tx channels (see section 3.1.9).

2.3.1.2 Transitions

2.3.1.2.1 ResetCompleted

ResetCompleted is specified as the transition from RESETTING to IDLE.

It is triggered once channels have completed the RESETTING state.
2.3.1.2.2 TuningStart

TuningStart is specified as the transition from IDLE state to TUNING.

It is triggered under control of CreationControl (see section 2.3.2).

2.3.1.2.3 TuningStop

TuningStop is specified as the transition from TUNING to IDLE.

It is triggered once channels have completed the TUNING state.

2.3.1.2.4 ProcessingStart

ProcessingStart is specified as the transition from IDLE to PROCESSING.

It is triggered under control of CreationControl (see section 2.3.2).

2.3.1.2.5 ProcessingStop

ProcessingStop is specified as the transition from PROCESSING to IDLE.

It is triggered by PROCESSING based on knowledge of last processed sample (see section 2.3.1.1.5).

2.3.1.2.6 RuntimeReset

RuntimeReset is specified as the transition from OPERATING toResetting.

It is triggered upon call of reset() (see section 3.1.1) by the radio application.

2.3.2 CreationControl

CreationControl is specified as the autonomous process followed by a transceiver instance for the control of creation of the bursts executed by channels.

An instance of CreationControl applies to all Tx channels of a transceiver instance.

An instance of CreationControl applies to all Rx channels of a transceiver instance.
The following figure is the statechart of *CreationControl* state machine:

![Image of statechart](image)

**Figure 10  CreationControl statechart**

2.3.2.1  States

2.3.2.1.1  AWAITING

**AWAITING** is specified as the state of *CreationControl* during which a *transceiver instance* stays until it triggers a *burst creation*.

A *transceiver instance shall*, during **AWAITING**, wait until a *creation command* is available in storage.

2.3.2.1.2  CREATING

**CREATING** is specified as the state of *CreationControl* during which a *transceiver instance* performs creation of a particular burst.

The *burst under creation* is defined as the burst that an instance of **CREATING** aims to create.

The *applied creation command* is defined as the *creation command* applied by **CREATING** for creation of the *burst under creation*.

The entry transition of **CREATING** is specified as a transition to the **INITIATING** sub-state.

The exit transition of **CREATING** is specified as a transition after completion of the **ACTUATION** sub-state.

2.3.2.1.3  INITIATING

**INITIATING** is specified as the sub-state of **CREATING** during which *burst creation* is initiated.

A *transceiver instance shall*, during **INITIATING**, make the oldest *creation command* available in storage the *applied creation command*, and remove it from storage.
A transceiver instance shall, during INITIATING, set value of applicableBurstLength to value of requestedLength as specified in the applied creation command.

A transceiver instance shall, during INITIATING, increment burstCount (see section 3.3.1.1) by 1 (one), rolling-over to 1 after 4,294,967,295 (2^32 – 1).

A transceiver instance shall, for Rx channels during INITIATING, set the length of Rx packets to the value of applicableRxPacketsLength (see section 3.3.1.2).

A transceiver instance shall, during INITIATING, search for stored tuning parameters set according to a condition specified by value of TUNING_ASSOCIATION (see section 4.3):
- For sequential: search for the oldest stored tuning parameters set,
- For burstReferencing: search for a stored tuning parameters set with value of requestedBurstNumber equal to value of burstCount.

A transceiver instance shall, during INITIATING, if a stored tuning parameters set was found, use it as the applicable tuning parameters set and remove it from storage.

A transceiver instance shall, during INITIATING, if no stored tuning parameters set was found, set the applicable tuning parameters set as follows:
- requestedTuningPreset equals to UndefinedTuningPreset,
- requestedCarrierFreq equals to UndefinedCarrierFreq,
- requestedDelay equals to UndefinedDelay.

2.3.2.1.4 SCHEDULING

SCHEDULING is specified as the sub-state of CREATING during which the start time, activation time and tuning time of a burst under creation are determined.

Start time corresponds to start of the core burst at radio signal level (see section 1.2.3).

A channel shall stay in SCHEDULING until all information required for determination of start time is known.

A channel shall, during SCHEDULING of a startBurst() creation command, make start time equal to the termination time of the previous burst plus INTER-PROCESSING (see section 3.1.4).

A channel shall, during SCHEDULING of a scheduleRelativeBurst() creation command, make start time equal to the start time of the previous burst on channels specified by value of requestedAlternate plus the value of requestedDelay (see section 3.1.4).

A channel shall, during SCHEDULING of a scheduleAbsoluteBurst() creation command, make start time equal to the value of requestedStartTime (see section 3.1.5).

A channel shall, during SCHEDULING of a scheduleStrobedBurst() creation command, make start time equal to the occurrence time of the next strobe triggered on the strobe source specified by requestedStrobeSource plus the value of requestedDelay (see section 3.1.6).

Activation time is defined as the time at which the startProcessing transition is triggered.

A channel shall, during SCHEDULING, determine activation time so that the effective start time belongs to start time ± START_TIME_ACC (see section 4.12).
Note: for Tx channels, activation time is equal to start time minus up-conversion latency (see Figure 2); for Rx channels, activation time is equal to start time (see Figure 5).

Tuning time is defined as the time at which the startTuning transition is triggered to ensure that the applicable tuning parameters set is implemented by the TUNING state with a TuningStop transition triggered before activation time.

A channel shall, during SCHEDULING, determine tuning time based on activation time.

2.3.2.1.5 ACTUATING

ACTUATING is specified as the sub-state of CREATING during which the transceiver instance triggers TuningStart and ProcessingStart transitions of the Channels state machine.

A transceiver instance shall, during ACTUATING, trigger a TuningStart transition at tuning time.

A transceiver instance shall, during ACTUATING of Tx channels if the applied creation operation is startBurst(), shift activation time until first baseband sample becomes available.

A transceiver instance shall, during ACTUATING, trigger a ProcessingStart transition at activation time.

2.3.2.2 Transitions

2.3.2.2.1 CreationStart

CreationStart is specified as the transition from AWAITING to CREATING.

It is triggered once a creation command is available in storage.

2.3.2.2.2 InitiationCompleted

CreationStart is specified as the transition from INITIATING to SCHEDULING.

It is triggered once a transceiver instance has completed INITIATING.

2.3.2.2.3 SchedulingCompleted

SchedulingCompleted is specified as the transition from SCHEDULING to ACTUATING.

It is triggered once a transceiver instance has completed SCHEDULING.

2.3.2.2.4 CreationCompleted

CreationCompleted is specified as the transition from CREATING to AWAITING.

It is triggered once a transceiver instance has completed ACTUATING.

2.3.3 RadioSilence

RadioSilence is specified as the state machine applicable if RadioSilence is active or if the channels can be turned to radio silence by an agent different from the radio application.
The following figure is the statechart of **RadioSilence**: 

![RadioSilence Statechart](image)

**RadioSilence** is a sub-state machine of **OPERATING**, parallel to the sub-state machine of **OPERATING** specified by **Channels** (see section 2.3.1).

### 2.3.3.1 States

#### 2.3.3.1.1 NORMAL

**NORMAL** is specified as the state during which the **channels** operate as specified for the **OPERATING** state of **Channels**.

The entry transition of **RadioSilence** brings to **NORMAL**.

#### 2.3.3.1.2 RADIO_SILENCE

**RADIO_SILENCE** is specified as the state during which **channels** minimize the radiated radio signal, preventing respect of the specified **tuning** during **PROCESSING** state.

The **RADIO_SILENCE** state does not impact any other aspect of the **OPERATING** state.

### 2.3.3.2 Transitions

#### 2.3.3.2.1 RadioSilenceStart

**RadioSilenceStart** is specified as the transition from **NORMAL** to **RADIO_SILENCE**.

It is triggered by invocation of **startRadioSilence()**.

#### 2.3.3.2.2 RadioSilenceStop

**RadioSilenceStop** is specified as the transition from **RADIO_SILENCE** to **NORMAL**.

It is triggered by invocation of **stopRadioSilence()**.
2.3.4 Retuning

*Retuning* is specified as the state machine applicable if *Retuning* is active.

The following figure is the statechart of *Retuning*:

![Retuning Statechart](image)

**Figure 12 Retuning statechart**

*Retuning* is a sub-state machine of *PROCESSING* (see section 2.3.1).

### 2.3.4.1 States

#### 2.3.4.1.1 TUNED

*TUNED* is specified as the sub-state of *PROCESSING* during which *channels* process with stable tuning characteristics that comply with the specified *tuning*.

The entry transition of *Retuning* brings to *TUNED*.

#### 2.3.4.1.2 RETUNING

*REتونING* is specified as the sub-state of *PROCESSING* during which *channels* change their *tuning* while continuing to perform *up-conversion* or *down-conversion*.

*REتونING_DURATION* (see section 4.8) specifies the maximum duration of *REتونING* state.

### 2.3.4.2 Transitions

#### 2.3.4.2.1 RetuningStart

*RetuningStart* is specified as the transition from *TUNED* to *REتونING*.

It is triggered when the *radio application* calls *retune()* (see section 3.1.12).

#### 2.3.4.2.2 RetuningStop

*RetuningStop* is specified as the transition from *REتونING* to *TUNED*.

It is triggered when the new tuning characteristics are stable and conform to the tuning changes commanded by *retune()*.
2.4 Services groups description

The class diagrams appearing in this section aim to comply with the OMG Unified Modeling Language v2.5, as specified in [Ref4].

2.4.1 Transceiver::Management

The Management services group enables radio applications to manage the Transceiver, and contains the following services:

![Diagram of Management services group]

**Figure 13 Services of Management services group**

The Reset service enables radio applications to reset channels.

The RadioSilence service enables radio applications to start and stop radio silence.

2.4.1.1 Transceiver::Management::Reset Interface Description

The Reset interface is composed of the reset() operation, as depicted in the following figure:

![Diagram of Reset interface]

**Figure 14 Management::Reset interface**

reset() enables radio applications to reset channels.

2.4.1.2 Transceiver::Management::RadioSilence Interface Description

The RadioSilence interface is composed of the startRadioSilence() and stopRadioSilence() operations, as depicted in the following figure:

![Diagram of RadioSilence interface]

**Figure 15 Management::RadioSilence interface**
startRadioSilence() enables radio applications to start radio silence. stopRadioSilence() enables radio applications to stop radio silence.

2.4.2 Transceiver::BurstControl

The BurstControl services group enables radio applications to control the creation of bursts, and contains the following services:

A creation service is defined as a service of BurstControl services group.

A creation operation is defined as one operation of a creation service: startBurst(), scheduleRelativeBurst(), scheduleAbsoluteBurst() or scheduleStrobedBurst().

The DirectCreation service enables radio applications to schedule the creation of a new burst with no specific requirement on its start time.

A timely creation service is defined as a RelativeCreation, AbsoluteCreation or StrobedCreation service.

A timely creation operation is defined as a creation operation of a timely creation service: scheduleRelativeBurst(), scheduleAbsoluteBurst() or scheduleStrobedBurst().

Timely creation services and operations enables to specify the start time of scheduled burst, measured at the radio signal level, as specified in section 1.2.

The RelativeCreation service enables radio applications to schedule the creation of a new burst with a start time delayed by a specified value from the start time of the previous burst.

The AbsoluteCreation service enables radio applications to schedule the creation of a new burst with a start time specified using the transceiver time.

The StrobedCreation service enables radio applications to schedule the creation of a new burst with a start time delayed by a specified value from the next occurrence of a strobe discrete signal on a specified strobe source.

All creation services enable radio applications to specify the length of the baseband block.
Radio applications must make calls to creation operations in the same order as the order of created bursts (see section 2.3.2), and can make up to CREATION_STORAGE (see section 4.8) anticipated calls to creation operations.

Radio applications must make calls to timely creation operations ensuring value of INTER-PROCESSING (see section 4.8) is respected.

The Termination service enables radio applications to control termination of an ongoing processing phase.

2.4.2.1 Transceiver::BurstControl::DirectCreation Interface Description

The DirectCreation interface is composed of the startBurst() operation, as depicted in the following figure:

![Figure 17 BurstControl::DirectCreation interface](image)

startBurst() enables radio applications to schedule the creation of a new burst with no specific requirement on its start time.

2.4.2.2 Transceiver::BurstControl::RelativeCreation Interface Description

The RelativeCreation interface is composed of the scheduleRelativeBurst() operation, as depicted in the following figure:

![Figure 18 BurstControl::RelativeCreation interface](image)

scheduleRelativeBurst() enables radio applications to schedule the creation of a new burst with a start time delayed by a specified value from the start time of the previous burst.

scheduleRelativeBurst() must be combined with another creation operation (e.g. startBurst() or scheduleStrobedBurst()), used to create the first burst of all series of bursts then created using scheduleRelativeBurst().
2.4.2.3 Transceiver::BurstControl::AbsoluteCreation Interface Description

The **AbsoluteCreation** interface is composed of the `scheduleAbsoluteBurst()` operation, as depicted in the following figure:

![Figure 19 BurstControl::AbsoluteCreation interface](image)

`scheduleAbsoluteBurst()` enables *radio applications* to schedule the creation of a new burst with a *start time* specified using the *transceiver time*.

`scheduleAbsoluteBurst()` must be used in conjunction with a mechanism enabling *radio applications* to get the *transceiver time* (e.g. the `TransceiverTime` service).

2.4.2.4 Transceiver::BurstControl::StrobedCreation Interface Description

The **StrobedCreation** interface is composed of the `scheduleStrobedBurst()` operation, as depicted in the following figure:

![Figure 20 BurstControl::StrobedCreation interface](image)

`scheduleStrobedBurst()` enables *radio applications* to schedule the creation of a new burst with a *start time* delayed by a specified value from the next occurrence of a strobe discrete signal on a specified strobe source.

The specified strobe source can be internal to the platform (e.g. the PPS signal of a GNSS system) or be provided by a component of the *radio application* (e.g. a FPGA component).
2.4.2.5 Transceiver::BurstControl::Termination Interface Description

The **Termination** interface is composed of the `setBlockLength()` and `stopBurst()` operations, as depicted in the following figure:

![Termination Interface Diagram](image)

**setBlockLength()** enables radio applications to set the length of the baseband block processed by channels during an ongoing processing phase.

**stopBurst()** enables radio applications command immediate termination of an ongoing processing phase.

2.4.3 Transceiver::BasebandSignal

The **BasebandSignal** services group enables radio applications to exchange blocks of baseband samples processed by channel, and contains the following services:

![BasebandSignal Services Diagram](image)

The **SamplesReception** service enables radio applications to obtain a receive baseband block from an Rx channel during a reception (see section 1.2.4).

The **SamplesTransmission** service enables radio applications to forward a transmit baseband block to a Tx channel during a transmission (see section 1.2.5).

The **RxPacketsLengthControl** service enables radio applications to set the value of the applicableRxPacketsLength attribute.
2.4.3.1 Transceiver::BasebandSignal::SamplesReception Interface Description

The SamplesReception interface is composed of the pushRxPacket() operation, as depicted in the following figure:

![Diagram of SamplesReception interface](image)

pushRxPacket() enables radio applications to obtain a baseband packet from an Rx channel and to be specified if the packet is the last packet of the Rx block.

2.4.3.2 Transceiver::BasebandSignal::SamplesTransmission Interface Description

The SamplesTransmission interface is composed of the pushTxPacket() operation, as depicted in the following figure:

![Diagram of SamplesTransmission interface](image)

pushTxPacket() enables radio applications to forward a baseband packet to a Tx channel and to specify if the packet is the last packet of the Tx forwarded block.
2.4.3.3 Transceiver::BasebandSignal::RxPacketsLengthControl Interface Description

The **RxPacketsLengthControl** interface is composed of the `setRxPacketsLength()` operation, as depicted in the following figure:

![Diagram](image)

**Figure 25** BasebandSignal::RxPacketsLengthControl interface

`setRxPacketsLength()` enables *radio applications* to set the value of the `applicableRxPacketsLength` attribute.

2.4.4 Transceiver::Tuning

The **Tuning** services group enables *radio applications* to control the tuning of *bursts*, and contains the following services:

![Diagram](image)

**Figure 26** Services of Tuning services group

The **InitialTuning** service enables *radio applications* to specify the *tuning preset*, *carrier frequency* and *gain* values to be applied at beginning of a future *burst*.

*Radio applications* can make up to `TUNING_STORAGE` (see section 4.8) anticipated calls to `setTuning()`.

*Radio applications* must use the **InitialTuning** service for a given *burst*, if needed, before the stored creation operation of the *burst* is used by **CreationControl** (see section 2.3.2).

The **Retuning** service enables *radio applications* to schedule and specify new values of *carrier frequency* and *gain* without interrupting an ongoing *processing phase*. 
2.4.4.1 Transceiver::Tuning::InitialTuning Interface Description

The **InitialTuning** interface is composed of the `setTuning()` operation, as depicted in the following figure:

![InitialTuning Interface Diagram](image)

**setTuning()** enables *radio applications* to specify the *tuning preset*, *carrier frequency* and *gain* values to be applied at beginning of a future *burst*.

2.4.4.2 Transceiver::Tuning::Retuning Interface Description

The **Retuning** interface is composed of `retune()` operation, as depicted in the following figure:

![Retuning Interface Diagram](image)

**retune()** enables *radio applications* to schedule and specify new values of *carrier frequency* and *gain* without interrupting an ongoing *processing phase*.

2.4.5 Transceiver::Notifications

The **Notifications** services group enables *radio applications* to be notified by *channels* of execution events and execution errors, and contains the following services:

![Notifications Services Diagram](image)
The **Events** service enables *radio applications* to be notified of *events* occurrences.
The **Errors** service enables *radio applications* to be notified of *errors* occurrences.

2.4.5.1 Transceiver::Notifications::Events Interface Description

The **Events** interface is composed of the `notifyEvent()` operation, as depicted in the following figure:

![Figure 30 Notifications::Events interface](image)

`notifyEvent()` enables *radio applications* to be notified of *events* occurrences.

2.4.5.2 Transceiver::Notifications::Errors Interface Description

The **Errors** interface is composed of the `notifyError()` operation, as depicted in the following figure:

![Figure 31 Notifications::Errors interface](image)

`notifyError()` enables *radio applications* to be notified of *errors* occurrences.
2.4.6 Transceiver::GainControl

The GainControl services group enables radio applications to be informed of aspects related to gain control, and contains the following service:

![Diagram of GainControl services group]

The GainChanges service enables radio applications to be notified of changes in gain values decided by channels during a processing phase.

The AGCActivation service enables radio applications to deactivate and reactivate permanent AGC while a reception is ongoing.

2.4.6.1 Transceiver::GainControl::GainChanges Interface Description

The GainChanges interface is composed of the indicateGain() operation, as depicted in the following figure:

![Diagram of GainChanges interface]

indicateGain() enables radio applications to be notified of changes in gain values decided by channels during a processing phase.
2.4.6.2 Transceiver::GainControl::AGCActivation Interface Description

The **AGCActivation** interface is composed of the *deactivateAGC()* and *reactivateAGC()* operations, as depicted in the following figure:

![Figure 34 GainControl::AGCActivation interface](image)

*deactivateAGC()* enables *radio applications* to deactivate AGC in the course of a *reception*. 
*reactivateAGC()* enables *radio applications* to reactivate a previously deactivated AGC.

2.4.7 Transceiver::TransceiverTime

The **TransceiverTime** services group enables *radio applications* to get values of *transceiver time*, and contains the following service:

![Figure 35 Service of TransceiverTime services group](image)

The **TimeAccess** service enables *radio applications* to get the current value of *transceiver time* and the value of *transceiver time* for the *start time* of the last created burst.

2.4.7.1 Transceiver::TransceiverTime::TimeAccess Interface Description

The **TimeAccess** interface is composed of the *getCurrentTime()* and *getLastStartTime()* operations, as depicted in the following figure:
**get CURRENT Time()** enables radio applications to get the current value of transceiver time. **get Last Start Time()** enables radio applications to get the value of transceiver time for the start time of the last created burst.

### 2.4.8 Transceiver::Strobing

The Strobing services group enables radio applications to trigger strobes that can be used for creation of bursts scheduled with StrobedCreation service, and contains the following interface:

![Strobing services group](image)

Figure 37  Service of Strobing services group

The Strobing service enables radio applications to trigger strobes that can be used for creation of a burst scheduled with StrobedCreation service.

#### 2.4.8.1 Transceiver::Strobing::ApplicationStrobe Interface Description

The ApplicationStrobe interface is composed of the triggerStrobe() operation, as depicted in the following figure:

![Strobing::ApplicationStrobe interface](image)

Figure 38  Strobing::ApplicationStrobe interface

triggerStrobe() enables radio applications to trigger occurrences of strobes that can be used for creation of a burst scheduled by a scheduleStrobedBurst() call (see section 3.1.6) with requestedStrobeSource parameter equal to ApplicationStrobe.
3 Service primitives and attributes

3.1 Service primitives

This section specifies the primitives of the services interfaces of the Transceiver API.

Each declaration of a primitive complies with the Full PIM IDL Profile of WInnForum IDL profiles for PIM of SDR Applications, specified in [Ref5].

The conformance criteria for Application-Specific Interfaces is applied (see [Ref5], section 1.3.2): “An Application-Specific Interface is conformant with one applicable IDL Profile if each of its operations exclusively uses capabilities of the applicable IDL Profile.”.

The declaration of each primitive also complies with SCA 4.1 Appendix E-1 [Ref6].

The specified declarations are common normative inputs for the PSMs (see section 1.1) specified in appendices of the specification.

The sequence diagrams appearing in this section are based on the OMG Unified Modeling Language v2.5, as specified in [Ref4].

3.1.1 Transceiver::Management::Reset

3.1.1.1 reset Operation

3.1.1.1.1 Overview

reset() commands channels to reset.

3.1.1.1.2 Associated properties

RESET_WCET (see section 4.15) specifies the worst-case execution time of the primitive.

3.1.1.1.3 Declaration

The declaration of the operation is specified as:

```c
void reset();
```

3.1.1.1.4 Parameters

None.

3.1.1.1.5 Returned value

None.

3.1.1.1.6 Originator

Radio application.
3.1.1.7 Exceptions
None.

3.1.1.8 Behavior requirements

An active instance of Reset shall, on a call to reset():

- Stop any ongoing activity,
- Trigger a RuntimeReset transition (see section 2.3.1.2.6),
- Complete the RESETTING state (see section 2.3.1.1.6),
- Trigger a ResetCompleted transition (see section 2.3.1.2.1),
- Return the call to the radio application.

3.1.2 Transceiver::Management::RadioSilence

3.1.2.1 startRadioSilence Operation

3.1.2.1.1 Overview

startRadioSilence() commands Tx channels to start a radio silence phase, as depicted in the following figure:

![Figure 39 Principle of startRadioSilence()](image)

3.1.2.1.2 Associated properties

START_SILENCE_WCET (see section 4.15) specifies the worst-case execution time of the primitive.

3.1.2.1.3 Declaration

The declaration of the operation is specified as:

```c
void startRadioSilence();
```

3.1.2.1.4 Parameters

None.
3.1.2.1.5 Returned value

None.

3.1.2.1.6 Originator

Radio application.

3.1.2.1.7 Exceptions

None.

3.1.2.1.8 Behavior requirements

An active instance of RadioSilence shall, on a call to startRadioSilence():

- Trigger a RadioSilenceStart transition (see section 2.3.2),
- Stop radiating any signal at RF level,
- Return the call to the radio application.

The RADIO_SILENCE state does not impact operation of the OPERATING state further than preventing RF radiation.

3.1.2.2 stopRadioSilence Operation

3.1.2.2.1 Overview

stopRadioSilence() commands the Tx channels to stop a radio silence phase, as depicted in the following figure:

![Figure 40 Principle of stopRadioSilence()](image)

3.1.2.2.2 Associated properties

STOP_SILENCE_WCET (see section 4.15) specifies the worst-case execution time of the primitive.

3.1.2.2.3 Declaration

The declaration of the operation is specified as:

```c
void stopRadioSilence();
```
3.1.2.2.4 Parameters

None.

3.1.2.2.5 Returned value

None.

3.1.2.2.6 Originator

Radio application.

3.1.2.2.7 Exceptions

None.

3.1.2.2.8 Behavior requirements

An active instance of RadioSilence shall, on a call to stopRadioSilence():

- Trigger a RadioSilenceStop transition (see section 2.3.2),
- Resume normal radio operation at RF level,
- Return the call to the radio application.

3.1.3 Transceiver::BurstControl::DirectCreation

3.1.3.1 startBurst Operation

3.1.3.1.1 Overview

startBurst() commands the channels to schedule creation of a burst with no specified start time, as depicted in the following figure:

![Figure 41 Principle of startBurst()](image-url)
3.1.3.1.2 Associated properties

**CREATION_STORAGE** (see section 4.8) specifies the maximum number of calls to *creation operations*, such as calls to `startBurst()`, which *channels* can store in advance.

**DIRECT_WCET** (see section 4.15) specifies the worst-case execution time of the primitive.

3.1.3.1.3 Declaration

The declaration of the operation is specified as:

```c
void startBurst(
    in BlockLength requestedLength);
```

3.1.3.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| requestedLength | BlockLength (see § 3.4.3) | Value of `applicableLength`. Number of baseband samples to be processed during the processing phase associated to the burst:  
  - If equal to `UndefinedBlockLength`: specifies an undefined value,  
  - If not equal to `UndefinedBlockLength`: specifies the number of baseband samples of the baseband block to be processed during PROCESSING (see section 2.3.1). |

Table 3 Specification of `startBurst()` parameters

The parameters validity properties are specified as (see section 4.7):

- For `requestedLength`: `MIN_BLOCK_LENGTH` and `MAX_BLOCK_LENGTH`.

3.1.3.1.5 Return value

None.

3.1.3.1.6 Originator

*Radio application*.

3.1.3.1.7 Exceptions

The exceptions of the operation are specified as (see section 3.2):

- `MinBlockLength` and `MaxBlockLength`.

3.1.3.1.8 Behavior requirements

An active instance of `DirectCreation` shall, on a call to `startBurst()`, handle the exceptions of the operation as specified in section 3.2.
An active instance of DirectCreation shall, on a call to startBurst() that raised no exception:

- If CREATION_STORAGE calls (see section 4.8) are stored, wait until storage becomes available,
- Store the call for later usage by CreationControl (see section 2.3.2),
- Return the call to the radio application.

3.1.4 Transceiver::BurstControl::RelativeCreation

3.1.4.1 scheduleRelativeBurst Operation

3.1.4.1.1 Overview

scheduleRelativeBurst() commands the channels to schedule creation of a burst starting at a specified delay after the start time of the previous burst of the referenced channel, as depicted in the following figure:

![Figure 42 Principle of scheduleRelativeBurst()](image)

3.1.4.1.2 Associated properties

CREATION_STORAGE (see section 4.8) specifies the maximum number of calls to creation operations that channels can store in advance, including calls to scheduleRelativeBurst().

RELATIVE_MILT (see section 4.13) specifies the minimum invocation lead time for correct real-time usage of the operation.

RELATIVE_WCET (see section 4.15) specifies the worst-case execution time of the primitive.
3.1.4.1.3 Declaration

The declaration of the operation is specified as:

```c
void scheduleRelativeBurst(
    in boolean requestedAlternate,
    in Delay requestedDelay,
    in BlockLength requestedLength);
```

3.1.4.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestedAlternate</td>
<td>boolean</td>
<td>For duplex transceivers, specifies the reference channels:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If equal to <code>false</code>: called channels are used,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If equal to <code>true</code>: alternate channels are used.</td>
</tr>
<tr>
<td>requestedDelay</td>
<td>Delay</td>
<td>Specifies the delay between the start time of the previous burst scheduled by reference channel and the start time of the burst to create.</td>
</tr>
<tr>
<td>requestedLength</td>
<td>BlockLength</td>
<td>Number of baseband samples to be processed during the processing phase associated to the burst:</td>
</tr>
<tr>
<td></td>
<td>(see § 3.4.3)</td>
<td>- If equal to UndefinedBlockLength: specifies an undefined value,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If not equal to UndefinedBlockLength: specifies the number of baseband samples of the baseband block to be processed during PROCESSING (see section 2.3.1).</td>
</tr>
</tbody>
</table>

The parameters validity properties are specified as (see section 4.7):
- For `requestedAlternate`: ALTERNATEREFERENCING,
- For `requestedDelay`: MIN_FROM_PREVIOUS and MAX_FROM_PREVIOUS,
- For `requestedLength`: MIN_BLOCK_LENGTH and MAX_BLOCK_LENGTH.

3.1.4.1.5 Return value

None.

3.1.4.1.6 Originator

Radio application.

3.1.4.1.7 Exceptions

The exceptions of the operation are specified as (see section 3.2):
- NoAlternateReferencing,
- MinFromPrevious and MaxFromPrevious,
- MinBlockLength and MaxBlockLength,
- RelativeMILT.
3.1.4.1.8 Behavior requirements

An active instance of `RelativeCreation` shall, on a call to `scheduleRelativeBurst()`, handle the exceptions of the operation as specified in section 3.2.

An active instance of `RelativeCreation` shall, on a call to `scheduleRelativeBurst()` that raised no exception:

- If `CREATION_STORAGE` calls (see section 4.6) are stored, wait until storage becomes available,
- Store the call for later usage by `CreationControl` (see section 2.3.2),
- Return the call to the radio application.

3.1.5 Transceiver::BurstControl::AbsoluteCreation

3.1.5.1 scheduleAbsoluteBurst Operation

3.1.5.1.1 Overview

`scheduleAbsoluteBurst()` commands the channels to schedule creation of a burst for which core burst will start at the specified `requestedStartTime`, as depicted in the following figure:

![Figure 43 Principle of scheduleAbsoluteBurst()](image)

3.1.5.1.2 Associated properties

`CREATION_STORAGE` (see section 4.8) specifies the maximum number of calls to creation operations that channels can store in advance, including calls to `scheduleAbsoluteBurst()`.

`TIME_COUPLING` (see section 4.2) specifies coupling of transceiver time with other times of the radio platform.

`ABSOLUTE_MILT` (see section 4.13) specifies the minimum invocation lead time for correct real-time usage of the operation.

`ABSOLUTE_WCET` (see section 4.15) specifies the worst-case execution time of the primitive.
3.1.5.1.3 Declaration

The declaration of the operation is specified as:

```c
void scheduleAbsoluteBurst(
    in TimeSpec requestedStartTime,
    in BlockLength requestedLength);
```

3.1.5.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestedStartTime</td>
<td>TimeSpec</td>
<td>Specifies the value of start time of the burst to create, expressed according to transceiver time.</td>
</tr>
</tbody>
</table>
| requestedLength   | BlockLength       | Number of baseband samples to be processed during the processing phase associated to the burst:  
|                  | (see § 3.4.3)     |  
|                  |                   | - If equal to UndefinedBlockLength: specifies an undefined value,  
|                  |                   | - If not equal to UndefinedBlockLength: specifies the number of baseband samples of the baseband block to be processed during PROCESSING (see section 2.3.1). |

Table 5 Specification of `scheduleAbsoluteBurst()` parameters

The parameters validity properties are specified as (see section 4.7):
- For `requestedLength`: MIN_BLOCK_LENGTH and MAX_BLOCK_LENGTH.

3.1.5.1.5 Return value

None.

3.1.5.1.6 Originator

Radio application.

3.1.5.1.7 Exceptions

The exceptions of the operation are specified as (see section 3.2):
- MaxNanoseconds,
- MinBlockLength and MaxBlockLength,
- AbsoluteMILT.

3.1.5.1.8 Behavior requirements

An active instance of `AbsoluteCreation` shall, on a call to `scheduleAbsoluteBurst()`, handle the exceptions of the operation as specified in section 3.2.
An **active instance of AbsoluteCreation** shall, on a call to `scheduleAbsoluteBurst()` that raised no exception:

- If `CREATION_STORAGE` calls (see section 4.8) are stored, wait until storage becomes available,
- Store the call for later usage by `CreationControl` (see section 2.3.2),
- Return the call to the **radio application**.

### 3.1.6 Transceiver::BurstControl::StrobedCreation

#### 3.1.6.1 scheduleStrobedBurst Operation

**3.1.6.1.1 Overview**

`scheduleStrobedBurst()` commands the *channels* to schedule creation of a burst for which the **core burst** will start at a specified delay taken after the *start time* of the next strobe occurrence of the specified strobe source, as depicted in the following figure:

![Figure 44 Principle of scheduleStrobedBurst()](image)

The standard strobe sources are specified by the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ApplicationStrobe</strong></td>
<td>Strobes delivered by the <em>radio application</em> using the <strong>ApplicationStrobe</strong> interface.</td>
</tr>
<tr>
<td><strong>TimeRef_PPS</strong></td>
<td>Strobes delivered by the PPS signal of a wired time reference.</td>
</tr>
<tr>
<td><strong>GNSS_PPS</strong></td>
<td>Strobes delivered by the PPS signal of a GNSS system.</td>
</tr>
<tr>
<td><strong>UserStrobe1</strong></td>
<td>User-defined strobe 1.</td>
</tr>
<tr>
<td><strong>UserStrobe2</strong></td>
<td>User-defined strobe 2.</td>
</tr>
<tr>
<td><strong>UserStrobe3</strong></td>
<td>User-defined strobe 3.</td>
</tr>
<tr>
<td><strong>UserStrobe4</strong></td>
<td>User-defined strobe 4.</td>
</tr>
</tbody>
</table>

*Table 6 Specification of strobe sources*
Additional *strobe sources* can be implemented if required by usage context.

### 3.1.6.1.2 Associated properties

**CREATION_STORAGE** (see section 4.8) specifies the maximum number of calls to *creation operations* that *channels* can store in advance, including calls to `scheduleStrobedBurst()`.

**STROBED_MILT** (see section 4.13) specifies the minimum invocation lead time for correct real-time usage of the operation.

**STROBED_WCET** (see section 4.15) specifies the worst-case execution time of the primitive.

### 3.1.6.1.3 Declaration

The declaration of the operation *is specified as*:

```c
void scheduleStrobedBurst(
    in StrobeSource requestedStrobeSource,
    in Delay requestedDelay,
    in BlockLength requestedLength);
```

### 3.1.6.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestedStrobeSource</td>
<td>StrobeSource</td>
<td>Specifies the <em>strobe source</em> to be used.</td>
</tr>
<tr>
<td>requestedDelay</td>
<td>Delay</td>
<td>Specifies the delay between the <em>next strobe occurrence on strobe source</em> and <em>start time of the burst</em> to create.</td>
</tr>
<tr>
<td>requestedLength</td>
<td>BlockLength</td>
<td>Number of <em>baseband samples</em> to be processed during the <em>processing phase</em> associated to the burst:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If equal to <em>UndefinedBlockLength</em>: specifies an undefined value,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- If not equal to <em>UndefinedBlockLength</em>: specifies the number of <em>baseband samples</em> of the <em>baseband block</em> to be processed during <em>PROCESSING</em> (see section 2.3.1).</td>
</tr>
</tbody>
</table>

| Table 7 Specification of `scheduleStrobedBurst()` parameters |

The *parameters validity properties are specified as* (see section 4.7):

- For `requestedStrobeSource`: `STROBE_SOURCES`,
- For `requestedDelay`: `MIN_FROM_STROBE` and `MAX_FROM_STROBE`,
- For `requestedLength`: `MIN_BLOCK_LENGTH` and `MAX_BLOCK_LENGTH`.

### 3.1.6.1.5 Return value

None.

### 3.1.6.1.6 Originator

*Radio application.*
3.1.6.1.7 Exceptions

The exceptions of the operation are specified as (see section 3.2):

- StrobeSource,
- MinFromStrobe and MaxFromStrobe,
- MinBlockLength and MaxBlockLength.

3.1.6.1.8 Behavior requirements

An active instance of StrobedCreation shall, on a call to scheduleStrobedBurst(), handle the exceptions of the operation as specified in section 3.2.

An active instance of StrobedCreation shall, on a call to scheduleStrobedBurst() that raised no exception:

- If CREATION_STORAGE calls (see section 4.8) are stored, wait until storage becomes available,
- Store the call for later usage by CreationControl (see section 2.3.2),
- Return the call to the radio application.

3.1.7 Transceiver::BurstControl::Termination

3.1.7.1 setBlockLength operation

3.1.7.1.1 Overview

setBlockLength() specifies the length of baseband block applicable for termination of an ongoing processing phase.

3.1.7.1.2 Associated properties

BLOCK_LENGTH_MILT (see section 4.13) specifies the minimum invocation lead time for correct real-time usage of the operation.

BLOCK_LENGTH_WCET (see section 4.15) specifies the worst-case execution time of the primitive.

3.1.7.1.3 Declaration

The declaration of the operation is specified as:

```plaintext
void setBlockLength(
    in BlockLength requestedLength);
```

3.1.7.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestedLength</td>
<td>BlockLen</td>
<td>Number of baseband samples to be processed during PROCESSING (see section 2.3.1).</td>
</tr>
</tbody>
</table>

Table 8 Specification of setBlockLength() parameters
The parameters validity properties are specified as (see section 4.7):

- For \textit{requestedLength}: \texttt{MIN\_BLOCK\_LENGTH} and \texttt{MAX\_BLOCK\_LENGTH}.

3.1.7.1.5 Return value

None.

3.1.7.1.6 Originator

Radio application.

3.1.7.1.7 Exceptions

The exceptions of the operation are specified as (see section 3.2):

- \texttt{NoOngoingProcessing},
- \texttt{MinBlockLength} and \texttt{MaxBlockLength}.

3.1.7.1.8 Behavior requirements

An active instance of \texttt{Termination} shall, on a call to \texttt{setBlockLength()}, handle the exceptions of the operation as specified in section 3.2.

An active instance of \texttt{Termination} shall, on a call to \texttt{setBlockLength()} that raised no exception:

- Set value of \texttt{applicableBurstLength} to value of \texttt{requestedLength},
- Notify the \texttt{PROCESSING} state of \texttt{Channels} of availability of a new \texttt{applicableBurstLength} value,
- Return the call to the radio application.

3.1.7.2 stopBurst operation

3.1.7.2.1 Overview

\texttt{stopBurst()} commands an ongoing processing phase to immediately terminate.

3.1.7.2.2 Associated properties

None.

3.1.7.2.3 Declaration

The declaration of the operation is specified as:

\begin{verbatim}
void stopBurst();
\end{verbatim}

3.1.7.2.4 Parameters

None.

3.1.7.2.5 Return value

None.
3.1.7.2.6 Originator
Radio application.

3.1.7.2.7 Exceptions
The exceptions of the operation are specified as (see section 3.2):

- NoOngoingProcessing.

3.1.7.2.8 Behavior requirements
An active instance of Termination shall, on a call to setBlockLength(), handle the exceptions of the operation as specified in section 3.2.

An active instance of Termination shall, on a call to stopBurst() that raised no exception:

- Set value of applicableBurstLength to the value enabling fastest possible termination of the ongoing processing phase,
- Notify the PROCESSING state of Channels of availability of a new applicableBurstLength value,
- Return the call to the radio application.

3.1.8 Transceiver::BasebandSignal::SamplesReception

3.1.8.1 pushRxPacket Operation

3.1.8.1.1 Overview
pushRxPacket() provides the radio application with the next packet of an Rx block received by one Rx channel, as depicted in following figure:

![Diagram of pushRxPacket()](image)

**Figure 45 Principle of pushRxPacket()**
**Rx packets** are sent by the **Rx channel** until the last processed sample (see section 2.3.1.1.5) has been transferred.

The first packet is sent after the **ProcessingStart** transition (see section 2.3.1).

A boolean flag specifies to the radio application that a received packet is the last packet of the received samples block. The next received packet is the first packet of the next received samples block.

### 3.1.8.1.2 Associated properties

**RX_MIN_BASEBAND_LEVEL** and **RX_MAX_BASEBAND_LEVEL** (see section 4.10) specify the interval into which the level of baseband signal fits.

**RX_META_DATA** (see section 4.5) specifies if meta-data is associated to Rx packets.

**RX_PACKET_WCET** (see section 4.15) specifies the worst-case execution time of the primitive for correct real-time operation of the transceiver instance.

### 3.1.8.1.3 Declaration

The declaration of the operation is specified as, if **RX_META_DATA** is equal to **FALSE**:

```c
void pushRxPacket(
    in BasebandPacket rxPacket,
    in boolean endOfBlock);
```

The declaration of the operation is specified as, if **RX_META_DATA** is equal to **TRUE**:

```c
void pushRxPacket(
    in BasebandPacket rxPacket,
    in boolean endOfBlock,
    in RxMetaData rxMetaData);
```

### 3.1.8.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rxPacket</td>
<td>BasebandPacket</td>
<td>The transferred Rx packet within the Rx block.</td>
</tr>
<tr>
<td>endOfBlock</td>
<td>boolean</td>
<td>Specifies if rxPacket is the last packet of the Rx block.</td>
</tr>
<tr>
<td>rxMetaData</td>
<td>RxMetaData</td>
<td>Specifies the user-defined meta-data associated to the Rx packet.</td>
</tr>
</tbody>
</table>

If **RX_META_DATA** is equal to **TRUE**.

Table 9 Specification of pushRxPacket() parameters

No **parameters validity property** is specified for use services.

### 3.1.8.1.5 Return value

None.

### 3.1.8.1.6 Originator

**Rx channel**.
3.1.8.1.7 Exceptions
Not applicable to a use service.

3.1.8.1.8 Behavior requirements

nbrFullPackets and tailPacketLength are defined as, respectively, the quotient and the remainder of the Euclidean division of applicableBlockLength by applicableRxPacketsLength.

Active instances of SamplesReception shall transfer the Rx block with a succession of nbrFullPackets calls to pushRxPacket(), with the length of rxPacket equal to applicableRxPacketsLength.

Active instances of SamplesReception shall, if tailPacketLength is greater than 0, make a last call to pushRxPacket() with the length of rxPacket equal to tailPacketLength.

Active instances of SamplesReception shall set the value of endOfBlock as follows:
- false: for all calls to pushRxPacket() except the last one,
- true: for the last call to pushRxPacket().

Active instances of SamplesReception shall wait for the radio application to return the previous call to pushRxPacket() before making a next call to pushRxPacket().

3.1.9 Transceiver::BasebandSignal::SamplesTransmission

3.1.9.1 pushTxPacket Operation

3.1.9.1.1 Overview

pushTxPacket() provides a Tx channel with the next packet of a Tx forwarded block to be stored prior to up-conversion, as depicted in following figure:

![Diagram of pushTxPacket()](image)

Figure 46 Principle of pushTxPacket()
The length of each packet is determined by the radio application, and can vary from one packet to another down to a single sample packet.

The first packet of the first Tx forwarded block is the first packet after ResetCompleted transition (see section 2.3.1.2.1).

A flag specifies to the Tx channel the last packet of the Tx forwarded block. Next transmitted packet is the first packet of the next Tx forwarded block.

### 3.1.9.1.2 Associated properties

**TX_META_DATA** (see section 4.5) specifies if meta-data is associated to Tx packets.

**TX_BASEBAND_STORAGE** (see section 4.8) specifies the number of baseband samples that a transceiver can store in advance of their usage by up-conversion.

**TX_PACKET_MILT** (see section 4.13) specifies the minimum invocation lead time for correct real-time usage of the operation.

**TX_MIN_BASEBAND_LEVEL** and **TX_MAX_BASEBAND_LEVEL** (see section 4.10) specify the interval into which the level of baseband signal must belong for correct Rx channel operation.

**TX_PACKET_WCET** (see section 4.15) specifies the worst-case execution time of the primitive.

### 3.1.9.1.3 Declaration

The declaration of the operation is specified as, if **TX_META_DATA** is equal to FALSE:

```c
void pushTxPacket(
    in BasebandPacket txPacket,
    in boolean endOfBlock);
```

The declaration of the operation is specified as, if **TX_META_DATA** is equal to TRUE:

```c
void pushTxPacket(
    in BasebandPacket txPacket,
    in boolean endOfBlock,
    in TxMetaData txMetaData);
```

### 3.1.9.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>txPacket</td>
<td>BasebandPacket (see § 0)</td>
<td>The transferred Tx packet.</td>
</tr>
<tr>
<td>endOfBlock</td>
<td>boolean</td>
<td>Specifies that txPacket is the last packet of the Tx forwarded block.</td>
</tr>
<tr>
<td>txMetaData</td>
<td>TxMetaData (see § 3.4.12)</td>
<td>Specifies the user-defined meta-data associated to the Tx packet.</td>
</tr>
</tbody>
</table>

Table 10 Specification of pushTxPacket() parameters

The parameters validity properties are specified as (see section 4.7):
- For length of txPacket: MAX_PACKETS_LENGTH.
3.1.9.1.5 Return value

None.

3.1.9.1.6 Originator

Radio application.

3.1.9.1.7 Exceptions

The exceptions of the operation are specified as (see section 3.2):

- MaxTxPacketsLength,
- TxPacketsMILT.

3.1.9.1.8 Behavior requirements

Active instances of SamplesTransmission shall, on a call to pushTxPacket(), handle the exceptions of the operation as specified in section 3.2.

Active instances of SamplesTransmission shall, on a call to pushTxPacket() that raised no exception:

- Wait, if needed, until all baseband samples of a previous burst have entered up-conversion,
- Wait, if storage is saturated, for consumption by up-conversion of previously stored samples to free storage capacity,
- Store the samples of txPacket for later usage by up-conversion,
- Depending on value of endOfBlock:
  - false: Tx forwarded block is not ended,
  - true: Tx forwarded block is ended, the last sample of txPacket is the last sample of the Tx forwarded block,
- Return the call to the radio application.

A channel shall be capable to store up to TX_BASEBAND_STORAGE (see section 4.6) baseband samples.

3.1.10 Transceiver::BasebandSignal::RxPacketsLengthControl

3.1.10.1 setRxPacketsLength operation

3.1.10.1.1 Overview

setRxPacketsLength() provides Rx channels with the size of received packets to be used at creation of forthcoming Rx bursts.

3.1.10.1.2 Associated properties

RX_PACKETS_LENGTH_WCET (see section 4.15) specifies the worst-case execution time of the primitive.
3.1.10.1.3 Declaration

The declaration of the operation is specified as:

```c
void setRxPacketsLength(
    in PacketLength requestedLength);
```

3.1.10.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestedLength</td>
<td>PacketLength</td>
<td>Specifies the new value for applicableRxPacketsLength attribute (see § 3.3.1.2).</td>
</tr>
</tbody>
</table>

Table 11 Specification of `setRxPacketsLength()` parameters

The parameters validity properties are specified as (see section 4.7):

- For `requestedLength`: `_MAX_PACKETS_LENGTH_`.

3.1.10.1.5 Return value

None.

3.1.10.1.6 Originator

Radio application.

3.1.10.1.7 Exceptions

The exceptions of the operation are specified as (see section 3.2):

- `MaxRxPacketsLength`.

3.1.10.1.8 Behavior requirements

An active instance of `RxPacketsLengthControl` shall, on a call to `setRxPacketsLength()`, handle the exceptions of the operation as specified in section 3.2.

An active instance of `RxPacketsLengthControl` shall, on a call to `setRxPacketsLength()` that raised no exception:

- Sets value of `applicableRxPacketsLength` attribute (see section 3.3.1.2) to value of `requestedLength` parameter,
- Return the call to the radio application.

3.1.11 Transceiver::Tuning::InitialTuning

3.1.11.1 setTuning Operation

3.1.11.1.1 Overview

`setTuning()` commands the channels to store a tuning parameters set (composed of tuning preset, carrier frequency \(f_c\) and gain \(G\), see section 1.1.4) than will be later applied to the tuned burst, as depicted in the following figure:
The call to `setTuning()` for a given burst needs to done, if needed, before `CreationControl` enters in `INITIATING` state for the considered burst (see section 2.3.2).

### 3.1.11.1.2 Associated properties

- **TUNING_ASSOCIATION** (see section 4.3) specifies how `CreationControl` (see section 2.3.2) associates stored tuning calls to created bursts during `INITIATING` state.
- **TUNING_STORAGE** (see section 4.8) specifies the maximum number of calls to `setTuning()` channels can store in advance.
- **TUNING_MILT** (see section 4.13) specifies the minimum invocation lead time in advance of the call to the `creation operation` of the tuned burst.
- **TUNING_WCET** (see section 4.15) specifies the worst-case execution time of the primitive.

### 3.1.11.1.3 Declaration

The declaration of the operation is specified as:

```c
void setTuning(
    in TuningPreset requestedPreset,
    in CarrierFreq requestedFrequency,
    in Gain requestedGain,
    in BurstNumber requestedBurstNumber);
```
### 3.11.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| requestedPreset| `TuningPreset` (see § 3.4.14) | Tuning preset to be applied under control of burst creation during a **TUNING** state:  
  - If equal to `UndefinedTuningPreset`: specifies to reuse the previously active tuning preset,  
  - If not equal to `UndefinedTuningPreset`: specifies the tuning preset to apply. |
| requestedFrequency | `CarrierFreq` (see § 3.4.6) | Carrier frequency ($f_c$) to be applied under control of burst creation during a **TUNING** state:  
  - If equal to `UndefinedCarrierFreq`: specifies to reuse the previously active carrier frequency,  
  - If not equal to `UndefinedCarrierFreq`: specifies the carrier frequency to apply. |
| requestedGain   | `Gain` (see § 3.4.10)     | Gain ($G$) to be applied under control of burst creation during a **TUNING** state:  
  - If equal to `UndefinedGain`: specifies to reuse the previously active gain,  
  - If not equal to `UndefinedGain`: specifies the gain to apply. |
| requestedBurstNumber | `BurstNumber` (see § 3.4.5) | Specifies a burst number for burst creation to determine the tuned burst for the specified tuning parameters set, depending on value of **TUNING_ASSOCIATION**:  
  - If equal to `sequential`: the value is ignored,  
  - If equal to `burstReferencing`: the specified number of the burst for which the specified tuning parameters set applies. |

Table 12  Specification of `setTuning()` parameters

The parameters validity properties are specified as (see section 4.7):

- For `requestedPreset`: `MAX_TUNING_PRESET`,
- For `requestedFrequency`: `MIN_CARRIER_FREQ` and `MAX_CARRIER_FREQ`,
- For `requestedGain`: `MIN_GAIN` and `MAX_GAIN`.

### 3.11.1.5 Return value
None.

### 3.11.1.6 Originator
Radio application.
3.1.11.1.7 Exceptions

The exceptions of the operation are specified as (see section 3.2):
- MaxTuningPreset,
- MinCarrierFreq and MaxCarrierFreq,
- MinGain and MaxGain,
- TuningMILT.

3.1.11.1.8 Behavior requirements

An active instance of InitialTuning shall, on a call to setTuning(), handle the exceptions of the operation as specified in section 3.2.

An active instance of InitialTuning shall, on a call to setTuning() that raised no exception:
- Wait, if storage is saturated, for usage of a previously stored call to free capacity,
- Store the call by order of arrival for later usage by CreationControl (see section 2.3.2),
- Return the call to the radio application.

A channel shall be capable to store up to TUNING_STORAGE (see section 4.8) setTuning() calls.

3.1.12 Transceiver::Tuning::Retuning

3.1.12.1 Retune Operation

3.1.12.1.1 Overview

retune() commands the channels to change the tuning during an ongoing processing phase, specifying the delay to take from the start time of the burst before starting to retune, as depicted in following figure:

An undefined delay specifies retuning to take place immediately after the call to retune().
3.1.12.1.2 Associated properties

**RETUNING_DURATION** (see section 4.8) specifies the maximum duration of **RETUNING** state (see section 2.3.4).

**RETUNING_MILT** (see section 4.13) specifies the minimum invocation lead time for correct real-time usage of the operation.

**RETUNING_WCET** (see section 4.15) specifies the worst-case execution time of the primitive.

3.1.12.1.3 Declaration

The declaration of the operation is specified as:

```c
void retune(
    in CarrierFreq requestedFrequency,
    in Gain requestedGain,
    in Delay requestedDelay);
```

3.1.12.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestedFrequency</td>
<td>CarrierFreq</td>
<td>Carrier frequency ($f_c$) to be applied by channels during the scheduled <strong>RETUNING</strong> state:</td>
</tr>
<tr>
<td></td>
<td>(see § 3.4.6)</td>
<td>▪ If equal to <strong>UndefinedCarrierFreq</strong>: specifies to reuse the previously active carrier frequency,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ If not equal to <strong>UndefinedCarrierFreq</strong>: specifies the carrier frequency to apply.</td>
</tr>
<tr>
<td>requestedGain</td>
<td>Gain</td>
<td>Gain ($G$) to be applied by channels during the scheduled <strong>RETUNING</strong> state:</td>
</tr>
<tr>
<td></td>
<td>(see § 3.4.10)</td>
<td>▪ If equal to <strong>UndefinedGain</strong>: specifies to reuse the previously active gain,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ If not equal to <strong>UndefinedGain</strong>: specifies the gain to apply.</td>
</tr>
<tr>
<td>requestedDelay</td>
<td>Delay</td>
<td>Delay to take after the start time of the ongoing processing phase for triggering the <strong>RetuningStart</strong> transition:</td>
</tr>
<tr>
<td></td>
<td>(see § 3.4.7)</td>
<td>▪ If equal to <strong>UndefinedDelay</strong>: specifies that the <strong>RetuningStart</strong> transition is triggered immediately,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ If not equal to <strong>UndefinedDelay</strong>: specifies the applicable delay.</td>
</tr>
</tbody>
</table>

Table 13 Specification of **retune()** parameters

The parameters validity properties are specified as (see section 4.7):

- For **requestedFrequency**: **MIN_CARRIER_FREQ** and **MAX_CARRIER_FREQ**,
- For **requestedGain**: **MIN_GAIN** and **MAX_GAIN**,
- For **requestedDelay**: **MIN_FROM_ONGOING** and **MAX_FROM_ONGOING**.

3.1.12.1.5 Return value

None.
3.1.12.1.6 Originator

*Radio application.*

3.1.12.1.7 Exceptions

The exceptions of the operation are specified as (see section 3.2):

- NoOngoingProcessing,
- MinCarrierFreq and MaxCarrierFreq,
- MinGain and MaxGain,
- MinFromOngoing and MaxFromOngoing,
- RetuningMILT.

3.1.12.1.8 Behavior requirements

An active instance of `Retuning` shall, on a call to `retune()`, handle the exceptions of the operation as specified in section 3.2.

An active instance of `Retuning` shall, on a call to `retune()` that raised no exception, with value of `requestedDelay` equal to `UndefinedDelay`:

- Return the call to `retune()` to the radio application,
- Trigger the `RetuningStart` transition (see section 2.3.4) immediately after.

An active instance of `Retuning` shall, on a call to `retune()` that raised no exception, with value of `requestedDelay` not equal to `UndefinedDelay`:

- Return the call to `retune()` to the radio application,
- Trigger the `RetuningStart` transition (see section 2.3.4) at `start time` of the ongoing processing phase plus value of `requestedDelay`.

A channel shall, during `RETUNING` state, act on the carrier frequency according to `requestedFrequency` value:

- If equal to `UndefinedCarrierFreq`: keep the carrier frequency unchanged.
- If not equal to `UndefinedCarrierFreq`: apply `requestedFrequency` as the new carrier frequency.

A channel shall, during `RETUNING` state, act on the gain according to `requestedGain` value:

- If equal to `UndefinedGain`: keep the gain unchanged,
- If not equal to `UndefinedGain`: apply `requestedGain` as the new gain.

3.1.13 Transceiver::Notifications::Events

3.1.13.1 notifyEvent Operation

3.1.13.1.1 Overview

An event is defined as occurrence of a condition related to operation of a channel.
notifyEvent() informs the radio application that a defined event has occurred as depicted in the following figure:

![Diagram showing the principle of notifyEvent()](image)

**Figure 49  Principle of notifyEvent()**

Events are specified by the following table:

<table>
<thead>
<tr>
<th>Name / &lt;eventName&gt;</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventProcessingStart</td>
<td>Applies to: all channels. Condition: channels make a ProcessingStart transition.</td>
<td>2.3.1</td>
</tr>
<tr>
<td>eventProcessingStop</td>
<td>Applies to: all channels. Condition: channels make a ProcessingStop transition.</td>
<td>2.3.1</td>
</tr>
<tr>
<td>eventSilenceStart</td>
<td>Applies to: channels capable of radio silence. Condition: channels make a SilenceStart transition.</td>
<td>2.3.2</td>
</tr>
<tr>
<td>eventSilenceStop</td>
<td>Applies to: channels capable of radio silence. Condition: channels make a SilenceStop transition.</td>
<td>2.3.2</td>
</tr>
</tbody>
</table>

Table 14 Specification of events

3.1.13.1.2 Associated properties

EVENTS (see section 4.40) specifies, for each event, if event notification has to be performed.

EVENTS_WCET (see section 4.15) specifies the worst-case execution time of the primitive for correct real-time operation of the transceiver instance.

3.1.13.1.3 Declaration

The declaration of the operation is specified as:

```c
void notifyEvent(
    in Event notifiedEvent);
```

3.1.13.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifiedEvent</td>
<td>Event</td>
<td>Enumerated value specifying the notified event.</td>
</tr>
</tbody>
</table>

Table 15 Specification of notifyEvent() parameters

No parameters validity property is specified for use services.
3.1.13.1.5 Return value

None.

3.1.13.1.6 Originator

Transceiver.

3.1.13.1.7 Exception

Not applicable to use services.

3.1.13.1.8 Behavior requirements

Channels with an active instance of Events shall, when <eventName> happens and EVENTS.<eventName> is equal to true, call notifyEvent() with notifiedEvent equal to <eventName>.

Channels with an active instance of Events shall, if channels have been set in radio silence by another agent than the radio application when INITIALIZATION terminates, call notifyEvent() with notifiedEvent equal to eventSilenceStart.

3.1.14 Transceiver::Notifications::Errors

3.1.14.1 notifyError Operation

3.1.14.1.1 Overview

An error is defined as an abnormal situation related to channels internal execution errors.

notifyError() informs the radio application that a defined error (see section 3.2) has occurred as depicted in following figure:

![Diagram](image)

Figure 50 Principle of notifyError()
Errors are specified by the following table:

<table>
<thead>
<tr>
<th>Name / &lt;errorName&gt;</th>
<th>Specification</th>
<th>See §</th>
</tr>
</thead>
</table>
| error1stSampleDelayed       | Applies to: Tx channels with at least one active instance of a timely creation service.  
                               Condition: the first sample of a Tx forwarded block is not available before activation time. | 2.3.2 |
| error1stSampleTimeout       | Applies to: Tx channels with at least one active instance of a timely creation service, if ERRORS.err1stSampleDelayed.reaction is equal to mitigating.  
                               Condition: during a burst creation, the first sample of a Tx forwarded block is not available once 1ST_SAMPLE_TIMEOUT nanoseconds have elapsed after activation time. | 2.3.2 |
| errorBurstOverlap           | Applies to: channels with at least one active instance of a timely creation service.  
                               Condition: activation time of a burst under creation and termination time of the previous burst do not enable respect value of INTER-PROCESS. | 2.3.2  
                               4.8 |
| errorRxOverflow             | Applies to: Rx channels.  
                               Condition: the radio application did not return a pushRxPacket() call when Rx channels have to make the next call. | 3.1.7 |
| errorShorterTxBlock         | Applies to: Tx channels.  
                               Condition: a Tx forwarded block is ended (value of endOfBlock in a call to pushTxPacket() is set to true) and requestedLength is equal to UndefinedBlockLength or length of the baseband block is smaller than a defined value of requestedLength. | 2.3.1 |
| errorTxUnderflow            | Applies to: Tx channels.  
                               Condition: baseband samples are not available early enough for a Tx channel to proceed with up-conversion during a PROCESSING state. | 3.1.8 |
| errorTuningDelayed          | Applies to: channels with an active instance of InitialTuning.  
                               Condition: during a burst creation, the TuningStop transition has not occurred at the time required for ProcessingStart transition to satisfy the start time. | 3.1.11 |
| errorTuningTimeout          | Applies to: channels with an active instance of InitialTuning, if ERRORS.errTuningDelayed.reaction is equal to mitigating.  
                               Condition: during a burst creation, TuningStop transition has not occurred once TUNING_TIMEOUT nanoseconds elapsed after the activation time. | 2.3.2 |

Table 16 Specification of errors
3.1.14.1.2 Associated properties

**ERRORS** (see section 4.4) specifies, for each *error*, how it is handled by *active instances* of *Errors*:
- Applicable behavior when the *error* happens,
- If *error notification* has to be performed.

**ERRORS_WCET** (see section 4.15) specifies the worst-case execution time of the primitive for correct real-time operation of the *transceiver instance*.

3.1.14.1.3 Declaration

The declaration of the operation *is specified as*:

```c
void notifyError(
    in Error notifiedError);
```

3.1.14.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifiedError</td>
<td>Error</td>
<td>Specifies the notified <em>error</em>. (see § 3.4.8)</td>
</tr>
</tbody>
</table>

| Table 17  Specification of notifyError() parameters

No *parameters validity property* is specified for *use services*.

3.1.14.1.5 Return value

None.

3.1.14.1.6 Originator

*Transceiver*.

3.1.14.1.7 Exceptions

Not applicable to *use services*.

3.1.14.1.8 Behavior requirements

*Error notification of <errorName>* is defined as a call to notifyError(), independently of other channels operation, with *notifiedError* parameter equal to <errorName>.

*Channels* with an active instance of *Errors*, when <errorName> happens and ERRORS.<errorName>.reaction is equal to *fatal*, have *unspecified* behavior.

*Channels* with an active instance of *Errors shall*, when <errorName> happens and ERRORS.<errorName>.reaction is equal to *resetting*:
- Trigger a RuntimeReset transition,
- If ERRORS.<errorName>.isNotified is equal to *true*, perform *error notification*. 
Channels with an active instance of Errors shall, when <errorName> happens and ERRORS.<errorName>.reaction is equal to mitigation:

- Perform the error mitigation behavior specified in Table 18,
- If ERRORS.<errorName>.isNotified is equal to true, perform error notification.

Errors mitigation behaviors are specified by the following table:

<table>
<thead>
<tr>
<th>Error name</th>
<th>Error mitigation behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>error1stSampleDelayed</td>
<td>Wait until the first baseband sample is available, then make a</td>
</tr>
<tr>
<td></td>
<td>ProcessingStart transition (entry in PROCESSING state, see section</td>
</tr>
<tr>
<td></td>
<td>2.3.1).</td>
</tr>
<tr>
<td>error1stSampleTimeout</td>
<td>Unspecified.</td>
</tr>
<tr>
<td>errorBurstOverlap</td>
<td>Call setBlockLength() with requestedLength shortening the</td>
</tr>
<tr>
<td></td>
<td>length of previous burst so that its termination time is smaller</td>
</tr>
<tr>
<td></td>
<td>than the tuning time of the burst under creation.</td>
</tr>
<tr>
<td>errorRxOverflow</td>
<td>Drop the baseband samples delivered by down-conversion until the</td>
</tr>
<tr>
<td></td>
<td>pending call to pushRxPacket() returns.</td>
</tr>
<tr>
<td>errorShorterTxBlock</td>
<td>Call setBlockLength() with requestedLength equal to the</td>
</tr>
<tr>
<td></td>
<td>length of the Tx forwarded block.</td>
</tr>
<tr>
<td>errorTxUnderflow</td>
<td>Pad missing baseband samples with unspecified samples until new</td>
</tr>
<tr>
<td></td>
<td>baseband samples are available.</td>
</tr>
<tr>
<td>errorTuningDelayed</td>
<td>Wait until TuningStop transition, then make a</td>
</tr>
<tr>
<td></td>
<td>ProcessingStart transition (entry in PROCESSING state, see section</td>
</tr>
<tr>
<td></td>
<td>2.3.1).</td>
</tr>
<tr>
<td>errorTuningTimeout</td>
<td>Unspecified.</td>
</tr>
</tbody>
</table>

Table 18 Specification of errors mitigation behaviors

3.1.15 Transceiver::GainControl::GainChanges

3.1.15.1 indicateGain Operation

3.1.15.1.1 Overview

indicateGain() provides the radio application with a new value of gain decided by channels during a processing phase as depicted in following figure:
3.1.15.1.2 Associated properties

**GAIN_CHANGE_WCET** (see section 4.15) specifies the worst-case execution time of the primitive for correct real-time operation of the *transceiver instance*.

3.1.15.1.3 Declaration

The declaration of the operation **is specified as**:

```c
void indicateGain(
    in Gain newGain,
    in SampleNumber firstValidSample
);
```

3.1.15.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>newGain</em></td>
<td><em>Gain</em></td>
<td>Specifies the new value of <em>gain</em>.</td>
</tr>
<tr>
<td></td>
<td>(see § 3.4.10)</td>
<td></td>
</tr>
<tr>
<td><em>firstValidSample</em></td>
<td><em>SampleNumber</em></td>
<td>Sample number of the first sample in the <em>Rx block</em> after which the tuning is stable again.</td>
</tr>
<tr>
<td></td>
<td>(see § 3.4.14)</td>
<td></td>
</tr>
</tbody>
</table>

Table 19 Specification of *indicateGain()* parameters

No *parameters validity property* is specified for *use services*.

3.1.15.1.5 Return value

None.

3.1.15.1.6 Originator

*Transceiver*.

3.1.15.1.7 Exceptions

Not applicable to *use services*.

3.1.15.1.8 Behavior requirements

An *active instance* of *GainChanges* **shall** indicate each new value of *gain* using *indicateGain()*.

3.1.16 *Transceiver::*GainControl::*GainLocking*

3.1.16.1 lockGain Operation

3.1.16.1.1 Overview

*lockGain()* commands *Rx channels* to lock the applied *Rx gain*, that becomes not modifiable by *AGC*.
3.1.16.1.2 Associated properties

**LOCK_GAIN_WCET** (see section 4.15) specifies the worst-case execution time of the primitive.

3.1.16.1.3 Declaration

The declaration of the operation is specified as:

```c
void lockGain();
```

3.1.16.1.4 Parameters

None.

3.1.16.1.5 Returned value

None.

3.1.16.1.6 Originator

*Radio application*.

3.1.16.1.7 Exceptions

The exceptions of the operation are specified as (see section 3.2):

- **NoOngoingProcessingException**.

3.1.16.1.8 Behavior requirements

An active instance of **GainLocking** shall, on a call to **lockGain()**:

- Lock value of *Rx gain* at its current value independently of *AGC* operation,
- Return the call to the *radio application*.

3.1.16.2 unlockGain Operation

3.1.16.2.1 Overview

**unlockGain()** commands *Rx channels* to unlock *Rx gain*, that becomes subject to modifications under control of *AGC*.
3.1.16.2.2 Associated properties

**UNLOCK_GAIN_WCET** (see section 4.15) specifies the *maximum processing time* for correct joint real-time operation of *radio application* and *transceiver*.

3.1.16.2.3 Declaration

The declaration of the operation is specified as:

```c
void unlockGain();
```

3.1.16.2.4 Parameters

None.

3.1.16.2.5 Returned value

None.

3.1.16.2.6 Originator

*Radio application*.

3.1.16.2.7 Exceptions

The *exceptions* of the operation are specified as (see section 3.2):

- **NoOngoingProcessingException**.

3.1.16.2.8 Behavior requirements

An active instance of **GainLocking** shall, on a call to **unlockGain()**:  
- Enable *Rx gain* to be modified by *AGC*,  
- Return the call to the *radio application*.  

![Figure 53 Principle of unlockGain()](image-url)
3.1.17 Transceiver::TransceiverTime::TimeAccess

3.1.17.1 getCurrentTime Operation

3.1.17.1.1 Overview

gCurrentTime() commands the channels to return the value of transceiver time corresponding to return time of the call, as depicted in following figure:

![Figure 54 Principle of getCurrentTime()](image)

3.1.17.1.2 Associated properties

CURRENT_TIME_ACC (see section 4.12) specifies the accuracy of the returned currentTime value. CURRENT_TIME_WCET (see section 4.15) specifies the worst-case execution time of the primitive.

3.1.17.1.3 Declaration

The declaration of the operation is specified as:

```c
void getCurrentTime(
    out TimeSpec currentTime);
```

3.1.17.1.4 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentTime</td>
<td>TimeSpec (see § 3.4.16)</td>
<td>Value of transceiver time when getCurrentTime() returns.</td>
</tr>
</tbody>
</table>

Table 20 Specification of getCurrentTime() parameters

No parameters validity property is associated to out parameters.

3.1.17.1.5 Return value

None.

3.1.17.1.6 Originator

Radio application.

3.1.17.1.7 Exceptions

None.
3.1.17.1.8 Behavior requirements

An active instance of **TimeAccess** shall, on a call to `getCurrentTime()`:

- Set the return value of `currentTime` to a value belonging to value of `transceiver time` when `getCurrentTime()` returns ± `CURRENT_TIME_ACC`,
- Return the call to the *radio application*.

3.1.17.2 `getLastStartTime` Operation

3.1.17.2.1 Overview

`getLastStartTime()` commands the *channels* to return the value of *transceiver time* corresponding to the start time of the last burst created by the *channels* for which `getLastStartTime()` is called, and to return its *burst number*, as depicted in following figure:

![Diagram](image)

**Figure 55**  Principle of `getLastStartTime()`

`LAST_START_TIME_ACC` (see section 4.12) specifies the accuracy of the returned `lastStartTime` value.

`LAST_START_TIME_wcet` (see section 4.15) specifies the **maximum processing time** for correct joint real-time operation of *radio application* and *transceiver*.

3.1.17.2.2 Declaration

The declaration of the operation **is specified as**:

```c
void getLastStartTime(
    out TimeSpec lastStartTime,
    out BurstNumber lastBurstNumber);
```

3.1.17.2.3 Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lastStartTime</td>
<td><code>TimeSpec</code></td>
<td>Value of <em>transceiver time</em> for the start time of last created burst.</td>
</tr>
<tr>
<td></td>
<td>(see § 3.4.16)</td>
<td></td>
</tr>
<tr>
<td>lastBurstNumber</td>
<td><code>BurstNumber</code></td>
<td>Number of the last created burst.</td>
</tr>
<tr>
<td></td>
<td>(see § 3.4.5)</td>
<td></td>
</tr>
</tbody>
</table>

**Table 21**  Specification of `getLastStartTime()` parameters

3.1.17.2.4 Return value

None.
3.1.17.2.5 Originator
Radio application.

3.1.17.2.6 Exceptions
None.

3.1.17.2.7 Behavior requirements
An active instance of **TimeAccess shall**, on a call to `getLastStartTime()`, if no burst was created by the `channels` before the call to `getLastStartTime()`:

- Set the return value of `lastStartTime` to `UndefinedTimeSpec` (see section 3.4.16),
- Set the return value of `lastBurstNumber` to zero (0),
- Return the call to the radio application.

An active instance of **TimeAccess shall**, on a call to `getLastStartTime()`, if at least one burst was created by the `channels` before the call to `getLastStartTime()`:

- Set the return value of `lastStartTime` to a value belonging to the actual start time of the last burst created by the `channels` ± `LAST_START_TIME_ACC`,
- Set the return value of `lastBurstNumber` to the burst number of the last burst created by the `channels`,
- Return the call to the radio application.

3.1.18 `Transceiver::Strobing::ApplicationStrobe`

3.1.18.1 triggerStrobe Operation

3.1.18.1.1 Overview
`triggerStrobe()` provides the `channel` with a strobe occurrence.

3.1.18.1.2 Associated properties
**TRIGGER_STROBE_WCET** (see section 4.15) specifies the worst-case execution time of the primitive.

3.1.18.1.3 Declaration
The declaration of the operation is specified as:

```c
void triggerStrobe( void);
```

3.1.18.1.4 Parameters
None.

3.1.18.1.5 Returned value
None.
3.1.18.1.6 Originator
Radio application.

3.1.18.1.7 Exceptions
None.

3.1.18.1.8 Behavior requirements
A channel shall, on a call to triggerStrobe():

- Register the triggered strobe as a strobe occurrence for the ApplicationStrobe strobe source,
- Return the call to the radio application.

### 3.2 Exceptions

#### 3.2.1 Specification

An exception is defined as an abnormal situation related to the calling context or to parameters values, detected during execution of a called operation of a provide service (see section 2.1).

General exceptions are specified by the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoAlternateReferencing</td>
<td>Applies to: an active instance of RelativeCreation in a simplex transceiver or in a duplex transceiver with ALTERNATEREFERENCING equal to false. Condition: the value of requestCrossReference in a call to createRelativeBurst() is equal to true while the transceiver instance is simplex or ALTERNATEREFERENCING is false.</td>
<td>3.1.4 4.7</td>
</tr>
<tr>
<td>NoOngoingProcessing</td>
<td>Applies to: active instance of TerminationContol or Retuning. Condition: setBlockLength() or retune() is called while the channels are not in PROCESSING state.</td>
<td>3.1.7 3.1.12 2.3.1</td>
</tr>
<tr>
<td>StrobeSource</td>
<td>Applies to: an active instance of StrobedCreation. Condition: the value of requestedStrobeSource in a call to createStrobedBurst() has a corresponding field in STROBE_SOURCES equal to false.</td>
<td>3.1.6 4.7</td>
</tr>
</tbody>
</table>

Table 22 Specification of general exceptions
Range exceptions are specified by the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinBlockLength MaxBlockLength</td>
<td>Applies to: an active instance of a burst creation service or Termination. Condition: the value of requestedLength in a call to a creation operation or setBlockLength() is not equal to UndefinedBlockLength and is lower / greater than MIN_BLOCK_LENGTH / MAX_BLOCK_LENGTH.</td>
<td>3.1.7 4.7</td>
</tr>
<tr>
<td>MinCarrierFreq MaxCarrierFreq</td>
<td>Applies to: an active instance of InitialTuning or Retuning. Condition: the value of requestedFrequency in a call to retune() or setTuning() is not equal to UndefinedCarrierFreq and is lower / greater than MIN_CARRIER_FREQ / MAX_CARRIER_FREQ.</td>
<td>3.1.11 3.1.12 4.7</td>
</tr>
<tr>
<td>MinFromOngoing MaxFromOngoing</td>
<td>Applies to: an active instance of Retuning. Condition: the value of requestedDelay in a call to retune() is not equal to UndefinedDelay and is lower / greater than MIN_FROM_ONGOING / MAX_FROM_ONGOING.</td>
<td>3.1.12 4.7</td>
</tr>
<tr>
<td>MinFromPrevious MaxFromPrevious</td>
<td>Applies to: an active instance of RelativeCreation. Condition: the value of requestedDelay in a call to createRelativeBurst() is lower / greater than MIN_FROM_PREVIOUS / MAX_FROM_PREVIOUS.</td>
<td>3.1.14 4.7</td>
</tr>
<tr>
<td>MinFromStrobe MaxFromStrobe</td>
<td>Applies to: an active instance of StrobedCreation. Condition: the value of requestedDelay in a call to createStrobedBurst() is lower / greater than MIN_FROM_STROBE / MAX_FROM_STROBE.</td>
<td>3.1.16 4.7</td>
</tr>
<tr>
<td>MinGain MaxGain</td>
<td>Applies to: an active instance of InitialTuning or Retuning. Condition: the value of requestedGain in a call to retune() or setTuning() is not equal to UndefinedGain and is lower / greater than MIN_GAIN / MAX_GAIN.</td>
<td>3.1.11 3.1.12 4.7</td>
</tr>
<tr>
<td>MaxNanoseconds</td>
<td>Applies to: an active instance of AbsoluteCreation. Condition: the value of field nanoseconds of requestedStartTime in a call to createAbsoluteBurst() is greater than 999,999,999.</td>
<td>3.1.5</td>
</tr>
<tr>
<td>MaxRxPacketsLength</td>
<td>Applies to: an active instance of RxPacketsLengthControl. Condition: the value of requestedLength in a call to setRxPacketsLength() is greater than MAX_PACKETS_LENGTH.</td>
<td>3.1.10 4.7</td>
</tr>
<tr>
<td>MaxTuningPreset</td>
<td>Applies to: an active instance of InitialTuning. Condition: the value of requestedPreset in a call to setTuning() is greater than MAX_TUNING_PRESET.</td>
<td>3.1.11 4.7</td>
</tr>
<tr>
<td>MaxTxPacketsLength</td>
<td>Applies to: an active instance of SamplesTransmission. Condition: the length of txPacket in a call to pushTxPacket() is greater than MAX_PACKETS_LENGTH.</td>
<td>3.1.8 4.7</td>
</tr>
</tbody>
</table>

Table 23 Specification of range exceptions
MILT exceptions are specified by the following table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsoluteMILT</td>
<td>Applies to: an active instance of AbsoluteCreation. Condition: the invocation time of scheduleAbsoluteBurst() does not respect ABSOLUTE_MILT.</td>
<td>3.1.5</td>
</tr>
<tr>
<td>RelativeMILT</td>
<td>Applies to: an active instance of RelativeCreation. Condition: the invocation time of scheduleRelativeBurst() does not respect RELATIVE_MILT.</td>
<td>3.1.4</td>
</tr>
<tr>
<td>RetuningMILT</td>
<td>Applies to: an active instance of Retuning. Condition: the invocation time retune() does not respect RETUNING_MILT.</td>
<td>3.1.12</td>
</tr>
<tr>
<td>TuningMILT</td>
<td>Applies to: an active instance of InitialTuning. Condition: the invocation time of setTuning() does not respect TUNING_MILT.</td>
<td>3.1.11</td>
</tr>
<tr>
<td>TxPacketsMILT</td>
<td>Applies to: an active instance of SamplesTransmission. Condition: the invocation time of pushTxPacket() does not respect TX_PACKET_MILT.</td>
<td>3.1.9</td>
</tr>
</tbody>
</table>

Table 24 Specification of MILT exceptions

3.2.2 Associated properties

EXCEPTIONS_SUPPORT (see section 4.4) specifies if exceptions are supported.

EXCEPTIONS (see section 4.4) specifies for each exception, if EXCEPTIONS_SUPPORT is equal to true, how any active instance of a provide service behave when the exception occurs:

- Reaction of the provide service,
- Need to raise the exception.

3.2.3 Behavior requirements

The applicative handler of an exception <exceptionName> is defined as a part of the radio application dedicated to handling of <exceptionName> occurrences.

The exception raising of an exception <exceptionName> is defined as branching the execution of the radio application to an applicative handler of <exceptionName> instead of waiting for the called operation to return.

The applied PSM (see section 1.1) specifies how exception raising is realized.

An active instance of a provide service, when <exceptionName> occurs and EXCEPTIONS.<exceptionName>.reaction is equal to fatal, has unspecified behavior.

An active instance of a provide service shall, when <exceptionName> occurs and EXCEPTIONS.<exceptionName>.reaction is equal to resetting:
- Trigger a `RuntimeReset` transition,
- If `EXCEPTIONS.<exceptionName>.isNotified` is equal to `true`, perform exception raising.

An active instance of a provide service shall, when `<exceptionName>` occurs and `EXCEPTIONS.<exceptionName>.reaction` is equal to `callIgnoring`:
- Implement no requirement of the nominal execution of the called operation,
- If `EXCEPTIONS.<exceptionName>.isNotified` is equal to `true`, perform exception raising.

### 3.3 Attributes

This section specifies `channels attributes` referenced by the remainder of the specification.

All channel attributes are virtual: `transceiver instances` are not required to make them accessible to `radio applications`.

#### 3.3.1 Channels attributes

The *initial value* of a `channels` attribute is defined as the value of an attribute when `channels` enter the `OPERATING` state (see section 2.3.1).

3.3.1.1 burstCount

`burstCount` attribute is specified as the number of `bursts` created since the last entry in the `OPERATING` state (see section 2.3).

The associated declaration is specified as:

```
BurstNumber burstCount;
```

The *initial value* of `burstCount` is specified as 0 (zero).

Value of `burstCount` is incremented during `INITIATING` state of `CreationControl`, as specified in section 2.3.2.

3.3.1.2 applicableRxPacketsLength

`applicableRxPacketsLength` attribute is specified as the length of the `Rx packets` sent by an `Rx channel` with `pushRxPacket()` (see section 3.1.7).

The associated declaration is specified as:

```
PacketLength applicableRxPacketsLength;
```

`INIT_RX_PACKETS_LENGTH` (see section 4.6) specifies the *initial value* of `applicableRxPacketsLength`.

Value of `applicableRxPacketsLength` is changed by `radio applications` using `setRxPacketsLength()` (see section 3.1.10).
3.3.2 Processing attributes

3.3.2.1 applicableTuningPreset

The applicableTuningPreset attribute is specified as a reference to the transmit transfer function (see section 1.2.4) or the receive transfer function (see section 1.2.5) applied by channels during PROCESSING state (see section 2.3).

The associated declaration is specified as:

```
applicableTuningPreset;  // TuningPreset applicableTuningPreset;
```

applicableTuningPreset ranges from 1 (one) to MAX_TUNING_PRESET (see section 4.7).

For channels with no active instance of InitialTuning, the value of applicableTuningPreset is equal to 1 and cannot be modified.

For channels with an active instance of InitialTuning, no initial value of applicableTuningPreset is specified.

Value of applicableTuningPreset is controlled by radio applications using setTuning() (see section 3.1.11).

3.3.2.2 applicableCarrierFreq

The applicableCarrierFreq attribute is specified as the carrier frequency (see section 1.2.2.2) applied by channels during PROCESSING state (see section 2.3).

The associated declaration is specified as:

```
applicableCarrierFreq;  // CarrierFreq applicableCarrierFreq;
```

applicableCarrierFreq ranges from MIN_CARRIER_FREQ to MAX_CARRIER_FREQ (see section 4.7).

For channels with no active instance of InitialTuning, INIT_CARRIER_FREQ (see section 4.6) specifies the value of applicableCarrierFreq at beginning of the first burst.

For channels with an active instance of InitialTuning, no initial value of applicableCarrierFreq is specified.

Value of applicableCarrierFreq is controlled by radio applications using setTuning() (see section 3.1.11) and retrun() (see section 3.1.12).

3.3.2.3 applicableGain

The applicableGain attribute is specified as the transmit gain (see section 1.2.4.4) or the receive gain (see section 1.2.5) applied by channels during PROCESSING state (see section 2.3).

The associated declaration is specified as:

```
applicableGain;  // Gain applicableGain;
```
applicableGain ranges from MIN_GAIN to MAX_GAIN (see section 4.7).

For channels with no active instance of InitialTuning, INIT_GAIN (see section 4.6) specifies the value of applicableGain at beginning of the first burst.

For channels with an active instance of InitialTuning, no initial value of applicableGain is specified.

Value of applicableGain is controlled by radio applications using setTuning() (see section 3.1.11) and retune() (see section 3.1.12).

3.3.2.4 applicableLength

applicableLength attribute is specified as the length of the baseband block to be processed by channels during PROCESSING state (see section 2.3).

The associated declaration is specified as:

```
BlockLength applicableLength;
```

Undefined applicableLength is equal to UndefinedBlockLength (see section 3.4.3).

Defined applicableLength ranges from MIN_BLOCK_LENGTH to MAX_BLOCK_LENGTH (see section 4.7).

No initial value of applicableLength is specified.

Value of applicableLength is controlled by radio applications using creation operations (see section 2.4.2) and setBlockLength() (see section 3.1.7).

3.3.2.5 sampleCount

sampleCount attribute is specified as the number of samples of the baseband block processed by channels since entry in the PROCESSING state (see section 2.3).

The associated declaration is specified as:

```
SampleNumber sampleCount;
```

The start value of sampleCount is specified as 1 (one) for the first sample of the baseband block. Value of sampleCount is incremented during PROCESSING state of Channels, as specified in section 2.3.1.
3.4 Types

3.4.1 Base assumptions

The IDL keywords used for specification of types are:

- For Basic Types:
  - 16-bit integers: `short`, `unsigned short`,
  - 32-bit integers: `long`, `unsigned long`,
  - 64-bit integers: `long long`, `unsigned long long`,
  - Others: `float`, `boolean`,
- For Constructed Types: `typedef`, `struct`, `enum`,
- For Template Types: `sequence`.

This makes the specification compliant with the Full Profile or [Ref5], and with the ULw Profile augmented by `long long` and `float` basic types.

3.4.2 BasebandPacket

`BasebandPacket` type is specified as a sequence of `baseband samples`.

The associated declaration is specified as:

```
typedef sequence <BasebandSample> BasebandPacket;
```

`BasebandPacket` is used by `pushRxPacket()` (see section 3.1.7) and `pushTxPacket()` (see section 3.1.9).

3.4.3 BlockLength

`BlockLength` type is specified as a 32-bit unsigned integer number of `baseband samples` to be processed by `Tx channels` or `Rx channels` during a `processing phase`.

`UndefinedBlockLength` is specified as the reserved value specifying an `undefined` value of `BlockLength`.

The associated declarations are specified as:

```
typedef unsigned long BlockLength;
const BlockLength UndefinedBlockLength = 0xFFFFFFFF;
```

`BlockLength` is used by `startBurst()` (see section 3.1.3), `scheduleRelativeBurst()` (see section 3.1.4), `scheduleAbsoluteBurst()` (see section 3.1.5) and `scheduleStrobedBurst()` (see section 3.1.6).
3.4.4 BasebandSample

BasebandSample type is specified as the structure representing baseband samples, with field valueI for the in-phase component and field valueQ for the quadrature component (see section 1.2.2.1).

The associated declaration is specified as:
```c
struct BasebandSample {IQ valueI, IQ valueQ};
```

BasebandSample is used by declaration of IQ type (see section 3.4.11).

3.4.5 BurstNumber

BurstNumber type is specified as a 32-bit unsigned integer that specifies a burst number.

The associated declaration is specified as:
```c
typedef unsigned long BurstNumber;
```

BurstNumber is used by setTuning() (see section 3.1.11) and burstCount attribute (see 3.3.1.1).

3.4.6 CarrierFreq

CarrierFreq type is specified as an unsigned integer that specifies a carrier frequency ($f_c$).

CARRIER_FREQ_TYPE (see section 4.3) specifies if CarrierFreq is 32-bit or 64-bit.

A CarrierFreq value is expressed in hertz (Hz).

UndefinedCarrierFreq is specified as the reserved value specifying an undefined value of CarrierFreq.

The associated declarations are specified as, if CARRIER_FREQ_TYPE is equal to 32bit:
```c
typedef unsigned long CarrierFreq; // in Hz
const CarrierFreq UndefinedCarrierFreq = 0xFFFFFFFF;
```

The associated declarations are specified as, if CARRIER_FREQ_TYPE is equal to 64bit:
```c
typedef unsigned long long CarrierFreq; // in Hz
const CarrierFreq UndefinedCarrierFreq = 0xFFFFFFFFFFFFFFFF;
```

CarrierFreq is used by setTuning() (see section 3.1.11) and retune() (see section 3.1.12).

3.4.7 Delay

Delay type is specified as an unsigned integer that specifies a delay from the start time of an ongoing processing phase.

DELAY_TYPE (see section 4.3) specifies if Delay is 32-bit or 64-bit.

A Delay value is expressed in nanoseconds (ns).
UndefinedDelay is specified as the reserved value specifying an undefined value of Delay.

The associated declarations are specified as, if \texttt{DELAY\_TYPE} is equal to 32bit:

```c
typedef unsigned long Delay; // in ns
const Delay UndefinedDelay = 0xFFFFFFFF;
```

The associated declarations are specified as, if \texttt{DELAY\_TYPE} is equal to 64bit:

```c
typedef unsigned long long Delay; // in ns
const Delay UndefinedDelay = 0xFFFFFFFFFFFFFFFF;
```

Delay is used by\ scheduleRelativeBurst() (see section 3.1.4), \ scheduleStrobedBurst() (see section 3.1.6) and \ retune() (see section 3.1.12).

### 3.4.8 Error

Error type is specified as an enumeration identifying an error.

The associated declaration is specified as:

```c
enum Error {
    errorDelayedTuning,
    errorTuningTimeout,
    errorDelayedFirstSample,
    errorFirstSampleTimeout,
    errorTransmissionUnderflow,
    errorReceptionOverflow,
    errorShorterTransmittedBlock,
    errorLongerTransmittedBlock;
}
```

Error is used by \ notifyError() (see section 3.1.14.1).

### 3.4.9 Event

Event type is specified as an enumeration identifying an event.

The associated declaration is specified as:

```c
enum Event {
    eventProcessingStart,
    eventProcessingStop,
    eventSilenceStart,
    eventSilenceStop;
}
```

Event is used by \ notifyEvent() (see section 3.1.13.1).

### 3.4.10 Gain

Gain type is specified as a signed 16-bit integer that specifies a gain ($G$).

A Gain value is expressed in tenths of decibels ($1/10\ dB$).

UndefinedGain is specified as the reserved value specifying an undefined value of Gain.
The associated declarations are specified as:

```c
typedef short Gain; // in 1/10 dB
const Gain UndefinedGain = 0xFFFF;
```

*Gain* is used by `setTuning()` (see section 3.1.11) and `retune()` (see section 3.1.12).

### 3.4.11 IQ

*IQ* type is specified as the representation of I (in-phase) and Q (quadrature) components of a baseband sample.

*IQ_TYPE* (see section 4.3) specifies if *IQ* type is 16-bit, 32-bit or floating point.

Integer values of *IQ* shall be signed 2-complement MSB-aligned.

The declaration of *IQ* is specified as, if *IQ_TYPE* is equal to 16bit:

```c
typedef short IQ;
```

The declaration of *IQ* is specified as, if *IQ_TYPE* is equal to 32bit:

```c
typedef long IQ;
```

The declaration of *IQ* is specified as, if *IQ_TYPE* is equal to floatingPoint,

```c
typedef float IQ;
```

*IQ* is used for declaration of *BasebandSample* type (see section 3.4.4).

### 3.4.12 MetaData

*TxMetaData* and *RxMetaData* types are specified as structures of unspecified fields optionally used to attach meta-data to transferred baseband packets.

The associated declarations are user-defined, and shall be specified as follows:

```c
typedef struct TxMetaData {
    <user-defined>;
} TxMetaData;

typedef struct RxMetaData {
    <user-defined>;
} RxMetaData;
```

*TxMetaData* is used by `pushTxPacket()` (see section 3.1.9) and *RxMetaData* is used by `pushRxPacket()` (see section 3.1.8).

### 3.4.13 PacketLength

*PacketLength* type is specified as a 32-bit unsigned integer that identifies the length of a packet.

The associated declarations are specified as:

```c
typedef unsigned long PacketLength;
```
PacketLength is used by setRxPacketsLength() (see section 3.1.10) and applicableRxPacketsLength (see section 3.3.1.2).

3.4.14 SampleNumber

SampleNumber type is specified as a 32-bit unsigned integer that specifies a sample number. The associated declaration is specified as:

```c
typedef unsigned long SampleNumber;
```

SampleNumber is used by indicateGain() (see section 3.1.15.1) and sampleCount attribute (see 3.3.1.2).

3.4.15 StrobeSource

StrobeSource type is specified as an enumeration that specifies the referenced strobe source for strobed creation of a burst, as specified in section 3.1.6. The associated declaration is specified as:

```c
enum StrobeSource {
    ApplicationStrobe,
    TimeRef_PPS,
    GNSS_PPS,
    UserStrobe1,
    UserStrobe2,
    UserStrobe3,
    UserStrobe4};
```

StrobeSource is used by scheduleStrobedBurst() (see section 3.1.6).

3.4.16 TimeSpec

TimeSpec type is specified as a structure that specifies a value of transceiver time, composed of 32-bit unsigned integer fields for seconds and nanoseconds. The seconds field value is expressed in seconds (s). The nanoseconds field value is expressed in nanoseconds (ns). UndefinedTimeSpec is specified as the reserved value specifying an undefined value of TimeSpec. The associated declarations are specified as:

```c
struct TimeSpec {
    unsigned long seconds,       // in seconds
    unsigned long nanoseconds};  // in nanoseconds (<1.000.000.000)
const TimeSpec UndefinedTimeSpec = {0xFFFFFFFF, 0xFFFFFFFF};
```

TimeSpec is used by scheduleAbsoluteBurst() (see section 3.1.5), getCurrentTime() and getLastStartTime() (see section 3.1.15).
3.4.17 TuningPreset

*TuningPreset* type is specified as a 16-bit unsigned integer that identifies a tuning preset. *UndefinedTuningPreset* is specified as the reserved value specifying an *undefined* value of *TuningPreset*.

The associated declarations are specified as:

```
typedef unsigned short TuningPreset;
const TuningPreset UndefinedTuningPreset = 0xFFFF;
```

*TuningPreset* is used by *setTuning()* (see section 3.1.11).
4 Properties

This section specifies the Transceiver Properties, which characterize a transceiver instance, once it has been reconfigured in accordance to needs of the supported radio application.

4.1 Introduction

4.1.1 Properties

A property is defined as an attribute of a transceiver instance which value is defined when the channels have reached the CONFIGURED state.

The value of a property cannot be modified until the channels have exited the CONFIGURED state.

Note: future versions of the specification may enable modification of property values.

The remainder of the section specifies properties and their base name, also denoted <BaseName>. Depending on cases, a unique property can fully characterize a transceiver instance, or multiple properties can be required.

4.1.2 Properties naming

The name of a unique property shall be the <BaseName> of the property.

The names of multiple properties are constructed from the <BaseName> of the property with usage of prefixes or postfixes.

The name of multiple properties that differ between Tx channels and Rx channels shall be constructed with TX_ and RX_ prefixes added before the <BaseName>.

A property which base name starts with TX_ (resp. RX_) only applies to Tx channels (resp. Rx channels).

The name of multiple properties that differ according to conditions shall be constructed with the condition-dependent <Condition> postfixes added after the <BaseName> and a separation composed of two (2) underscores (__).

Any conditions and associated <Condition> postfixes can be user-defined.

For rapidity properties, section 4.8 specifies standard conditions and <Condition> postfixes.

4.1.3 Portability engineering support

The configuration expectations of a radio application are defined as the properties values of each used transceiver instance required for correct operation after the CONFIGURED state is reached.

The configuration capabilities of a transceiver implementation are defined as the properties values possibly supported by the transceiver after the CONFIGURED state is reached.
Porting feasibility can be evaluated through comparison of the radio application’s configuration expectations with transceiver’s configuration capabilities.

Note: derived specifications may standardize machine readable meta-data for expression of configuration expectations, enabling automation of porting feasibility evaluations and, for some advanced implementations, of the configuration of the transceiver instances.

4.1.4 Profiles

A profile of the specification is defined as a standard that specifies values of properties for radio applications and transceivers to facilitate or even guarantee that porting of any compliant radio application is feasible on any compliant transceiver implementation.

Note: development of profiles is out of the scope of the specification, but may be standardized by derived specifications.
4.2 Structure

A structure property is defined as a property that specifies an aspect related to the structure of a transceiver instance.

Structure properties are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Type</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX_CHANNELS</td>
<td>unsigned short</td>
<td>Applies to: any transceiver instance. Specifies: number of Tx channels (equal to number of active instances of SamplesTransmission).</td>
<td>1.2.1</td>
</tr>
<tr>
<td>RX_CHANNELS</td>
<td>unsigned short</td>
<td>Applies to: any transceiver instance. Specifies: number of Rx channels (equal to number of active instances of SamplesReception).</td>
<td>1.2.1</td>
</tr>
<tr>
<td>DUPLEX</td>
<td>Enumeration</td>
<td>Applies to: a duplex transceiver (TX_CHANNELS &gt; 0 and RX_CHANNELS &gt; 0). Specifies: duplex type of the transceiver instance: fullDuplex, halfDuplex.</td>
<td>1.2.1</td>
</tr>
<tr>
<td>TX_SHAPING</td>
<td>Enumeration</td>
<td>Applies to: Tx channels. Specifies: shaping of Tx bursts: nominal, specific.</td>
<td>1.2.4</td>
</tr>
<tr>
<td>TX_SERVICES</td>
<td>ActiveServices</td>
<td>Applies to: Tx channels. Specifies: for each service except SamplesTransmission, if one active instance is attached to Tx channels.</td>
<td>1.3.3</td>
</tr>
<tr>
<td>RX_SERVICES</td>
<td>ActiveServices</td>
<td>Applies to: Rx channels. Specifies: for each service except SamplesReception, if one active instance is attached to Rx channels.</td>
<td>1.3.3</td>
</tr>
<tr>
<td>TIME_COUPLING</td>
<td>Enumeration</td>
<td>Applies to: channels with active instance of AbsoluteCreation. Specifies: coupling of transceiver time: autonomous: uncorrelated with any other time, coupled: identical to another time, coupledToTerminalTime: identical to Terminal Time of Timing Service API (see Ref7).</td>
<td>3.1.5</td>
</tr>
</tbody>
</table>

Table 25 Structure properties

The declaration of DUPLEX is specified as:

```c
enum DUPLEX {fullDuplex, halfDuplex};
```

The declaration of TX_SHAPING is specified as:

```c
enum TX_SHAPING {nominal, specific};
```
The declarations for `TX_SERVICES` and `RX_SERVICES` are specified as:

```c
typedef boolean isActive;

typedef struct {
    // Provide services
    isActive reset,
    isActive radioSilence,
    isActive directCreation,
    isActive relativeCreation,
    isActive absoluteCreation,
    isActive strobedCreation,
    isActive termination,
    isActive rxPacketsLengthControl,
    isActive initialTuning,
    isActive retuning,
    isActive gainLocking,
    isActive timeAccess,
    isActive applicationStrobe,

    // Use services
    isActive events,
    isActive errors
    isActive gainChanges,
} ActiveServices;

ActiveServices TX_SERVICES;
ActiveServices RX_SERVICES;
```

The following consistency conditions apply to fields of `TX_SERVICES` and `RX_SERVICES`:

- At least one among `directCreation`, `relativeCreation`, `absoluteCreation` and `strobedCreation` is equal to `true`,
- `rxPacketsLengthControl` of `TX_SERVICES` is equal to `false`,
- `timeAccess` is equal to `false` if `relativeCreation` is equal to `false`,
- `applicationStrobe` is equal to `false` if `strobedCreation` is equal to `false`.

The declaration of `TIME_COUPLING` is specified as:

```c
enum TIME_COUPLING {autonomous, coupled, coupledToTerminalTime};
```
4.3 Behavior

A behavior property is defined as a property that specifies an aspect relative to the behavior of a transceiver instance.

Behavior properties are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Type</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
</table>
| TUNING_ASSOCIATION | Enumeration (see below) | Applies to: channels with an active instance of InitialTuning.  
                      |                     | Specifies: search condition among stored tuning parameters sets applicable during INITIATING:  
                      |                     | • sequential,  
                      |                     | • burstReferencing.  | 2.3.2 |
| AGC                | Enumeration (see below) | Applies to: Rx channels.  
                      |                     | Specifies: nature of the implemented AGC:  
                      |                     | • noAGC,  
                      |                     | • earlyControl,  
                      |                     | • permanentControl.  | 2.3.1 |
| ALC                | Enumeration (see below) | Applies to: Tx channels.  
                      |                     | Specifies: nature of the implemented ALC:  
                      |                     | • noALC,  
                      |                     | • activeALC.  | 2.3.1 |
| TUNING_TIMEOUT     | unsigned long | Applies to: channels with an active instance of InitialTuning, if  
                      |                     | ERRORS.errTuningDelayed.reaction is equal to mitigating.  
                      |                     | Specifies: timeout value, in nanoseconds (ns), for triggering of errorTuningTimeout.  | 3.1.14 |
| 1ST_SAMPLE_TIMEOUT | unsigned long | Applies to: Tx channels with at least one active instance of timely creation services, if  
                      |                     | ERRORS.err1stSampleDelayed.reaction is equal to mitigating.  
                      |                     | Specifies: timeout value, in nanoseconds (ns), for triggering of error1stSampleTimeout.  | 3.1.14 |

Table 26 Behavior properties

The declaration of TUNING_ASSOCIATION is specified as:
```
enum TUNING_ASSOCIATION {sequential, burstReferencing};
```

The declaration of AGC is specified as:
```
enum AGC {noAGC, startupAGC, permanentAGC};
```

The declaration of ALC is specified as:
```
enum ALC {noALC, activeALC};
```
4.4 Notifications

A notification property is defined as a property that specifies an aspect relative to notifications made by a transceiver instance to the radio application.

Notification properties are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Type</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCEPTIONS_SUPPORT</td>
<td>boolean</td>
<td>Applies to: all channels. Specifies: if exceptions are supported.</td>
<td>3.2</td>
</tr>
<tr>
<td>EXCEPTIONS</td>
<td>Structure (see below)</td>
<td>Applies to: all channels. Specifies: an exceptionHandling field for each standard exception, which specifies the reaction to occurrences of the exception and if the exception is raised to the radio application with the exception notification mechanism.</td>
<td>3.2</td>
</tr>
<tr>
<td>EVENTS</td>
<td>Structure (see below)</td>
<td>Applies to: channels with an active instance of Events. Specifies: an isNotified field for each event, which specifies if occurrences are notified to the radio application with notifyEvent().</td>
<td>3.1.13</td>
</tr>
<tr>
<td>ERRORS</td>
<td>Structure (see below)</td>
<td>Applies to: channels with an active instance of Errors. Specifies: an errorHandling field for each error, which specifies the reaction to occurrences of the error and if occurrences are notified to the radio application with notifyError().</td>
<td>3.1.14</td>
</tr>
</tbody>
</table>

Table 27 Notification properties

The declarations for ERRORS are specified as:

```c
typedef struct{
    enum reaction {fatal, reset, mitigation},
    boolean isNotified
} errorHandling;

struct ERRORS {
    errorHandling error1stSampleDelayed,
    errorHandling error1stSampleTimeout,
    errorHandling errorBurstOverlap,
    errorHandling errorRxOverflow,
    errorHandling errorShorterTxBlock,
    errorHandling errorTxUnderflow,
    errorHandling errorTuningDelayed,
    errorHandling errorTuningTimeout;
}
```
The declarations for **EXCEPTIONS** are specified as:

```c
typedef struct{
    enum reaction {fatal, resetting, callIgnoring}
    boolean isRaised}
exceptionHandling;

struct EXCEPTIONS {
    // General exceptions
    exceptionHandling NoAlternateReferencing,
    exceptionHandling NoOngoingProcessing,
    exceptionHandling StrobeSource,

    // Range exceptions
    exceptionHandling MaxBlockLength,
    exceptionHandling MinBlockLength,
    exceptionHandling MaxCarrierFreq,
    exceptionHandling MinCarrierFreq,
    exceptionHandling MaxFromOngoing,
    exceptionHandling MinFromOngoing,
    exceptionHandling MinFromPrevious,
    exceptionHandling MaxFromPrevious,
    exceptionHandling MaxFromStrobe,
    exceptionHandling MinFromStrobe,
    exceptionHandling MaxGain,
    exceptionHandling MinGain,
    exceptionHandling MaxNanoseconds,
    exceptionHandling MaxRxPacketsLength,
    exceptionHandling MaxTuningPreset,
    exceptionHandling MaxTxPacketsLength

    // MILT exceptions
    exceptionHandling AbsoluteMILT,
    exceptionHandling RelativeMILT,
    exceptionHandling RetuningMILT,
    exceptionHandling TuningMILT,
    exceptionHandling TxPacketsMILT};
```

The declarations for **EVENTS** are specified as:

```c
typedef boolean isNotified;

struct EVENTS {
    isNotified eventProcessingStart,
    isNotified eventProcessingStop,
    isNotified eventSilenceStart,
    isNotified eventSilenceStop};
```
# 4.5 Interface declaration

An interface declaration property is defined as a property that specifies an aspect relative to the declaration of a service interface.

Interface declaration properties are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Type</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARRIER_FREQ_TYPE</td>
<td>Enumeration (see below)</td>
<td>Applies to: CarrierFreq type. Specifies: type used (32-bit or 64-bit).</td>
<td>3.4.6</td>
</tr>
<tr>
<td>DELAY_TYPE</td>
<td>Enumeration (see below)</td>
<td>Applies to: Delay type. Specifies: type used (32-bit or 64-bit).</td>
<td>3.4.7</td>
</tr>
<tr>
<td>IQ_TYPE</td>
<td>Enumeration (see below)</td>
<td>Applies to: IQ type. Specifies: type used (16-bit, 32-bit or floating point).</td>
<td>3.4.11</td>
</tr>
<tr>
<td>TX_META_DATA</td>
<td>boolean</td>
<td>Specifies if user-defined meta-data are attached to the Tx packets forwarded to Tx channels.</td>
<td>3.1.9</td>
</tr>
<tr>
<td>RX_META_DATA</td>
<td>boolean</td>
<td>Specifies if user-defined meta-data are attached to the Rx packets obtained from Rx channels.</td>
<td>3.1.8</td>
</tr>
</tbody>
</table>

Table 28 Interface declaration properties

The associated declarations are specified as:

```c
enum CARRIER_FREQ_TYPE {int32, int64};
enum DELAY_TYPE {int32, int64};
enum IQ_TYPE {int16, int32, float32};
```

# 4.6 Initialization

An initialization property is defined as a property that specifies the conditions to be met by a transceiver instance when the CONFIGURED state is reached by its Tx channels and Rx channels.

Initialization properties are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Type</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>INIT_RX_PACKETS_LENGTH</td>
<td>PacketLength (see § 3.4.12)</td>
<td>Applies to: all Rx channels. Specifies: initial value of applicableRxPacketsLength.</td>
<td>3.3.1</td>
</tr>
<tr>
<td>INIT_CARRIER_FREQ</td>
<td>CarrierFreq (see § 3.4.6)</td>
<td>Applies to: channels with no active instance of InitialTuning. Specifies: the value of applicableCarrierFreq at beginning of the first burst.</td>
<td>3.3.2</td>
</tr>
<tr>
<td>INIT_GAIN</td>
<td>Gain (see § 3.4.10)</td>
<td>Applies to: channels with no active instance of InitialTuning. Specifies: the value of applicableGain at beginning of first burst.</td>
<td>3.3.2</td>
</tr>
</tbody>
</table>

Table 29 Initialization properties
### 4.7 Parameters validity

A parameter validity property is defined as a property that specifies the validity conditions applicable to a parameter of a primitive of a service interface.

Parameters validity properties are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Type</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN_BLOCK_LENGTH</td>
<td>BlockLength</td>
<td>Applies to: requestedLength not equal to UndefinedBlockLength in a call to a creation operation.</td>
<td>3.1.3</td>
</tr>
<tr>
<td>MAX_BLOCK_LENGTH</td>
<td></td>
<td>Specifies: minimum and maximum value.</td>
<td>3.1.4</td>
</tr>
<tr>
<td>ALTERNATEREFERENCING</td>
<td>boolean</td>
<td>Applies to: requestedAlternate in a call to scheduleRelativeBurst().</td>
<td>3.1.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifies: if true value is supported.</td>
<td>3.1.6</td>
</tr>
<tr>
<td>MIN_FROM_PREVIOUS</td>
<td>Delay</td>
<td>Applies to: requestedDelay in a call to scheduleRelativeBurst().</td>
<td>3.1.7</td>
</tr>
<tr>
<td>MAX_FROM_PREVIOUS</td>
<td></td>
<td>Specifies: minimum and maximum value.</td>
<td></td>
</tr>
<tr>
<td>STROBEOURCES</td>
<td>Structure</td>
<td>Applies to: requestedStrobeSource in scheduleStrobedBurst().</td>
<td>3.1.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifies: for each boolean field attached to a strobe source, if the corresponding value of</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>requestedStrobeSource is supported.</td>
<td></td>
</tr>
<tr>
<td>MIN_FROM_STROBE</td>
<td>Delay</td>
<td>Applies to: requestedDelay in a call to scheduleStrobedBurst().</td>
<td>3.1.9</td>
</tr>
<tr>
<td>MAX_FROM_STROBE</td>
<td></td>
<td>Specifies: minimum and maximum value.</td>
<td>3.1.5</td>
</tr>
<tr>
<td>MAX_PACKETS_LENGTH</td>
<td>PacketLength</td>
<td>Applies to: length of txPacket in a call to pushTxPacket() or requestedLength in a call to</td>
<td>3.1.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setRxPacketsLength().</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifies: maximum value.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: minimum value is constant and equal to 1.</td>
<td></td>
</tr>
<tr>
<td>MAX_TUNING_PRESET</td>
<td>TuningPreset</td>
<td>Applies to: requestedPreset in a call to setTuning().</td>
<td>3.1.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifies: maximum value.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: minimum value is constant and equal to 1.</td>
<td></td>
</tr>
<tr>
<td>MIN_CARRIER_FREQ</td>
<td>CarrierFreq</td>
<td>Applies to: requestedFrequency not equal to UndefinedCarrierFreq in a call to setTuning() or</td>
<td>3.1.12</td>
</tr>
<tr>
<td>MAX_CARRIER_FREQ</td>
<td></td>
<td>retune().</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifies: minimum and maximum value.</td>
<td></td>
</tr>
<tr>
<td>MIN_GAIN</td>
<td>Gain</td>
<td>Applies to: requestedGain not equal to UndefinedGain in a call to setTuning() or retune().</td>
<td>3.1.11</td>
</tr>
<tr>
<td>MAX_GAIN</td>
<td></td>
<td>Specifies: minimum and maximum value.</td>
<td>3.1.12</td>
</tr>
<tr>
<td>MIN_FROMONGOING</td>
<td>Delay</td>
<td>Applies to: requestedDelay in a call to retune().</td>
<td>3.1.12</td>
</tr>
<tr>
<td>MAX_FROMONGOING</td>
<td></td>
<td>Specifies: minimum and maximum value.</td>
<td></td>
</tr>
</tbody>
</table>

**Table 30 Parameters validity properties**
The declaration of `STROBE_SOURCES` is specified as:

```c
typedef boolean isSupported;

struct STROBE_SOURCES {
    isSupported ApplicationStrobe,
    isSupported TimeRef_PPS,
    isSupported GNSS_PPS,
    isSupported UserStrobe1,
    isSupported UserStrobe2,
    isSupported UserStrobe3,
    isSupported UserStrobe4);
```

### 4.8 Rapidity

A *rapidity property* is defined as a property that specifies the rapidity of execution of a transceiver instance.

Rapidity properties are specified as indicated in the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Type</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTER-PROCESSING</td>
<td><code>unsigned long</code></td>
<td>Applies to: channels. Specifies: minimum time, in nanoseconds (ns), between:</td>
<td>1.2.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Termination time of a burst (a <code>StopProcessing</code> transition),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Activation time of the next burst (<code>StartProcessing</code> transition).</td>
<td></td>
</tr>
<tr>
<td>INTER-BURST</td>
<td><code>unsigned long</code></td>
<td>Applies to: channels. Specifies: minimum time, in nanoseconds (ns), between:</td>
<td>1.2.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Stop time of a burst (end of its core burst, at its start time plus block length / (F_{BB})),</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Start time of the next burst (end of its core burst).</td>
<td></td>
</tr>
<tr>
<td>TUNING_DURATION</td>
<td><code>unsigned long</code></td>
<td>Applies to: channels with an active instance of <code>Tuning</code>. Specifies: maximum duration, in nanoseconds (ns), of the TUNING state.</td>
<td>2.3.1</td>
</tr>
<tr>
<td>RETUNING_DURATION</td>
<td><code>unsigned long</code></td>
<td>Applies to: channels with an active instance of <code>Retuning</code>. Specifies: maximum duration, in nanoseconds (ns), of the RETUNING state.</td>
<td>2.3.4</td>
</tr>
<tr>
<td>EARLY_AGC_DELAY</td>
<td><code>unsigned long</code></td>
<td>Applies to: Rx channels with AGC equal to <code>earlyControl</code>. Specifies: delay available after start time of a Rx burst for the AGC to have set the receive gain.</td>
<td>2.3.1</td>
</tr>
</tbody>
</table>

Table 31  Rapidity properties
**Tuning conditions** are specified as indicated in the following table:

<table>
<thead>
<tr>
<th>&lt;Condition&gt; postfix</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_TUNING_CHANGE</td>
<td>Applies to: <strong>INTER-BURST</strong>, <strong>INTER-PROCESSING</strong> and <strong>TUNING-DURATION</strong> of channels with an active instance of InitialTuning. Condition: the applicable tuning parameters set specifies no tuning change (requestedTuningPreset is equal to undefinedTuningPreset, requestedCarrierFreq is equal to UndefinedCarrierFreq and requestedDelay is equal to UndefinedDelay).</td>
</tr>
<tr>
<td>NEW_TUNING_PRESET</td>
<td>Applies to: <strong>INTER-BURST</strong>, <strong>INTER-PROCESSING</strong> and <strong>TUNING-DURATION</strong> of channels with an active instance of InitialTuning. Condition: the applicable tuning parameters set specifies a new tuning preset (requestedTuningPreset is not equal to undefinedTuningPreset).</td>
</tr>
<tr>
<td>NEW_FREQUENCY</td>
<td>Applies to: <strong>INTER-BURST</strong>, <strong>INTER-PROCESSING</strong> and <strong>TUNING-DURATION</strong> of channels with an active instance of InitialTuning and <strong>RETUNING-DURATION</strong> of channels with an active instance of Retuning. Condition: the applicable tuning parameters set specifies a new frequency with no tuning preset change (requestedTuningPreset is equal to undefinedTuningPreset and requestedCarrierFreq is not equal to UndefinedCarrierFreq).</td>
</tr>
<tr>
<td>NEW_GAIN</td>
<td>Applies to: <strong>INTER-BURST</strong>, <strong>INTER-PROCESSING</strong> and <strong>TUNING-DURATION</strong> of channels with an active instance of InitialTuning and <strong>RETUNING-DURATION</strong> of channels with an active instance of Retuning. Condition: the applicable tuning parameters set specifies a new gain with no other change (requestedTuningPreset is equal to undefinedTuningPreset, requestedCarrierFreq is equal to UndefinedCarrierFreq and requestedDelay is not equal to UndefinedDelay).</td>
</tr>
</tbody>
</table>

Table 32 Tuning conditions

See section 2.3.2.1.3 for further information regarding applicable tuning parameters set.

**Duplex conditions** are specified as indicated in the following table:

<table>
<thead>
<tr>
<th>&lt;Condition&gt; postfix</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX-TX</td>
<td>Applicable to: <strong>INTER-BURST</strong>, <strong>INTER-PROCESSING</strong> and <strong>TUNING-DURATION</strong> of all Tx channels. Condition: the consecutive bursts are Tx bursts.</td>
</tr>
<tr>
<td>RX-RX</td>
<td>Applicable to: <strong>INTER-BURST</strong>, <strong>INTER-PROCESSING</strong> and <strong>TUNING-DURATION</strong> all Rx channels. Condition: the consecutive bursts are Rx bursts.</td>
</tr>
<tr>
<td>TX-RX</td>
<td>Applicable to: <strong>INTER-BURST</strong>, <strong>INTER-PROCESSING</strong> and <strong>TUNING-DURATION</strong> of half-duplex transceivers. Condition: the previous burst is a Tx burst and the next burst is a Rx burst.</td>
</tr>
<tr>
<td>RX-TX</td>
<td>Applicable to: <strong>INTER-BURST</strong>, <strong>INTER-PROCESSING</strong> and <strong>TUNING-DURATION</strong> half-duplex transceivers. Condition: the previous burst is a Tx burst and the next burst is a Rx burst.</td>
</tr>
</tbody>
</table>

Table 33 Duplex conditions
4.9 Storage

A storage property is defined as a property that specifies the number of calls to certain operations a transceiver instance can store before blocking further calls until storage is freed.

Storage properties are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Type</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATION_STORAGE</td>
<td>unsigned short</td>
<td>Applies to: all Tx channels and Rx channels. Specifies: maximum number of creation operations calls the transceiver instance can store.</td>
<td>3.1.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.1.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.1.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.1.6</td>
</tr>
<tr>
<td>TUNING_STORAGE</td>
<td>unsigned short</td>
<td>Applies to: channels with an active instance of InitialTuning. Specifies: maximum number of setTuning() calls the transceiver instance can store.</td>
<td>3.1.11</td>
</tr>
<tr>
<td>TX_BASEBAND_STORAGE</td>
<td>unsigned long</td>
<td>Applies to: Tx channels. Specifies: maximum number of baseband samples the transceiver instance can store for each active instance of SamplesTransmission.</td>
<td>3.1.9</td>
</tr>
</tbody>
</table>

Table 34 Storage properties

4.10 Levels

A level property is defined as a property that specifies the range of signal levels at the boundary of channels.

Level properties are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Type</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX_MIN_BASEBAND_LEVEL</td>
<td>short</td>
<td>Applies to: Tx channels. Specifies: minimum and maximum values of the level of baseband signal at input of Tx channels, in tenth of decibels relative to full scale (1/10 dBFS).</td>
<td>2.3.1</td>
</tr>
<tr>
<td>TX_MAX_BASEBAND_LEVEL</td>
<td></td>
<td></td>
<td>2.3.1</td>
</tr>
<tr>
<td>RX_MIN_RADIO_LEVEL</td>
<td>short</td>
<td>Applies to: Rx channels. Specifies: minimum and maximum values of the level of radio signal at input of Rx channels, in tenth of decibels relative to one milliwatt (1/10 dBm).</td>
<td>2.3.1</td>
</tr>
<tr>
<td>RX_MAX_RADIO_LEVEL</td>
<td></td>
<td></td>
<td>2.3.1</td>
</tr>
<tr>
<td>RX_MIN_BASEBAND_LEVEL</td>
<td>short</td>
<td>Applies to: Rx channels. Specifies: minimum and maximum values of the level of baseband signal at output of Rx channels, in tenth of decibels relative to full scale (1/10 dBFS).</td>
<td>2.3.1</td>
</tr>
<tr>
<td>RX_MAX_BASEBAND_LEVEL</td>
<td></td>
<td></td>
<td>2.3.1</td>
</tr>
</tbody>
</table>

Table 35 Level properties

4.11 Channelization

A channelization property is defined as a property that specifies each tuning preset supported by a transceiver instance.
Channelization properties are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Type</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHANNEL_MASK</td>
<td>Structure</td>
<td>Applies to: all tuning presets. Specifies: the channel mask for the transfer function, to be respected during the PROCESSING state.</td>
<td>2.3.1</td>
</tr>
<tr>
<td>SAMPLING_FREQ_ACC</td>
<td>unsigned long</td>
<td>Applies to: channels. Specifies: accuracy of the baseband sampling frequency, in hertz (Hz), to be respected during the PROCESSING state.</td>
<td>2.3.1</td>
</tr>
<tr>
<td>CARRIER_FREQ_ACC</td>
<td>CarrierFreq</td>
<td>Applies to: channels. Specifies: accuracy of the carrier frequency, to be respected during the PROCESSING state.</td>
<td>2.3.1</td>
</tr>
<tr>
<td>GAIN_ACC</td>
<td>Gain</td>
<td>Applies to: channels. Specifies: accuracy of the gain, to be respected during the PROCESSING state.</td>
<td>2.3.1</td>
</tr>
</tbody>
</table>

Table 36 Channelization properties

One property instance of CHANNEL MASK is specified for each value of tuning preset between 1 and MAX_TUNING_PRESET (see section 4.7).

The associated names are specified as:

- CHANNEL_MASK if MAX_TUNING_PRESET is equal to 1,
- CHANNEL_MASK_<PresetNumber> if MAX_TUNING_PRESET is greater than 1.
The fields of *channel masks* are specified by the following figure:

![Figure 56 Specification of fields of channel masks](image-url)
The declaration for `CHANNEL_MASK` is specified as, taking the previous figure as reference for specification of the structure’s fields:

```c
typedef struct {
    // Sampling frequency
    unsigned long basebandSamplingFreq, // in Hz
    // Useful signal
    unsigned long channelBandwidth, // in Hz
    unsigned short ripple, // in the nth of dB
    unsigned short groupDelayDistorsion, // in ns
    // Proximity protection
    unsigned short lowerRejectionFreq, // in Hz
    unsigned short lowerRejectionGain, // in dB
    unsigned short lowerRejectionSlope, // in dB/kHz
    unsigned short upperRejectionFreq, // in Hz
    unsigned short upperRejectionGain, // in dB
    unsigned short upperRejectionSlope // in dB/kHz
} ChannelMask;
```

### 4.12 Temporal accuracy

A temporal accuracy property is defined as a property that specifies the temporal accuracy of a transceiver instance.

The type of a temporal accuracy property is specified as `unsigned long`.

Temporal accuracy properties are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Description</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>START_TIME_ACC</code></td>
<td>Applies to: channels with at least one active instance of a timely creation service.</td>
<td>2.3.2</td>
</tr>
<tr>
<td></td>
<td>Specifies: maximum absolute difference, in nanoseconds (ns), between:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Actual start time of a created burst,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Start time specified by the creation operation.</td>
<td></td>
</tr>
<tr>
<td><code>CURRENT_TIME_ACC</code></td>
<td>Applies to: channels with an active instance of <code>TimeAccess</code>.</td>
<td>3.1.15</td>
</tr>
<tr>
<td></td>
<td>Specifies: maximum absolute difference, in nanoseconds (ns), between:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Actual return time of <code>getCurrentTime()</code>,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Returned <code>currentTime</code> value.</td>
<td></td>
</tr>
<tr>
<td><code>LAST_START_TIME_ACC</code></td>
<td>Applies to: channels with an active instance of <code>TimeAccess</code>.</td>
<td>3.1.15</td>
</tr>
<tr>
<td></td>
<td>Specifies: maximum absolute difference, in nanoseconds (ns), between:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Actual start time of the last burst,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Returned <code>lastStartTime</code> value.</td>
<td></td>
</tr>
</tbody>
</table>

Table 37 Temporal accuracy properties

### 4.13 Invocation lead time

The invocation lead time of a `provide service` primitive is defined as the time elapsing, in nanoseconds (ns), between invocation of the primitive by the radio application and occurrence within the transceiver instance of the future related event.
The invocation lead time property of a provide service is defined as a property that specifies the minimum value of invocation lead time supported by the service.

The type of an invocation lead time property is specified as unsigned long.

Invocation lead time properties are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Provide service primitive</th>
<th>(Future) Related event</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELATIVE_MILT</td>
<td>RelativeCreation.</td>
<td>Start time of the burst.</td>
<td>3.1.4</td>
</tr>
<tr>
<td></td>
<td>scheduleRelativeBurst()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABSOLUTE_MILT</td>
<td>AbsoluteCreation.</td>
<td>Start time of the burst.</td>
<td>3.1.5</td>
</tr>
<tr>
<td></td>
<td>scheduleAbsoluteBurst()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STROBED_MILT</td>
<td>StrobedCreation.</td>
<td>Start time of the burst.</td>
<td>3.1.6</td>
</tr>
<tr>
<td></td>
<td>scheduleStrobedBurst()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TX_PACKET_MILT</td>
<td>SamplesTransmission.</td>
<td>First sample of the pushed packet is used by up-conversion.</td>
<td>3.1.9</td>
</tr>
<tr>
<td></td>
<td>pushTxPacket()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLOCK_LENGTH_MILT</td>
<td>Termination.</td>
<td>Stop time of the ongoing processing phase.</td>
<td>3.1.7</td>
</tr>
<tr>
<td></td>
<td>setBlockLength</td>
<td>If value of requestedLength is not equal to UndefinedBlockLength</td>
<td></td>
</tr>
<tr>
<td>TUNING_MILT</td>
<td>InitialTuning.</td>
<td>Usage of the creation operation of the burst by CreationControl.</td>
<td>3.1.11</td>
</tr>
<tr>
<td></td>
<td>setTuning()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETUNING_MILT</td>
<td>Retuning::retune()</td>
<td>Start of the RETUNING state.</td>
<td>3.1.12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If value of requestedDelay is not equal to UndefinedDelay</td>
<td></td>
</tr>
</tbody>
</table>

Table 38 Invocation lead time properties

4.14 Invocation delay

The invocation delay of a use service primitive is defined as the time elapsing, in nanoseconds (ns), between occurrence within a transceiver instance of the past related event and invocation of the primitive by the transceiver instance.

The invocation delay property of a use service is defined as a property that specifies the maximum value of invocation delay guaranteed by the service.

The type of an invocation delay property is specified as unsigned long.

Invocation delay properties are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Use service primitive</th>
<th>(Past) Related event</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH_RX_PACKET_MID</td>
<td>SamplesReception.</td>
<td>Down-conversion outputs the last sample of the pushed packet.</td>
<td>3.1.8</td>
</tr>
<tr>
<td></td>
<td>pushRxPacket()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOTIFY_EVENT_MID</td>
<td>Events.</td>
<td>The notified error occurs.</td>
<td>3.1.13</td>
</tr>
<tr>
<td></td>
<td>notifyEvent()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOTIFY_ERROR_MID</td>
<td>Errors.</td>
<td>The notified error is detected.</td>
<td>3.1.14</td>
</tr>
<tr>
<td></td>
<td>notifyError()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDICATE_GAIN_MID</td>
<td>GainChanges.</td>
<td>The indicated Gain starts to be applied in application of an AGC algorithm decision.</td>
<td>3.1.15</td>
</tr>
<tr>
<td></td>
<td>indicateGain()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 39 Invocation delay properties
### 4.15 Worst-case execution time (WCET)

The worst case execution time (WCET) of a service primitive is defined as the maximum length of time, in nanoseconds (ns), possibly taken between the invocation and the return of the primitive.

The WCET property of a primitive of a provide service is defined as a property that specifies the maximum value of the WCET of the primitive.

The WCET property of a primitive of a use service is defined as a property that specifies the maximum value of the WCET of the primitive for correct real-time behavior of the transceiver instance.

The type of a WCET property is specified as unsigned long.

WCET properties of primitives of provide services are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Related primitive</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESET_WCET</td>
<td>Reset::reset()</td>
<td>3.1.1</td>
</tr>
<tr>
<td>START_SILENCE_WCET</td>
<td>RadioSilence::startRadioSilence()</td>
<td>3.1.2</td>
</tr>
<tr>
<td>STOP_SILENCE_WCET</td>
<td>RadioSilence::stopRadioSilence()</td>
<td>3.1.2</td>
</tr>
<tr>
<td>DIRECT_WCET</td>
<td>DirectCreation::startBurst()</td>
<td>3.1.3</td>
</tr>
<tr>
<td>RELATIVE_WCET</td>
<td>RelativeCreation::scheduleRelativeBurst()</td>
<td>3.1.4</td>
</tr>
<tr>
<td>ABSOLUTE_WCET</td>
<td>AbsoluteCreation::scheduleAbsoluteBurst()</td>
<td>3.1.5</td>
</tr>
<tr>
<td>STROBED_WCET</td>
<td>StrobedCreation::scheduleStrobedBurst()</td>
<td>3.1.6</td>
</tr>
<tr>
<td>BLOCK_LENGTH_WCET</td>
<td>Termination::setBlockLength()</td>
<td>3.1.7</td>
</tr>
<tr>
<td>STOP_BURST_WCET</td>
<td>Termination::stopBurst()</td>
<td>3.1.7</td>
</tr>
<tr>
<td>TX_PACKET_WCET</td>
<td>SamplesTransmission::pushTxPacket()</td>
<td>3.1.9</td>
</tr>
<tr>
<td>RX_PACKETS_LENGTH_WCET</td>
<td>RxPacketsLengthControl::setRxPacketsLength()</td>
<td>3.1.10</td>
</tr>
<tr>
<td>TUNING_WCET</td>
<td>InitialTuning::setTuning()</td>
<td>3.1.11</td>
</tr>
<tr>
<td>RETUNING_WCET</td>
<td>Retuning::retune()</td>
<td>3.1.12</td>
</tr>
<tr>
<td>LOCK_GAIN_WCET</td>
<td>GainLocking::lockGain()</td>
<td>3.1.15</td>
</tr>
<tr>
<td>UNLOCK_GAIN_WCET</td>
<td>GainLocking::unlockGain()</td>
<td>3.1.16</td>
</tr>
<tr>
<td>CURRENT_TIME_WCET</td>
<td>TimeAccess::getCurrentTime()</td>
<td>3.1.17</td>
</tr>
<tr>
<td>LAST_START_TIME_WCET</td>
<td>TimeAccess::getLastStartTime()</td>
<td>3.1.17</td>
</tr>
<tr>
<td>TRIGGER_STROBE_WCET</td>
<td>ApplicationStrobe::triggerStrobe()</td>
<td>3.1.18</td>
</tr>
</tbody>
</table>

Table 40 WCET properties of provide operations

WCET properties of primitives of use services are specified by the following table:

<table>
<thead>
<tr>
<th>Base name</th>
<th>Related primitive</th>
<th>See §</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX_PACKET_WCET</td>
<td>SamplesReception::pushRxPacket()</td>
<td>3.1.8</td>
</tr>
<tr>
<td>EVENTS_WCET</td>
<td>Events::notifyEvent()</td>
<td>3.1.13</td>
</tr>
<tr>
<td>ERRORS_WCET</td>
<td>Errors::notifyError()</td>
<td>3.1.14</td>
</tr>
<tr>
<td>GAIN_CHANGE_WCET</td>
<td>GainChanges::indicateGain()</td>
<td>3.1.15</td>
</tr>
</tbody>
</table>

Table 41 WCET properties of use operations
5 References

5.1 Referenced documents

  


[Ref4] *OMG Unified Modeling Language (OMG UML)*, The Object Management Group,
  formal/2015-03-01, Version 2.5, March 2015
  
  http://www.omg.org/spec/UML/2.5

  
  https://sds.wirelessinnovation.org/specifications-and-recommendations
  https://winnf.memberclicks.net/assets/work_products/Specifications/winnf-14-s-0016-v2.0.2.pdf

[Ref6] *Application Interface Definition Language Platform Independent Model Profiles, SCA 4.1
  Appendix E-1*, Joint Tactical Networking Center, 20 August 2015
  
  http://www.public.navy.mil/jtnc/sca/Documents/SCAv4_1_Final/SCA_4.1_App_E-
  1_ApplicationIdlPimProfiles.pdf

  Tactical Networking Center, Version 1.4.4, 26 June 2013
  

The URLs above were successfully accessed at release date.