Transceiver Facility Native C++ PSM specification

Document WINNF-TS-0008-App01

Version V2.1.0

20 January 2022

TERMS, CONDITIONS & NOTICES

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 It may be amended or withdrawn at a later time and it is not binding on any member of the Forum or of the Transceiver Next work group.

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

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

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

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

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

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

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

Table of Contents

List of Figures

List of Tables

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,
- Claude Bélisle, NordiaSoft,
- Neal Buchmeyer, Collins,
- Jean-Philippe Delahaye, DGA,
- Guillaume Delbarre, DGA,
- David Hagood, Cynosure,
- Olivier Kirsch, KEREVAL,
- David Murotake, HiKE,
- Eric Nicollet, Thales,
- Kevin Richardson, MITRE,
- Dmitri Zvernick, NordiaSoft.

Transceiver Facility Native C++ PSM specification

1 Introduction

This document WINNF-TS-0008-A.1 is the *native C++ PSM specification* of the *Transceiver Facility* V2.1.0.

It derives from the *Transceiver Facility PIM Specification* [Ref1] in accordance with *Principles for WInnForum Facility Standards* [Ref2].

It addresses the Native C++ programming paradigm, applying the mapping rules of the Native C++ section of *WInnForum Facilities PSMs Mapping Rules* [Ref3] and specifically reporting any deviation to those rules.

The following figure positions the interfaces addressed by the *native C++ PSM specification*:

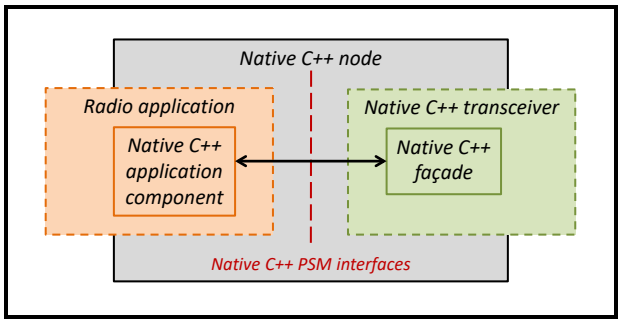

Figure 1 Positioning of *native C++ PSM interfaces*

As depicted, the *native C++ PSM specification* addresses the *native C++ PSM interfaces* of *transceivers*, positioned, within a *native C++ node*, between the *native C++ application components* of *radio applications* and *native C++ façades* of *native C++ transceivers*.

The *native C++ PSM specification* addresses two versions of the C++ language:

- C++03, specified by [Ref4], typically applicable for DSP (Digital Signal Processors) processing nodes where compilers do not support recent C++ language versions,
- C++11, specified by [Ref5], suited to state-of-the art programming environments.

It uses the abbreviation "xvcr" to identify *Transceiver Facility* in formal identifiers.

1.1 Reference definitions

The *Transceiver Facility native C++ PSM specification* applies the following definitions from *Transceiver Facility PIM Specification* [Ref1]:

Topic	Used definitions
Transceiver technical concepts	*transceiver*, *Transceiver Facility*, *Tx channel*, *Rx channel*

Table 1 Definitions from *Transceiver Facility PIM Specification*

The *Transceiver Facility native C++ PSM specification* applies the following definitions from *Principles for WInnForum Facility Standards* [Ref2]:

Topic	Applied definitions
Base concepts	*radio application*, *radio platform*
Architecture concepts	*application component, processing node*, *façade*
WInnForum facility	*facility, PIM specification*, *PSM specification*
Services	*service*, *service interface, provide service*, *use service, services group*
Primitives	*primitive*, *parameter*, *type*, *exception*
Attributes	*attribute*, *property*

Table 2 Definitions from *Principles for WInnForum Facility Standards*

The *Transceiver Facility native C++ PSM specification* applies the following definitions from the Native C++ section of *WInnForum Facilities PSMs Mapping Rules* [Ref3]:

Topic	Used definitions
Specification purpose	*native C++ PSM specification*, *native C++ PSM interface*
Software architecture	*native C++ application component*, *native C++ node*, *native C++ façade*

Table 3 Definitions from Native C++ section of *WInnForum Facilities PSMs Mapping Rules*

The term "*unspecified"* indicates an aspect explicitly left to implementer's decisions.

1.2 Conformance

1.1.1 Radio platform items

A *native C++ façade* of a *transceiver* implementation **is conformant with** the *Transceiver Facility native C++ PSM specification* if it provides an implementation of the `Facade` class and its related *service interfaces*.

A *native C++ transceiver* **is defined as** a *transceiver* implementation with all of its *native C++ façades* being conformant with the *native C++ PSM specification*.

1.1.2 Radio application items

A *native C++ application component* **is conformant with** the *Transceiver Facility native C++ PSM specification* if it can use *native C++ façades* conformant with the *native C++ PSM specification*, without using any non-standard *service interface* for the *transceiver*.

1.3 Document structure

Section 2 specifies the normative classes for the *native C++ PSM interfaces*.

Section 3 specifies the header files to be used by *native C++ transceivers*.

2 Native C++ PSM classes

This normative section specifies the C++ classes for *native C++ PSM interfaces*.

2.1 Provide and Use Services Interfaces

The *service interfaces* for the *provide services* of the *native C++ PSM interfaces* **are specified by** the following table:

PIM *service interface* (in `Transceiver::`)	PIM section	Native C++ PSM *service interface* (in `WInnF_Cpp::Transceiver::`)
`Management::Reset`	2.4.1.1	`Management::Reset`
`Management::RadioSilence`	2.4.1.2	`Management::RadioSilence`
`BurstControl::DirectCreation`	2.4.2.1	`BurstControl::DirectCreation`
`BurstControl::RelativeCreation`	2.4.2.2	`BurstControl::RelativeCreation`
`BurstControl::AbsoluteCreation`	2.4.2.3	`BurstControl::AbsoluteCreation`
`BurstControl::StrobedCreation`	2.4.2.4	`BurstControl::StrobedCreation`
`BurstControl::Termination`	2.4.2.5	`BurstControl::Termination`
`BasebandSignal::SamplesTransmission`	2.4.3.2	`BasebandSignal::SamplesTransmission`
`BasebandSignal::RxPacketsLengthControl`	2.4.3.3	`BasebandSignal::RxPacketsLengthControl`
`Tuning::InitialTuning`	2.4.4.1	`Tuning::InitialTuning`
`Tuning::Retuning`	2.4.4.2	`Tuning::Retuning`
`GainControl::GainLocking`	2.4.6.2	`GainControl::GainLocking`
`TransceiverTime::TimeAccess`	2.4.7.1	`TransceiverTime::TimeAccess`
`Strobing::ApplicationStrobe`	2.4.8.1	`Strobing::ApplicationStrobe`

Table 4 Provide services mapping

The *service interfaces* for the *use services* of the *native C++ PSM interfaces* **are specified by** the following table:

PIM *service interface* (in `Transceiver::`)	PIM section	Native C++ PSM *service interface* (in `WInnF_Cpp::Transceiver::`)
`BasebandSignal::SamplesReception`	2.4.3.1	`BasebandSignal::SamplesReception`
`Notifications::Events`	2.4.5.1	`Notifications::Events`
`Notifications::Errors`	2.4.5.2	`Notifications::Errors`
`GainControl::GainChanges`	2.4.6.1	`GainControl::GainChanges`

Table 5 Use services mapping

2.2 Facade classes

The `WInnF_Cpp::Transceiver::TxFacade` class **is specified as** the class dedicated to *Tx channels*, derived from the `Facade` class of the Native C++ section of *WInnForum Facilities PSMs Mapping Rules* [Ref3].

The **WInnF_Cpp::Transceiver::RxFacade** class **is specified as** the class dedicated to *Rx channels*, derived from the **Facade** class of the Native C++ section of *WInnForum Facilities PSMs Mapping Rules* [Ref3].

Since a *transceiver* features a **CONFIGURED** state, the **Facade** class features the methods *activeServicesInitialized()* and *activeServicesReleased()*.

Two classes are available for *active services access*, as specified by [Ref3]:

- **ExplicitServicesAccess**,
- **GenericServicesAccess**.

Usage of those classes by *façades* is controlled by the preprocessing variables EXPLICIT_SERVICES_ACCESS and GENERIC_SERVICES_ACCESS.

A *façade* may implement the two possibilities, in which case the two preprocessing variables are set.

Typical possibilities for the **TxFacade** and **RxFacade** classes are represented by the following class diagram:

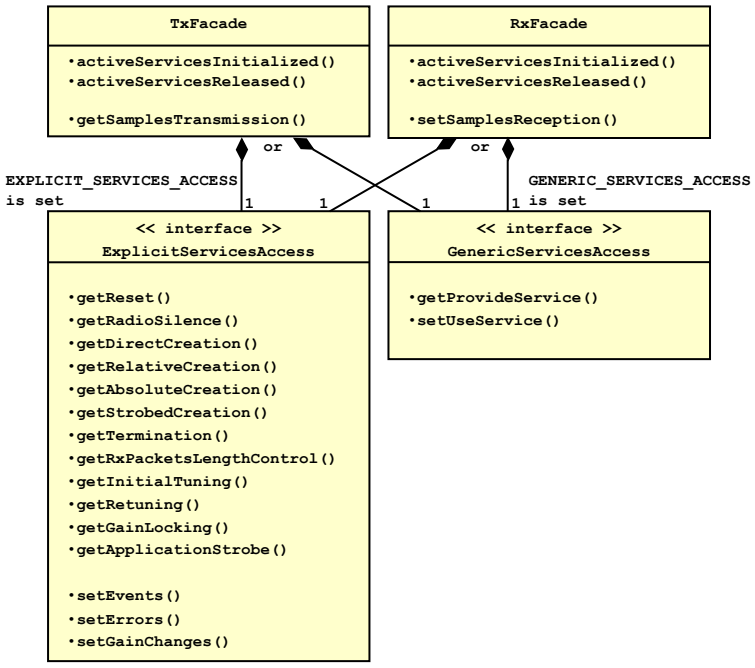

Figure 2 Class diagram of Native C++ Facades

The lifecycle of a **Façade** instance is greater than the lifecycle of the *application component* using the *transceiver*. Other aspects are *unspecified*.

2.2.1 TxFacade

TxFacade owns the *transceiver*-specific *getSamplesTransmission()* primitive.

2.2.1.1 *getSamplesTransmission* method

2.2.1.1.1 Overview

getSamplesTransmission() returns a pointer referencing the instance of **`SamplesTransmission`** interface implemented by the *native C++ façade* for the required *Tx channel*.

2.2.1.1.2 Associated properties

Not applicable.

2.2.1.1.3 Declaration

The C++ signature of the method **is specified as**:

```
Transceiver::BasebandSignal::SamplesTransmission* getSamplesTransmission(
      uint16_t channelNumber);
```

2.2.1.1.4 Parameters

Name	Type	Description
channelNumber	**`uint16_t`**	Number of the required *Tx channel*.

Table 6 *getSamplesTransmission()* parameters

The parameters validity ranges are:

- For *channelNumber*: **`1`** to **`TX_CHANNELS`**.

2.2.1.1.5 Returned value

If an instance is available, the method returns a **`Transceiver::BasebandSignal::SamplesTransmission*`** pointer to the *service interface* for the required *Tx channel*.

Otherwise, the function returns **`NULL`**.

2.2.1.1.6 Originator

Native C++ application component.

2.2.1.1.7 Exceptions

The *exceptions* (see section 2.2.3) of the method **are specified as**:

- **`MaxChannelNumberException`**,
- **`UnavailableServiceException`**.

2.2.1.1.8 Behavior requirements

A *native C++ façade* needs to, on a call to *getSamplesTransmission()*:

- If no instance of the **`SamplesTransmission`** *provide service* is available:
  - If *exception*s are supported, throw **`UnavailableServiceException`**,
  - If *exception*s are not supported, set the return value to **`NULL`**,
- If the **`SamplesTransmission`** *provide service* is available, set the return value with a reference to its *service interface*,
- Return the call,
- Keep the *provide service* active at least until the instance of the *transceiver* implementation has exited the **`CONFIGURED`** state.

2.2.2 RxFacade

`RxFacade` owns the *transceiver*-specific *setSamplesReception()* primitive.

2.2.2.1 *setSamplesReception* method

2.2.2.1.1 Overview

setSamplesReception() passes a pointer used by the *façade* to reference the instance of **`SamplesReception`** implemented by the *native C++ application component* for the required *Rx channel*.

2.2.2.1.2 Associated properties

Not applicable.

2.2.2.1.3 Declaration

The C++ signature of the method **is specified as**:

```
void setSamplesReception(
      Transceiver::BasebandSignal::SamplesReception* reference,
      uint16_t channelNumber);
```

2.2.2.1.4 Parameters

Name	Type	Description
reference	`Transceiver::BasebandSignal::SamplesReception*`	Reference of a `SamplesReception` *service interface* implemented by the *application component*.
channelNumber	`uint16_t`	Number of the required *Rx channel*.

Table 7 Specification of *setSamplesReception()* parameters

The parameters validity ranges are:

- For *channelNumber*: `1` to `RX_CHANNELS`.

2.2.2.1.5 Returned value

None.

2.2.2.1.6 Originator

Native C++ application component.

2.2.2.1.7 Exceptions

The *exceptions* (see section 2.2.3) of the method **are specified as**:

- `MaxChannelNumberException`,
- `UnavailableServiceException`.

2.2.2.1.8 Behavior requirements

A *native C++ façade* needs to, on a call to a *setSamplesReception()*:

- If no instance of the `SamplesReception` *use service* is available:
  - If *exceptions* are supported, throw `UnavailableServiceException`.
- If the `SamplesReception` *use service* is available, use *reference* to set a pointer of the *native C++ façade* to reference the *service interface* implemented by the *native C++ application component*,
- Return the call,
- Use the referenced *service interface* until the instance of the *transceiver* exits the `CONFIGURED` state.

Note: this implies that the `SamplesReception` *use service* of the *native C++ application component* remains valid until the *transceiver* exits the `CONFIGURED` state.

2.2.3 Facade exceptions

The exceptions associated to *native C++ façades* **are specified by** the following table:

Exception	Standard C++ parent	Condition
`MaxChannelNumberException`	`std::range_error`	The value of *channelNumber* parameter exceeds `TX_CHANNELS` or `RX_CHANNELS`.
`UnavailableServiceException`	`std::range_error`	The accessed *service* is not available.

Table 8 *Façades* exceptions

3 Native C++ PSM header files

This normative section specifies the standard C++ header files (`.hpp`) to be used by conformant (see section 1.2) *native C++ transceivers* and *native C++ application components*.

Conformant *native C++ façades* and *native C++ application components* have only to include `XcvrFacades.hpp`.

`XcvrPlatformProviderAdaptations.hpp` captures the adaptations decided by *native C++ transceivers* implementers.

The supported C++ versions are C++03 and C++11. The standard macro `__cplusplus` indicates the supported C++ version.

The specified header files have been successfully compiled using:

- GCC 4.8.5 and 7.5.0 for C++ 98, C++ 11, C++14 and C++17.

The used code formatting rules are specified by the Native C++ section of *WInnForum Facilities PSMs Mapping Rules* [Ref3].

3.1 <cstdint> (stdint.h)

The `<cstdint> (stdint.h)` library is used for C++ standard declarations for basic types.

3.2 XcvrFacades.hpp

The `XcvrFacades.hpp` file **is specified as** the header file that declares the `Facade` classes.

The content of **`XcvrFacades.hpp`** **is specified as**:

```
#ifndef XCVR_FACADES
#define XCVR_FACADES

#include "XcvrPlatformProviderAdaptations.hpp"
#include "XcvrServices.hpp"

#ifdef EXPLICIT_SERVICES_ACCESS
   #include "XcvrExplicitServicesAccess.hpp"
#endif

#ifdef GENERIC_SERVICES_ACCESS
   #include "XcvrGenericServicesAccess.hpp"
#endif

namespace WInnF_Cpp
{
  namespace Transceiver
  {

    // TxFacade (section 2.2.1 of [PSM])
    class TxFacade
    {
     public:
#ifdef EXPLICIT_SERVICES_ACCESS
       virtual ActiveServicesAccess::ExplicitServicesAccess *
         getExplicitServicesAccess() NOEXCEPT = 0;
#endif

#ifdef GENERIC_SERVICES_ACCESS
       virtual ActiveServicesAccess::GenericServicesAccess *
         getGenericServicesAccess() NOEXCEPT = 0;
#endif
       virtual void activeServicesInitialized()= 0;
       virtual void activeServicesReleased()= 0;

       virtual void getSamplesTransmission(
         uint16_t channelNumber) NOEXCEPT = 0;
       virtual ~TxFacade() NOEXCEPT {}
    };
     // RxFacade (section 2.2.1 of [PSM])
    class RxFacade
    {
     public:
#ifdef EXPLICIT_SERVICES_ACCESS
       virtual ActiveServicesAccess::ExplicitServicesAccess *
         getExplicitServicesAccess() NOEXCEPT = 0;
#endif

#ifdef GENERIC_SERVICES_ACCESS
       virtual ActiveServicesAccess::GenericServicesAccess *
         getGenericServicesAccess() NOEXCEPT = 0;
#endif

       virtual void activeServicesInitialized()= 0;
       virtual void activeServicesReleased()= 0;

       virtual void setSamplesReception(
         Transceiver::BasebandSignal::SamplesReception* const reference,
         uint16_t channelNumber) NOEXCEPT = 0;
       virtual ~RxFacade() NOEXCEPT {}
     };
```

```
  } // namespace Transceiver
} // namespace WInnF_Cpp

#endif // ifndef XCVR_FACADES
```

3.3 XcvrPlatformProviderAdaptations.hpp

The **`XcvrPlatformProviderAdaptations.hpp`** file **is specified as** the header file specifying all user adaptations:

- *Interface declaration properties*, specified by section 4.5 of the *Transceiver Facility PIM Specification* [Ref1],
- Tx or Rx metadata, if needed,
- Preprocessing variables for *active services access.*

`XcvrPlatformProviderAdaptations.hpp` has to be used with user adaptations at areas marked with *** PROVIDER-ADAPTATION ***.

The content of **`XcvrPlatformProviderAdaptations.hpp` is specified as**:

```
// Specifies user-adaptations

#ifndef XCVR_USER_ADAPTATIONS
#define XCVR_USER_ADAPTATIONS

// Active Services Access and Exceptions Support
// *** PROVIDER-ADAPTATION ***
#define EXPLICIT_SERVICES_ACCESS // comment if EXPLICIT is not supported
#define GENERIC_SERVICES_ACCESS  // comment if GENERIC is not supported
#define EXCEPTIONS_SUPPORT       // comment if EXCEPTIONS is not supported

// Interface declaration properties
namespace WInnF_Cpp
{
  namespace Transceiver
  {
    // PIM enumerated identifiers : int16, int32, int64, float32
    // Datatype_ prefix for disambiguation vs. <cstdint> types
    enum{ Datatype_int16, Datatype_int32, Datatype_int64, Datatype_float32};

    // *** PROVIDER-ADAPTATION *** CARRIER_FREQ_TYPE value
    const int XCVR_CARRIER_FREQ_TYPE = Datatype_int32; // Specify
"Datatype_int32" or "Datatype_int64"

    // *** PROVIDER-ADAPTATION *** DELAY_TYPE value
    const int XCVR_DELAY_TYPE = Datatype_int32; // Specify "Datatype_int32"
or "Datatype_int64"

    // *** PROVIDER-ADAPTATION *** IQ_TYPE value
    const int XCVR_IQ_TYPE = Datatype_int32;  // Specify "Datatype_int16",
"Datatype_int32" or " Datatype_float32"
  }
}

// TX_META_DATA and RX_META_DATA values
// *** PROVIDER-ADAPTATION *** XCVR_TX_META_DATA value
// #define XCVR_TX_META_DATA FALSE // Uncomment if FALSE
#if XCVR_TX_META_DATA == TRUE
namespace WInnF_Cpp
{
  namespace Transceiver
  {
    typedef struct TxMetaData
    {
      // *** PROVIDER-ADAPTATION ***
      // Specify declaration of Tx metadata fields
    };
  }
}
#endif // #if XCVR_TX_META_DATA == TRUE
```

```
// *** PROVIDER-ADAPTION *** XCVR_RX_META_DATA value
// #define XCVR_RX_META_DATA FALSE // Uncomment if FALSE
#if XCVR_RX_META_DATA == TRUE
namespace WInnF_Cpp
{
  namespace Transceiver
  {
    typedef struct RxMetaData
    {
      // *** USER-ADAPTATION ***
      // Specify declaration of Rx metadata fields
    };
  }
}
#endif // #if XCVR_RX_META_DATA == TRUE

#endif // #ifndef XCVR_USER_ADAPTATIONS
```

3.4 XcvrTypes.hpp

The **`XcvrTypes.hpp`** file **is specified as** the header file that declares the *API types* specified by section 3.4 of the *Transceiver Facility PIM Specification* [Ref1].

The content of **`XcvrTypes.hpp` is specified as**:

```
#ifndef XCVR_TYPES
#define XCVR_TYPES

#if __cplusplus >= 201103L // C++2011
  #include <cstdint>
#else
  #include <stdint.h>
#endif

#include "XcvrPlatformProviderAdaptations.hpp"

#if __cplusplus >= 201103L // C++2011
  #include <vector>
#endif // #if __cplusplus >= 201103L

// Support of exceptions (section 3.2 of [PIM])
#ifdef EXCEPTIONS_SUPPORT
  #include "XcvrExceptions.hpp"
#else
// If exceptions are not supported, make sure NOEXCEPT is defined
   #ifndef NOEXCEPT
      #if __cplusplus < 201103L
         #define NOEXCEPT
      #else
         #define NOEXCEPT noexcept
      #endif
   #endif
#endif

namespace WInnF_Cpp
{
  namespace Transceiver
  {
    // Optional types templated declarations
    template<int I> struct OptionalTypes{};
    template<> struct OptionalTypes<Datatype_int16>
    {
      typedef uint16_t IQ;
    };
    template<> struct OptionalTypes<Datatype_int32>
    {
      typedef uint32_t IQ;
      typedef uint32_t Delay;
      typedef uint32_t CarrierFreq;
    };
    template<> struct OptionalTypes<Datatype_int64>
    {
      typedef uint64_t Delay;
      typedef uint64_t CarrierFreq;
    };
    template<> struct OptionalTypes<Datatype_float32>
    {
      typedef float IQ;
    };

    // IQ (section 3.4.11 of [PIM])
    typedef OptionalTypes<XCVR_IQ_TYPE>::IQ IQ;

    // BasebandSample (section 3.4.4 of [PIM])
    struct BasebandSample {IQ valueI; IQ valueQ;};
```

```
    // BasebandPacket (section 3.4.2 of [PIM])
    #if __cplusplus == 199711L // C++1998/2003
    // IDL to C++ mapping of "typedef sequence <BasebandSample>
BasebandPacket;"
    class BasebandPacket
    {
      public:
      BasebandPacket();
      BasebandPacket( uint32_t max);
      BasebandPacket( uint32_t max, uint32_t length, BasebandSample *bbs,
bool release = false);
      ~ BasebandPacket();
      BasebandPacket &operator =(const BasebandPacket &);
      uint32_t maximum() const;
      void length( uint32_t);
      uint32_t length() const;
      BasebandSample &operator []( uint32_t index);
      const BasebandSample &operator []( uint32_t index) const;
      bool release() const;
      void replace( uint32_t max, uint32_t length, BasebandSample *bbs, bool
release = false);
      BasebandSample *get_buffer( bool orphan = false);
      const BasebandSample *get_buffer() const;
      static BasebandSample *allocbuff( uint32_t nelems);
      static void freebuff(BasebandSample *);
    };
    #elif __cplusplus>= 201103L // C++2011
      typedef std::vector<BasebandSample> BasebandPacket;
    #endif // #if __cplusplus == 199711L // C++1998/2003

    // BlockLength (section 3.4.3 of [PIM])
    typedef uint32_t BlockLength;
    const BlockLength UndefinedBlockLength = 0xFFFFFFFF;

    // BurstNumber (section 3.4.5 of [PIM])
    typedef uint32_t BurstNumber;

    // CarrierFreq (section 3.4.6 of [PIM]), in Hz
    typedef OptionalTypes<XCVR_CARRIER_FREQ_TYPE>::CarrierFreq CarrierFreq;

    #if XCVR_CARRIER_FREQ_TYPE == int32
      const CarrierFreq UndefinedCarrierFreq = 0xFFFFFFFF;
    #elif XCVR_CARRIR_FREQ_TYPE == int64
      const CarrierFreq UndefinedCarrierFreq = 0x7FFFFFFFFFFFFFFF;
    #else
      #error const XCVR_CARRIER_FREQ_TYPE must be equal to "int32" or "int64"
    #endif // #if XCVR_CARRIER_FREQ_TYPE == int32

    // Delay (section 3.4.7 of [PIM]), in ns
    typedef OptionalTypes<XCVR_DELAY_TYPE>::Delay Delay;

    #if XCVR_DELAY_TYPE == int32
      const Delay UndefinedDelay = 0xFFFFFFFF;
    #elif XCVR_DELAY_TYPE == int64
      const Delay UndefinedDelay = 0x7FFFFFFFFFFFFFFF;
    #else
      #error cont XCVR_DELAY_TYPE must be equal to "int32" or "int64"
    #endif // #if XCVR_DELAY_TYPE == int32

    // Error (section 3.4.8 of [PIM])
    #if __cplusplus == 199711L // C++1998/2003
    enum Error
    {
```

```
#elif __cplusplus >= 201103L // C++2011
enum class Error
{
#endif // #if __cplusplus == 199711L // C++1998/2003
  DelayedTuningError,
  TuningTimeoutError,
  DelayedFirstSampleError,
  FirstSampleTimeoutError,
  TransmissionUnderflowError,
  ReceptionOverflowError,
  ShorterTransmittedBlockError,
  LongerTransmittedBlockError,
  DelayedTuning,
  TuningTimeout,
  DelayedFirstSample,
  FirstSampleTimeout,
  TransmissionUnderflow,
  ReceptionOverflow,
  ShorterTransmittedBlock,
  LongerTransmittedBlock
};

// Event (section 3.4.9 of [PIM])
#if __cplusplus == 199711L // C++1998/2003
enum Event
{
#elif __cplusplus >= 201103L // C++2011
enum class Event
{
#endif // #if __cplusplus == 199711L // C++1998/2003
  ProcessingStartEvent,
  ProcessingStopEvent,
  SilenceStartEvent,
  SilenceStopEvent
  };

// Gain (section 3.4.10 of [PIM]), in 1/10 dB
typedef int16_t Gain;
const Gain UndefinedGain = 0xFFFF;

// MetaData (section 3.4.12 of [PIM])
// Declared in XcvrUserAdaptations.hpp

// PacketLength (section 3.4.13 of [PIM])
typedef uint32_t PacketLength;

// SampleNumber (section 3.4.14 of [PIM])
typedef uint32_t SampleNumber;

// StrobeSource (section 3.4.15 of [PIM])
enum StrobeSource
{
  ApplicationStrobe,
  TimeRef_PPS,
  GNSS_PPS,
  UserStrobe1,
  UserStrobe2,
  UserStrobe3,
  UserStrobe4
};

// TimeSpec (section 3.4.16 of [PIM])
struct TimeSpec
```

```
    {
      uint32_t seconds;             // in seconds
      uint32_t nanoseconds;
    };       // in nanoseconds (<1.000.000.000)
    const TimeSpec UndefinedTimeSpec = {0xFFFFFFFF, 0xFFFFFFFF};

    // TuningPreset (section 3.4.17 of [PIM])
    typedef uint16_t TuningPreset;
    const TuningPreset UndefinedTuningPreset = 0xFFFF;
  }
}

#endif // #ifndef XCVR_TYPES
```

3.5 XcvrServices.hpp

The **XcvrServices.hpp** file **is specified as** the header file that declares the *service interfaces* specified by section 3.1 of the *Transceiver Facility PIM Specification* [Ref1].

It declares **XCVR_PIM_VERSION** in accordance with section "Referenced PIM version" of the Native C++ section of *WInnForum Facilities PSMs Mapping Rules* [Ref3].

The content of **XcvrServices.hpp** is specified as:

```
#ifndef XCVR_SERVICES
#define XCVR_SERVICES

#if __cplusplus < 199711L
  #error C++ version must be C++98/2003 or later
#endif

// XCVR_PIM_VERSION property
#define XCVR_PIM_VERSION 0x020100L // For V2.1.0

#include "XcvrTypes.hpp"

namespace WInnF_Cpp
{
  namespace Transceiver
  {
    namespace Management
    {
      // Management::Reset (section 2.4.1.1 of [PIM])
      class Reset
      {
       public:
         virtual void reset() = 0;
         virtual ~Reset() NOEXCEPT {}
      };

      // Management::RadioSilence (section 2.4.1.2 of [PIM])
      class RadioSilence
      {
       public:
         virtual void startRadioSilence() = 0;
         virtual void stopRadioSilence() = 0;
         virtual ~RadioSilence() NOEXCEPT {}
      };
    }

    namespace BurstControl
    {
      // BurstControl::DirectCreation (section 2.4.2.1 of [PIM])
      class DirectCreation
      {
       public:
         virtual void startBurst(
           BlockLength requestedLength) = 0;
         virtual ~DirectCreation() NOEXCEPT {}
      };

      // BurstControl::RelativeCreation (section 2.4.2.2 of [PIM])
      class RelativeCreation
      {
       public:
         virtual void scheduleRelativeBurst(
```

```
        bool requestedAlternate,
        Delay requestedDelay,
        BlockLength requestedLength) = 0;
      virtual ~RelativeCreation() NOEXCEPT {}
  };

  // BurstControl::AbsoluteCreation (section 2.4.2.3 of [PIM])
  class AbsoluteCreation
  {
   public:
    virtual void scheduleAbsoluteBurst(
      TimeSpec requestedStartTime,
      BlockLength requestedLength) = 0;
    virtual ~AbsoluteCreation() NOEXCEPT {}
  };

  // BurstControl::StrobedCreation (section 2.4.2.4 of [PIM])
  class StrobedCreation
  {
   public:
    virtual void scheduleStrobedBurst(
      StrobeSource requestedStrobeSource,
      Delay requestedDelay,
      BlockLength requestedLength) = 0;
      virtual ~StrobedCreation() NOEXCEPT {}
  };

  // BurstControl::Termination (section 2.4.2.5 of [PIM])
  class Termination
  {
   public:
    virtual void setBlockLength(
      BlockLength requestedLength) = 0;
    virtual void stopBurst() = 0;
    virtual ~Termination() NOEXCEPT {}
  };
}

namespace BasebandSignal
{
  // BasebandSignal::SamplesReception (section 2.4.3.1 of [PIM])
  class SamplesReception
  {
   public:
   #if XCVR_RX_META_DATA == TRUE
    virtual void pushRxPacket(
      BasebandPacket rxPacket,
      bool endOfBlock,
      RxMetaData rxMetaData) = 0;
   #else
    virtual void pushRxPacket(
      BasebandPacket rxPacket,
      bool endOfBlock) = 0;
   #endif // #if XCVR_TX_META_DATA == TRUE
    virtual ~SamplesReception() NOEXCEPT {}
};
```

```
    // BasebandSignal::SamplesTransmission (section 2.4.3.2 of [PIM])
    class SamplesTransmission
    {
     public:
     #if TX_META_DATA == TRUE
       virtual void pushTxPacket(
         BasebandPacket txPacket,
         bool endOfBlock,
         TxMetaData txMetaData) = 0;
     #else
       virtual void pushTxPacket(
         BasebandPacket txPacket,
         bool endOfBlock) = 0;
     #endif // #if TX_META_DATA == TRUE
       virtual ~SamplesTransmission() NOEXCEPT {}
    };

    // BasebandSignal::RxPacketsLengthControl (section 2.4.3.3 of [PIM])
    class RxPacketsLengthControl
    {
     public:
       virtual void setRxPacketsLength(
         PacketLength requestedLength) = 0;
       virtual ~RxPacketsLengthControl() NOEXCEPT {}
    };
  }

  namespace Tuning
  {
    // Tuning::InitialTuning (section 2.4.4.1 of [PIM])
    class InitialTuning
    {
     public:
       virtual void setTuning(
         TuningPreset requestedPreset,
         CarrierFreq requestedFrequency,
         Gain requestedGain,
         BurstNumber requestedBurstNumber) = 0;
       virtual ~InitialTuning() NOEXCEPT {}
    };

    // Tuning::Retuning (section 2.4.4.2 of [PIM])
    class Retuning
    {
     public:
       virtual void retune(
         CarrierFreq requestedFrequency,
         Gain requestedGain,
         Delay requestedDelay) = 0;
       virtual ~Retuning() NOEXCEPT {}
    };
  }
```

```
namespace Notifications
{
  // Notifications::Events (section 2.4.5.1 of [PIM])
  class Events
  {
   public:
     virtual void notifyEvent(
       Event notifiedEvent) = 0;
     virtual ~Events() NOEXCEPT {}
  };

  // Notifications::Errors (section 2.4.5.2 of [PIM])
  class Errors
  {
   public:
     virtual void notifyError(
       Error notifiedError) = 0;
     virtual ~Errors() NOEXCEPT {}
  };
}

namespace GainControl
{
  // GainControl::GainChanges (section 2.4.6.1 of [PIM])
  class GainChanges
  {
   public:
     virtual void indicateGain(
      Gain newGain,
      SampleNumber firstValidSample) = 0;
     virtual ~GainChanges() NOEXCEPT {}
  };

  // GainControl::GainLocking (section 2.4.6.2 of [PIM])
  class GainLocking
  {
   public:
     virtual void lockGain() = 0;
     virtual void unlockGain() = 0;
     virtual ~GainLocking() NOEXCEPT {}
  };
}

namespace TransceiverTime
{
  // TransceiverTime::TimeAccess (section 2.4.7.1 of [PIM])
  class TimeAccess
  {
   public:
     virtual void getCurrentTime(
       TimeSpec* currentTime) const = 0;
     virtual void getLastStartTime(
       TimeSpec* lastStartTime,
       BurstNumber* lastBurstNumber) const = 0;
     virtual ~TimeAccess() NOEXCEPT {}
  };
}
```

```
    namespace Strobing
    {
      // Strobing::ApplicationStrobe (section 2.4.8.1 of [PIM])
      class ApplicationStrobe
      {
       public:
         virtual void triggerStrobe(void);
         virtual ~ApplicationStrobe() NOEXCEPT {}
      };
    }
  }
}

#endif // ifndef XCVR_SERVICES
```

3.6 XcvrExplicitServicesAccess.hpp

The content of **`XcvrExplicitServicesAccess.hpp`** **is specified as**:

```
#ifndef XCVR_EXPLICIT_SERVICES_ACCESS
#define XCVR_EXPLICIT_SERVICES_ACCESS

namespace WInnF_Cpp
{
  namespace Transceiver
  {
    namespace ActiveServicesAccess
    {
      // Access to specific active services
      class ExplicitServicesAccess
      {
       public:
         // Provide services
         // Management services group
         virtual Transceiver::Management::Reset* getReset() = 0;
         virtual Transceiver::Management::RadioSilence* getRadioSilence() =
0;

         // BurstControl services group
         virtual Transceiver::BurstControl::DirectCreation*
getDirectCreation()
            = 0;
         virtual Transceiver::BurstControl::RelativeCreation*
getRelativeCreation() = 0;
         virtual Transceiver::BurstControl::AbsoluteCreation*
getAbsoluteCreation() = 0;
         virtual Transceiver::BurstControl::StrobedCreation*
getStrobedCreation() = 0;
         virtual Transceiver::BurstControl::Termination* getTermination() =
0;

         // BasebandSignal services group
         virtual Transceiver::BasebandSignal::RxPacketsLengthControl*
getRxPacketsLengthControl() = 0;

         // Tuning services group
         virtual Transceiver::Tuning::InitialTuning* getInitialTuning() = 0;
         virtual Transceiver::Tuning::Retuning* getRetuning() = 0;

         // GainControl services group
         virtual Transceiver::GainControl::GainLocking* getGainLocking() = 0;
```

```
        // Strobing services group
        virtual Transceiver::Strobing::ApplicationStrobe*
getApplicationStrobe() = 0;

        // Use services
        // Notifications services group
        virtual void setEvents( Transceiver::Notifications::Events*
reference)
          = 0;
        virtual void setErrors( Transceiver::Notifications::Errors*
reference)
          = 0;

        // GainControl services group
        virtual void setGainChanges( Transceiver::GainControl::GainChanges*
reference) = 0;
        virtual ~ExplicitServicesAccess() NOEXCEPT {}
      };
    } // namespace ActiveServicesAccess
  } // namespace Transceiver
} // namespace WInnF_Cpp
#endif // ifndef XCVR_EXPLICIT_SERVICES_ACCESS
```

3.7 XcvrGenericServicesAccess.hpp

The content of **`XcvrGenericServicesAccess.hpp`** **is specified as**:

```
#ifndef XCVR_GENERIC_SERVICES_ACCESS
#define XCVR_GENERIC_SERVICES_ACCESS

#include "XcvrTypes.hpp"

namespace WInnF_Cpp
{
  namespace Transceiver
  {
    namespace ActiveServicesAccess
    {
      class Object
      {
       public:
        virtual ~Object() NOEXCEPT {}
      };

      class GenericServicesAccess
      {
       public:
        // Access to active provide services
        virtual Object *getProvideService(
          const char *ServiceName) = 0;

        // Access to active use services
        virtual void setUseService(const char *ServiceName,
                                   Object *    provider) = 0;

        virtual ~GenericServicesAccess() NOEXCEPT {}
      };
    } // namespace ActiveServicesAccess
  } // namespace Transceiver
} // namespace WInnF_Cpp
#endif // ifndef XCVR_GENERIC_SERVICES_ACCESS
```

3.8 XcvrExceptions.hpp

The **XcvrExceptions.hpp** file **is specified as** the header file that declares all useable *exceptions*. **XcvrExceptions.hpp** is used if **EXCEPTION_USE** is equal to **true**.

The content of **XcvrExceptions.hpp** is specified as:

```
#ifndef XCVR_EXCEPTIONS
#define XCVR_EXCEPTIONS

#include <stdexcept>

#ifndef NOEXCEPT
   #if __cplusplus < 201103L
      #define NOEXCEPT
   #else
      #define NOEXCEPT noexcept
   #endif
#endif

// Exceptions identifiers (section 3.1.19 of [PIM])
namespace WInnF_Cpp
{
  namespace Transceiver
  {
    // General exceptions
    class NoAlternateReferencingException: public std::invalid_argument
    {
     public:
       NoAlternateReferencingException (char const *msg = "") NOEXCEPT
             : ::std::invalid_argument(msg)
       {
       }
       NoAlternateReferencingException(::std::string const &msg) NOEXCEPT
             : ::std::invalid_argument(msg)
       {
       }
    };

    class NoOngoingProcessingException: public std::runtime_error
    {
     public:
       NoOngoingProcessingException(char const *msg = "") NOEXCEPT
             : ::std::runtime_error(msg)
       {
       }
       NoOngoingProcessingException(::std::string const &msg) NOEXCEPT
             : ::std::runtime_error(msg)
       {
       }
    };

    class StrobeSourceException: public std::invalid_argument
    {
     public:
       StrobeSourceException(char const *msg = "") NOEXCEPT
             : ::std::invalid_argument(msg)
       {
       }
       StrobeSourceException(::std::string const &msg) NOEXCEPT
             : ::std::invalid_argument(msg)
       {
```

```
    }
};

// Range exceptions
class MinBlockLengthException: public std::range_error
{
 public:
   MinBlockLengthException(char const *msg = "") NOEXCEPT
         : ::std::range_error(msg)
   {
   }
   MinBlockLengthException(::std::string const &msg) NOEXCEPT
         : ::std::range_error(msg)
   {
   }
};

class MaxBlockLengthException : public std::range_error
{
 public:
   MaxBlockLengthException(char const *msg = "") NOEXCEPT
         : ::std::range_error(msg)
   {
   }
   MaxBlockLengthException(::std::string const &msg) NOEXCEPT
         : ::std::range_error(msg)
   {
   }
};

class MinCarrierFreqException : public std::range_error
{
 public:
   MinCarrierFreqException(char const *msg = "") NOEXCEPT
         : ::std::range_error(msg)
   {
   }
   MinCarrierFreqException(::std::string const &msg) NOEXCEPT
         : ::std::range_error(msg)
   {
   }
};

class MaxCarrierFreqException : public std::range_error
{
 public:
   MaxCarrierFreqException(char const *msg = "") NOEXCEPT
         : ::std::range_error(msg)
   {
   }
   MaxCarrierFreqException(::std::string const &msg) NOEXCEPT
         : ::std::range_error(msg)
   {
   }
};

class MinFromOngoingException: public std::range_error
{
 public:
   MinFromOngoingException(char const *msg = "") NOEXCEPT
         : ::std::range_error(msg)
   {
   }
```

```
    MinFromOngoingException(::std::string const &msg) NOEXCEPT
          : ::std::range_error(msg)
    {
    }
};

class MaxFromOngoingException: public std::range_error
{
 public:
    MaxFromOngoingException(char const *msg = "") NOEXCEPT
          : ::std::range_error(msg)
    {
    }
    MaxFromOngoingException(::std::string const &msg) NOEXCEPT
          : ::std::range_error(msg)
    {
    }
};

class MinFromPreviousException: public std::range_error
{
 public:
    MinFromPreviousException(char const *msg = "") NOEXCEPT
          : ::std::range_error(msg)
    {
    }
    MinFromPreviousException(::std::string const &msg) NOEXCEPT
          : ::std::range_error(msg)
    {
    }
};

class MaxFromPreviousException: public std::range_error
{
 public:
    MaxFromPreviousException(char const *msg = "") NOEXCEPT
          : ::std::range_error(msg)
    {
    }
    MaxFromPreviousException(::std::string const &msg) NOEXCEPT
          : ::std::range_error(msg)
    {
    }
};

class MinFromStrobeException: public std::range_error
{
 public:
    MinFromStrobeException(char const *msg = "") NOEXCEPT
          : ::std::range_error(msg)
    {
    }
    MinFromStrobeException(::std::string const &msg) NOEXCEPT
          : ::std::range_error(msg)
    {
    }
};

class MaxFromStrobeException: public std::range_error
{
 public:
    MaxFromStrobeException(char const *msg = "") NOEXCEPT
          : ::std::range_error(msg)
```

```
    {
    }
    MaxFromStrobeException(::std::string const &msg) NOEXCEPT
          : ::std::range_error(msg)
    {
    }
};

class MinGainException: public std::range_error
{
 public:
    MinGainException(char const *msg = "") NOEXCEPT
          : ::std::range_error(msg)
    {
    }
    MinGainException(::std::string const &msg) NOEXCEPT
          : ::std::range_error(msg)
    {
    }
};

class MaxGainException: public std::range_error
{
 public:
    MaxGainException(char const *msg = "") NOEXCEPT
          : ::std::range_error(msg)
    {
    }
    MaxGainException(::std::string const &msg) NOEXCEPT
          : ::std::range_error(msg)
    {
    }
};

class MaxNanoSecondsException: public std::range_error
{
 public:
    MaxNanoSecondsException(char const *msg = "") NOEXCEPT
          : ::std::range_error(msg)
    {
    }
    MaxNanoSecondsException(::std::string const &msg) NOEXCEPT
          : ::std::range_error(msg)
    {
    }
};

class MaxRxPacketsLengthException: public std::range_error
{
 public:
    MaxRxPacketsLengthException(char const *msg = "") NOEXCEPT
          : ::std::range_error(msg)
    {
    }
    MaxRxPacketsLengthException(::std::string const &msg) NOEXCEPT
          : ::std::range_error(msg)
    {
    }
};

class MaxTuningPresetException: public std::range_error    {
 public:
    MaxTuningPresetException(char const *msg = "") NOEXCEPT
```

```
            : ::std::range_error(msg)
      {
      }
      MaxTuningPresetException(::std::string const &msg) NOEXCEPT
            : ::std::range_error(msg)
      {
      }
};

class MaxTxPacketsLengthException: public std::range_error
{
 public:
      MaxTxPacketsLengthException(char const *msg = "") NOEXCEPT
            : ::std::range_error(msg)
      {
      }
      MaxTxPacketsLengthException(::std::string const &msg) NOEXCEPT
            : ::std::range_error(msg)
      {
      }
};

 // MILT exceptions
 class AbsoluteMILTException: public std::runtime_error
{
 public:
      AbsoluteMILTException(char const *msg = "") NOEXCEPT
            : ::std::runtime_error(msg)
      {
      }
      AbsoluteMILTException(::std::string const &msg) NOEXCEPT
            : ::std::runtime_error(msg)
      {
      }
};

 class RelativeMILTException: public std::runtime_error
{
 public:
      RelativeMILTException(char const *msg = "") NOEXCEPT
            : ::std::runtime_error(msg)
      {
      }
      RelativeMILTException(::std::string const &msg) NOEXCEPT
            : ::std::runtime_error(msg)
      {
      }
};

 class RetuningMILTException: public std::runtime_error
{
 public:
      RetuningMILTException(char const *msg = "") NOEXCEPT
            : ::std::runtime_error(msg)
      {
      }
      RetuningMILTException(::std::string const &msg) NOEXCEPT
            : ::std::runtime_error(msg)
      {
      }
};

 class Tuning_MILTException: public std::runtime_error
```

```
{
 public:
   Tuning_MILTException(char const *msg = "") NOEXCEPT
         : ::std::runtime_error(msg)
   {
   }
   Tuning_MILTException(::std::string const &msg) NOEXCEPT
         : ::std::runtime_error(msg)
   {
   }
};

 class TxPacketsMILTException: public std::runtime_error
{
 public:
   TxPacketsMILTException(char const *msg = "") NOEXCEPT
         : ::std::runtime_error(msg)
   {
   }
   TxPacketsMILTException(::std::string const &msg) NOEXCEPT
         : ::std::runtime_error(msg)
   {
   }
};

// Services access exceptions
class AlreadyConfiguredException: public std::runtime_error
{
 public:
   AlreadyConfiguredException(char const *msg = "") NOEXCEPT
       : ::std::runtime_error(msg)
   {
   }
   AlreadyConfiguredException(::std::string const &msg) NOEXCEPT
       : ::std::runtime_error(msg)
   {
   }
};

class MaxChannelNumberException: public std::range_error
{
 public:
   MaxChannelNumberException(char const *msg = "") NOEXCEPT
       : ::std::range_error(msg)
   {
   }
   MaxChannelNumberException(::std::string const &msg) NOEXCEPT
       : ::std::range_error(msg)
   {
   }
};

class UnavailableServiceException : public std::runtime_error
{
 public:
   UnavailableServiceException(char const *msg = "") NOEXCEPT
       : ::std::runtime_error(msg)
   {
   }
   UnavailableServiceException(::std::string const &msg) NOEXCEPT
       : ::std::runtime_error(msg)
   {
   }
```

```
    };

    class InvalidReferenceException : public std::runtime_error
    {
     public:
       InvalidReferenceException(char const *msg = "") NOEXCEPT
           : ::std::runtime_error(msg)
       {
       }
       InvalidReferenceException(::std::string const &msg) NOEXCEPT
           : ::std::runtime_error(msg)
       {
       }
    };
  } // namespace Transceiver
} // namespace WInnF_Cpp

#endif // ifndef XCVR_EXCEPTIONS
```

4 References

4.1 Referenced documents

[Ref1] *Transceiver Facility PIM Specification*, The Wireless Innovation Forum, WINNF-TS-0008 V2.1.0, 20 January 2022
https://sds.wirelessinnovation.org/specifications-and-recommendations
https://winnf.memberclicks.net/assets/work_products/Specifications/WINNF-TS-0008-V2.1.0.pdf

[Ref2] *Principles for WInnForum Facility Standards*, The Wireless Innovation Forum, WINNF-TR-2007 V1.0.0, 13 October 2020
https://sds.wirelessinnovation.org/specifications-and-recommendations
https://winnf.memberclicks.net/assets/work_products/Reports/WINNF-TR-2007-V1.0.0.pdf

[Ref3] *WInnForum Facilities PSMs Mapping Rules*, The Wireless Innovation Forum, WINNF-TR-2008, V1.0.1, 18 January 2022
https://sds.wirelessinnovation.org/specifications-and-recommendations
https://winnf.memberclicks.net/assets/work_products/Reports/WINNF-TR-2008-V1.0.1.pdf

[Ref4] *Information technology – Programming languages – C++*, ISO/IEC 14882:2003
http://www.cplusplus.com/

[Ref5] *Information technology – Programming languages – C++*, ISO/IEC 14882:2011
http://www.cplusplus.com/

The URLs above were successfully accessed at release date.

END OF THE DOCUMENT