Time Service Facility PIM

Document WINNF-TS-3004

Version V1.0.0

9 December 2020

TERMS, CONDITIONS & NOTICES

Table of Contents

TERMS, CONDITIONS & NOTICES 1
Table of Contents ii
List of Figures iv
List of Tables vi
Contributors vii
Time Service Facility PIM 8
1 Introduction 8
1.1. The Time service facility 8
1.1.1 Purpose 8
1.1.2 Formal positioning 8
1.1.3 Composition 9
1.1.4 Origin 9
1.2. The PIM specification 9
1.2.1 Purpose 9
1.2.2 Reference definitions 10
1.2.3 Writing conventions 10
1.3. Conformance 10
1.3.1 Conformance principles 10
1.3.2 Definitions usage 10
1.4. Implementations and usages 11
1.4.1 Multiple time services 11
1.4.2 Multiple radio applications 11
1.4.3 Services distribution 12
1.5. Time concepts 13
1.5.1 Base notions 13
1.5.2 Standard times 13
1.5.3 Implemented times 15
1.6. Time handling 24
1.6.1 Time stamps 24
1.6.2 Estimation uncertainty 25
1.6.3 Time uncertainty 27
1.6.4 Time references 27
2 Services 29
2.1. Provide services 29
2.2. Use services 29
2.3. Service-level conformance 29
2.3.1 Services scope 29
2.3.2 selectedOptionalServices capability 30
2.3.3 Services implementation conformance 30
2.4. States machines 30
2.5. Services groups description 31
2.5.1 TimeService::TerminalTime 31

2.5.2 TimeService::SystemTime....31
2.5.3 TimeService::StandardTimes....32
2.5.4 TimeService::SpecificTimes....33
3 Service primitives and attributes....34
3.1. Service primitives....34
3.1.1 Specification approach....34
3.1.2 TimeService::TerminalTime::TerminalTimeAccess....35
3.1.3 TimeService::SystemTime::SystemTimeAccess....39
3.1.4 TimeService::SystemTime::StandardTimeProvision....47
3.1.5 TimeService::StandardTimes::ReferencesNotification....52
3.1.6 TimeService::SpecificTimes::SpecificTimeHandling....55
3.1.7 TimeService::SpecificTimes::SettingsNotification....59
3.2. Exceptions....62
3.2.1 Specification....62
3.2.2 Associated capabilities....63
3.3. Attributes....63
3.3.1 Overview....63
3.3.2 Attributes conformance....63
3.3.3 Capabilities....64
3.3.4 Properties....64
3.3.5 Variables....64
3.4. Types....65
3.4.1 TimeValue....65
3.4.2 TimeUncertainty....65
3.4.3 RateUncertainty....67
4 References....68
4.1. Referenced documents....68
5 Acronyms list....69
6 Reference tables....70
6.1. Definitions....70
6.2. Notations....72
6.3. Equations....72
6.4. Requirements....73
END OF THE DOCUMENT....73

List of Figures

Figure 1 Overview of *time service facility* 8
Figure 2 Usage configurations of multiple *time services* 11
Figure 3 Usage configurations of multiple *radio applications* 12
Figure 4 Notion of *services* distribution 12
Figure 5 Time function of the *TAI* 14
Figure 6 Time function of the *UTC* depicting leap seconds 14
Figure 7 Time functions of *terminal time* 17
Figure 8 Notion of *terminal time rate error* (*TTRE*) 18
Figure 9 *Time functions* of *system time* 20
Figure 10 Principle of a *system time update* 21
Figure 11 Typical *system time* in case of GNSS interruption 21
Figure 12 *Time functions* of a *specific time* 23
Figure 13 Possible *specific time* discontinuities 23
Figure 14 Notion of *stamping uncertainty* 25
Figure 15 Typical *system time uncertainty* evolution in case of GNSS interruption 27
Figure 16 TimeService statechart 30
Figure 17 TerminalTime *services group* 31
Figure 18 SystemTime *services group* 31
Figure 19 Principle of SystemTime and StandardTimes *services groups* 32
Figure 20 StandardTimes *services group* 32
Figure 21 SpecificTimes *services group* 33
Figure 22 Principle of SpecificTimes *services group* 33
Figure 23 TerminalTime::TerminalTimeAccess *service interface* 35
Figure 24 *getTerminalTime()* overview 36
Figure 25 *getTerminalTime()* real-time capabilities 37
Figure 26 *getTerminalTimeRateUncertainty()* overview 38
Figure 27 SystemTime::SystemTimeAccess *service interface* 39
Figure 28 *getCurrentTAI()* overview 39
Figure 29 *getCurrentTAI()* real-time capabilities 41
Figure 30 *getCurrentUTC()* overview 41
Figure 31 *getCurrentUTC()* real-time capabilities 43
Figure 32 Principle of *getLastUpdateTAI()* 43
Figure 33 *getLastUpdateTAI()* capabilities 45
Figure 34 Principle of *getLastUpdateUTC()* 45
Figure 35 Meaning of *getLastUpdateUTC()* capabilities 47
Figure 36 SystemTime::StandardTimeProvision *service interface* 47
Figure 37 *provideTAI()* overview 47
Figure 38 *provideTAI()* real-time capabilities 49
Figure 39 Principle of *provideUTC()* 50
Figure 40 *provideUTC()* real-time capabilities 52
Figure 41 StandardTimes::ReferencesNotification *service interface* 52
Figure 42 *notifyStandardTimeReference()* overview 53
Figure 43 *notifyStandardTimeReference()* real-time capabilities 55
Figure 44 SystemTimes::SpecificTimeHandling *service interface* 55

Figure 45 *setSpecificTime()* overview 55
Figure 46 *setSpecificTime ()* real-time *capabilities* 57
Figure 47 *getSpecificTime()* overview 57
Figure 48 *getSpecificTime()* real-time capabilities 59
Figure 49 SpecificTimes::SettingsNotification *service interface* 59
Figure 50 *notifySpecificTimeSetting()* overview 60
Figure 51 *notifySpecificTimeSetting()* real-time capabilities 62

List of Tables

Table 1 Applied general definitions 10
Table 2 *Provide services* 29
Table 3 *Use services* 29
Table 4 *getTerminalTime()* parameters 36
Table 5 *getTerminalTime()* real-time capabilities 37
Table 6 *getTerminalTimeRateUncertainty()* parameters 38
Table 7 *getTerminalTimeRateUncertainty()* real-time capabilities 38
Table 8 *getCurrentTAI()* parameters 40
Table 9 *getCurrentTAI()* real-time capabilities 40
Table 10 *getCurrentUTC()* parameters 42
Table 11 *getCurrentUTC()* real-time capabilities 42
Table 12 *getLastUpdateTAI()* parameters 44
Table 13 *getLastUpdateTAI()* real-time capabilities 44
Table 14 *getLastUpdateUTC()* parameters 46
Table 15 *getLastUpdateUTC()* real-time capabilities 46
Table 16 *provideTAI()* parameters 48
Table 17 *provideTAI()* real-time capabilities 49
Table 18 *provideUTC()* parameters 51
Table 19 *provideUTC()* real-time capabilities 52
Table 20 *notifyStandardTimeReference()* parameters 54
Table 21 *notifyStandardTimeReference()* real-time capabilities 54
Table 22 *setSpecificTime()* parameters 56
Table 23 *setSpecificTime()* real-time capabilities 57
Table 24 *getSpecificTime()* parameters 58
Table 25 *getSpecificTime()* real-time capabilities 59
Table 26 *notifySpecificTimeSetting()* parameters 61
Table 27 *notifySpecificTimeSetting()* real-time capabilities 61
Table 28 Specification of general *exceptions* 62
Table 29 Specification of range *exceptions* 62
Table 30 *Time service* general capabilities 64
Table 31 Variables 64
Table 32 Variables access primitives 64
Table 33 Possible values for TimeUncertainty 66
Table 34 Acronyms list 69
Table 35 Specified definitions 71
Table 36 Specified notations 72
Table 37 Specified equations 72
Table 38 Specified requirements 73

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, Fraunhofer FKIE,
- Jean-Philippe Delahaye, DGA,
- Guillaume Delbarre, DGA,
- David Hagood, VIAVI Solutions,
- Olivier Kirsch, KEREVAL,
- Francois Levesque, NordiaSoft,
- Charles Linn, L3Harris,
- David Murotake, HKE,
- Eric Nicollet, Thales,
- Kevin Richardson, MITRE,
- Robert Sklut, JTNC.

Time Service Facility PIM

1 Introduction

This document is the *PIM specification* (Platform-Independent Model) of the WInnForum *time service facility*.

1.1. The Time service facility

1.1.1 Purpose

The *time service facility* is the WInnForum standard for *time services*, which enable *radio platforms* to provide *radio applications* with knowledge of time.

It supports *portability* of *radio applications* and *hospitality* of *radio platforms*, through a generic specification of the *time service capability*, with the associated API and attributes.

An overview of *time service facility* is provided by the following figure:

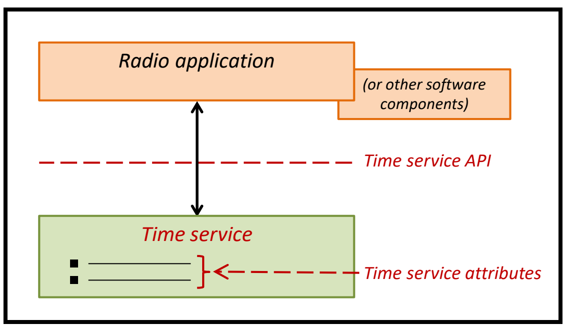

Figure 1 Overview of *time service facility*

Usage of the specified API is not limited to *radio applications*, e.g. components of the *radio platform* may use it as needed.

For the purpose of simplicity, "*radio application*" is used in the remainder of the document either designates a *radio application* or any other kind of client of a *time service*.

A *time service* can be distributed across several processing nodes within a *radio platform.*

1.1.2 Formal positioning

D01 A *time service capability* **is defined as** a *functional support capability* of a *radio platform* that provides *radio applications* with knowledge of time.

D02 A *time service* **is defined as** an instantiation of a *time service capability*.

D03 The *time service facility* **is defined as** the WInnForum *facility* specified for *time services*.

1.1.3 Composition

The *time service facility* is composed of the following specification documents:

- The *PIM specification* (Platform-Independent Model, this document),
- Several *PSM specifications* (Platform-Specific Models).

This general structure complies with "*Principles of WInnForum Facility Standards*" (see [Ref1]).

1.1.4 Origin

The *time service facility* results from a development effort conducted by the WInnForum during years 2018-2020.

The development joined international contributors experienced in usage of the JTNC "*JTRS Standard Timing Service API*" (see [Ref2]) and derived APIs such as the ESSOR Architecture *Timing Service API* specified by "*Radio Services API Description Document*" (see [Ref3]) or the SVFuA corresponding API.

1.2. The PIM specification

1.2.1 Purpose

The *PIM specification* is the entry specification of the *time service facility*.

Disambiguation note related to "*service*" usage:

- "*time service*", with the two terms systematically coupled, corresponds to one instance of a *time service capability*, as specified by **D02**,
- "*service*", "*use service*" or "*provide service*", without "*time*" before, refer to elementary *services* of a *time service*.

1.2.2 *Reference definitions*

The *PIM specification* applies the following reference definitions, specified in "*Principles for WInnForum Facility Standards*" (see [Ref1]):

Topic	Applied definitions
Base concepts	*radio application*, *radio platform, portability*, *hospitality*
Architecture concepts	*application component, processing node*, *façade*, *functional support capability*
WinnForum facility	*facility, PIM specification*, *PSM specification*
Services	*service*, *service implementation*, *service interface, provide service*, *use service, services group*
Primitives	*primitive*, *primitive implementation signature*, *parameter*, *type*, *direction*, *semantics*, *exception*
Attributes	*attribute*, *capability, property*, *variable*
Execution time	*call time*, *return time*, *worst-case execution time, worst-case external execution time*

Table 1 Definitions used from "Principles for WInnForum Facility Standards"

1.2.3 *Writing conventions*

The *PIM specification* applies the writing conventions specified in "*Principles for WInnForum Facility Standards*" (see [Ref1]).

All class diagrams, sequence diagrams and state charts are based on the Unified Modeling Language (UML), specified in "*OMG Unified Modeling Language (OMG UML)*" (see [Ref4]).

1.3. Conformance

1.3.1 *Conformance principles*

In order to be "conformant with the *time service facility*", a *time service* has to satisfy all the requirements specified in the *PIM specification*.

Waivers may be attached to a conformant *time service*.

The concept of "partial conformance" may not be applied.

1.3.2 *Definitions usage*

Any documentation and technical artifact related to a *time service* should apply the definitions specified in the *PIM specification*.

1.4. Implementations and usages

1.4.1 Multiple time services

Multiple *time services* are required in cases multiple uncorrelated time domains coexist within a *radio platform* (e.g. uncorrelated multi-channel systems).

The following figure illustrates the associated usage configurations:

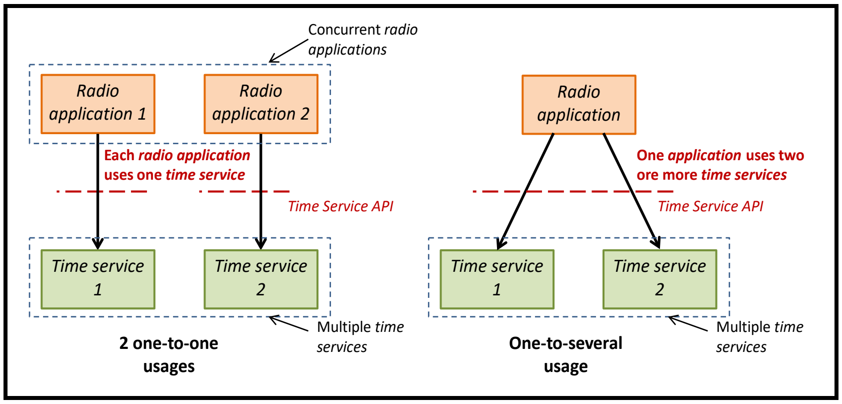

Figure 2 Usage configurations of multiple *time services*

One-to-several usage should be limited to cases where the *radio application* is meant to handle unsynchronized time domains.

Different *time services* are not expected to be synchronized, but this does not preclude implementations from doing so.

<u>Illustration: chassis with N processing blades</u>

> If each of the N blades has an independent local oscillator (i.e. not synchronized with the local oscillator of any other blades), the only possibility is to implement one *time service* per processing blade (total: N *time services*).
>
> If a time synchronization mechanism is implemented across the processing blades (e.g. a master clock is driving the oscillators of all the processing blades), two possibilities exist.
>
> First, one can keep implementing N different *time services*, even if they actually run at the same rate, they may not be synchronized in value, and the executing *radio applications* may not assume synchronization.
>
> Second, one can implement a common *time service* in which time is unified across all blades and the executing *radio applications* may assume such unicity.

1.4.2 Multiple radio applications

Multiple *radio applications* happen when multiple independent *radio applications* simultaneously execute within a *radio platform*.

The following figure illustrates the associated usage configurations:

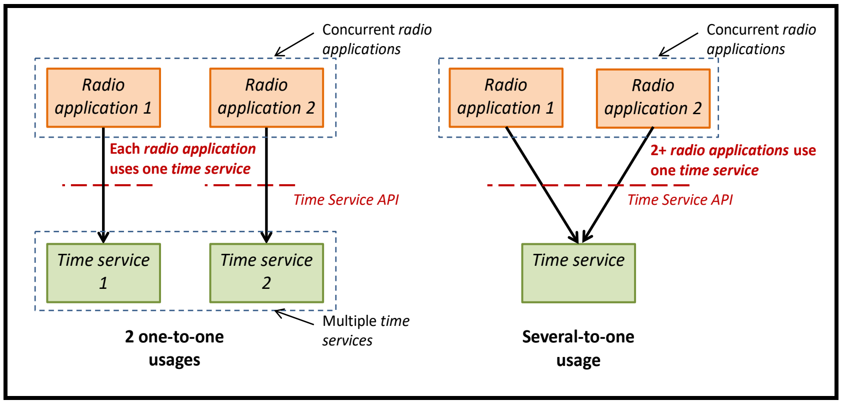

Figure 3 Usage configurations of multiple *radio applications*

The *time service facility* does not prevent several-to-one usage, which can be used to minimize complexity of the overall design in mutualizing access to a shared *time service*.

This said, safety or security considerations might prevent such approaches.

1.4.3 Services distribution

Services distribution happens where a same *service* of a given *time service* is simultaneously accessible on *façades* available on several *processing nodes*.

In such situations, the *time service* is responsible for the distribution and synchronization of time across *façades*.

This typically enables *application components* of a *radio application* to synchronize their real-time execution.

This is illustrated on the following figure:

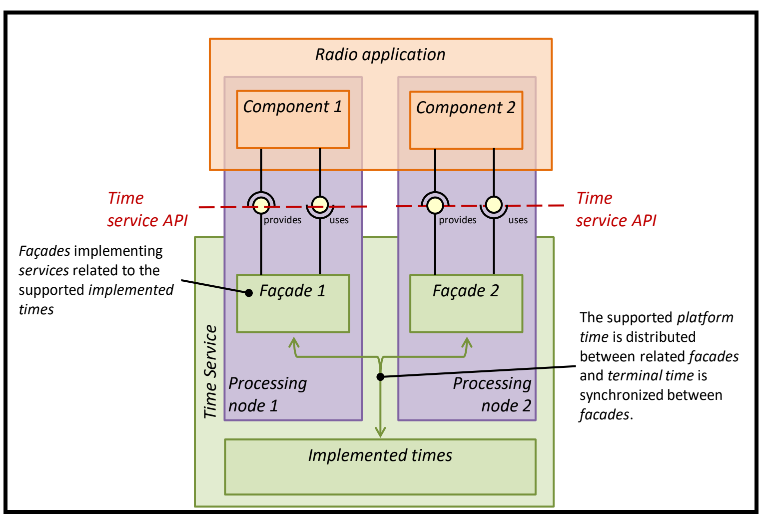

Figure 4 Notion of *services* distribution

1.5. Time concepts

1.5.1 Base notions

1.5.1.1 Physical time

D04 The *physical time* **is defined as** the time physically elapsing within the *radio platform*.

The *physical time* is the fundamental time from which all the other times are defined.

D05 A *physical instant* **is defined as** an infinitesimal moment of the *physical time*, whose passage is instantaneous.

N01 *t* **denotes** an undefined *physical instant* of the *physical time*.

N02 $t_{<qualifier>}$ **denotes** the *physical instant* characterized by the used <*qualifier*>.

The notations *t* and $t_{<qualifier>}$ are independent from how *physical instants* are measured.

1.5.1.2 Time function

The *time function* is the concept enabling the characterization of any time through its dependency to the *physical time*.

D06 A *time value* **is defined as** the value taken by a time at a certain *physical instant*.

D07 A *time function* **is defined as** the mathematical function that relates, for any time notion measuring time, the taken *time values* to the *physical time*.

1.5.2 Standard times

D08 A *standard time* **is defined as** the *International Atomic Time* (*TAI*) or the *Universal Coordinated Time* (*UTC*).

1.5.2.1 International Atomic Time (TAI)

D09 The *International Atomic Time* **is defined as** an ITU-standardized *physical time* measure built from an international network of permanently running reference atomic clocks.

The *International Atomic Time* is the most accurate available measure of the elapsing *physical time*.

See [Ref5] for the related Wikipedia article.

N03 *TAI* **denotes** the *International Atomic Time*.

N04 TAI(*t*) **denotes** the *time function* of the *TAI*.

TAI(t) is a strictly linear function of slope 1 :

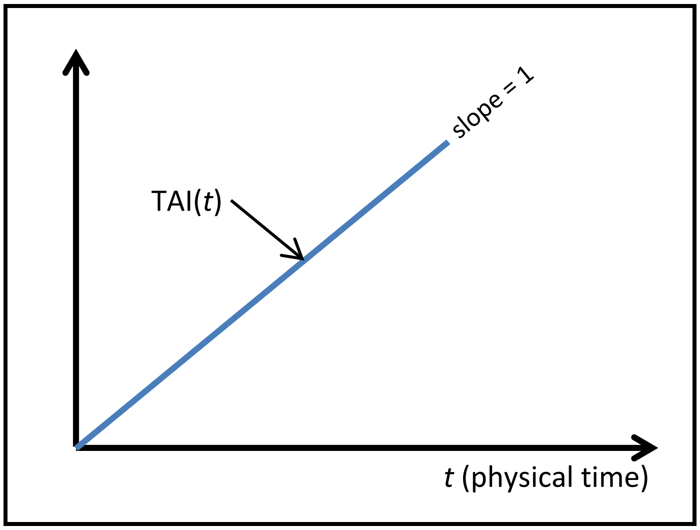

Figure 5 Time function of the *TAI*

1.5.2.2 Universal Coordinated Time (UTC)

D10 The *Universal Coordinated Time* **is defined as** the ITU-standardized measurement of time that adjusts the *TAI* using the concept of leap seconds (LS) to compensate the long term drifts in the apparent position of the sun.

See [Ref6] for the related Wikipedia article.

N05 *UTC* **denotes** the *Universal Coordinated Time*.

N06 UTC(t) **denotes** the *time function* of the *UTC*.

UTC(t) is a piecewise linear function of slope 1, separated by transitions of 1 second in the advent of leap seconds (at most once every 6 months):

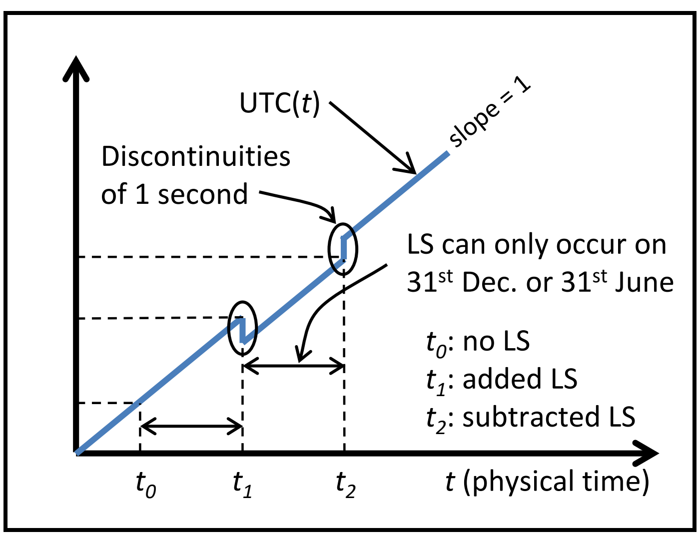

Figure 6 Time function of the *UTC* depicting leap seconds

1.5.2.3 Standard time sources

D11 A *standard time source* **is defined as** any source of *standard time* supporting implementation of a *time service*.

A *standard time source* can:

- Be internal to the *time service*,
- Come from a *radio application,* or from a *radio platform* client.

Typical examples of internal *standard time sources* are user inputs, GNSS or precise time devices.

1.5.2.4 Standard time sources identification

The *standard time sources* are identified using integer identifiers.

For the *standard time sources* that are part of the *time service* implementation, the identifiers are implementation-dependent constant values chosen between 1 and 128.

For the *standard time sources* that are *radio application* components (or a *radio platform* client) employing the *time service*, the identifiers are *radio application*-dependent values, which have to be greater than 128.

How *standard time sources* identifiers are provided to *radio applications*, if needed, is unspecified.

Some possibilities are setting of values at deployment-time (e.g. using SCA configuration properties), or porting time rebuild of *radio applications* after setting of the values in their source code.

1.5.3 Implemented times

D12 An *implemented time* **is defined as** a time implemented by a *time service*.

There are 3 sorts of *implemented times*:

- *Terminal time*: permanent internal concept of time (see section 1.5.3.1),
- *System time*: estimation of the TAI or the UTC (see section 1.5.3.1.8),
- *Specific times*: non-standard times controlled by *radio application* (see section 1.5.3.3).

1.5.3.1 Terminal time

1.5.3.1.1 Definition

D13 The *terminal time* **is defined as** the *implemented time* that measures the time elapsing within the *time service*.

Within the time domain of the *time service*, *terminal time* is synchronized across *processing nodes* and may be common between the *time service* and some other *functional support capability* of the *radio platform*.

1.5.3.1.2 Usage

A *radio application* typically uses the *terminal time*:

- To interpret *time stamps* attached to *time values*,
- To coordinate real-time execution of *application components* across *processing nodes*,
- As a time shared with other capabilities of the *radio platform*, e.g. WInnForum "*Transceiver Facility PIM Specification*" (see [Ref7]) or timers.

1.5.3.1.3 Implementation

The *time service* implements the *terminal time* as a strictly monotonic increasing function of *physical time* measuring the time having elapsed since an implementation-specific initial instant.

There is no roll-over possibility within the lifetime of the *time service* and that the *time service* never resets the *terminal time* to align it with any other *implemented time*.

A *terminal time* is typically implemented using a counter incremented by progress of a local oscillator of the *radio platform*.

The rate of the *terminal time* may be disciplined using a time reference in order to be closer to the rate of the *physical time*, while never introducing a time discontinuity or moving backwards in time.

In case of *radio platforms* with several *time services,* their *terminal times* are assumed to be independent from each other.

Illustration

> The *terminal time* can be available on a variety of *processing nodes*, e.g., from SCA-compliant GPPs to FPGAs.
>
> The *terminal time* is not required to correspond to any *standard time* (UTC or TAI).
>
> POSIX could be used to implement the *terminal time* on GPPs.
>
> The *terminal time* can be implemented with a standard epoch (e.g. POSIX 1 Jan 1970) or start at any value (e.g. "00:00").

1.5.3.1.4 Time functions

N07 TT(t) **denotes** the idealized *time function* of *terminal time*.

TT(t) is a linear continuous function.

N08 TT[t] **denotes** the digitized *time function* of *terminal time*.

TT[t] reflects the quantization effects on *terminal time* implementation, and is a discontinuous step function.

N09 t_{TTinit} **denotes** the initial instant from which *terminal time* is measured.

N10 v_{TTinit} **denotes** the initial value taken by *terminal time* at t_{TTinit}.

The following figure illustrates the *time functions* of *terminal time*:

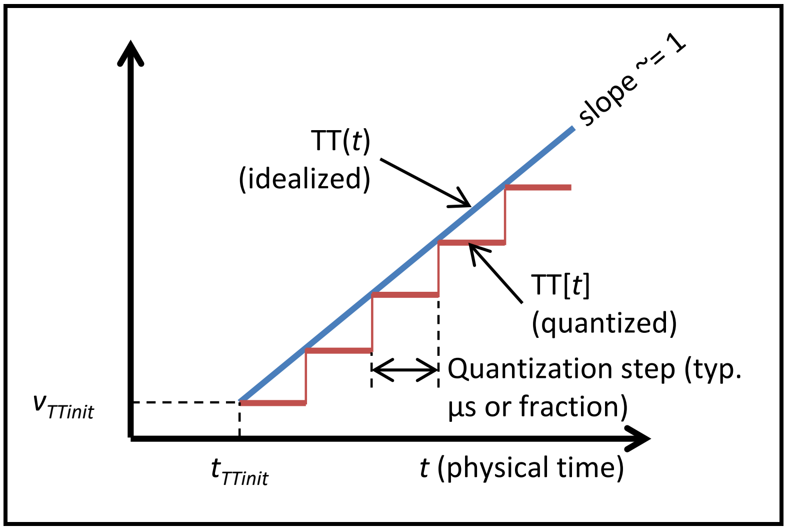

Figure 7 Time functions of *terminal time*

1.5.3.1.5 Terminal time rate error

D14 The *terminal time rate error* **is defined as** the relative rate error of the *terminal time* versus the *physical time*.

N11 *TTRE* **denotes** *terminal time rate error*.

TTRE can take negative or positive values.

A *terminal time* implementation with high accuracy has a low absolute value of *TTRE*.

The typical order of magnitude for the absolute value of *TTRE* is between 10^{-5} (poor accuracy) and 10^{-9} (good accuracy).

E01 The *terminal time rate error* equation **captures** its mathematical definition:

$$\boxed{\mathrm{TT}(t_0 + \Delta t) = \mathrm{TT}(t_0) + \left(1 + TTRE_{t_0}\right) \cdot \Delta t} \qquad \text{Eq. 1.}$$

In Eq. 1:

- $\mathrm{TT}(t_0)$ and $\mathrm{TT}(t_0 + \Delta t)$ denote idealized *terminal time* values,
- $TTRE_{t_0}$ denotes *TTRE* value at t_0,
- Δt denotes a small increment of *physical time*.

Eq. 1 uses the idealized *terminal time* representation TT(*t*), neglecting the quantization effects appearing in actual *terminal time* implementations TT[*t*].

The following figure illustrates the previous notions:

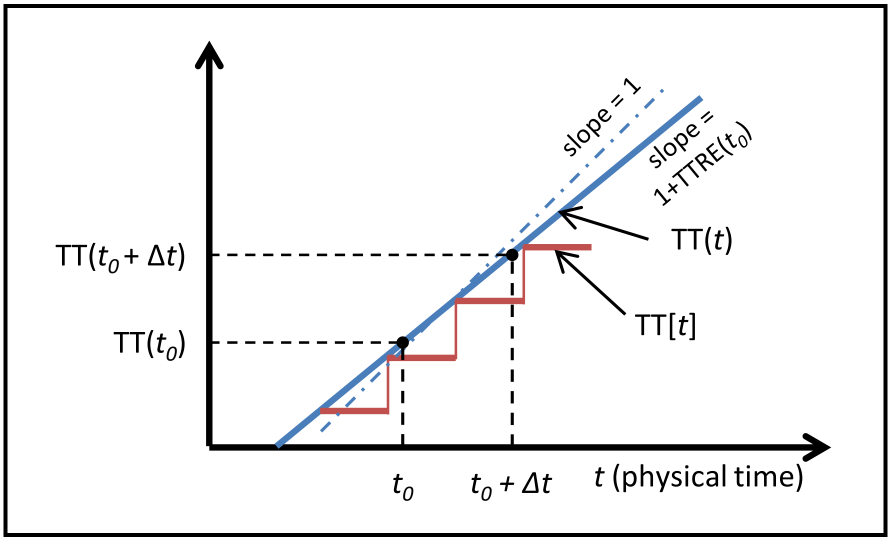

Figure 8 Notion of *terminal time rate error* (*TTRE*)

TTRE may slowly vary over time, e.g., due to influence of temperature and aging of the oscillator.

1.5.3.1.6 Terminal time rate uncertainty

D15 The *terminal time rate uncertainty* **is defined as** an upper bound of the absolute value of its *terminal time rate error*.

N12 *TTRU* **denotes** *terminal time rate uncertainty*.

E02 The *terminal time rate uncertainty* equation **captures** its mathematical definition:

$$\boxed{|TTRE| < TTRU} \qquad \text{Eq. 2.}$$

TTRU is typically equal to the accuracy of the oscillator used to implement *terminal time*.

1.5.3.1.7 terminalTimeRateMaxUncertainty capability

The **`terminalTimeRateMaxUncertainty`** *capability* **is specified as** a scalar that reflects the maximum *terminal time rate uncertainty* of a *time service*, in parts-per-billion (ppb).

1.5.3.1.8 terminalTimeRateUncertainty variable

The **`terminalTimeRateUncertainty`** *variable* **is specified as** a scalar that reflects the current *terminal time rate uncertainty* of a *time service*, in parts-per-billion (ppb).

The *getTerminalTimeRateUncertainty()* *primitive* (see section 3.1.2.2) enables a *radio application* to access to **`terminalTimeRateUncertainty`**.

1.5.3.2 System time

1.5.3.2.1 Definition

D16 The *system time* **is defined as** the *implemented time* that jointly estimates the *TAI* and the *UTC*.

The *system time*, being an estimate, may lead or lag the *standard times*, with an uncertainty influenced by items such as the figure of merit of values from *standard time sources* used by the implementation, the quantization error and the *terminal time rate error*.

1.5.3.2.2 Usage

A *radio application* typically uses the *system time* to coordinate with other systems or other *radio applications*.

1.5.3.2.3 Implementation

A *system time* is typically derived from one or more sources (e.g. a GNSS device, a chronometer device, or an operator input).

Implementation of *system time* typically has knowledge of leap seconds.

1.5.3.2.4 Time representation

The *time service* represents, for *TAI*, *system time* as a measure of the *physical time* having elapsed since epoch 00:00:00, 1 January 2000 (GMT).

The *time service* represents, for *UTC*, *system time* as a measure of the *physical time* having elapsed since epoch 00:00:00, 1 January 2000 (GMT), minus leap seconds.

The previous statements make the *UTC* representation correspond to the concept of Unix time (see [Ref8]), with an epoch changed from 1 Jan. 1970 to 1 Jan. 2000.

1.5.3.2.5 Time functions

N13 ST(t) **denotes** the idealized *time function* of a *system time*.

ST(t) is a piecewise continuous function.

N14 ST[t] **denotes** the digitized *time function* of a *system time*.

ST[t] is a piecewise step function.

The following figure illustrates the *time services* of *system time*:

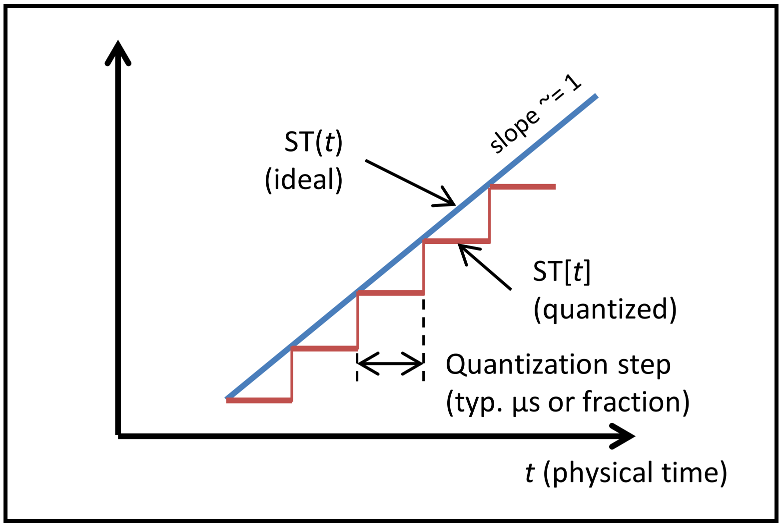

Figure 9 *Time functions* of *system time*

1.5.3.2.6 Leap seconds handling

When a leap second occurs, the *UTC* is delayed or advanced by one second at midnight of the application day (30-June or 31-Dec).

The *system time* behavior in the advent of a leap second is *unspecified*.

Illustration

For added leap second, the seconds count

er of the *system time* can stay on the same second for one additional second, on the second before midnight or on the second after midnight,

For subtracted leap second, the seconds counter of the *system time* can skip one second, on the second before midnight or on the second after midnight.

1.5.3.2.7 System time updates

D17 A *system time update* **is defined as** a point in time when the *time service* updates the *system time* in order to decrease its *estimation uncertainty*.

The *time service* makes a *system time update* when a *standard time reference* delivered by a *standard time source* results in a decrease of the *estimation uncertainty* of the *system time*.

See section 1.6 for definitions of *estimation uncertainty* and *standard time reference*.

A *system time update* can be caused by arrival of any *standard time reference*, from any *standard time source*.

A *system time update* generally causes a discontinuity in *system time*. Such discontinuity can move the *system time* backwards resulting in a non-monotonic behavior.

The updated time is then used as the reference from which the *time service* determines the *system time* based on *terminal time* rate until the next *system time update* occurs.

Depending on the nature of the *time service* and the course of events, the occurrence of *system time updates* can be periodic (with possible interruptions), or sporadic.

The following figure illustrates the principle of a *system time update*:

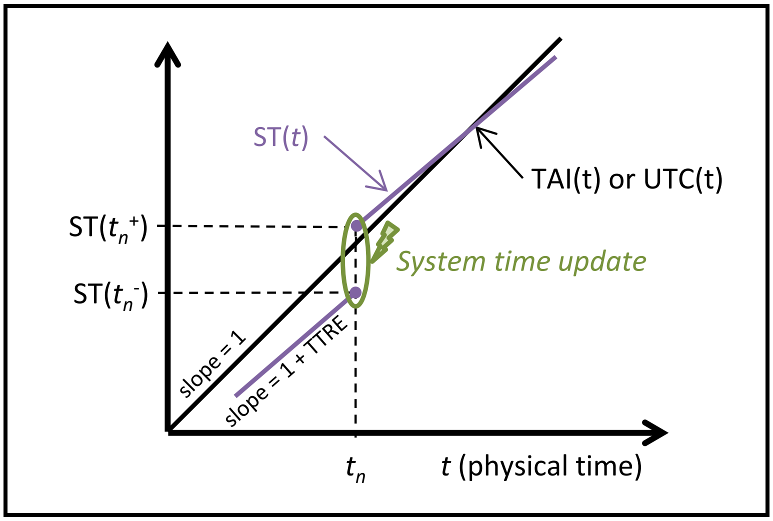

Figure 10 Principle of a *system time update*

1.5.3.2.8 GNSS usage example

A GNSS (e.g., GPS, Galileo) device can be used as a *standard time source,* typically delivering every second a *standard time reference* (see section 1.6.4.2) with the current second *standard time* value and the associated figure of merit.

The following figure depicts the typical evolution of *system time* in case of a GNSS interruption:

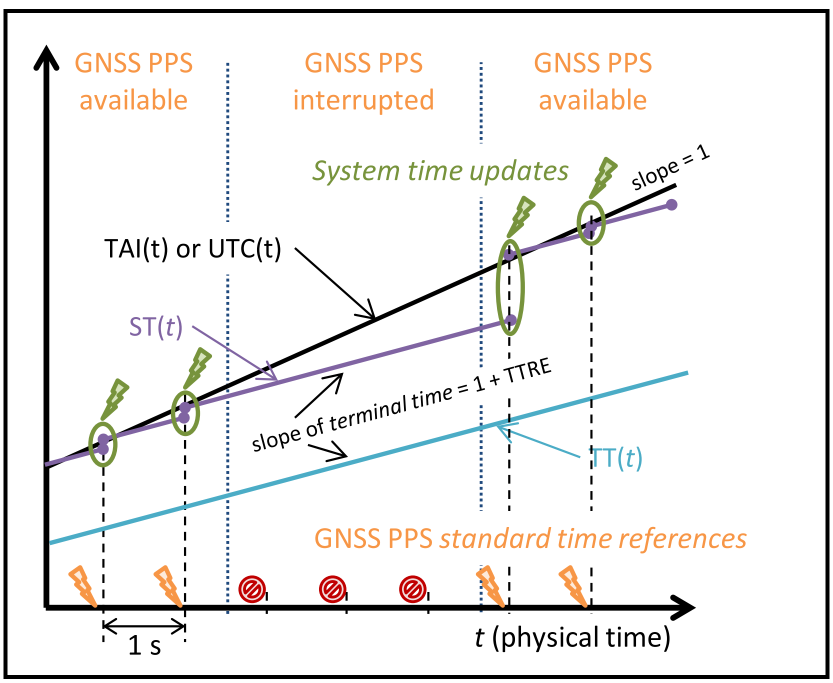

Figure 11 Typical *system time* in case of GNSS interruption

The previous figure illustrates that the instantaneous slope of *system time* is the slope of *terminal time*, in which inaccuracy is compensated by *system time updates*.

1.5.3.3 Specific times

1.5.3.3.1 Definitions

D18 A *specific time* **is defined as** a monotonically increasing *implemented time* maintained as closely as possible to the *physical time* rate since it was last set.

D19 A *setting time* **is defined as** the *physical time* value at which a *specific time* was last set.

D20 A *setting value* **is defined as** the value to which a *specific time* was set to at a *setting time*.

A *specific time* becomes defined once any *radio application* sets it for the first time in the lifetime of the *time service*.

A *radio application* can set a *specific time* as often as needed, possibly creating discontinuities in the *specific time*.

By contrast to *system time*, occurrences of *standard time references* will not generate *specific time* updates, but may be used to improve the accuracy of *specific time* maintenance.

1.5.3.3.2 Usage

A *radio application* typically uses a *specific time* for the following purposes:

- To use a non-standard time of its own,
- To recover a previously set *specific time* after a period of de-instantiation.

<u>Illustration: faster resynchronization</u>

> In anticipation of an inactive period, a *radio application* can set a *specific time* with a value, let the *time service* maintain the *specific time*, and retrieve the updated value once the *radio application* is active again, in order to reduce duration of its over-the-air resynchronization procedure with the other radios of the radio network.

Combinations of different *radio applications* may also use a same *specific time* for cross-applications purposes, while such usages need to be considered with care.

<u>Illustration: master/slave time</u>

> A master *radio application* can set a *specific time*, for a slave *radio application* to align with this *specific time*.

1.5.3.3.3 Implementation

A *time service* represents a *specific time* adding to its *setting value* the number of seconds and nanoseconds having elapsed since its *setting time*.

A *specific time* is typically computed and monotonically maintained using a combination of time keeping mechanisms available to the *time service* (e.g., a local oscillator, a GNSS device, a chronometer device).

1.5.3.3.4 Time functions

N15 SpeT(t) **denotes** the idealized *time function* of a *specific time*.

SpeT(*t*) is a piecewise continuous function.

Each continuous portion of SpeT(*t*) reflects the monotonic maintenance of the *specific time* since last *setting time*.

N16 SpeT[*t*] **denotes** the digitized *time function* of a *specific time*.

SpeT[*t*] reflects the quantization effects on *specific time* implementation, and is a piecewise step function.

N17 $t_{setting}$ **denotes** a *setting time*.

N18 $v_{setting}$ **denotes** a *setting value*.

The following figure illustrates the *time functions* of a *specific time*:

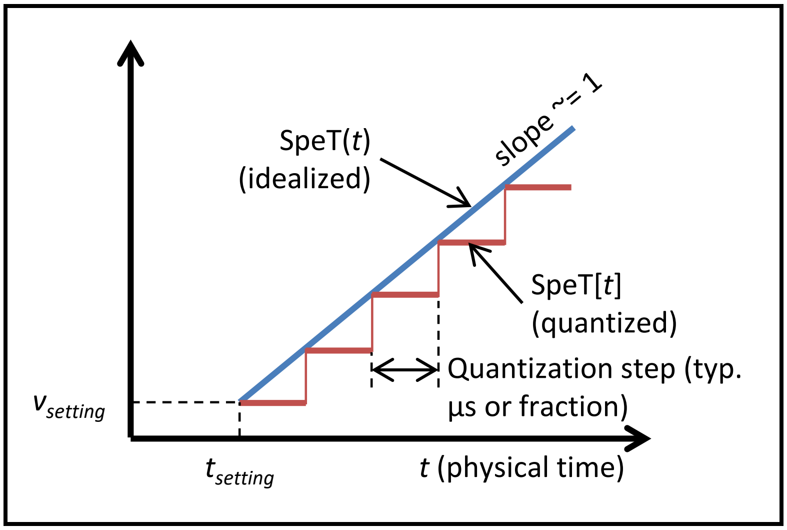

Figure 12 *Time functions* of a *specific time*

The following figure illustrates the possible discontinuities in the *time function* of a *specific time*:

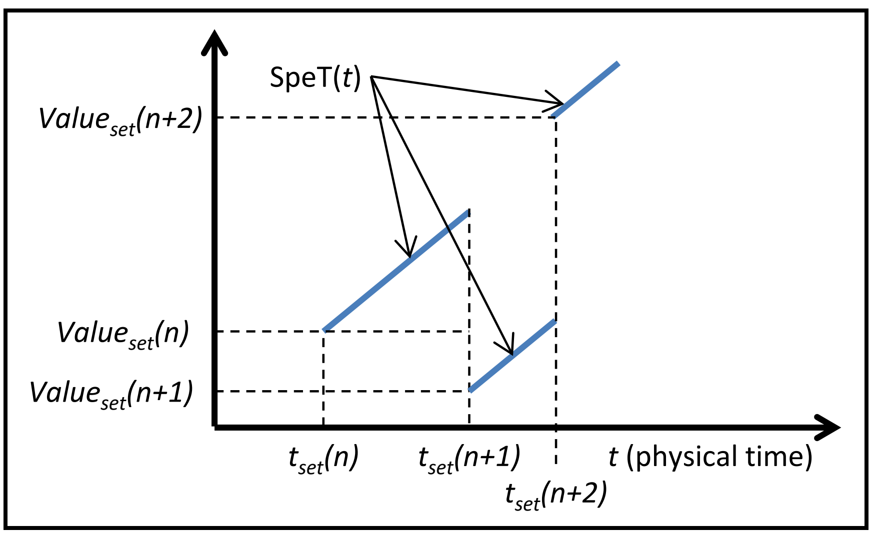

Figure 13 Possible *specific time* discontinuities

1.5.3.3.5 maxSpecificTimes capability

The `maxSpecificTimes` *capability* **is specified as** an integer that captures the number of *specific times* implemented by a *time service*.

1.5.3.3.6 Specific times identification

Integer constants ranging from 1 to `maxSpecificTimes` identify *specific times*.

Those identifiers are global to the *time service*, and need to be consistently assigned between *radio applications* for correct overall behavior.

How *specific times* identifiers are assigned to *radio applications* is unspecified.

Some possibilities are setting of values at deployment-time (e.g. using SCA configuration properties), or porting time rebuild of *radio applications* after setting of the values in their source code.

1.6. Time handling

1.6.1 Time stamps

1.6.1.1 Definition

D21 A *time stamp* **is defined as**, for a stamped instant, a *terminal time* value measured at a *physical time* close to when the stamped instant occurs.

N19 *TS* **denotes** a *time stamp*.

Time stamps are used to indicate a correspondence between a time of interest and *terminal time*.

The times of interest can be:

- An instant associated to *implemented times*, e.g. a *system time* current value, a last *system time update* or the last *setting time* of a *specific time*,
- A *standard time reference* (see section 1.6.4.2) delivered by a *standard time source*.

This design paradigm enables *radio applications* and *time service* to interact with no hardware interface needed.

Like all measurements of real-world values, the *time stamp* value makes a stamping error influenced by implementation-specific factors such as quantization error, timing rate error, etc.

1.6.1.2 Stamping uncertainty

D22 A *stamping uncertainty* **is defined as** a maximum possible error between a stamped instant and the associated *time stamp*.

N20 *SU* **denotes** the *stamping uncertainty*.

N21 $t_{stamped}$ **denotes** the *physical time* value of the stamped instant.

E03 The *stamping uncertainty* equation **captures** its mathematical definition:

$$\boxed{TS \in \mathrm{TT}(t_{stamped}) \pm SU} \quad \text{Eq. 3.}$$

In an ideal implementation, *stamping uncertainty* would be equal to zero, with *TS* = TT($t_{stamped}$).

The following figure illustrates the notion of *stamping uncertainty*:

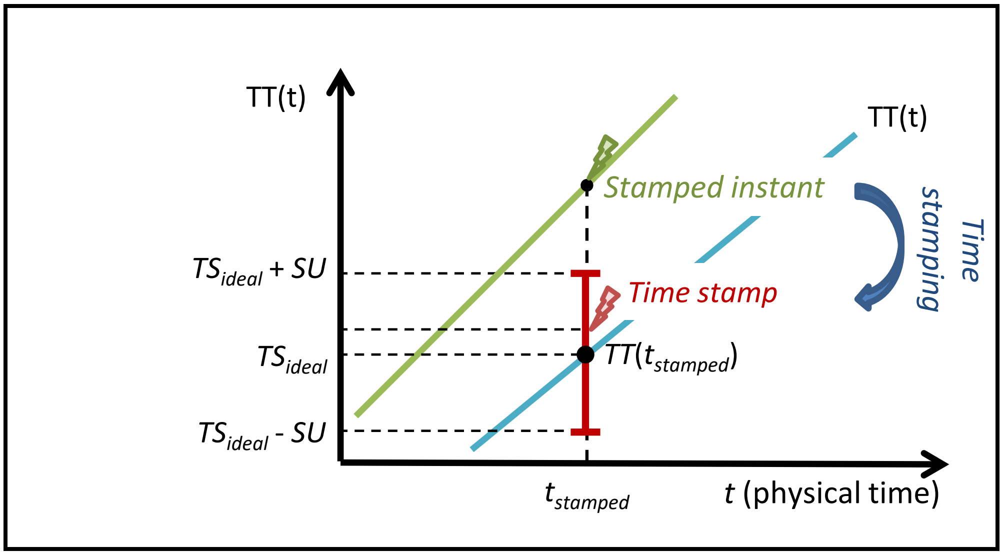

Figure 14 Notion of *stamping uncertainty*

1.6.1.3 stampingUncertainty capability

A `stampingUncertainty` *capability* **is specified as** a scalar reflecting the *stamping uncertainty* achieved *services* of a *time service*, expressed in ns.

Depending on implementation choices, a unique `stampingUncertainty` can be assigned to the entire *time service,* or can be specialized on a finer per-*primitive* and/or per-*façade* basis.

Illustration:

> For a GPP *façade* where the *time stamp* would be measured using a timer of the GPP, clocked by an oscillator decoupled from the *terminal time* with regular resynchronization with the actual *terminal time* deported on an FPGA, the `stampingUncertainty` value will encompass the uncertainties introduced between two resynchronizations.

1.6.2 Estimation uncertainty

1.6.2.1 Definition

D23 An *estimation uncertainty* **is defined as** an upper bound of the absolute value of the difference between a *time value* and a time it estimates, valid within a specified confidence percentage.

An *estimation uncertainty* characterizes the quality of a *system time* or *specific time* value, enabling decisions of various nature to be taken (choice of the most relevant time source, adjustment of a time, …).

The concept of *estimation uncertainty* applies to:

- *System time*, where the related time is a *standard time*,
- *Specific time*, where the related time is the time maintained from last *setting time* according to *physical time* rate.

The concept of *estimation uncertainty* does not apply to *terminal time* since it is not meant to implement any particular time.

1.6.2.2 System time uncertainty

D24 The *system time uncertainty* **is defined as** an *estimation uncertainty,* at 95% confidence, between a *time value* of a *system time* and the *standard time* it estimates (*TAI* and/or *UTC*).

N22 *STU* **denotes** a *system time uncertainty*.

E04 The *system time uncertainty* equation **captures** its mathematical definition:

$$\boxed{|ST(t) - StdT(t)| \leq STU, \text{ in 95\% of cases}} \quad \text{Eq. 4,}$$

where:

- $ST(t)$ denotes the *system time* value (estimated *standard time*) at *t*,
- $StdT(t)$ denotes the value of the estimated *standard time* at *t*.

Illustration

> For a *time service* not assisted by whichever external mechanism (such a GNSS or a precise time source), the *system time uncertainty* typically increases each second by the accuracy of the local oscillator used to maintain the *terminal time*.
>
> For a *time service* using a GNSS receiver, *system time uncertainty* depends on the number of received satellites.
>
> A 95% confidence interval corresponds to a standard deviation of $2.\sigma$ in case the *standard time* estimation follows a Gaussian random distribution.

Illustration:

> For a typical *time service* using a GNSS receiver as a *standard time source*, when all satellites are in sight, *STU* < *SU*, and *SU* is be the dominant factor in *system time uncertainty*.
>
> Upon loss of GNSS signals, assuming the *time service* uses *terminal time* to maintain *system time*, *STU* increases by *terminal time rate uncertainty* every elapsing second. Eventually *STU* > *SU*, and *STU* becomes the dominant factor in *system time uncertainty*.

1.6.2.3 Specific time uncertainty

D25 A *specific time uncertainty* **is defined as** an *estimation uncertainty*, at 95% confidence, between a *time value* of a *specific time* and the time that would have been maintained from the last *setting time* using *physical time* rate.

N23 *SpeTU* **denotes** a *specific time uncertainty*.

E05 The *specific time uncertainty* equation **captures** its mathematical definition:

$$\left|SpeT(t) - \left(SpeT\left(t_{setting}\right) + \left(t - t_{setting}\right)\right)\right| \leq SpeTU, \text{ at 95\% confidence} \quad \text{Eq. 5,}$$

Where:

- $SpeT(t)$ is the value at *t* of the *specific time*,
- $\left(SpeT(t) + \left(t - t_{setting}\right)\right)$ is the value at *t* of a time ideally maintained from $t_{setting}$.

Illustration:

For a *time service* directly using its *terminal time* to maintain *specific time*: if *TTRU* is known by the *time service* as a constant, EU_{SpeT} increase rate is equal to *terminal time rate uncertainty* ($SpeTU(t) = SpeTU(t_{set}) + (t - t_{set}).TTRU$); if, more accurately, *TTRU* is known by the *time service* as a function of time, one has $SpeTU(t) = SpeTU(t_{set}) + \int_{t_{set}}^{t} TTRU\ (t).\mathrm{dt}$.

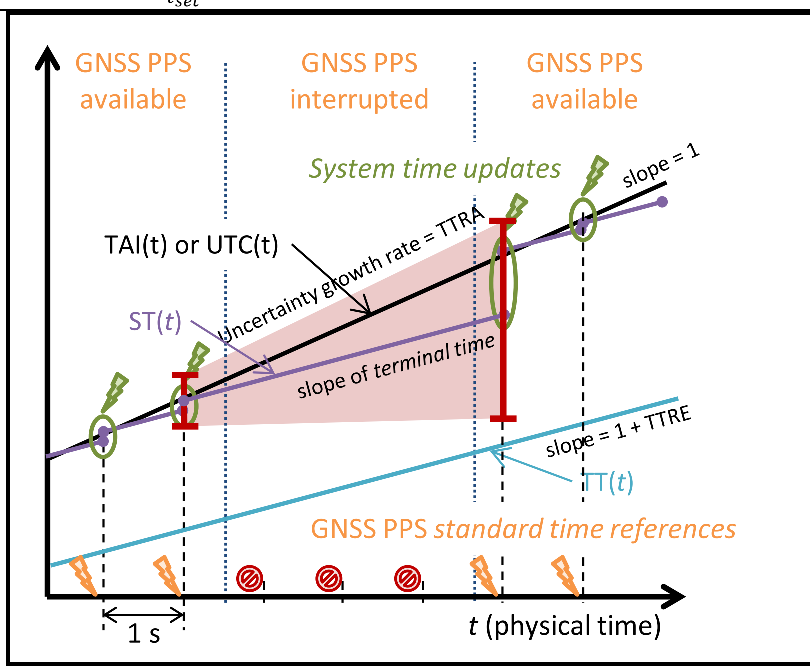

Figure 15 Typical *system time uncertainty* evolution in case of GNSS interruption

1.6.3 *Time uncertainty*

D26 A *time uncertainty* **is defined as** the addition of the *estimation uncertainty* of a *time value* of interest and the *stamping uncertainty* of the associated *time stamp*.

1.6.4 *Time references*

D27 A *time reference* **is defined as** a triplet composed of a *time value* of interest, the associated *time stamp* and their *time uncertainty*.

Time references are the essential unified structure of information exchanged by the *primitives* of the *time service* API.

1.6.4.1 System time references

D28 A *system time reference* **is defined as** a *time reference* which *time value* of interest relates to *system time*.

When a *radio application* requests value of *system time*, using *getCurrentTAI()*, *getCurrentUTC()*, *getLastUpdateTAI()* or *getLastUpdateUTC()* (see section 3.1.3), the *time service* returns a *system time reference*.

1.6.4.2 Standard time references

D29 A *standard time reference* **is defined as** a *time reference* which *time value* of interest relates to a *standard time*.

When a *standard time reference* is delivered to a *time service* by a *standard time source*:

- The *time service* evaluates it for eventual update of the *system time* (see section 1.5.3.2.7),
- The *time service* may notify *radio application* of the *standard time reference*, using *notifyStandardTimeReference()* (see section 3.1.5).

When a *radio application* provides an estimate of a *standard time*, using *provideTAI()* or *provideUTC()* (see section 3.1.4), it provides the corresponding *standard time reference* to the *time service*.

1.6.4.3 Specific time references

D30 A *specific time reference* **is defined as** a *time reference* which *time value* of interest relates to a *specific time*.

When a *radio application* sets a *specific time*, using *setSpecificTime()* (see section 3.1.6.1), it provides the corresponding *specific time reference* to the *time service*.

Following a call to *setSpecificTime()*, the *time service* can notify a *radio application* using *notifySpecificTimeSetting()* (see section 3.1.7), sending the corresponding *specific time reference*.

When a *radio application* gets the current value of a *specific time*, using *getSpecificTime()* (see section 3.1.6.2), the *time service* returns the corresponding *specific time reference*.

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 *time service*):

Services groups (same as Modules)	Services (same as Interfaces)	Primitives	Optionality
`TerminalTime`	`TerminalTimeAccess`	*getTerminalTime()* *getTerminalTimeRateUncertainty()*	**M**
`SystemTime`	`SystemTimeAccess`	*getCurrentTAI()* *getCurrentUTC()* *getLastUpdateTAI()* *getLastUpdateUTC()*	**M**
	`StandardTimeProvision`	*provideTAI()* *provideUTC()*	**O**
`SpecificTimes`	`SpecificTimeHandling`	*setSpecificTime()* *getSpecificTime()*	**O**

Table 2 ***Provide services***

The column "Optionality" specifies if a *provide service* is **mandatory (M)** or **optional (O)**.

2.2. Use services

The following table lists the *use service* of the API (provided by a *radio application* and used by a *time service*):

Services groups (same as Modules)	Services (same as Interfaces)	Primitives	Optionality
`StandardTimes`	`ReferencesNotification`	*notifyStandardTimeReference()*	**O**
`SpecificTimes`	`SettingsNotification`	*notifySpecificTimeSetting()*	**O**

Table 3 ***Use services***

The column "Optionality" specifies if a *use service* is **mandatory (M)** or **optional (O)**.

2.3. Service-level conformance

2.3.1 Services scope

A *service implementation* is an implementation of a particular *service* on a particular *façade* (see [Ref1]).

R01 A *time service* **shall** present a *service implementation* for each **mandatory** *service*.

R02 A *time service* **shall** present a *service implementation* for each selected **optional** *service*.

2.3.2 selectedOptionalServices capability

The selectedOptionalServices *capability* **is specified as** the set of boolean values indicating which **optional** *services* specified by the *time service facility* are implemented.

2.3.3 Services implementation conformance

A *service implementation* **needs to** comply with the applicable normative content of the *PIM specification* and of the applicable *PSM specification*.

A *service implementation* **needs to** present a *primitive implementation* for each *primitive* specified in its *service interface*.

2.4. States machines

All specified transitions are instantaneous.

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

2.4.1 TimeService

TimeService **is specified as** the main state machine followed by *time service*.

An instance of TimeService is followed by each *time service* instance.

The following figure is the statechart of TimeService state machine:

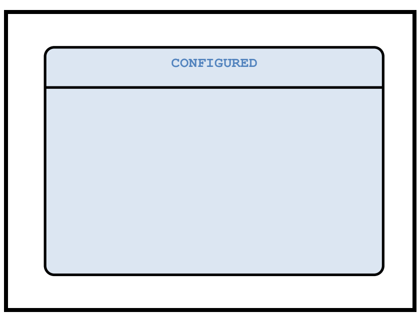

Figure 16 TimeService statechart

2.4.1.1 States

1.1.1.1.1 CONFIGURED

CONFIGURED **is specified as** the unique state of TimeService, during which a *time service* is configured according to the needs of all the *radio applications* to be supported during the CONFIGURED state.

CONFIGURED is reached by a *time service* when it:

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

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

2.5. Services groups description

2.5.1 *TimeService::TerminalTime*

The TerminalTime *services group* enables *radio applications* to access to *terminal time*, and contains the TerminalTimeAccess *service*:

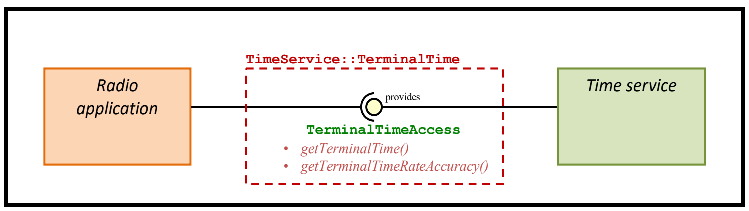

Figure 17 TerminalTime *services group*

TerminalTimeAccess is a *provide service* enabling to get the *terminal time* current value.

2.5.2 *TimeService::SystemTime*

The SystemTime *services group* enables *radio applications* to use *system time*, and contains SystemTimeAccess and StandardTimeProvision *services*:

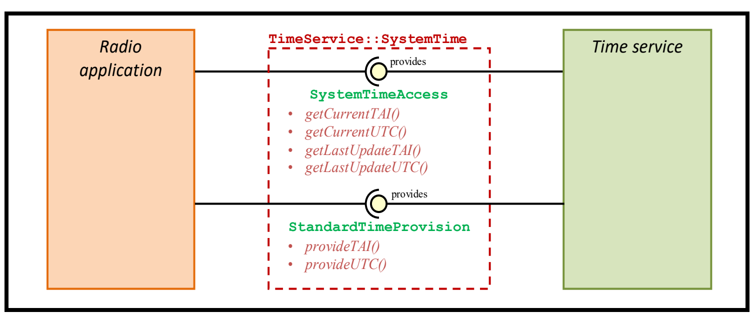

Figure 18 SystemTime *services group*

SystemTimeAccess is a *provide service* enabling to get current or last update value of a *standard time* (*TAI* or *UTC*). It offloads the *radio application* from duties related to estimation of *standard times*.

`StandardTimeProvision` is a *provide service* enabling to provide a *standard time reference* to the *time service* in order to support *system time* implementation. It allows a *radio application* to act as a *standard time source*.

The principle of `SystemTime` and `StandardTimes` *services groups* (see section 2.5.3) is summarized by the following figure:

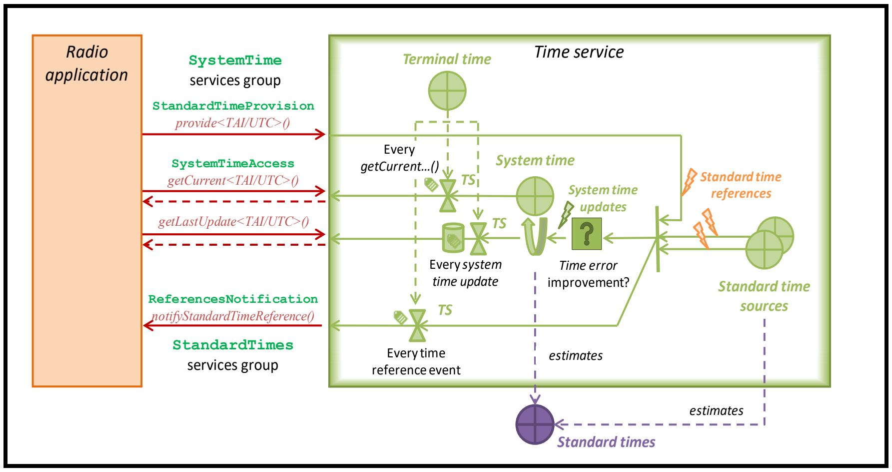

Figure 19 Principle of SystemTime and StandardTimes *services groups*

2.5.3 TimeService::StandardTimes

The `StandardTimes` *services group* enables *radio applications* to be notified of all *standard time references* generated by *standard time sources*, and contains the `ReferencesNotification` *service*:

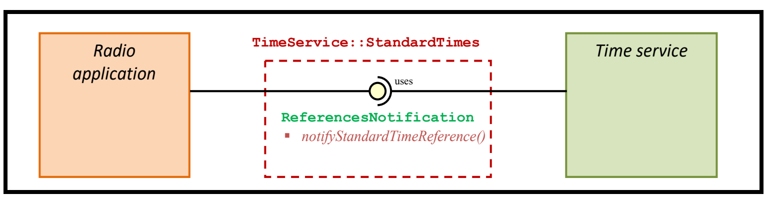

Figure 20 StandardTimes *services group*

`ReferencesNotification` is a *use service* notifying *radio applications* of occurrence of a *standard time reference*. It allows a *radio application* to directly use *standard time sources*.

2.5.4 TimeService::SpecificTimes

The **SpecificTimes** *services group* enables *radio applications* to use *specific times*, and contains **SpecificTimeHandling** and **SettingsNotification** *services*:

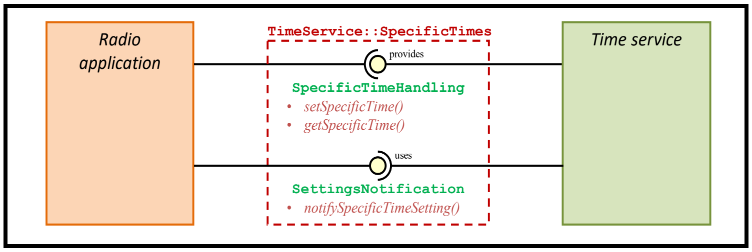

Figure 21 SpecificTimes *services group*

SpecificTimeHandling is a *provide service* enabling to set and get current value of a *specific time.*

SettingsNotification is a *use service* enabling to notify occurrence of a *specific time* setting by a *radio application*.

The principle of **SpecificTimes** *services group* is summarized by the following figure:

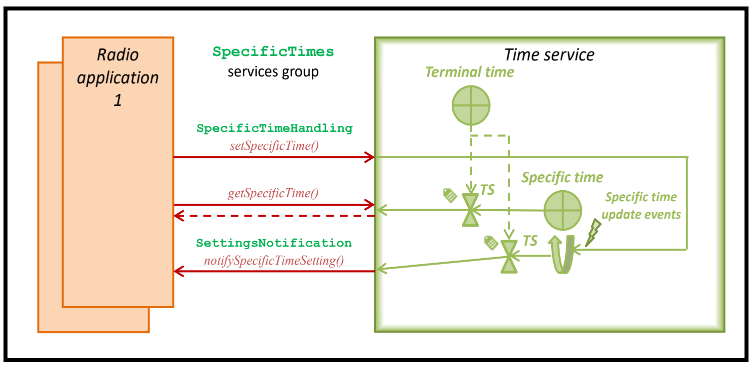

Figure 22 Principle of SpecificTimes *services group*

3 Service primitives and attributes

3.1. Service primitives

This section specifies the *primitives* of the *service interfaces* of the *time service* API.

3.1.1 Specification approach

3.1.1.1 Primitive conformance

A *primitive implementation* **needs to** comply with the applicable normative content of the *PIM specification* and of the applicable *PSM specification*.

A *primitive implementation* **needs to** comply with:

- Its *signature*, specified in section "Signature",
- The *semantics* of *parameters*, specified in section "Parameters",
- The *exceptions* listed in section "Exceptions",
  - The *attributes* specified in section "Attributes".

3.1.1.2 Overview

The section "Overview" provides an informative overview of the *primitive*'s purpose, composed of a short description and a sequence diagram.

For each *primitive*, t_{call} and t_{return} denote the *call time* and *return time* of a *primitive* invocation (see [Ref1]).

3.1.1.3 Signatures

The section "Signature" specifies the *signature* of a *primitive* using the OMG Interface Definition Language (IDL).

A *signature* specifies the name of the *primitive*, its ordered set of *parameters*, with each *parameter*'s name, *direction* and applicable *type*.

The possible values for *direction* are `in` or `out`.

The applicable *types* are specified in Section *type*, which specifies all the *types* of the *time service* API.

The *signatures* specified for *primitives* comply with the Ultra Lightweight (ULw) PIM IDL Profile of the "*IDL Profiles for Platform-Independent Modeling of SDR Applications*" (see [Ref9]).

The conformance criteria for Application-Specific Interfaces specified by [Ref9], section 1.3.2 applies to the specified *service interfaces*: "*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 specified *signatures* also comply with the Ultra Lightweight (ULw) PIM IDL Profile of the SCA 4.1 Appendix E-1 "*Application Interface Definition Language Platform Independent Model Profiles*" (see [Ref10]).

The specified *signatures* are common inputs to all *PSM specifications*.

3.1.1.4 Parameters

The section "Parameters" completes specification of *parameters* in specifying, in the column "Content" of the *parameters* table, the *semantics* of each *parameter*.

3.1.1.5 Exceptions

The section "Exceptions" specifies the list of *exceptions* applicable for the *primitive*.

The listed *exceptions* are specified in Section 3.2, which specifies all the *exceptions* of the *time service* API.

3.1.1.6 Attributes

The section "Attributes" specifies the *attributes* applicable to the *primitive*.

Most *attributes* are real-time *capabilities*.

Specification of values for real-time *capabilities* is **optional**.

Real-time *capabilities* can have *façade*-specific values.

A **WCET** real-time *capability* reflects the *worst-case execution time* (see [Ref1]) of a *primitive implementation* of a *provide service*.

A **WCEET** real-time *capability* reflects the *worst-case execution time* (see [Ref1]) of a *primitive implementation* of a *use service*.

As stated in [Ref1], **WCEET** are difficult to verify and are more likely to be left unspecified.

3.1.2 TimeService::TerminalTime::TerminalTimeAccess

The *service interface* of the **TerminalTimeAccess** *service* is:

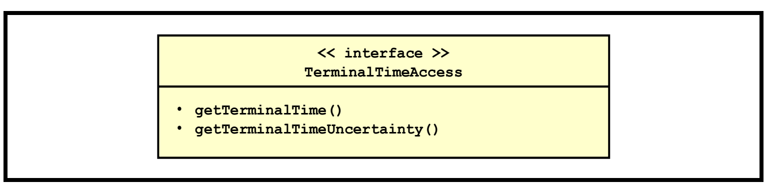

Figure 23 TerminalTime::TerminalTimeAccess *service interface*

3.1.2.1 getTerminalTime()

3.1.2.1.1 *Overview*

getTerminalTime() returns a *terminal time* value representative of when the call returns:

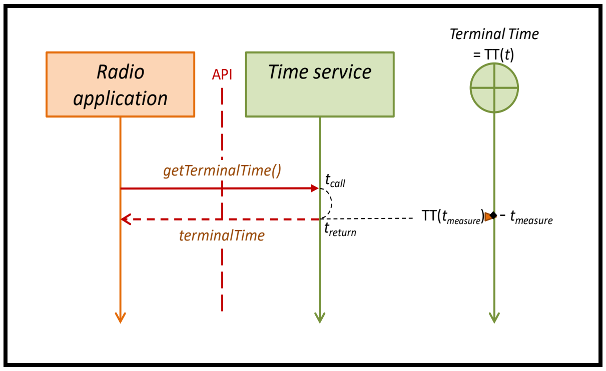

Figure 24 *getTerminalTime()* overview

$t_{measure}$ **denotes** the *physical instant* close to t_{return} such that *terminalTime* = TT($t_{measure}$).

3.1.2.1.2 *Signature*

The *signature* of *getTerminalTime()* **is specified as**:

```
void getTerminalTime(
    out TimeValue terminalTime
);
```

3.1.2.1.3 *Parameters*

The *parameters* of *getTerminalTime()* **are specified as**:

Name	Type	Direction	Semantics
terminalTime	`TimeValue` See 3.4.1	`out`	*Terminal time* value representative of when *getTerminalTime()* returns. Referenced to the initial value of *terminal time*. Never equal to `UndefinedTime`.

Table 4 *getTerminalTime()* parameters

3.1.2.1.4 *Exceptions*

None.

3.1.2.1.5 *Attributes*

The real-time *capabilities* attached to *getTerminalTime()* **are specified as**:

Name	Unit	Description
`timeliness`	ns	$\lvert t_{measure} - t_{return} \rvert <$ `timeliness`
`WCET`	ns	$t_{return} < t_{call} +$ `WCET`

Table 5 *getTerminalTime()* real-time capabilities

`timeliness` characterizes the maximum absolute difference between the *return time* and measured *terminalTime*.

For reduction of the value of **`timeliness`**, a *time service* may compensate systematic delays adding a positive offset to the actually measured *terminal time*. This may cause the returned values to be in the future compared to *return time*.

Illustration

On FPGA *façades*, **`timeliness`** is typically equal to some clock cycles, e.g. some 10 ns for a 100 MHz clock,

On GPP *façades*, **`timeliness`** is typically close to the RTOS tasking transition time, e.g. some ms.

The following figure illustrates the real-time *capabilities* of *getTerminalTime()*:

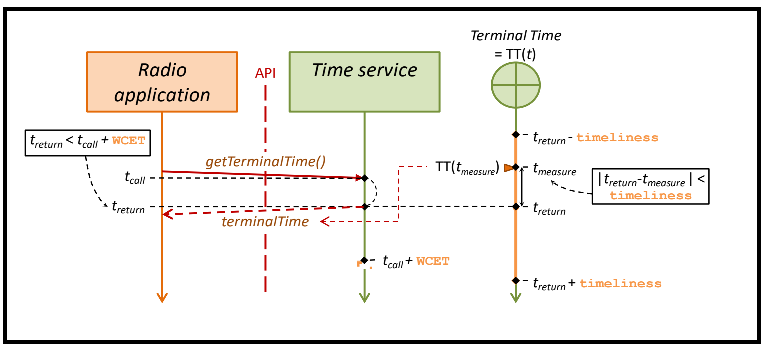

Figure 25 *getTerminalTime()* real-time capabilities

3.1.2.2 getTerminalTimeRateUncertainty()

3.1.2.2.1 *Overview*

getTerminalTimeRateUncertainty() returns the current value of **`terminalTimeRateUncertainty`** *variable*:

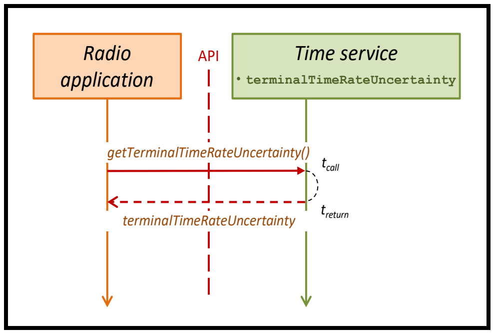

Figure 26 *getTerminalTimeRateUncertainty()* overview

3.1.2.2.2 *Signature*

The *signature* of *getTerminalTimeRateUncertainty()* **is specified as**:

```
void getTerminalTimeRateUncertainty(
    out RateUncertainty terminalTimeRateUncertainty
);
```

3.1.2.2.3 *Parameters*

The *parameters* of *getTerminalTimeRateUncertainty()* **are specified as**:

Name	Type	Direction	Semantics
terminalTimeRateUncertainty	***`RateUncertainty`*** See 3.4.3	***`out`***	Current value of **`terminalTimeRateUncertainty`**. Equal to **`UnknownRateUncertainty`** if the *time service* has no trustable value.

Table 6 *getTerminalTimeRateUncertainty()* parameters

3.1.2.2.4 *Exceptions*

None.

3.1.2.2.5 *Attributes*

The real-time *capabilities* attached to *getTerminalTimeRateUncertainty()* **are specified as**:

Name	Unit	Description
`WCET`	ns	$t_{return} < t_{call} +$ **`WCET`**

Table 7 *getTerminalTimeRateUncertainty()* real-time capabilities

3.1.3 TimeService::SystemTime::SystemTimeAccess

The *service interface* of the **`SystemTimeAccess`** *service* is:

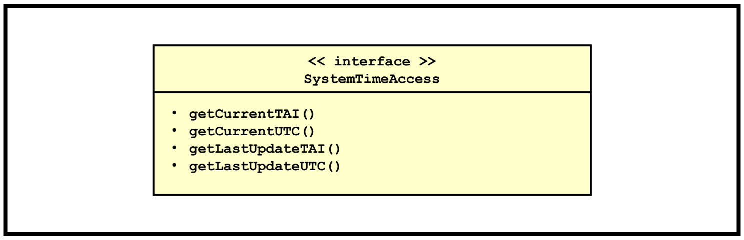

Figure 27 SystemTime::SystemTimeAccess *service interface*

3.1.3.1 getCurrentTAI()

3.1.3.1.1 *Overview*

getCurrentTAI() returns the current *TAI* value of *system time*, with the associated *time stamp* and *time uncertainty*:

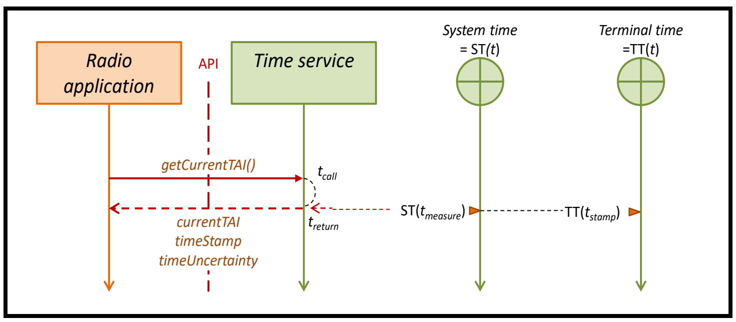

Figure 28 *getCurrentTAI()* overview

$t_{measure}$ **denotes** the *physical instant* close to t_{return} such that *currentTAI* = ST($t_{measure}$).

t_{stamp} **denotes** the *physical instant* matching $t_{measure}$ such that *timeStamp* = TT(t_{stamp}).

3.1.3.1.2 *Signature*

The *signature* of *getCurrentTAI()* **is specified as**:

```
void getCurrentTAI(
    out TimeValue currentTAI,
    out TimeValue timeStamp,
    out TimeUncertainty timeUncertainty
);
```

3.1.3.1.3 *Parameters*

The *parameters* of *getCurrentTAI()* **are specified as**:

Name	Type	Direction	Semantics
currentTAI	`TimeValue` See 3.4.1	*out*	*TAI* value estimated by *system time* close to *return time*. Referenced to 00:00:00, 1 January 2000 UTC, with no adjustment for leap seconds. Equal to `UndefinedTime` if the *time service* has not acquired *system time*.
timeStamp	`TimeValue` See 3.4.1	*out*	*Time stamp* (see 1.6.1) attached to *currentTAI*. Referenced to the initial value of *terminal time*. Possibly equal to `UndefinedTime` if the *time service* has not acquired *system time*.
timeUncertainty	`TimeUncertainty` See 3.4.2	*out*	*Time uncertainty* attached to *currentTAI*. Equal to `UnknownTimeUncertainty` if value is unknown to the *time service*.

Table 8 *getCurrentTAI()* parameters

3.1.3.1.4 *Exceptions*

None.

3.1.3.1.5 *Attributes*

The real-time *capabilities* attached to *getCurrentTAI()* **are specified as**:

Name	Unit	Description
`timeliness`	ns	$\|t_{measure} - t_{return}\| <$ `timeliness`
`WCET`	ns	$t_{return} < t_{call} +$ `WCET`

Table 9 *getCurrentTAI()* real-time capabilities

`timeliness` characterizes the maximum absolute difference between the *return time* and returned *currentTAI*.

For reduction of the value of `timeliness`, a *time service* may compensate systematic delays adding a positive offset to the actually measured *system time*. This may cause the returned values to be in the future compared to *return time*.

The following figure illustrates the real-time *capabilities* of *getCurrentTAI()*:

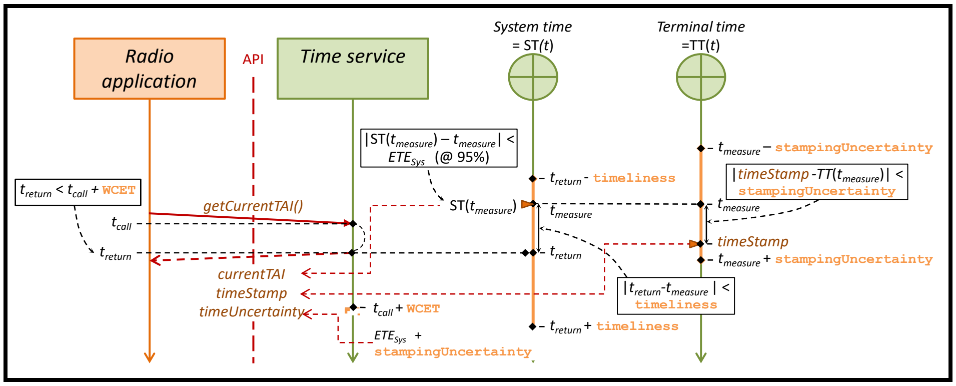

Figure 29 *getCurrentTAI()* real-time capabilities

3.1.3.2 getCurrentUTC()

3.1.3.2.1 *Overview*

getCurrentUTC() returns the current *UTC* value of *system time*, with the associated *time stamp* and *time uncertainty*:

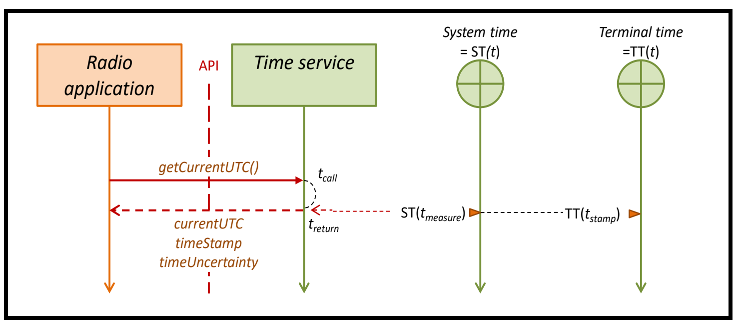

Figure 30 *getCurrentUTC()* overview

$t_{measure}$ **denotes** the *physical instant* close to t_{return} such that *currentUTC* = ST($t_{measure}$).

t_{stamp} **denotes** the *physical instant* matching $t_{measure}$ such that *timeStamp* = TT(t_{stamp}).

3.1.3.2.2 *Signature*

The *signature* of *getCurrentUTC()* **is specified as**:

```
void getCurrentUTC(
    out TimeValue currentUTC,
    out TimeValue timeStamp,
    out TimeUncertainty timeUncertainty
);
```

3.1.3.2.3 *Parameters*

The *parameters* of *getCurrentUTC()* **are specified as**:

Name	Type	Direction	Semantics
currentUTC	`TimeValue` See 3.4.1	`out`	*UTC* value estimated by *system time* close to *return time*. Referenced to 00:00:00, 1 January 2000 UTC, adjusted to reflect leap seconds. Equal to `UndefinedTime` if the *time service* has not acquired *system time*.
timeStamp	`TimeValue` See 3.4.1	`out`	*Time stamp* (see 1.6.1) attached to *currentUTC*. Referenced to the initial value of *terminal time*. Possibly equal to `UndefinedTime` if the *time service* has not acquired *system time*.
timeUncertainty	`TimeUncertainty` See 3.4.2	`out`	*Time uncertainty* attached to *currentUTC*. Equal to `UnknownTimeUncertainty` if value is unknown to the *time service*.

Table 10 *getCurrentUTC()* parameters

3.1.3.2.4 *Exceptions*

None.

3.1.3.2.5 *Attributes*

The real-time *capabilities* attached to *getCurrentUTC()* **are specified as**:

Name	Unit	Description
`timeliness`	ns	$\lvert t_{measure} - t_{return} \rvert <$ `timeliness`
`WCET`	ns	$t_{return} < t_{call} +$ `WCET`

Table 11 *getCurrentUTC()* real-time capabilities

`timeliness` characterizes the maximum absolute difference between the *return time* and returned *currentUTC*.

For reduction of the value of `timeliness`, a *time service* may compensate systematic delays adding a positive offset to the actually measured *system time*. This may cause the returned values to be in the future compared to *return time*.

The following figure illustrates the real-time *capabilities* of *getCurrentUTC()*:

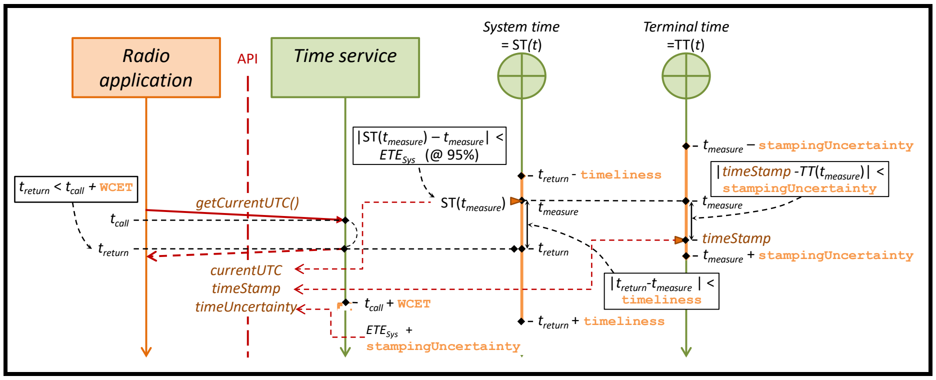

Figure 31 *getCurrentUTC()* real-time capabilities

3.1.3.3 getLastUpdateTAI()

3.1.3.3.1 *Overview*

getLastUpdateTAI() returns a *system time reference* (see 1.5.2.3) in *TAI* form corresponding to the last *system time update*:

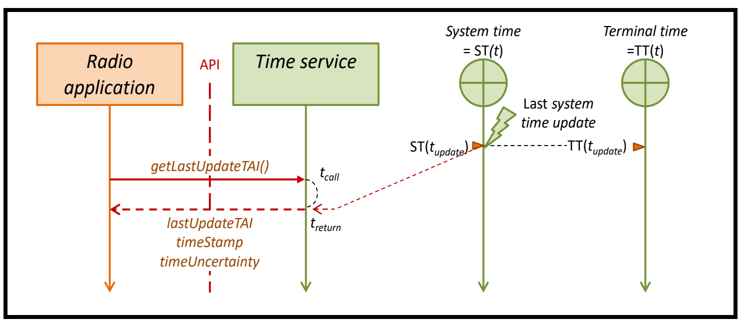

Figure 32 Principle of *getLastUpdateTAI()*

t_{update} **denotes** the *physical instant* at which the last *system time update* occurred.

3.1.3.3.2 *Signature*

The *signature* of *getLastUpdateTAI()* **is specified as**:

```
void getLastUpdateTAI(
    out TimeValue lastUpdateTAI,
    out TimeValue timeStamp,
    out TimeUncertainty timeUncertainty
);
```


3.1.3.3.3 *Parameters*

The *parameters* of *getLastUpdateTAI()* **are specified as**:

Name	Type	Direction	Semantics
lastUpdateTAI	`TimeValue` See 3.4.1	`out`	*TAI* value estimated by *system time* when the last *system time update* occurred. Referenced to 00:00:00, 1 January 2000 UTC, with no adjustment for leap seconds. Equal to `UndefinedTime` if the *time service* has not acquired *system time*.
timeStamp	`TimeValue` See 3.4.1	`out`	*Time stamp* (see 1.6.1) attached to *lastUpdateTAI*. Referenced to the initial value of *terminal time*. Possibly equal to `UndefinedTime` if the *time service* has not acquired *system time*.
timeUncertainty	`TimeUncertainty` See 3.4.2	`out`	*Time uncertainty* attached to *lastUpdateTAI*, corresponding to when the last *system time update* occurred. Equal to `UnknownTimeUncertainty` if value is unknown to the *time service.*

Table 12 *getLastUpdateTAI()* parameters

3.1.3.3.4 *Exceptions*

None.

3.1.3.3.5 *Attributes*

The real-time *capabilities* attached to *getLastUpdateTAI()* **are specified as**:

Name	Unit	Description
`reactivity`	ns	$t_{update} < t_{call}$ - `reactivity`
`WCET`	ns	$t_{return} < t_{call} +$ `WCET`

Table 13 *getLastUpdateTAI()* real-time capabilities

`reactivity` characterizes how much in advance from *call time* does the *system time update* need to have taken place in order to guarantee that it is reflected by the call.

The following figure illustrates the real-time *capabilities* of *getLastUpdateTAI()*:

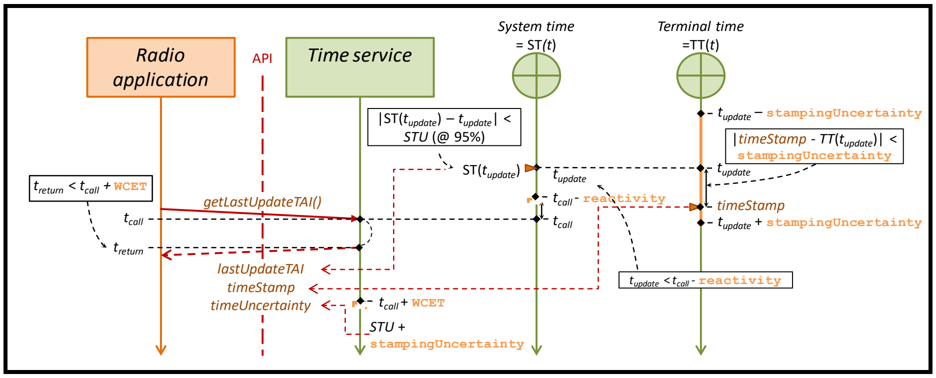

Figure 33 *getLastUpdateTAI()* capabilities

3.1.3.4 getLastUpdateUTC()

3.1.3.4.1 *Overview*

getLastUpdateUTC() returns the *system time reference* (see 1.5.2.3) in *UTC* form corresponding to the last *system time update*:

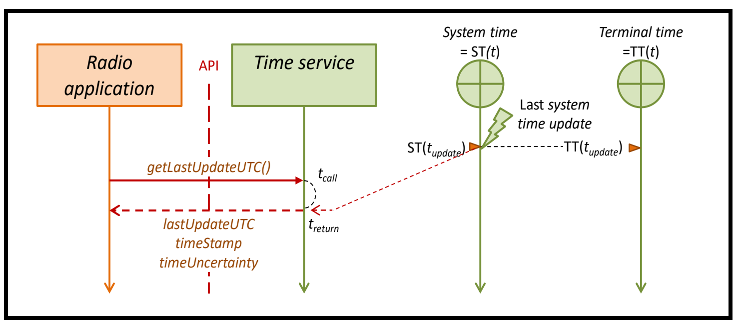

Figure 34 Principle of *getLastUpdateUTC()*

t_{update} **denotes** the *physical instant* at which the last *system time update* occurred.

3.1.3.4.2 *Signature*

The *signature* of *getLastUpdateUTC()* **is specified as**:

```
void getLastUpdateUTC(
    out TimeValue lastUpdateUTC,
    out TimeValue timeStamp,
    out TimeUncertainty timeUncertainty
);
```

3.1.3.4.3 *Parameters*

The *parameters* of *getLastUpdateUTC()* **are specified as**:

Name	Type	Direction	Semantics
lastUpdateUTC	`TimeValue` See 3.4.1	`out`	*UTC* value estimated by *system time* when the last *system time update* occurred. Referenced to 00:00:00, 1 January 2000 UTC, adjusted to reflect leap seconds. Equal to `UndefinedTime` if the *time service* has not acquired *system time*.
timeStamp	`TimeValue` See 3.4.1	`out`	*Time stamp* (see 1.6.1) attached to *lastUpdateUTC*. Referenced to the initial value of *terminal time*. Possibly equal to `UndefinedTime` if the *time service* has not acquired *system time*.
timeUncertainty	`TimeUncertainty` See 3.4.2	`out`	*Time uncertainty* attached to *lastUpdateUTC*, corresponding to when the last *system time update* occurred. Equal to `UnknownTimeUncertainty` if value is unknown to the *time service.*

Table 14 *getLastUpdateUTC()* parameters

3.1.3.4.4 *Exceptions*

None.

3.1.3.4.5 *Attributes*

The real-time *capabilities* attached to *getLastUpdateUTC()* **are specified as**:

Name	Unit	Description
`reactivity`	Ns	$t_{update} < t_{call}$ - `reactivity`
`WCET`	ns	$t_{return} < t_{call} +$ `WCET`

Table 15 *getLastUpdateUTC()* real-time capabilities

`reactivity` characterizes how much in advance from *call time* does the *system time update* need to have taken place in order to guarantee that it reflected by the call.

The following figure illustrates the real-time capabilities of *getLastUpdateUTC()*:

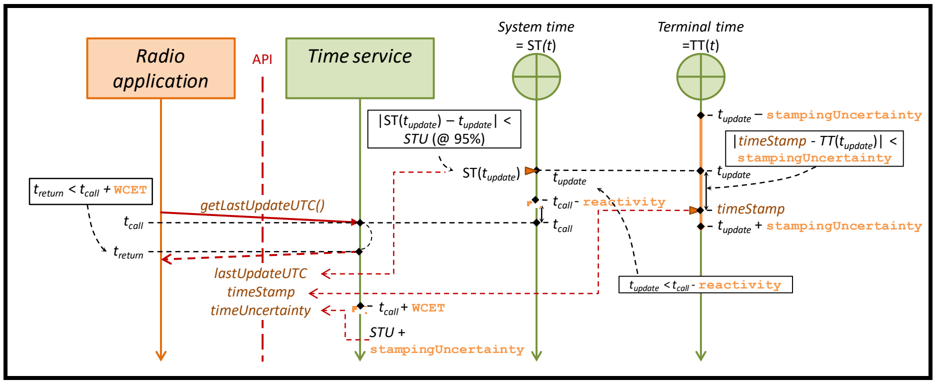

Figure 35 Meaning of *getLastUpdateUTC()* capabilities

3.1.4 *TimeService::SystemTime::StandardTimeProvision*

The *service interface* of the **StandardTimeProvision** *service* is:

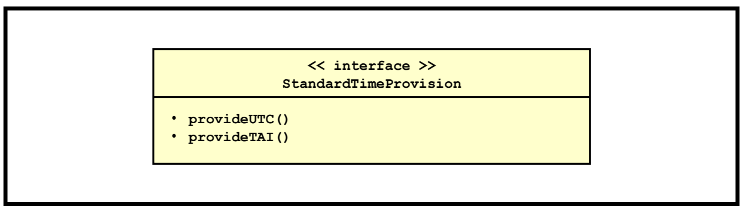

Figure 36 SystemTime::StandardTimeProvision *service interface*

3.1.4.1 provideTAI()

3.1.4.1.1 *Overview*

provideTAI() provides a *standard time reference* (see 1.6.4.2) in the past using *TAI*; for *system time* update to take place the provided information must decrease *time uncertainty* of the *system time*:

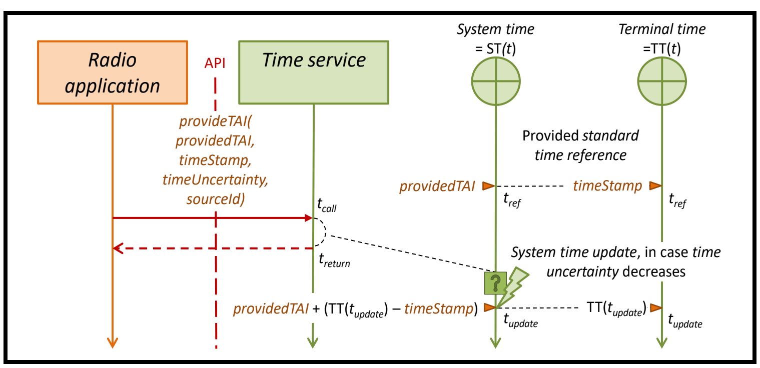

Figure 37 *provideTAI()* overview

t_{ref} **denotes** the *physical instant* corresponding to the provided *standard time reference*.

t_{update} **denotes** the *physical instant* at which the *system time update* occurs, if the provided *standard time reference* improved *time uncertainty*.

3.1.4.1.2 *Signature*

The *signature* of *provideTAI()* **is specified as**:

```
void provideTAI(
    in TimeValue providedTAI,
    in TimeValue timeStamp,
    in TimeUncertainty timeUncertainty,
    in int sourceId
);
```

3.1.4.1.3 *Parameters*

The *parameters* of *provideTAI()* **are specified as**:

Name	Type	Direction	Semantics
providedTAI	***TimeValue*** See 3.4.1	***in***	*TAI* value to be considered for eventual *system time update*. Referenced to 00:00:00, 1 January 2000 UTC, with no adjustment for leap seconds. The candidate value for *system time update* is equal to: *providedTAI* + TT(t_{update}) – *timeStamp.* If equal to **UndefinedTime**, *time service* behavior is unspecified.
timeStamp	***TimeValue*** See 3.4.1	***in***	*Time stamp* (see 1.6.1) attached to *providedTAI*. Referenced to the initial value of *terminal time*. If equal to **UndefinedTime**, the behavior of *time service* is unspecified.
timeUncertainty	***TimeUncertainty*** See 3.4.2	***in***	*Time uncertainty* attached to *providedTAI*. The candidate value for *system time update* is equal to: *timeUncertainty* $+\int_{t_{ref}}^{t_{update}} TTRU(t) \cdot dt.$ If equal to **UnknownTimeUncertainty**, the behavior of *time service* is unspecified.
sourceId	***int***	***in***	*Radio application* identification as the *standard time source* having delivering the *standard time reference.*

Table 16 *provideTAI()* parameters

sourceId should uniquely identify a given *standard time source*.

3.1.4.1.4 *Exceptions*

The *exceptions* attached to the *primitive* **are specified as** (see section 3.2.1):

- `FutureTimeStamp`.

3.1.4.1.5 *Attributes*

The real-time *capabilities* attached to *provideTAI()* **are specified as**:

Name	Unit	Description
`reactivity`	ns	$t_{update} < t_{call} +$ `reactivity`
`WCET`	ns	$t_{return} < t_{call} +$ `WCET`

Table 17 *provideTAI()* real-time capabilities

`reactivity` characterizes how far from *call time* the *system time update* can take place.

The following figure illustrates the real-time capabilities of *provideTAI()*:

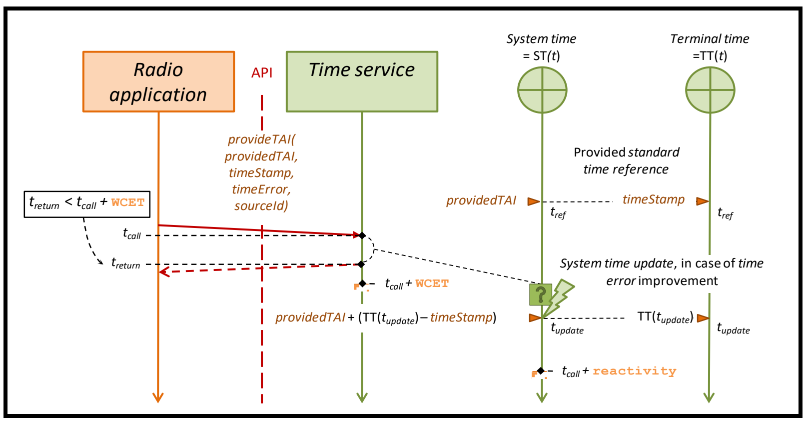

Figure 38 *provideTAI()* real-time capabilities

3.1.4.2 provideUTC()

3.1.4.2.1 *Overview*

provideUTC() provides a *standard time reference* (see 1.6.4.2) in the past using *UTC*; for *system time* update to take place the provided information must decrease *time uncertainty* of the *system time*:

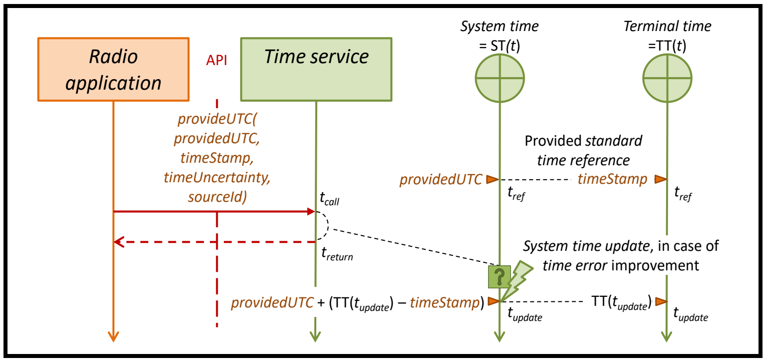

Figure 39 Principle of *provideUTC()*

t_{ref} **denotes** the *physical instant* corresponding to the provided *standard time reference*.

t_{update} **denotes** the *physical instant* at which the *system time update* occurs, if the provided *standard time reference* improved *time uncertainty*.

3.1.4.2.2 *Signature*

The *signature* of *provideUTC()* **is specified as**:

```
void provideUTC(
    in TimeValue providedUTC,
    in TimeValue timeStamp,
    in TimeUncertainty timeUncertainty,
    in int sourceId
);
```

3.1.4.2.3 *Parameters*

The *parameters* of *provideUTC()* **are specified as**:

Name	Type	Direction	Semantics
providedUTC	***TimeValue*** See 3.4.1	***in***	*UTC* value to be considered for eventual *system time update*. Referenced to 00:00:00, 1 January 2000 UTC, adjusted to reflect leap seconds. The candidate value for *system time update* is equal to *providedUTC* + TT(t_{update}) – *timeStamp.* If equal to **UndefinedTime**, *time service* behavior is unspecified.
timeStamp	***TimeValue*** See 3.4.1	***in***	*Time stamp* (see 1.6.1) attached to *providedUTC*. Referenced to the initial value of *terminal time*. If equal to **UndefinedTime**, the behavior of *time service* is unspecified.
timeUncertainty	***TimeUncertainty*** See 3.4.2	***in***	*Time uncertainty* attached to *providedUTC*. The candidate value for *system time update* is equal to: *timeUncertainty* $+\int_{t_{ref}}^{t_{update}} TTRU(t) \cdot dt.$ If equal to **UnknownTimeUncertainty**, the behavior of *time service* is unspecified.
sourceId	***int***	***in***	*Radio application* identification as the *standard time source* having delivering the *standard time reference*.

Table 18 *provideUTC()* parameters

sourceId should uniquely identify a given *standard time source*.

3.1.4.2.4 *Exceptions*

The *exceptions* attached to the *primitive* **are specified as** (see section 3.2.1):

- **FutureTimeStamp**.

3.1.4.2.5 *Attributes*

The real-time *capabilities* attached to *provideUTC()* **are specified as**:

Name	Unit	Description
`reactivity`	ns	$t_{update} < t_{call} +$ `reactivity`
`WCET`	ns	$t_{return} < t_{call} +$ `WCET`

Table 19 *provideUTC()* real-time capabilities

`reactivity` characterizes how long, after *call time*, may the *system time update* take place.

The following figure illustrates the real-time capabilities of *provideUTC()*:

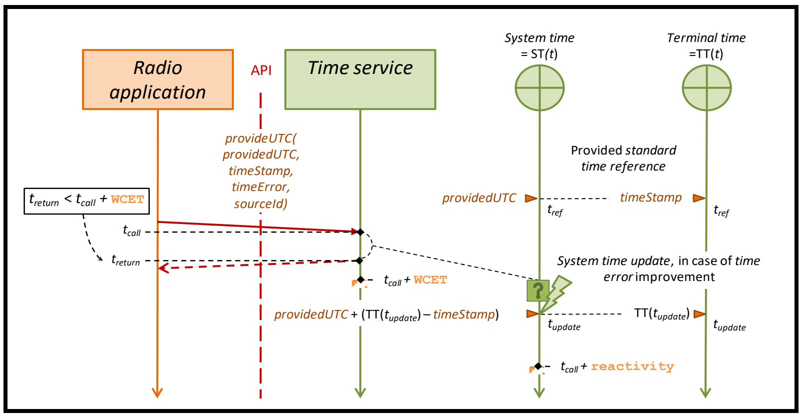

Figure 40 *provideUTC()* real-time capabilities

3.1.5 *TimeService::StandardTimes::ReferencesNotification*

The *service interface* of the **ReferencesNotification** *service* is:

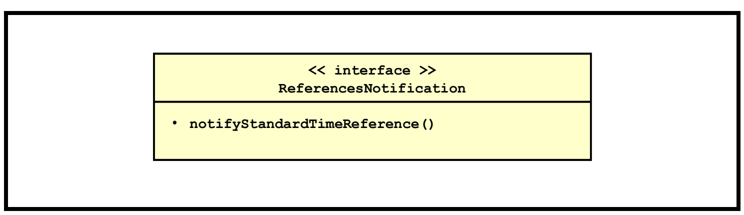

Figure 41 StandardTimes::ReferencesNotification *service interface*

3.1.5.1 notifyStandardTimeReference()

3.1.5.1.1 *Overview*

notifyStandardTimeReference() notifies the *radio application* of an occurrence of a *standard time reference* (see 1.5.2.3) delivered by a *standard time source* using *TAI* and *UTC* representations:

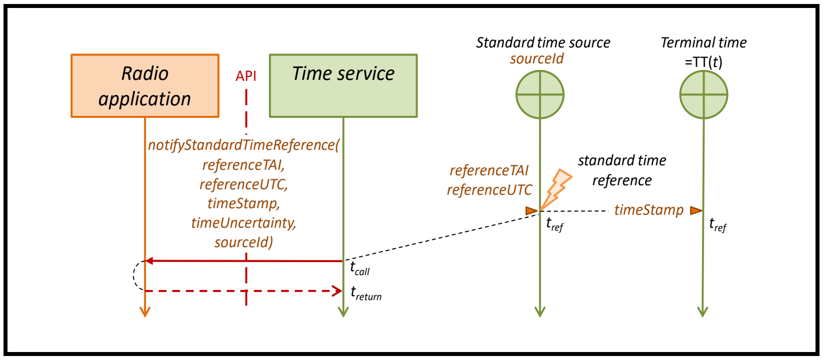

Figure 42 *notifyStandardTimeReference()* overview

Design of *radio applications* need to take into account that, when all *standard time sources* are inactive, no *standard time references* are notified.

Each *standard time reference* generated by *standard time source* is notified to the *radio application* via a dedicated call. The *standard time source* having generated the *standard time reference* has an identifier which is unique within the *time service*.

3.1.5.1.2 *Signature*

The *signature* of *notifyStandardTimeReference()* **is specified as**:

```
void notifyStandardTimeReference(
    in TimeValue referenceTAI,
    in TimeValue referenceUTC,
    in TimeValue timeStamp,
    in TimeUncertainty timeUncertainty,
    in int sourceId
);
```

3.1.5.1.3 *Parameters*

The *parameters* of *notifyStandardTimeReference()* **are specified as**:

Name	Type	Direction	Semantics
referenceTAI	`TimeValue` See 3.4.1	`in`	*TAI* value estimated by the *standard time source*. Referenced to 00:00:00, 1 January 2000 UTC, with no adjustment for leap seconds. Never equal to `UndefinedTime`.
referenceUTC	`TimeValue` See 3.4.1	`in`	*UTC* value estimated by the *standard time source.* Referenced to 00:00:00, 1 January 2000 UTC, adjusted to reflect leap seconds. Never equal to `UndefinedTime`.
timeStamp	`TimeValue` See 3.4.1	`in`	*Time stamp* (see 1.6.1) attached to *referenceTAI* and *referenceUTC*. Referenced to the initial value of *terminal time*. Never equal to `UndefinedTime`.
timeUncertainty	`TimeUncertainty` See 3.4.2	`in`	*Time uncertainty* attached to *referenceTAI* and *referenceUTC*. Equal to `UnknownTimeUncertainty` if *time service* is not able to determine *time uncertainty*. Never equal to `UndefinedTime`.
sourceId	`int`	`in`	Identifier of the *standard time source* that provided the *standard time reference* notified by the call.

Table 20 *notifyStandardTimeReference()* parameters

3.1.5.1.4 *Exceptions*

None.

3.1.5.1.5 *Attributes*

The real-time *capabilities* attached to *notifyStandardTimeReference()* **are specified as**:

Name	Unit	Description
`reactivity`	ns	$t_{call} < t_{ref} +$ `reactivity`
`WCEET`	ns	$t_{return} < t_{call} +$ `WCEET`

Table 21 *notifyStandardTimeReference()* real-time capabilities

`reactivity` characterizes how long after occurrence of the *standard time reference* the call to *notifyStandardTimeReference()* can take place.

The following figure illustrates the real-time capabilities of *notifyStandardTimeReference()*:

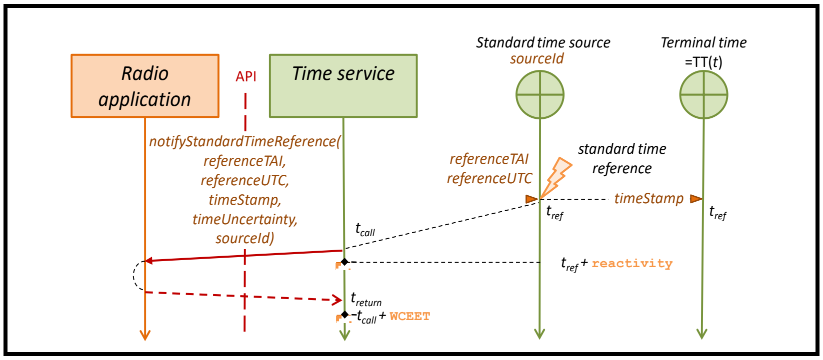

Figure 43 *notifyStandardTimeReference()* real-time capabilities

3.1.6 *TimeService::SpecificTimes::SpecificTimeHandling*

The *service interface* of the **`SpecificTimeHandling`** *service* is:

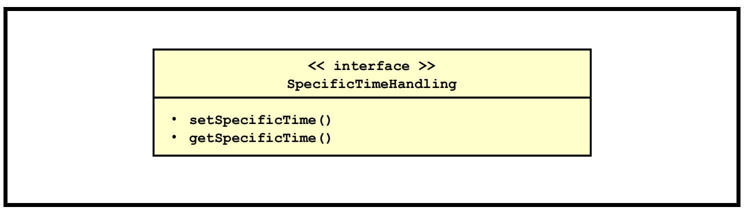

Figure 44 SystemTimes::SpecificTimeHandling *service interface*

3.1.6.1 setSpecificTime()

3.1.6.1.1 *Overview*

setSpecificTime() provides the *time service* with a *specific time reference* in the past used to set the identified *specific time*:

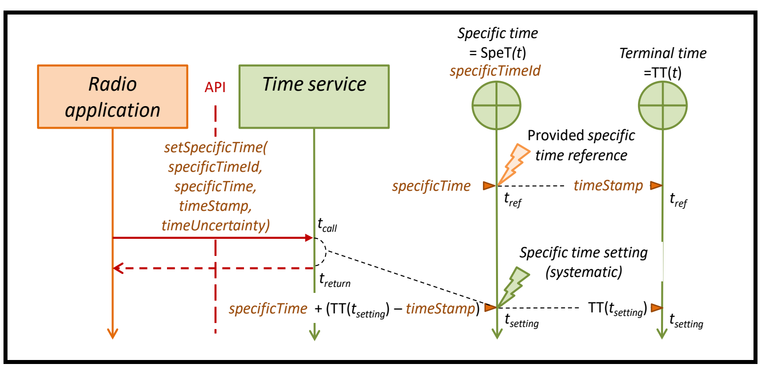

Figure 45 *setSpecificTime()* overview

t_{ref} **denotes** the *physical instant* corresponding to the provided *specific time reference*.

$t_{setting}$ **denotes** the *physical instant* at which setting of *specific time* occurs.

3.1.6.1.2 *Signature*

The *signature* of *setSpecificTime ()* **is specified as**:

```
void setSpecificTime(
    in int specificTimeId,
    in TimeValue specificTime,
    in TimeValue timeStamp,
    in TimeUncertainty timeUncertainty
);
```

3.1.6.1.3 *Parameters*

The *parameters* of *setSpecificTime()* **are specified as**:

Name	Type	Direction	Semantics
specificTimeId	`int`	`in`	Identifier of the *specific time* to be set.
specificTime	`TimeValue` See 3.4.1	`in`	The value to be used for *specific time* setting is equal to *specificTime* + TT($t_{setting}$) – *timeStamp*. If equal to `UndefinedTime`, the behavior of *time service* is unspecified.
timeStamp	`TimeValue` See 3.4.1	`in`	*Time stamp* (see 1.6.1) attached to *specificTime*. Referenced to the initial value of *terminal time*. If equal to `UndefinedTime`, the behavior of *time service* is unspecified.
timeUncertainty	`TimeUncertainty` See 3.4.2	`in`	*Time uncertainty* attached to *specificTime*. The value at *specific time* setting is equal to: *timeUncertainty* $+\int_{t_{ref}}^{t_{setting}} TTRU(t) \cdot dt$. If equal to `UndefinedTime`, the behavior of *time service* is *unspecified*. If equal to `UnknownTimeUncertainty`, *time uncertainty* will remain unknown until a value is set.

Table 22 *setSpecificTime()* parameters

3.1.6.1.4 *Exceptions*

The *exceptions* attached to the *primitive* **are specified as** (see section 3.2.1):

- **`FutureTimeStamp`**,
- **`InvalidSpecificTimeId`**.

3.1.6.1.5 *Attributes*

The **`maxSpecificTimes`** *capability* (see section 1.5.3.3.5) sets the validity range of *specificTimeId* being from 1 to **`maxSpecificTimes`**.

The real-time *capabilities* attached to *setSpecificTime()* **are specified as**:

Name	Unit	Description
`reactivity`	ns	$t_{setting} < t_{call} +$ `reactivity`
`WCET`	ns	$t_{return} < t_{call} +$ `WCET`

Table 23 *setSpecificTime()* real-time capabilities

`reactivity` characterizes how far from *call time* the *specific time update event* can take place.

The following figure illustrates the real-time capabilities of *setSpecificTime()*:

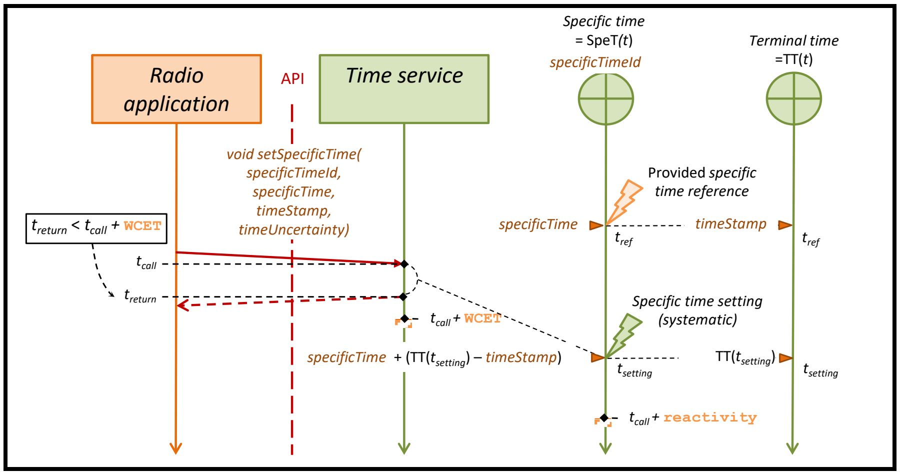

Figure 46 *setSpecificTime ()* real-time *capabilities*

3.1.6.2 getSpecificTime()

3.1.6.2.1 *Overview*

getSpecificTime() returns the current value of *specific time*, with the associated *time stamp* and *time uncertainty*:

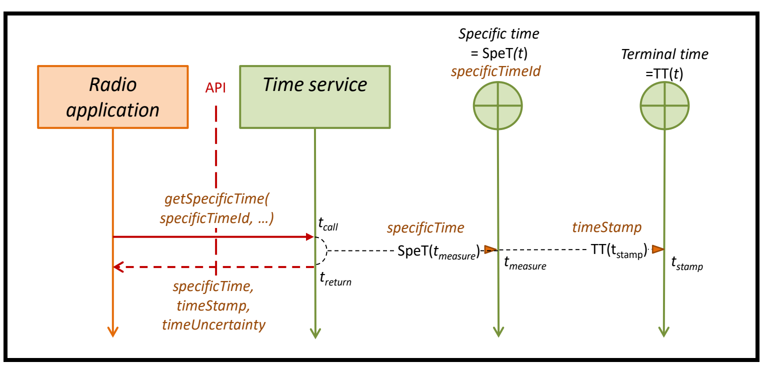

Figure 47 *getSpecificTime()* overview

$t_{measure}$ **denotes** the *physical instant* close to t_{return} such that *specificTime* = SpeT($t_{measure}$).

t_{stamp} **denotes** the *physical instant* matching $t_{measure}$ such that *timeStamp* = TT(t_{stamp}).

3.1.6.2.2 *Signature*

The *signature* of *getSpecificTime()* **is specified as**:

```
void getSpecificTime(
    in int specificTimeId,
    out TimeValue specificTime,
    out TimeValue timeStamp,
    out TimeUncertainty timeUncertainty
);
```

3.1.6.2.3 *Parameters*

The *parameters* of *getSpecificTime()* **are specified as**:

Name	Type	Direction	Semantics
specificTimeId	***int***	***in***	Identifier of the *specific time* to be get.
specificTime	***TimeValue*** See 3.4.1	***out***	Value of *specific time* close to when *getSpecificTime()* returns. Equal to **UndefinedTime** if the *specific time* was not set.
timeStamp	***TimeValue*** See 3.4.1	***out***	*Time stamp* (see 1.6.1) attached to *specificTime*. Referenced to the initial value of *terminal time*. May be equal to **UndefinedTime** if the *specific time* was not set.
timeUncertainty	***TimeUncertainty*** See 3.4.2	***out***	*Time uncertainty* attached to *specificTime*. Equal to **UnknownTimeUncertainty** if the *specific time* was not set or set with **UnknownTimeUncertainty**. May be equal to **UndefinedTime** if the *specific time* was not set.

Table 24 *getSpecificTime()* parameters

3.1.6.2.4 *Exceptions*

The *exceptions* attached to the *primitive* **are specified as** (see section 3.2.1):

- **InvalidSpecificTimeId**.

3.1.6.2.5 *Attributes*

The **maxSpecificTimes** *capability* (see section 1.5.3.3.5) sets the validity range of *specificTimeId* being from 1 to **maxSpecificTimes.**

The real-time *capabilities* attached to *getSpecificTime()* **are specified as**:

Name	Unit	Description
timeliness	ns	$\lvert t_{measure} - t_{return} \rvert <$ timeliness
WCET	ns	$t_{return} < t_{call} +$ WCET

Table 25 *getSpecificTime()* real-time capabilities

timeliness characterizes how close to the *return time* the returned *specificTime* was measured.

The following figure illustrates the real-time capabilities of *getSpecificTime()*:

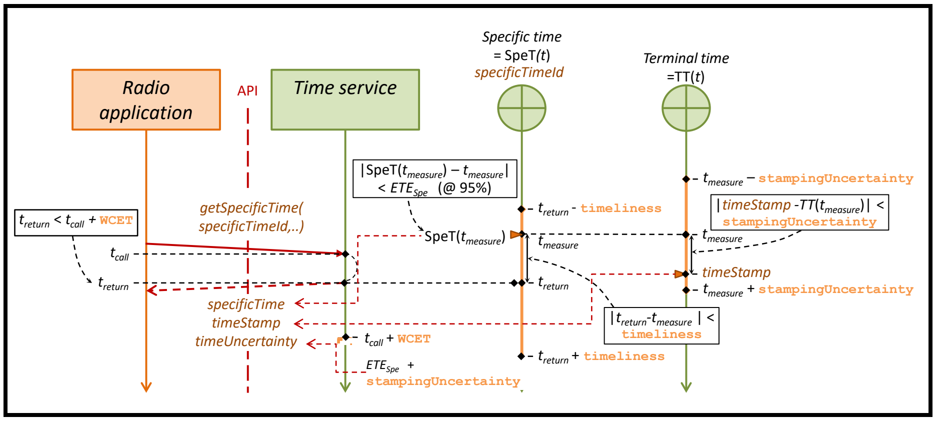

Figure 48 *getSpecificTime()* real-time capabilities

3.1.7 TimeService::SpecificTimes::SettingsNotification

The *service interface* of the **SettingsNotification** *service* is:

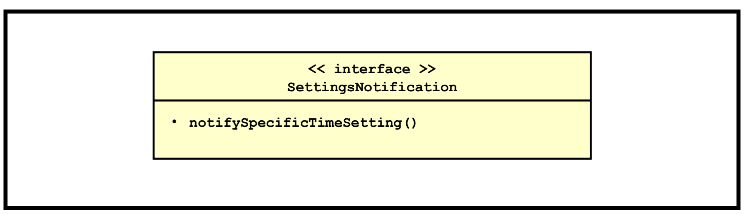

Figure 49 SpecificTimes::SettingsNotification *service interface*

3.1.7.1 notifySpecificTimeSetting()

3.1.7.1.1 *Overview*

notifySpecificTimeSetting() notifies the *radio application* that a *specific time* has been set by a *radio application*, using *setSpecificTime()*:

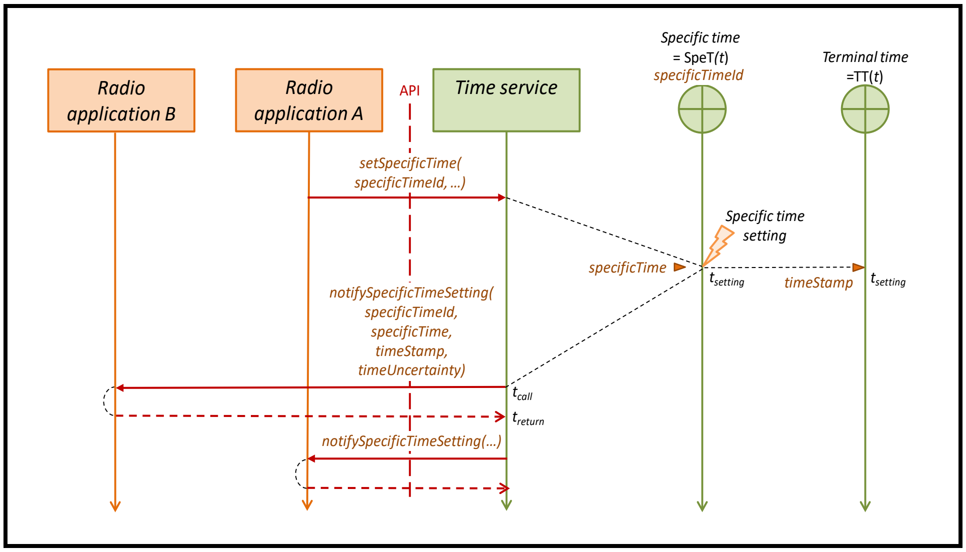

Figure 50 *notifySpecificTimeSetting()* overview

3.1.7.1.2 *Signature*

The *signature* of *notifySpecificTimeSetting()* **is specified as**:

```
void notifySpecificTimeSetting(
    in int specificTimeId,
    in TimeValue specificTime,
    in TimeValue timeStamp,
    in TimeUncertainty timeUncertainty
);
```

3.1.7.1.3 *Parameters*

The *parameters* of *notifySpecificTimeSetting()* **are specified as**:

Name	Type	Direction	Semantics
specificTimeId	`int`	`in`	Identifier of the *specific time* for which a setting took place.
specificTime	`TimeValue` See 3.4.1	`in`	*Specific time* value applied by the reported setting. Never equal to `UndefinedTime`.
timeStamp	`TimeValue` See 3.4.1	`in`	*Time stamp* (see 1.6.1) attached to *specificTime*. Referenced to the initial value of *terminal time*. Never equal to `UndefinedTime`.
timeUncertainty	`TimeUncertainty` See 3.4.2	`in`	*Time uncertainty* attached to *specificTime*. Equal to `UnknownTimeUncertainty` if the *specific time* was set with `UnknownTimeUncertainty`. Never equal to `UndefinedTime`.

Table 26 *notifySpecificTimeSetting()* parameters

3.1.7.1.4 *Exceptions*

None.

3.1.7.1.5 *Attributes*

The real-time *capabilities* attached to *notifySpecificTimeSetting()* **are specified as**:

Name	Unit	Description
`Reactivity`	ns	$t_{call} < t_{setting}$ + `reactivity`
`WCEET`	ns	$t_{return} < t_{call}$ + `WCEET`

Table 27 *notifySpecificTimeSetting()* real-time capabilities

`reactivity` characterizes how long, after occurrence of the *specific time* setting, the call to *notifySpecifcTimeSetting()* can take place.

The following figure illustrates the real-time *capabilities* of *notifySpecificTimeSetting()*:

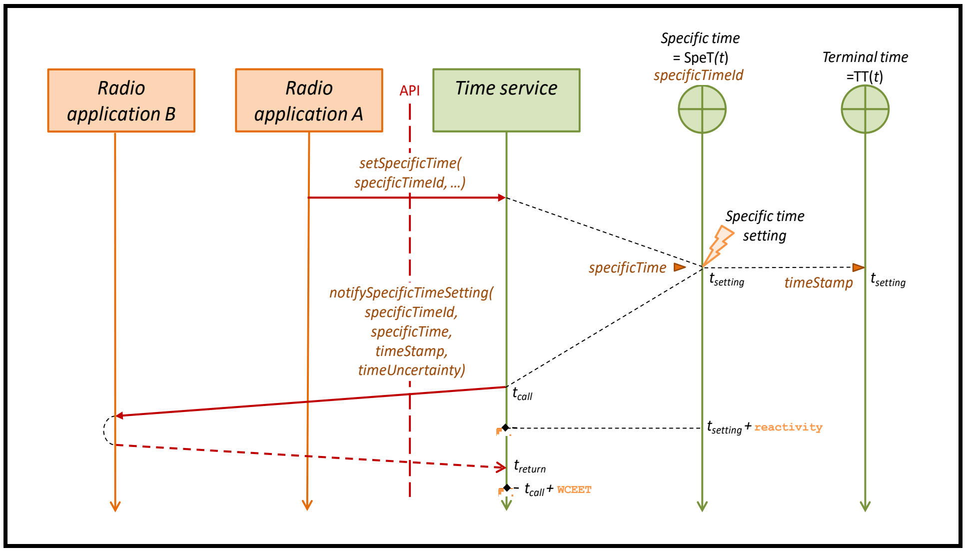

Figure 51 *notifySpecificTimeSetting()* real-time capabilities

3.2. Exceptions

An *exception* is an abnormal situation related to the calling context or to parameters values, detected during execution of a called *primitive* (see [Ref1]).

Exceptions are only specified for *provide services.*

3.2.1 Specification

General *exceptions* **are specified by** the following table:

Name	Applies to	Description
`FutureTimeStamp`	*provideTAI()* *provideUTC()* *setSpecificTime()*	The provided *standard time reference* is in the future compared to t_{call}.

Table 28 Specification of general *exceptions*

Range *exceptions* **are specified by** the following table:

Name	Applies to	Description
`InvalidSpecificTimeId`	*setSpecificTime()* *getSpecificTime()*	*specificTimeId* is greater than `maxSpecificTimes` or lower than `1`.

Table 29 Specification of range *exceptions*

3.2.2 *Associated capabilities*

The **exceptionsActive** *capability* **is specified as** a boolean that indicates if the *time service* raises *exceptions*.

The **supportedExceptions** *capability* **is specified as** a set of booleans indicating, for each specified *exception*, if it is raised by the *time service*.

3.3. Attributes

This section specifies the *attributes* attached to *time service*.

3.3.1 Overview

Attributes are characterized by the lifespan of their constant value (see [Ref1]):

- *Capabilities*: *attributes* constant over the lifetime of *time service*,
- *Properties*: *attributes* constant during the **CONFIGURED** state,
- *Variables*: *attributes* not expected to be constant.

How *capabilities* and *properties* values are accessible to the *radio application* is unspecified.

Variables are only accessible to the *radio application* via access *primitives*.

See section 2.4.1 for specification of the **CONFIGURED** state.

3.3.2 Attributes conformance

The specification of a value for an *attribute* of a *time service* is either **mandatory** or **optional**.

For mandatory *attributes* the implementer must specify a value whereas for optional *attributes* the implementer may or may not specify a value.

R03 A *time service* **shall**, for each *attribute* for which a value is specified, comply with the related specified content.

Depending on implementation choices, specification of *attribute* values can be part of the documentation of the *time service* and/or can be implemented in software.

3.3.3 Capabilities

The following table lists the general *capabilities* of a *time service*:

Capability	Meaning	Section
terminalTimeRateMaxUncertainty	Scalar that reflects the maximum *terminal time rate uncertainty* of a *time service*, in ppb.	1.5.3.1.7
maxSpecificTimes	Integer that captures the number of *specific times* implemented by a *time service*.	1.5.3.3.5
stampingUncertainty	Scalar reflecting the *stamping uncertainty* achieved by a *service* of a *time service*, expressed in ns.	1.6.1.3
selectedOptionalServices	Set of boolean values indicating which optional *services* specified by the *time service facility* are implemented.	2.3.2
exceptionsActive	Boolean that indicates if the *time service* raises exceptions.	3.2.2
supportedExceptions	Set of booleans indicating, for each specified *exception*, if it is raised by the *time service*.	3.2.2

Table 30 ***Time service*** **general capabilities**

Specification of values for all general *capabilities* is **mandatory**.

Section 3.1 specifies the *capabilities* attached to *primitives*. Specification of values for these *capabilities* is **optional**.

3.3.4 Properties

None.

3.3.5 Variables

The following table lists the specified *variables*:

Variable	Meaning	Section
terminalTimeRateUncertainty	Scalar that reflects the current *terminal time rate uncertainty* of a *time service*, in ppb.	1.5.3.1.8

Table 31 Variables

The following table lists the access primitives and the *variables* they give access to:

Variable access primitive	Accessed variable	Section
getTerminalTimeRateUncertainty()	terminalTimeRateUncertainty	3.1.2.2

Table 32 Variables access primitives

3.4. Types

3.4.1 TimeValue

The ***TimeValue*** *type* **is specified as** a structure that reflects a *time value* using a ***seconds*** and ***nanoseconds*** 32-bit signed fields, where the ***seconds*** field reflects the number of entire seconds that have physically elapsed since a referenced time and the ***nanoseconds*** field reflects the remaining number of nanoseconds.

UndefinedTime **is specified as** the reserved value reflecting an undefined *time value*.

The associated declarations **are specified as**:

```
struct TimeValue {
    long seconds,              // in seconds
    long nanoseconds};         // in nanoseconds (<1.000.000.000)

const TimeValue UndefinedTime = {0xFFFFFFFF, 0xFFFFFFFF};
```

3.4.2 TimeUncertainty

The ***TimeUncertainty*** *type* **is specified as** a 32-bit signed integer that reflects a *time uncertainty* value, in ns.

A *time uncertainty* is the addition of an *estimation uncertainty* (see section 1.6.2) and a *stamping uncertainty* (see section 1.6.1.2).

The natural valid values for ***TimeUncertainty*** range from **0** to **2,000,000,000**.

Beyond<2^n>SecTimeUncertainty, for **n** = 1 to 14, **are specified as** reserved values that reflect higher values of *time uncertainty*, ranging from beyond 2 s to beyond 16384 s.

UnknownTimeUncertainty **is specified as** the reserved value reflecting that the *time service* is unable to evaluate *time uncertainty* while the time itself is valid.

UndefinedTime **is specified as** the reserved value reflecting that the *time service* is unable to return a *time value*. This implies that the *time uncertainty* is unknown.

The following table summarizes the possible *`TimeUncertainty`* values:

Identifier	Meaning	Hexadecimal	Decimal
None	*Time uncertainty*, in ns (for values up to 2 s)	`0x0` to `0x77359400`	`0` to `2,000,000,000`
`Beyond2SecTimeUncertainty`	2 s < *time uncertainty* ≤ 4 s	`0xFFFFFFF0`	`-16`
`Beyond4SecTimeUncertainty`	4 s < *time uncertainty* ≤ 8 s	`0xFFFFFFF1`	`-15`
`Beyond8SecTimeUncertainty`	8 s < *time uncertainty* ≤ 16 s	`0xFFFFFFF2`	`-14`
`Beyond16SecTimeUncertainty`	16 s < *time uncertainty* ≤ 32 s	`0xFFFFFFF3`	`-13`
`Beyond32SecTimeUncertainty`	32 s < *time uncertainty* ≤ 64 s	`0xFFFFFFF4`	`-12`
`Beyond64SecTimeUncertainty`	64 s < *time uncertainty* ≤ 128 s	`0xFFFFFFF5`	`-11`
`Beyond128SecTimeUncertainty`	128 s < *time uncertainty* ≤ 256 s	`0xFFFFFFF6`	`-10`
`Beyond256SecTimeUncertainty`	256 s < *time uncertainty* ≤ 521 s	`0xFFFFFFF7`	`-9`
`Beyond512SecTimeUncertainty`	512 s < *time uncertainty* ≤ 1024 s	`0xFFFFFFF8`	`-8`
`Beyond1024SecTimeUncertainty`	1024 s < *time uncertainty* ≤ 2048 s	`0xFFFFFFF9`	`-7`
`Beyond2048SecTimeUncertainty`	2048 s < *time uncertainty* ≤ 4096 s	`0xFFFFFFFA`	`-6`
`Beyond4096SecTimeUncertainty`	4096 s < *time uncertainty* ≤ 8192 s	`0xFFFFFFFB`	`-5`
`Beyond8192SecTimeUncertainty`	8192 s < *time uncertainty* ≤ 16384 s	`0xFFFFFFFC`	`-4`
`Beyond16384SecTimeUncertainty`	*time uncertainty* > 16384 s	`0xFFFFFFFD`	`-3`
`UnknownTimeUncertainty`	Unknown *time uncertainty*	`0xFFFFFFFE`	`-2`
`UndefinedTime`	Undefined *time*	`0xFFFFFFFF`	`-1`
Not allowed	Not allowed positive	`0x77359401` to `0x7FFFFFFF`	`> 2,000,000,000`
Not allowed	Not allowed negative	`0x80000000` to `0xFFFFFFEF`	`< -16`

Table 33 Possible values for TimeUncertainty

The associated declarations **are specified as**:

```
typedef long TimeUncertainty;

const TimeUncertainty Beyond2SecTimeUncertainty = 0xFFFFFFF0;
const TimeUncertainty Beyond4SecTimeUncertainty = 0xFFFFFFF1;
const TimeUncertainty Beyond8SecTimeUncertainty = 0xFFFFFFF2;
const TimeUncertainty Beyond16SecTimeUncertainty = 0xFFFFFFF3;
const TimeUncertainty Beyond32SecTimeUncertainty = 0xFFFFFFF4;
const TimeUncertainty Beyond64SecTimeUncertainty = 0xFFFFFFF5;
const TimeUncertainty Beyond128SecTimeUncertainty = 0xFFFFFFF6;
const TimeUncertainty Beyond256SecTimeUncertainty = 0xFFFFFFF7;
const TimeUncertainty Beyond512SecTimeUncertainty = 0xFFFFFFF8;
const TimeUncertainty Beyond1024SecTimeUncertainty = 0xFFFFFFF9;
const TimeUncertainty Beyond2048SecTimeUncertainty = 0xFFFFFFFA;
const TimeUncertainty Beyond4096SecTimeUncertainty = 0xFFFFFFFB;
const TimeUncertainty Beyond8192SecTimeUncertainty = 0xFFFFFFFC;
const TimeUncertainty Beyond16384SecTimeUncertainty = 0xFFFFFFFD;

const TimeUncertainty UnknownTimeUncertainty = 0xFFFFFFFE;
const TimeUncertainty UndefinedTime = 0xFFFFFFFF;
```

3.4.3 *RateUncertainty*

`RateUncertainty` type **is specified as** a 32-bit signed integer that reflects a time rate uncertainty value, expressed in parts-per-billion (ppb).

`UnknownRateUncertainty` **is specified as** the reserved value reflecting that *time service* is not able to evaluate the considered time rate uncertainty.

The associated declarations **are specified as**:

```
typedef long RateUncertainty;

const RateUncertainty UnknownRateUncertainty = 0xFFFFFFFF;
```

4 References

4.1. Referenced documents

[Ref1] *Principles for WInnForum Facility Standards*, The Wireless Innovation Forum, WINNF-TR-2007, V1.0.0, 9-Dec-2020
https://winnf.memberclicks.net/assets/work_products/reports/WINNF-TR-2007

[Ref2] *Joint Tactical Radio System Standard Timing Service Application Program Interface*, Joint Tactical Networking Center, Version 1.4.4, 26 June 2013
https://www.jtnc.mil/Resources-Catalog/Resource-Catalog-Article-View/Article/2084734/timingservice-api/

[Ref3] *Radio Services API Description Document*, ESSOR Architecture, 2019
http://www.occar.int/sites/default/files/downloads/Radio%20Services%20API%20Description%20Document.pdf

[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

[Ref5] *International Atomic Time (TAI)*, Wikipedia
https://en.wikipedia.org/wiki/International_Atomic_Time

[Ref6] *Coordinated Universal Time (UTC)*, Wikipedia
https://en.wikipedia.org/wiki/Coordinated_Universal_Time

[Ref7] *Transceiver Facility PIM Specification*, The Wireless Innovation Forum, WINNF-TS-0008, Version 2.0.0, 9 November 2017
https://sdsw.memberclicks.net/assets/docs/WINNF-TS-0008-V2.0.0%20Transceiver%20Facility%20PIM%20Specification.pdf

[Ref8] *Unix Time*, Wikipedia
https://en.wikipedia.org/wiki/Unix_time

[Ref9] *IDL Profiles for Platform-Independent Modeling of SDR Applications*, The Wireless Innovation Forum, WINNF-14-S-0016, Version 2.0.1, 12 June 2015
http://www.wirelessinnovation.org/assets/work_products/Specifications/winnf-14-s-0016-v1.0.0%20-%20pim%20idl%20profiles.zip

[Ref10] *Application Interface Definition Language Platform Independent Model Profiles, SCA 4.1 Appendix E-1*, Joint Tactical Networking Center, 20 August 2015
https://www.jtnc.mil/Resources-Catalog/Resource-Catalog-Article-View/Article/2083328/sca-41-appendix-e-1-application-idl-pim-profiles/

The provided URLs were successfully accessed at the release date of the specification.

5 Acronyms list

The following table lists the acronyms appearing in the *time service facility*:

Acronym	Signification
API	Application Programming Interface
ESSOR	European Secure Software Radio
FPGA	Field Programmable Gate Array
GNSS	Global Navigation Satellite System
GPP	General Purpose Processor
GPS	Global Positioning System
IDL	Interface Description Language
ITU	International Telecommunication Union
JTNC	Joint Tactial Networking Center
OMG	Object Management Group
PIM	Platform-Independent Model
POSIX	Portable Operating System Interface
PSM	Platform-Specific Model
RTOS	Real-Time Operating System
SCA	Software Communications Architecture
SVFuA	Streitkräftegemeinsame Verbundfähige Funkgeräte-Ausstattung
TAI	International Atomic Time
UTC	Coordinated Universal Time
WInnForum	Wireless Innovation Forum

Table 34 Acronyms list

6 Reference tables

The prefix "**TSF**", for "**T**ime **S**ervice **F**acility", is used for construction of identifiers.

6.1. Definitions

The following table lists the definitions specified by the *time service facility*:

Identifier	Concept	Meaning	Page
TSF.D01	*Time service capability*	A *functional support capability* of a *radio platform* that provides *radio applications* with knowledge of time.	8
TSF.D02	*Time service*	An instantiation of a *time service capability*.	8
TSF.D03	*Time service facility*	The WInnForum *facility* specified for *time services*.	8
TSF.D04	*Physical time*	The time physically elapsing within the *radio platform*.	13
TSF.D05	*Physical instant*	An infinitesimal moment of the *physical time*, whose passage is instantaneous.	13
TSF.D06	*Time value*	Value taken by a time at a certain *physical instant*.	13
TSF.D07	*Time function*	The mathematical function that relates, for any time notion measuring time, the taken *time values* to the *physical time*.	13
TSF.D08	*Standard times*	The *International Atomic Time* (*TAI*) and the *Universal Coordinated Time* (*UTC*)	13
TSF.D09	*International Atomic Time*	An ITU-standardized *physical time* measure built from an international network of permanently running reference atomic clocks.	13
TSF.D10	*Universal Coordinated Time*	The ITU-standardized measurement of time that adjusts the *TAI* using the concept of leap seconds (LS) to compensate the long term drifts in the apparent position of the sun.	14
TSF.D11	*Standard time source*	Any source of *standard time* supporting implementation of a *time service*.	15
TSF.D12	*Implemented time*	A time implemented by a *time service*.	15
TSF.D13	*Terminal time*	The *implemented time* that measures the time elapsing within the *time service*.	15
TSF.D14	*Terminal time rate error*	The relative rate error of the *terminal time* versus the *physical time*.	17
TSF.D15	*Terminal time rate uncertainty*	An upper bound of the absolute value of its *terminal time rate error*.	18
TSF.D16	*System time*	The *implemented time* that jointly estimates the *TAI* and the *UTC*.	19
TSF.D17	*System time update*	A point in time when the *time service* updates the *system time* in order to decrease its *time uncertainty*.	20
TSF.D18	*Specific time*	A monotonically increasing *implemented time* maintained as closely as possible to the *physical time* rate since it was last set.	22
TSF.D19	*Setting time*	The *physical time* value at which a *specific time* was last set.	22
TSF.D20	*Setting value*	The value to which a *specific time* was set to at a *setting time*.	22
TSF.D21	*Time stamp*	For a stamped instant, a *terminal time* value measured at a *physical time* close to when the stamped instant occurs.	24
TSF.D22	*Stamping uncertainty*	The maximum possible error between a stamped instant and the associated *time stamp*.	24

Identifier	Concept	Meaning	Page
TSF.D23	*Estimation uncertainty*	An upper bound of the absolute value of the difference between a time value and the related *time notion*, valid within a specified confidence percentage.	25
TSF.D24	*System time uncertainty*	An *estimation uncertainty,* at 95% confidence, between a time value of a *system time* and the *standard time* it estimates (TAI and/or UTC).	26
TSF.D25	*Specific time uncertainty*	An *estimation uncertainty*, at 95% confidence, between a time value of a *specific time* and the *specific time* that would have been maintained from the last *setting time* using *physical time*.	26
TSF.D27	*Time reference*	A triplet composed of a *time value* of interest, the associated *time stamp* and *time uncertainty*.	27
TSF.D28	*System time reference*	A *time reference* which *time value* of interest relates to *system time*.	28
TSF.D29	*Standard time reference*	A *time reference* which *time value* of interest relates to a *standard time*.	28
TSF.D30	*Specific time reference*	A *time reference* which *time value* of interest relates to a *specific time*.	28

Table 35 Specified definitions

6.2. Notations

The following table lists the notations specified by the *time service facility*:

Identifier	Notation	Denoted concept	Page
TSF.N01	t	Undefined *physical instant* of the *physical time*.	13
TSF.N02	$t_{<qualifier>}$	*Physical instant* characterized by the used <*qualifier*>.	13
TSF.N03	*TAI*	*International Atomic Time*	13
TSF.N04	TAI(t)	*Time function* of the *TAI*.	13
TSF.N05	*UTC*	*Universal Coordinated Time*	14
TSF.N06	UTC(t)	*Time function* of the *UTC*.	14
TSF.N07	TT(t)	Idealized *time function* of *terminal time*.	16
TSF.N08	TT[t]	Digitized *time function* of *terminal time*.	16
TSF.N09	t_{TTinit}	Initial instant from which *terminal time* is measured.	16
TSF.N10	v_{TTinit}	Initial value taken by *terminal time* at t_{TTinit}.	16
TSF.N11	*TTRE*	*Terminal time rate error*.	17
TSF.N12	*TTRU*	*Terminal time rate accuracy*.	18
TSF.N13	ST(t)	Idealized *time function* of a *system time*.	19
TSF.N14	ST[t]	Digitized *time function* of a *system time*.	19
TSF.N15	SpeT(t)	Idealized *time function* of a *specific time*.	22
TSF.N16	SpeT[t]	Digitized *time function* of a *specific time*.	23
TSF.N17	$t_{setting}$	*Setting time*.	23
TSF.N18	$v_{setting}$	*Setting value*.	23
TSF.N19	*TS*	*Time stamp*.	24
TSF.N20	*SU*	*Stamping uncertainty*.	24
TSF.N21	$t_{stamped}$	*Stamping instant*.	24
TSF.N22	*STU*	*System time uncertainty*.	26
TSF.N23	*SpeTU*	*Specific time uncertainty*.	26

Table 36 Specified notations

6.3. Equations

The following table lists the equations specified by the *time service facility*:

Identifier	Equation content	Page
TSF.E01	Mathematical definition of *terminal time rate error*.	17
TSF.E02	Mathematical definition of *terminal time rate uncertainty*.	18
TSF.E03	Mathematical definition of *stamping uncertainty*.	24
TSF.E04	Mathematical definition of *system time uncertainty*.	26
TSF.E05	Mathematical definition of *specific time uncertainty*.	27

Table 37 Specified equations

6.4. Requirements

The following table lists the requirements specified by the *time service facility*:

Identifier	Requirement clause	Page
TSF.R01	A *time service* **shall** present a *service implementation* for each mandatory *service*.	29
TSF.R02	A *time service* **shall** present a *service implementation* for each selected optional *service*.	29
TSF.R03	A *time service* **shall**, for each *attribute* for which a value is specified, comply with the related specified content.	63

Table 38 Specified requirements

END OF THE DOCUMENT