libadm Documentation

Release 0.10.0

Institut für Rundfunktechnik GmbH

Mar 03, 2023

CONTENTS

CHAPTER

ONE

GETTING STARTED

1.1 Requirements and dependencies

The library aims to minimize dependencies to limit the integration work necessary to use it.

- compiler with C++11 support
- Boost header libraries (version 1.57 or later)
  - Boost.Optional
  - Boost.Variant
  - Boost.Range
  - Boost.Iterator
  - Boost.Functional
  - Boost.Format
- CMake build system (version 3.5 or later)

1.2 Installation

1.2.1 macOS

On macOS you can use homebrew to install the library. You just have to add the NGA homebrew tap and can then use the usual install command.

```
brew tap ebu/homebrew-nga
brew install libadm
```

1.2.2 Manual installation

To manually install the library you have to clone the git repository and then use the CMake build system to build and install it.

```
git clone git@github.com:ebu/libadm.git
cd libadm
mkdir build && cd build
cmake ..
make
make install
```

1.3 CMake

As the library uses CMake as a build system it is really easy to set up and use if your project does too. Assuming you have installed the library, the following code shows a complete CMake example to compile a program which uses the libadm.

```
cmake_minimum_required(VERSION 3.5)
project(libadm_example VERSION 1.0.0 LANGUAGES CXX)

find_package(adm REQUIRED)

add_executable(examples example.cpp)
target_link_libraries(example PRIVATE adm)
```

If you prefer not to install the library on your system you can also use the library as a subproject. You can just add the library as a CMake subproject. Just add the folder containing the repository to your project and you can use the `adm` target.

```
cmake_minimum_required(VERSION 3.5)
project(libadm_example VERSION 1.0.0 LANGUAGES CXX)

add_subdirectory(submodules/libadm)

add_executable(example example.cpp)
target_link_libraries(example PRIVATE adm)
```

Note: If `libadm` is used as a CMake subproject the default values of the options

- `ADM_UNIT_TESTS`
- `ADM_EXAMPLES`
- `ADM_PACKAGE_AND_INSTALL`

are automatically set to `FALSE`.

CHAPTER

TWO

TUTORIAL

In this tutorial we will create a simple object-based ADM document and write it to `std::cout`. We assume that `libadm` is installed and, that the `include` path is added to the `PATH` and you are linking with the library. We also assume, that you have at least basic knowledge on how an ADM file is structured.

2.1 First example

Let us have a look at the following first example.

```
#include <iostream>
#include <adm/adm.hpp>
#include <adm/write.hpp>

int main() {
  auto admDocument = adm::Document::create();
  adm::writeXml(std::cout, admDocument);
  return 0;
}
```

For most of the functionality of the library only the header `adm/adm.hpp` has to be included. As we simultaneously want to see how our ADM document takes shape we also included `adm/write.hpp`. This header contains the declaration of the `adm::writeXml()` functions. These functions can be used to write an ADM document to an `std::ostream` or a file. Apart from that not much is happening yet. We just create an *adm::Document*, which is the class representation of a whole ADM file.

```
<?xml version="1.0" encoding="utf-8"?>
<ebuCoreMain xmlns:dc="http://purl.org/dc/elements/1.1/"
             xmlns="urn:ebu:metadata-schema:ebuCore_2014"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             schema="EBU_CORE_20140201.xsd" xml:lang="en">
  <coreMetadata>
    <format>
      <audioFormatExtended/>
    </format>
  </coreMetadata>
</ebuCoreMain>
```

As our ADM document is still empty the output is no surprise: the `audioFormatExtended` node does not contain any ADM elements, but the basic `ebuCore` structure is already there.

Note: For all following XML code examples we are omitting the EBUcore structure for reasons of simplicity and only include the `audioFormatExtended` node.

2.2 Adding content

So let us fill our ADM document with some content.

```
#include <iostream>
#include <adm/adm.hpp>
#include <adm/write.hpp>

int main() {
  auto admDocument = adm::Document::create();

  auto admProgramme = adm::AudioProgramme::create(
      adm::AudioProgrammeName("Alice and Bob talking in the forrest"));
  auto speechContent = adm::AudioContent::create(adm::AudioContentName("Speech"));
  auto atmoContent = adm::AudioContent::create(adm::AudioContentName("Atmo"));

  admDocument->add(admProgramme);
  admDocument->add(speechContent);
  admDocument->add(atmoContent);

  adm::writeXml(std::cout, admDocument);

  return 0;
}
```

We have created an audioProgramme and two audioContent ADM elements and added them to our document.

```
<audioFormatExtended>
  <audioProgramme audioProgrammeID="APR_1001" audioProgrammeName="Alice and Bob talking␣
↪in the forrest"/>
  <audioContent audioContentID="ACO_1001" audioContentName="Speech"/>
  <audioContent audioContentID="ACO_1002" audioContentName="Atmo"/>
</audioFormatExtended>
```

Note that the IDs of the ADM elements are already properly set. This automatically happens when an ADM element is added to an *adm::Document*. This is usually very convenient. But in some cases one might want to manually set the ID. If an ADM element already has an ID, the `adm::IdAssigner` will use the ID if the ID is not already in use in the document. If it is, the `adm::IdAssigner` will increase the ID value until it finds an ID which is not used yet.

```
#include <iostream>
#include <adm/adm.hpp>
#include <adm/write.hpp>
#include <adm/utilities/object_creation.hpp>

int main() {
  auto admDocument = adm::Document::create();
```

(continues on next page)

(continued from previous page)

```
  auto admProgramme = adm::AudioProgramme::create(
      adm::AudioProgrammeName("Alice and Bob talking in the forest"));
  auto speechContent = adm::AudioContent::create(adm::AudioContentName("Speech"));
  auto atmoContent = adm::AudioContent::create(adm::AudioContentName("Atmo"));
  auto aliceHolder = adm::createSimpleObject("Alice");
  auto bobHolder = adm::createSimpleObject("Bob");

  admDocument->add(admProgramme);
  admDocument->add(speechContent);
  admDocument->add(atmoContent);
  admDocument->add(aliceHolder.audioObject);
  admDocument->add(bobHolder.audioObject);

  admProgramme->addReference(speechContent);
  admProgramme->addReference(atmoContent);
  speechContent->addReference(aliceHolder.audioObject);
  speechContent->addReference(bobHolder.audioObject);

  adm::writeXml(std::cout, admDocument);

  return 0;
}
```

As a next step we added two "objects". In an object-based situation we usually always have the same composition of audioObject, audioTrackUID, audioPackFormat, audioChannelFormat, audioStreamFormat, audioTrackFormat ADM elements. To simplify the process of creating an "object", we use the utility function *adm::createSimpleObject()*. It creates all the necessary ADM elements and adds the references.

The output of our programme is now as follows:

```
<audioFormatExtended>
  <audioProgramme audioProgrammeID="APR_1001" audioProgrammeName="Alice and Bob talking␣
↪in the forrest">
    <audioContentIDRef>ACO_1001</audioContentIDRef>
    <audioContentIDRef>ACO_1002</audioContentIDRef>
  </audioProgramme>
  <audioContent audioContentID="ACO_1001" audioContentName="Speech">
    <audioObjectIDRef>AO_1001</audioObjectIDRef>
    <audioObjectIDRef>AO_1002</audioObjectIDRef>
  </audioContent>
  <audioContent audioContentID="ACO_1002" audioContentName="Atmo"/>
  <audioObject audioObjectID="AO_1001" audioObjectName="Alice">
    <audioPackFormatIDRef>AP_00031001</audioPackFormatIDRef>
    <audioTrackUIDRef>ATU_00000001</audioTrackUIDRef>
  </audioObject>
  <audioObject audioObjectID="AO_1002" audioObjectName="Bob">
    <audioPackFormatIDRef>AP_00031002</audioPackFormatIDRef>
    <audioTrackUIDRef>ATU_00000002</audioTrackUIDRef>
  </audioObject>
  <audioPackFormat audioPackFormatID="AP_00031001" audioPackFormatName="Alice" typeLabel=
↪"0003" typeDefinition="Objects">
    <audioChannelFormatIDRef>AC_00031001</audioChannelFormatIDRef>
  </audioPackFormat>
```

(continues on next page)

(continued from previous page)

```
  <audioPackFormat audioPackFormatID="AP_00031002" audioPackFormatName="Bob" typeLabel=
↪"0003" typeDefinition="Objects">
    <audioChannelFormatIDRef>AC_00031002</audioChannelFormatIDRef>
  </audioPackFormat>
  <audioChannelFormat audioChannelFormatID="AC_00031001" audioChannelFormatName="Alice"␣
↪typeLabel="0003" typeDefinition="Objects"/>
  <audioChannelFormat audioChannelFormatID="AC_00031002" audioChannelFormatName="Bob"␣
↪typeLabel="0003" typeDefinition="Objects"/>
  <audioStreamFormat audioStreamFormatID="AS_00031001" audioStreamFormatName="Alice"␣
↪formatLabel="0001" formatDefinition="PCM">
    <audioChannelFormatIDRef>AC_00031001</audioChannelFormatIDRef>
    <audioTrackFormatIDRef>AT_00031001_01</audioTrackFormatIDRef>
  </audioStreamFormat>
  <audioStreamFormat audioStreamFormatID="AS_00031002" audioStreamFormatName="Bob"␣
↪formatLabel="0001" formatDefinition="PCM">
    <audioChannelFormatIDRef>AC_00031002</audioChannelFormatIDRef>
    <audioTrackFormatIDRef>AT_00031002_01</audioTrackFormatIDRef>
  </audioStreamFormat>
  <audioTrackFormat audioTrackFormatID="AT_00031001_01" audioTrackFormatName="Alice"␣
↪formatLabel="0001" formatDefinition="PCM">
    <audioStreamFormatIDRef>AS_00031001</audioStreamFormatIDRef>
  </audioTrackFormat>
  <audioTrackFormat audioTrackFormatID="AT_00031002_01" audioTrackFormatName="Bob"␣
↪formatLabel="0001" formatDefinition="PCM">
    <audioStreamFormatIDRef>AS_00031002</audioStreamFormatIDRef>
  </audioTrackFormat>
  <audioTrackUID UID="ATU_00000001">
    <audioTrackFormatIDRef>AT_00031001_01</audioTrackFormatIDRef>
    <audioPackFormatIDRef>AP_00031001</audioPackFormatIDRef>
  </audioTrackUID>
  <audioTrackUID UID="ATU_00000002">
    <audioTrackFormatIDRef>AT_00031002_01</audioTrackFormatIDRef>
    <audioPackFormatIDRef>AP_00031002</audioPackFormatIDRef>
  </audioTrackUID>
</audioFormatExtended>
```

But wait, we only added the audioObject to our document and all the elements created by *adm::createSimpleObject()* are now also part of the document. This is because the *adm::Document::add()* function automatically adds all referenced ADM elements too. Knowing this we can simplify our programme, while still getting the exact same output. We just add all our references first and only add the audioProgramme to the document.

```
#include <iostream>
#include <adm/adm.hpp>
#include <adm/write.hpp>
#include <adm/utilities/object_creation.hpp>

int main() {
  auto admDocument = adm::Document::create();

  auto admProgramme = adm::AudioProgramme::create(
      adm::AudioProgrammeName("Alice and Bob talking in the forest"));
```

(continues on next page)

(continued from previous page)

```
  auto speechContent = adm::AudioContent::create(adm::AudioContentName("Speech"));
  auto atmoContent = adm::AudioContent::create(adm::AudioContentName("Atmo"));
  auto aliceHolder = adm::createSimpleObject("Alice");
  auto bobHolder = adm::createSimpleObject("Bob");

  admProgramme->addReference(speechContent);
  admProgramme->addReference(atmoContent);
  speechContent->addReference(aliceHolder.audioObject);
  speechContent->addReference(bobHolder.audioObject);

  admDocument->add(admProgramme);

  adm::writeXml(std::cout, admDocument);

  return 0;
}
```

2.3 Using Common Definitions

As a next step we will add a channel bed to our document. The channel bed we are adding is a standard stereo signal. So we are going to use the common definitions. The first thing we need to do is add them to our document.

```
#include <adm/common_definitions.hpp>
...
auto admDocument = adm::Document::create();
addCommonDefinitionsTo(admDocument);    // add common definitions to our doc
```

Then we manually create our audioObject and the two audioTrackUIDs for the left and right channel.

```
auto atmoObject = adm::AudioObject::create(adm::AudioObjectName("Forest Atmo"));
auto trackUidLeft = adm::AudioTrackUid::create();
auto trackUidRight = adm::AudioTrackUid::create();
```

What is now missing is the connection between our object and the common definition ADM elements. To simplify the identification of the necessary ADM elements there are two lookup tables you can use. Those map the loudspeaker IDs and speaker labels specified in ITU-R BS.2051 to the corresponding ADM element IDs. To get the right ADM elements those IDs can then be used to look them up in the ADM document.

```
auto packFormatLookup = adm::audioPackFormatLookupTable();
auto trackFormatLookup = adm::audioTrackFormatLookupTable();

auto packFormatStereo = admDocument->lookup(packFormatLookup.at("0+2+0"));
auto trackFormatLeft = admDocument->lookup(trackFormatLookup.at("M+030"));
auto trackFormatRight = admDocument->lookup(trackFormatLookup.at("M-030"));

trackUidLeft->setReference(trackFormatLeft);
trackUidRight->setReference(trackFormatRight);
trackUidLeft->setReference(packFormatStereo);
trackUidRight->setReference(packFormatStereo);
```

(continues on next page)

(continued from previous page)

```
atmoObject->addReference(trackUidLeft);
atmoObject->addReference(trackUidRight);
atmoObject->addReference(packFormatStereo);
```

That's it. We are done.

```
#include <iostream>
#include <adm/adm.hpp>
#include <adm/write.hpp>
#include <adm/utilities/object_creation.hpp>
#include <adm/common_definitions.hpp>
#include <adm/utilities/copy.hpp>

int main() {
  auto admDocument = adm::Document::create();

  auto admProgramme = adm::AudioProgramme::create(
      adm::AudioProgrammeName("Alice and Bob talking in the forest"));
  auto speechContent = adm::AudioContent::create(adm::AudioContentName("Speech"));
  auto atmoContent = adm::AudioContent::create(adm::AudioContentName("Atmo"));
  auto aliceHolder = adm::createSimpleObject("Alice");
  auto bobHolder = adm::createSimpleObject("Bob");

  auto commonDefDoc = adm::getCommonDefinitions();
  adm::deepCopyTo(commonDefDoc, admDocument);

  auto atmoObject = adm::AudioObject::create(adm::AudioObjectName("Forrest Atmo"));
  auto trackUidLeft = adm::AudioTrackUid::create();
  auto trackUidRight = adm::AudioTrackUid::create();

  auto packFormatLookup = adm::audioPackFormatLookupTable();
  auto trackFormatLookup = adm::audioTrackFormatLookupTable();

  auto packFormatStereo = admDocument->lookup(packFormatLookup.at("0+2+0"));
  auto trackFormatLeft = admDocument->lookup(trackFormatLookup.at("M+030"));
  auto trackFormatRight = admDocument->lookup(trackFormatLookup.at("M-030"));

  trackUidLeft->setReference(trackFormatLeft);
  trackUidRight->setReference(trackFormatRight);
  trackUidLeft->setReference(packFormatStereo);
  trackUidRight->setReference(packFormatStereo);

  atmoObject->addReference(trackUidLeft);
  atmoObject->addReference(trackUidRight);
  atmoObject->addReference(packFormatStereo);

  admProgramme->addReference(speechContent);
  admProgramme->addReference(atmoContent);
  atmoContent->addReference(atmoObject);
  speechContent->addReference(aliceHolder.audioObject);
  speechContent->addReference(bobHolder.audioObject);
```

(continues on next page)

(continued from previous page)

```
  admDocument->add(admProgramme);

  adm::writeXml(std::cout, admDocument);  // write XML data to stdout

  return 0;
}
```

Now let us have a final look at the output.

```
<audioFormatExtended>
  <audioProgramme audioProgrammeID="APR_1001" audioProgrammeName="Alice and Bob talking␣
↪in the forrest">
    <audioContentIDRef>ACO_1002</audioContentIDRef>
    <audioContentIDRef>ACO_1001</audioContentIDRef>
  </audioProgramme>
  <audioContent audioContentID="ACO_1001" audioContentName="Atmo">
    <audioObjectIDRef>AO_1001</audioObjectIDRef>
  </audioContent>
  <audioContent audioContentID="ACO_1002" audioContentName="Speech">
    <audioObjectIDRef>AO_1002</audioObjectIDRef>
    <audioObjectIDRef>AO_1003</audioObjectIDRef>
  </audioContent>
  <audioObject audioObjectID="AO_1001" audioObjectName="Forrest Atmo">
    <audioPackFormatIDRef>AP_00010002</audioPackFormatIDRef>
    <audioTrackUIDRef>ATU_00000001</audioTrackUIDRef>
    <audioTrackUIDRef>ATU_00000002</audioTrackUIDRef>
  </audioObject>
  <audioObject audioObjectID="AO_1002" audioObjectName="Alice">
    <audioPackFormatIDRef>AP_00031001</audioPackFormatIDRef>
    <audioTrackUIDRef>ATU_00000003</audioTrackUIDRef>
  </audioObject>
  <audioObject audioObjectID="AO_1003" audioObjectName="Bob">
    <audioPackFormatIDRef>AP_00031002</audioPackFormatIDRef>
    <audioTrackUIDRef>ATU_00000004</audioTrackUIDRef>
  </audioObject>
  <audioPackFormat audioPackFormatID="AP_00031001" audioPackFormatName="Alice" typeLabel=
↪"0003" typeDefinition="Objects">
    <audioChannelFormatIDRef>AC_00031001</audioChannelFormatIDRef>
  </audioPackFormat>
  <audioPackFormat audioPackFormatID="AP_00031002" audioPackFormatName="Bob" typeLabel=
↪"0003" typeDefinition="Objects">
    <audioChannelFormatIDRef>AC_00031002</audioChannelFormatIDRef>
  </audioPackFormat>
  <audioChannelFormat audioChannelFormatID="AC_00031001" audioChannelFormatName="Alice"␣
↪typeLabel="0003" typeDefinition="Objects"/>
  <audioChannelFormat audioChannelFormatID="AC_00031002" audioChannelFormatName="Bob"␣
↪typeLabel="0003" typeDefinition="Objects"/>
  <audioStreamFormat audioStreamFormatID="AS_00031001" audioStreamFormatName="Alice"␣
↪formatLabel="0001" formatDefinition="PCM">
    <audioChannelFormatIDRef>AC_00031001</audioChannelFormatIDRef>
    <audioTrackFormatIDRef>AT_00031001_01</audioTrackFormatIDRef>
  </audioStreamFormat>
```

(continues on next page)

(continued from previous page)

```
  <audioStreamFormat audioStreamFormatID="AS_00031002" audioStreamFormatName="Bob"␣
↪formatLabel="0001" formatDefinition="PCM">
    <audioChannelFormatIDRef>AC_00031002</audioChannelFormatIDRef>
    <audioTrackFormatIDRef>AT_00031002_01</audioTrackFormatIDRef>
  </audioStreamFormat>
  <audioTrackFormat audioTrackFormatID="AT_00031001_01" audioTrackFormatName="Alice"␣
↪formatLabel="0001" formatDefinition="PCM">
    <audioStreamFormatIDRef>AS_00031001</audioStreamFormatIDRef>
  </audioTrackFormat>
  <audioTrackFormat audioTrackFormatID="AT_00031002_01" audioTrackFormatName="Bob"␣
↪formatLabel="0001" formatDefinition="PCM">
    <audioStreamFormatIDRef>AS_00031002</audioStreamFormatIDRef>
  </audioTrackFormat>
  <audioTrackUID UID="ATU_00000001">
    <audioTrackFormatIDRef>AT_00010001_01</audioTrackFormatIDRef>
    <audioPackFormatIDRef>AP_00010002</audioPackFormatIDRef>
  </audioTrackUID>
  <audioTrackUID UID="ATU_00000002">
    <audioTrackFormatIDRef>AT_00010002_01</audioTrackFormatIDRef>
    <audioPackFormatIDRef>AP_00010002</audioPackFormatIDRef>
  </audioTrackUID>
  <audioTrackUID UID="ATU_00000003">
    <audioTrackFormatIDRef>AT_00031001_01</audioTrackFormatIDRef>
    <audioPackFormatIDRef>AP_00031001</audioPackFormatIDRef>
  </audioTrackUID>
  <audioTrackUID UID="ATU_00000004">
    <audioTrackFormatIDRef>AT_00031002_01</audioTrackFormatIDRef>
    <audioPackFormatIDRef>AP_00031002</audioPackFormatIDRef>
  </audioTrackUID>
</audioFormatExtended>
```

As the idea of the common definitions is, that those ADM elements don't need to be written, even though we added common definition ADM elements to our document the XML writer does not write them.

2.4 Setting block format durations

Multiple `AudioBlockFormat` s in an *adm::AudioChannelFormat* should all have an *adm::Rtime* and a *adm::Duration*.

In practice, however, it can be very hard to determine the duration of an `adm::AudioBlockFormat` during its creation or setup. This is due to the fact that an *adm::AudioChannelFormat*, and thus its blocks and their durations, is bound to the parent *adm::AudioObject* duration. The lifetime of the *adm::AudioObject*, if not given explicitly, is bound to the length of the *adm::AudioProgramme* or, if that's not set either, to the length of the file.

Thus, it's easy to imagine situations where not all information is available during the setup of `adm::AudioBlockFormat` s.

This library provides some utility functions that are supposed to postpone the duration setting to a later point in time when all information is available, and therefore should help in writing standard conform ADM documents.

Consider the following code:

```
auto document = adm::Document::create();
auto programme = adm::AudioProgramme::create(adm::AudioProgrammeName{"main"});
auto content1 = adm::AudioContent::create(adm::AudioContentName{"main"});
programme->addReference(content1);
auto object1 = adm::AudioObject::create(adm::AudioObjectName{"object1"});
content1->addReference(object1);
auto pack1 = adm::AudioPackFormat::create(
  adm::AudioPackFormatName{"pack1"},
  adm::TypeDefinition::OBJECTS);
object1->addReference(pack1);
auto channel1 = adm::AudioChannelFormat::create(
  adm::AudioChannelFormatName{"channel1"},
  adm::TypeDefinition::OBJECTS);
channel1->add(adm::AudioBlockFormatObjects(
  adm::SphericalPosition{},
  adm::Rtime{std::chrono::milliseconds(0)}));
channel1->add(adm::AudioBlockFormatObjects(
  adm::SphericalPosition{},
  adm::Rtime{std::chrono::milliseconds(100)}));
```

Neither the referencing *adm::AudioObject* nor the main *adm::AudioProgramme* might have a duration or an end-time. Thus, the duration of the second block added to the *adm::AudioChannelFormat* `channel1` depends on the length of the audio signals, which might not be known at this point in time.

When it is known, for example when writing a `BW64` file with the ADM document contained in an `axml` chunk, one might known the actual length of the file. Then, one can use the utility function *adm::updateBlockFormatDurations()* to, well, update all block format durations with their correct values:

```
// ... somehow we know that our file will be 5 seconds long

updateBlockFormatDurations(document, std::chrono::seconds(5));

// now, continue with writing the xml chunk to disk or something similar
```

Depending on the use case, the file length might not be necessary or there might not even be a file with audio signals. Multiple variants of *adm::updateBlockFormatDurations()* are therefore provided to accommodate all use cases.

CHAPTER

THREE

LIBRARY DESIGN

The API of the libadm library is probably not self-explanatory if some of the concepts are not known. This sections aims at filling the gaps.

3.1 Named types

The libadm library makes an extensive use of so called named types. Thus, it is essential to understand named types to understand the design of the library. So we begin with a short introduction into named types. For more information on the implementation details of named types please refer to the blog post series on FluentCpp.

The idea of named types is to not use standard types directly but instead wrap them into a class. This approach has several advantages. Most importantly it makes the code safer and more expressive. To illustrate this have a look at the following snippet, which creates an *adm::AudioContent* object.

```
auto speechContentDe = AudioContent::create(AudioContentName("Speech"),
                                            AudioContentLanguage("de"));
```

It is obvious, that `Speech` is the name of the *adm::AudioContent* and `de` is the language. Yet still you can compare these types to a `std::string` like this:

```
auto myContentName = AudioContentName("Speech");
if (myContentName == "Speech") {
  std::cout << "Name of Content is Speech" << std::endl;
}
```

If for whatever reason it is really necessary to get the underlying type we can get it using the get method of the NamedTypes. But this should usually be avoided.

```
std::string myContentName = AudioContentName("Speech").get();
```

As we don't want to manually write a class for each type, named types are declared using the templated class `adm::detail::NamedType`. This makes declaring a new named type quite simple. The declaration for the *adm::AudioContentName* for example looks like this.

```
struct AudioContentNameTag {};
using AudioContentName = detail::NamedType<std::string, AudioContentNameTag>;
```

But the named types offer even more functionality. We can add a validator to it. Some basic validators are already implemented. E. g. the *adm::Importance* within the ADM can only have values between 0 and 10. To achieve this we add a `adm::detail::RangeValidator` to the type declaration.

```
struct ImportanceTag {};
using Importance = detail::NamedType<int, ImportanceTag, detail::RangeValidator<0, 10>>;
```

3.2 Basic structure

The libadm library is (for now) a quite low level library. Every ADM element has a class representation (either an ordinary class or a named type). Every class or named type is named exactly the same as in the ADM. The main ADM elements (see following list) are then managed by an *adm::Document*.

- *adm::AudioProgramme*
- *adm::AudioContent*
- *adm::AudioObject*
- *adm::AudioTrackUid*
- *adm::AudioPackFormat*
- *adm::AudioChannelFormat*
- *adm::AudioStreamFormat*
- *adm::AudioTrackFormat*

Note: At the moment there are still some sub-elements missing. Please refer to the documentation of the main ADM elements for a list of supported/unsupported sub-elements.

The *adm::Document* and the main ADM elements always have to be `std::shared_ptr<>`. This is enforced by making the constructors private and adding static `create` functions to each class, which return a `std::shared_ptr<>`.

Note: An ADM element can only belong to one *adm::Document*!

Once added to an *adm::Document* they cannot be added to another one. Trying to do so will result in a `std::runtime_error`. If you really want to move or copy an ADM element to another *adm::Document* the `adm::Document::move()` and `adm::Document::copy()` functions of the *adm::Document* have to be used.

As you have maybe noticed the `AudioBlockFormats` are not part of the previous list of main ADM Elements. That's because they are more like a special attribute of the *adm::AudioChannelFormat*. As the main ADM elements they also can only be created as `std::shared_ptr<>` s, but instead of the *adm::Document* they are managed by the *adm::AudioChannelFormat* they belong to. The same principles as for the main ADM elements and the *adm::Document* apply here. An `AudioBlockFormat` can only belong to one *adm::AudioChannelFormat* and if you want to move or copy it you have to use the corresponding functions of the *adm::AudioChannelFormat*.

3.3 References

References between the basic ADM elements can be established using the `addReference` or `addReferences` methods. Trying to establish a reference between two ADM elements which belong to different *`adm::Document`* results in a `std::runtime_error`. Adding an ADM element to an *`adm::Document`* will automatically add the referenced ADM elements too.

3.4 Overloaded/templated methods whenever possible

As we use classes or named types everywhere, it is quite straight forward to overload or use templated functions. Sub-elements or attributes of an ADM element can all be accessed using the following set of functions:

Function	Explanation
`Parameter get<Parameter>();`	Templated getter method
`void set(Parameter parameter);`	Overloaded setter method
`bool has<Parameter>();`	Returns true if the ADM parameter is set or has a default value.
`void unset<Parameter>();`	Removes the ADM parameter if it is optional or resets it to the default value if there is one.
`bool isDefault<Parameter>();`	Returns true if the current ADM parameter is the default value.

To illustrate the usage here is a simple example which uses them.

```
JumpPosition jumpPosition;

if(jumpPosition.has<InterpolationLength>() == true &&
   jumpPosition.isDefault<InterpolationLength>() == true) {
  std::cout << "JumpPositon has a default value for InterpolationLength: "
            << jumpPosition.get<InterpolationLength>() << std::endl;
}

jumpPosition.set(InterpolationLength(1.5f));
if(jumpPosition.has<InterpolationLength>() == true &&
   jumpPosition.isDefault<InterpolationLength>() == false) {
  std::cout << "InterpolationLength is now set to a custom value: "
            << jumpPosition.get<InterpolationLength>() << std::endl;
}
```

For more detail, see *Element API*.

3.5 Constructors with optional arguments in arbitrary order

The constructors (or the `create` functions in case of the main ADM elements) also make use of the named types. Using some black template magic they support optional arguments in arbitrary order. So let us revisit our first named type example.

```
auto speechContentDe = AudioContent::create(AudioContentName("Speech"),
                                            AudioContentLanguage("de"));
```

We can simply reorder *adm::AudioContentName* and *adm::AudioContentLanguage* and have the same result.

```
auto speechContentDe = AudioContent::create(AudioContentLanguage("de"),
                                            AudioContentName("Speech"));
```

3.6 Reading/writing ADM data

Parsing ADM data is as easy as it gets. You just have to include the file `adm/parse.hpp` and use one of the `adm::parseXml()` functions. You can either pass an `std::istream`

```
std::istream myAdmDataStream;
// Add XML data to stream ...
auto admDocument = adm::parseXml(myAdmDataStream);
```

or use the convenience function and pass the name of the input file as a `std::string`

```
std::string myFilename("./my_adm_file.xml");
auto admDocument = adm::parseXml(myFilename);
```

The same applies for writing an *adm::Document* to a file or stream. You just have to include the file `adm/write.hpp` and use one of the `adm::writeXml()` functions. You can either pass an `std::ostream`

```
auto admDocument = adm::Document::create();
// Add ADM elements ...
std::ostream xmlStream(...);
adm::writeXml(xmlStream, admDocument);
```

or use the convenience function and pass the name of the output file as a *std::string*

```
auto admDocument = adm::Document::create();
// Add ADM elements ...
adm::writeXml("outFilename.xml", admDocument);
```

CHAPTER

FOUR

CHANGELOG

4.1 Unreleased

4.1.1 Added

- Added support for silent audioTrackUid references with ID 0. See `AudioTrackUid::isSilent` and `AudioTrackUid::getSilent`.

4.1.2 Changed

- Decimal times are now written without trailing zeros past 5 decimal places. To interoperate with ADM parsers which don't support more than 5 digits, users should round times in the ADM document before writing.

4.1.3 Fixed

- Complementary audio object references are now read by the xml parser.

4.2 0.14.0 (September 12, 2022)

4.2.1 Added

- Added support for AudioChannelFormatIDRef in AudioTrackUID as per BS.2076-2
- Added support for dB gains. For clarity, `Gain{1.0}` should be replaced with `Gain::fromLinear(1.0)`, and `b.get<Gain>().get()` should be replaced with `b.get<Gain>().asLinear()`, though the old API should continue to work.
- Added BS.2076-2 gain attribute to audioObjects and all audioBlockFormat types.
- Added BS.2076-2 headLocked attribute to audioObjects and audioBlockFormats.
- Added support for headphoneVirtualise in audioBlockFormat as per BS.2076-2.
- Added support for importance in all audioBlockFormat types as per BS.2076-2.
- Added support for Label elements in AudioProgramme, AudioContent and AudioObject, and AudioComplementaryObjectGroupLabel elements in AudioObject.
- Added support for PositionOffset sub-element in AudioObject.

4.2.2 Changed

- Most single-argument constructors have been made explicit. For most code this should not be a problem, but it may sometimes require an extra constructor call when making elements.
- updated required C++ standard from C++11 to C++14
- implemented fractional time format from BS.2076-2
- audioProgramme and audioContent may now have multiple loudnessMetadata elements, as per BS.2076-2
- admConfig.cmake updated to behave better with find_package calls - errors are now reported correctly and info messages are silenced if QUIET has been requested.
- libadm_INCLUDE_DIRS and libadm_LIBRARY_DIRS were removed from admConfig.cmake. Users of these should link to the adm targets instead, as per the documentation.
- CMake GNUInstallDirs module used to determine default install locations
- INSTALL_XXX_DIR cache variables prefixed with ADM
- Install path for .dll on Windows changed to binary dir
- .pdb files now installed for Windows Debug and RelWithDebInfo configurations

4.2.3 Fixed

- `has` for `NfcRefDist`, `ScreenRef` and `Normalization` in HOA audioBlockFormat and audioPackFormat now always return true, as these parameters have defaults.

4.3 0.13.0 (February 15, 2022)

4.3.1 Added

- Added support for Cartesian speaker positions.

4.3.2 Changed

- SpeakerPosition is now a boost::variant that can be either a CartesianSpeakerPosition or a SphericalSpeakerPosition
- The previous SpeakerPosition type has been renamed to SphericalSpeakerPosition.
- included mono (0+1+0) to the common definitions lookup tables
- corrected (0+5+0) to point to 5.1 pack (AP_00010003) in common definitions lookup table
- included LFE in common definitions lookup table
- multiple incorrect references to LFE1 changed to LFE in common definitions lookup tables
- corrected B-045 AudioTrackFormat reference in common definitions lookup table
- fixed erroneous test acceptance data
- replaced resource embedder with a cmake function to fix cross-compilation

4.3.3 Fixed

- updateBlockFormatDurations now throws an exception when given an audioChannelFormat with no audioBlockFormats, rather than segfaulting
- fixed crash when parsing empty ADM documents

4.4 0.12.0 (April 18, 2020)

4.4.1 Added

- new `addSimpleCommonDefinitionsObjectTo` function
- new `addSimpleObjectTo` function
- added support to lookup HOA common definitions AudioPackFormatIDs and AudioTrackFormatIDs
- added missing ITU-R BS.2051 setups 0+7+0 and 4+7+0 to common definition lookup tables

4.4.2 Changed

- improved `AudioChannelFormat::assignId` logic - huge performance increase for large documents

4.4.3 Fixed

- fixed bug were not all references were removed if AudioPackFormat was removed from document

4.5 0.11.0 (Oktober 11, 2019)

4.5.1 Added

- library can now also be used as a CMake subproject
- new CMake option `ADM_HIDE_INTERNAL_SYMBOLS`
- new CMake option `ADM_PACKAGE_AND_INSTALL`
- new CMake option `BUILD_SHARED_LIBS`
- `audioPackFormat` now supports typeDefinition `HOA`

4.5.2 Changed

- Renamed CMake library target name from `libadm` to `adm`
- Renamed CMake option `UNIT_TESTS` to `ADM_UNIT_TESTS`
- Renamed CMake option `EXAMPLES` to `ADM_EXAMPLES`
- properly implemented the `LoudnessMetadata` class
- improved common definitions handling
- drastically improved performance by enhancing hex and ID parsing

- boost will automatically be found when finding libadm
- hide symbols only is shared library is build

4.5.3 Fixed

- An unresolvable reference will now result in an exception instead of a segfault when parsing XML.
- Always return true for values with default values in has<...>() methods.
- The dialogue subelement will now be written by the xml writer

4.6 0.10.0 (November 30, 2018)

4.6.1 Added

- Added helper function to access optional properties from elements, return a supplied default value if it hasn't been set
- Add utility functions to (re-)calculate block format durations
- Added `adm::ReaderOption` to select `AudioFormatExtended` node search mode

4.6.2 Changed

- Use `Catch2` instead of `Boost.Test` for unit testing
- Refactored XmlParser tests to use separate files for test data
- Improved search for `AudioFormatExtended` node when parsing XML

4.6.3 Fixed

- Documentation fixes and clarifications
- Fixed visibility issues of some methods that prevented linking with the shared library

4.7 0.9.0 (July 23, 2018)

Initial release

CHAPTER

FIVE

ADM DOCUMENT

class adm::**Document** : public std::enable_shared_from_this<*Document*>
Class representation of a whole ADM document.

Add ADM elements

If the ADM element was already added to the *Document*, it will not be added again.

bool **add**(std::shared_ptr<*AudioProgramme*> programme)
Add an *AudioProgramme*.

bool **add**(std::shared_ptr<*AudioContent*> content)
Add an *AudioContent*.

bool **add**(std::shared_ptr<*AudioObject*> object)
Add an *AudioObject*.

bool **add**(std::shared_ptr<*AudioPackFormat*> packFormat)
Add an *AudioPackFormat*.

bool **add**(std::shared_ptr<*AudioChannelFormat*> channelFormat)
Add an *AudioChannelFormat*.

bool **add**(std::shared_ptr<*AudioStreamFormat*> streamFormat)
Add an *AudioStreamFormat*.

bool **add**(std::shared_ptr<*AudioTrackFormat*> trackFormat)
Add an *AudioTrackFormat*.

bool **add**(std::shared_ptr<*AudioTrackUid*> trackUid)
Add an *AudioTrackUid*.

Remove ADM elements

References from and to the ADM element will automatically be removed too.

bool **remove**(std::shared_ptr<*AudioProgramme*> programme)
Remove an *AudioProgramme*.

bool **remove**(std::shared_ptr<*AudioContent*> content)
Remove an *AudioContent*.

bool **remove**(std::shared_ptr<*AudioObject*> object)
Remove an *AudioObject*.

bool **remove**(std::shared_ptr<*AudioPackFormat*> packFormat)
Remove an *AudioPackFormat*.

bool **remove**(std::shared_ptr<*AudioChannelFormat*> channelFormat)
Remove an *AudioChannelFormat*.

bool **remove**(std::shared_ptr<*AudioStreamFormat*> streamFormat)
Remove an *AudioStreamFormat*.

bool **remove**(std::shared_ptr<*AudioTrackFormat*> trackFormat)
Remove an *AudioTrackFormat*.

bool **remove**(std::shared_ptr<*AudioTrackUid*> trackUid)
Remove an *AudioTrackUid*.

Lookup ADM elements by its Id

Lookup the first ADM element with the given Id.

std::shared_ptr<*AudioProgramme*> **lookup**(const *AudioProgrammeId* &programmeId)
Lookup *AudioProgramme* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<const *AudioProgramme*> **lookup**(const *AudioProgrammeId* &programmeId) const
Lookup *AudioProgramme* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<*AudioContent*> **lookup**(const *AudioContentId* &contentId)
Lookup *AudioContent* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<const *AudioContent*> **lookup**(const *AudioContentId* &contentId) const
Lookup *AudioContent* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<*AudioObject*> **lookup**(const *AudioObjectId* &objectId)
Lookup *AudioObject* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<const *AudioObject*> **lookup**(const *AudioObjectId* &objectId) const
Lookup *AudioObject* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<*AudioPackFormat*> **lookup**(const *AudioPackFormatId* &packFormatId)
Lookup *AudioPackFormat* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<const *AudioPackFormat*> **lookup**(const *AudioPackFormatId* &packFormatId) const
Lookup *AudioPackFormat* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<*AudioChannelFormat*> **lookup**(const *AudioChannelFormatId* &channelFormatId)
Lookup *AudioChannelFormat* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<const *AudioChannelFormat*> **lookup**(const *AudioChannelFormatId* &channelFormatId) const

Lookup *AudioChannelFormat* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<*AudioStreamFormat*> **lookup**(const *AudioStreamFormatId* &streamFormatId)

Lookup *AudioStreamFormat* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<const *AudioStreamFormat*> **lookup**(const *AudioStreamFormatId* &streamFormatId) const

Lookup *AudioStreamFormat* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<*AudioTrackFormat*> **lookup**(const *AudioTrackFormatId* &trackFormatId)

Lookup *AudioTrackFormat* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<const *AudioTrackFormat*> **lookup**(const *AudioTrackFormatId* &trackFormatId) const

Lookup *AudioTrackFormat* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<*AudioTrackUid*> **lookup**(const *AudioTrackUidId* &trackUidId)

Lookup *AudioTrackUid* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

std::shared_ptr<const *AudioTrackUid*> **lookup**(const *AudioTrackUidId* &trackUidId) const

Lookup *AudioTrackUid* using its id.

Returns `shared_ptr` to the element, `nullptr` if not found

Public Functions

std::shared_ptr<*Document*> **deepCopy**() const

Create a copy of the *Document* including all elements.

template<typename **Element**>
ElementRange<const *Element*> **getElements**() const

ADM elements getter template.

Templated getter with the wanted ADM element type as template argument.

template<typename **Element**>
ElementRange<*Element*> **getElements**()

ADM elements getter template.

Templated getter with the wanted ADM element type as template argument.

Public Static Functions

static std::shared_ptr<*Document*> **create**()
Static helper function to create an *Document*.

The constructor is private. This way it is ensured, that an *Document* can only be created as a `std::shared_ptr`.

CHAPTER

SIX

ELEMENT API

This page documents the API convention for accessing parameters of ADM element classes. Element classes like `AudioBlockFormatObjects` contain many parameters, each with different types (thanks to using named types). These parameters can be accessed through a common set of templated and overloaded methods defined on elements. Compared to standard accessors methods, this makes it easy to write templates which handle parameters generically.

Note: The classes listed on this page do not actually exist, they just serve to show the API on element classes themselves.

The set of methods which may be defined for each parameter of type `Parameter` are shown below:

class `adm::Element`

template<typename **Parameter**>
Parameter **get**()
: Get the value of a parameter. If the parameter has no default value and has not been set then an error is raised.

void **set**(Parameter parameter)
: Set the value of a parameter.

template<typename **Parameter**>
bool **has**<*Parameter*>()
: Returns true if the ADM parameter is set or has a default value, and therefore *get()* will not raise an error.

template<typename **Parameter**>
void **unset**<*Parameter*>()
: Removes the ADM parameter if it is optional or resets it to the default value if there is one.

template<typename **Parameter**>
bool **isDefault**<*Parameter*>()
: Returns true if the parameter has a default and has not been set; this is useful to see if the default value was specified explicitly in the ADM XML (in which case this would return false), and is used to control whether default values are written out in XML.

bool **add**(Parameter parameter)
: For parameters with multiple values, add a new value, ensuring uniqueness. Returns true if the parameter was added, or false if it already had a parameter with this value.

void **remove**(Parameter parameter)
: For parameters with multiple values, remove one.

These methods are implemented in some common patterns for parameters which behave in different ways:

template<typename **T**>

class adm::**RequiredParameter**
Required parameters must be specified in the ADM XML.

template<>
T **get**<*T*>()
see *Element::get()*

void **set**(*T*)
see *Element::set()*

template<>
bool **has**<*T*>()
always returns true

template<typename **T**>
class adm::**OptionalParameter**
Optional parameters may or may not be specified.

template<>
T **get**<*T*>()
see *Element::get()*

void **set**(*T*)
see *Element::set()*

template<>
bool **has**<*T*>()
see Element::has()

template<>
void **unset**<*T*>()
see Element::unset()

template<>
bool **isDefault**<*T*>()
always returns false

template<typename **T**>
class adm::**DefaultParameter**
Default parameters may or may not be specified, but have a default defined in the ADM.

template<>
T **get**<*T*>()
see *Element::get()*

void **set**(*T*)
see *Element::set()*

template<>
bool **has**<*T*>()
see Element::has()

template<>
void **unset**<*T*>()
see Element::unset()

template<>
bool **isDefault**<*T*>()
see Element::isDefault()

template<typename **VectorT**>

class adm::**VectorParameter**

Vector parameters have multiple values, and some defined concept of equality.

`get` and `set` methods get and set a `std::vector<T>` holding the parameters, while `add` and `remove` add and remove individual values.

using **T** = *VectorT*::value_type

template<>
VectorT **get**<*VectorT*>()

get a vector of parameters.

void **set**(*VectorT*)

Set a vector of parameters.

template<>
bool **has**<*VectorT*>()

Have any parameters been set?

template<>
void **unset**<*VectorT*>()

Clear the list of parameters.

template<>
bool **isDefault**<*VectorT*>()

Always returns false.

bool **add**(*T*)

Add a new value, ensuring uniqueness. Returns true if the parameter was added, or false if it already had a parameter with this value.

void **remove**(*T*)

Remove a parameter from the list.

template<typename **ParameterT**>
class adm::**VariantParameter**

Variant parameters have a single value, but that value can be one of two or more types, stored in a *boost::variant*.

This is used for types where there is no obvious conversion between the possible types. For types that just have multiple representations of the same data, a wrapper class is used instead.

Access to the variant type follows one of the above schemes (*adm::RequiredParameter*, *adm::OptionalParameter* etc.). In addition, methods are provided for each `T` in the variant to access the individual types:

template<>
T **get**<T>()

Get T; if the parameter is not set, or is not of the specified type, an error is raised.

void **set**(T)

Set the parameter.

template<>
bool **has**<T>()

Is the parameter set (or is it defaulted) and is it of the specified type?

template<>
void **unset**<T>()

Unset the parameter if it is set and of the specified type. If it's a different type this does nothing – to unset any type, use the variant type instead.

template<>

bool **isDefault**<T>()
Returns true if the variant has the default value, and the default is of the correct type.

CHAPTER

SEVEN

ADM ELEMENTS

7.1 AudioProgramme

class adm::**AudioProgramme** : public std::enable_shared_from_this<*AudioProgramme*>, private AudioProgrammeBase

Class representation of the audioProgramme ADM element.

ADM Parameter	Parameter Type	Pattern Type
audioProgrammeID	*AudioProgrammeId*	*RequiredParameter*
audioProgrammeName	*AudioProgrammeName*	*RequiredParameter*
audioProgrammeLanguage	*AudioProgrammeLanguage*	*OptionalParameter*
start	*Start*	*DefaultParameter*
end	*End*	*OptionalParameter*
audioProgrammeLabel	*Labels*	*VectorParameter*
loudnessMetadata	*LoudnessMetadatas*	*VectorParameter*
maxDuckingDepth	*MaxDuckingDepth*	*OptionalParameter*
audioProgrammeReferenceScreen	*AudioProgrammeReferenceScreen*	*OptionalParameter*

Note that start has a default time of 0, contrary to BS.2076-2 which does not define a default. `isDefault<Start>()` should be used in place of of `has<Start>()`.

Public Types

typedef AudioProgrammeTag **tag**

typedef *AudioProgrammeId* **id_type**

Type that holds the id for this element;.

Public Functions

std::shared_ptr<*AudioProgramme*> **copy**() const
Copy *AudioProgramme*.

The actual copy constructor is private to ensure that an *AudioProgramme* can only be created as a `std::shared_ptr`. This is not a deep copy! All referenced objects will be disconnected.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*AudioProgrammeId* id)
AudioProgrammeId setter.

void **set**(*AudioProgrammeName* name)
AudioProgrammeName setter.

void **set**(*AudioProgrammeLanguage* language)
AudioProgrammeLanguage setter.

void **set**(*Start* start)
Start setter.

void **set**(*End* end)
End setter.

void **set**(*MaxDuckingDepth* depth)
MaxDuckingDepth setter.

void **set**(*AudioProgrammeReferenceScreen* refScreen)
AudioProgrammeReferenceScreen setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

bool **addReference**(std::shared_ptr<*AudioContent*> content)
Add reference to an *AudioContent*.

template<typename **Element**>

ElementRange<*Element*> **getReferences**()
Get references to ADM elements template.

Templated get method with the ADM parameter type as template argument. Returns a set of all references to the ADM elements with the specified type.

template<typename **Element**>
ElementRange<const *Element*> **getReferences**() const
Get references to ADM elements template.

Templated get method with the ADM parameter type as template argument. Returns a set of all references to the ADM elements with the specified type.

void **removeReference**(std::shared_ptr<*AudioContent*> content)
Remove reference to an *AudioContent*.

template<typename **Element**>
void **clearReferences**()
Clear references to Elements template.

Templated clear method with the ADM parameter type as template argument. Removes all references to the ADM elements with the specified type.

void **print**(std::ostream &os) const
Print overview to ostream.

const std::weak_ptr<*Document*> &**getParent**() const
Get *adm::Document* this element belongs to.

Public Static Functions

template<typename ...**Parameters**>
static std::shared_ptr<*AudioProgramme*> **create**(*AudioProgrammeName* name, *Parameters*... optionalNamedArgs)
Static create function template.

Templated static create function which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters. The actual constructor is private. This way it is ensured, that an *AudioProgramme* object can only be created as a `std::shared_ptr`.

class `adm::`**AudioProgrammeId**
Representation of an *AudioProgrammeId*.

Unnamed Group

bool **operator==**(const *AudioProgrammeId* &other) const
Operator overload.

Compares the string representation of the *AudioProgrammeId*.

bool **operator!=**(const *AudioProgrammeId* &other) const

bool **operator<**(const *AudioProgrammeId* &other) const

Public Types

typedef AudioProgrammeIdTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **AudioProgrammeId**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

AudioProgrammeId(AudioProgrammeIdValue)

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(AudioProgrammeIdValue value)
Set value.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**AudioProgrammeName** = detail::NamedType<std::string, AudioProgrammeNameTag>
NamedType for the audioProgrammeName attribute.

using adm::**AudioProgrammeLanguage** = detail::NamedType<std::string, AudioProgrammeLanguageTag>
NamedType for the language attribute of the audioProgramme element.

using adm::**End** = detail::NamedType<Time, EndTag>
NamedType for the end attribute.

using adm::**MaxDuckingDepth** = detail::NamedType<double, MaxDuckingDepthTag, detail::RangeValidator<-62, 0>>

NamedType for the maxDuckingDepth attribute element.

Valid values are in the range [-62, 0]

class adm::**AudioProgrammeReferenceScreen**

Public Types

typedef AudioProgrammeReferenceScreenTag **tag**

Public Functions

inline **AudioProgrammeReferenceScreen**()

inline void **print**(std::ostream &os) const

Print overview to ostream.

7.2 AudioContent

class adm::**AudioContent** : public std::enable_shared_from_this<*AudioContent*>, private AudioContentBase

Class representation of the audioContent ADM element.

ADM Parameter	Parameter Type	Pattern Type
audioContentID	*AudioContentId*	*RequiredParameter*
audioContentName	*AudioContentName*	*RequiredParameter*
audioContentLanguage	*AudioContentLanguage*	*OptionalParameter*
audioContentLabel	*Labels*	*VectorParameter*
loudnessMetadata	*LoudnessMetadatas*	*VectorParameter*
• dialogue • nonDialogueContentKind • dialogueContentKind • mixedContentKind	• *DialogueId* • *ContentKind* • *NonDialogueContentKind* • *DialogueContentKind* • *MixedContentKind*	custom

For the behaviour of dialogue elements, see *set(DialogueId)*.

Unnamed Group

void **set**(*DialogueId* kind)
Dialogue setter.

Note: Setting one of the dialogue kinds always ensures consistency. If *DialogueId* is set the corresponding ContentKind will be set to undefined. If one of the ContentKinds is set *DialogueId* will be set accordingly.

---

void **set**(*ContentKind* kind)

void **set**(*NonDialogueContentKind* kind)

void **set**(*DialogueContentKind* kind)

void **set**(*MixedContentKind* kind)

Public Types

typedef AudioContentTag **tag**

typedef *AudioContentId* **id_type**
Type that holds the id for this element;.

Public Functions

std::shared_ptr<*AudioContent*> **copy**() const
Copy *AudioContent*.

The actual copy constructor is private to ensure that an *AudioContent* can only be created as a `std::shared_ptr`. This is not a deep copy! All referenced objects will be disconnected.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the ADM parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*AudioContentId* id)
AudioContentId setter.

void **set**(*AudioContentName* name)
AudioContentName setter.

void **set**(*AudioContentLanguage* language)
AudioContentLanguage setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

Note: Unsetting *DialogueId* automatically unsets the corresponding ContentKind. Unsetting one of the ContentKinds automatically unsets *DialogueId*.

---

bool **addReference**(std::shared_ptr<*AudioObject*> object)
Add reference to an *AudioObject*.

template<typename **Element**>
ElementRange<const *Element*> **getReferences**() const
Get references to ADM elements template.

Templated get method with the ADM parameter type as template argument. Returns a set of all references to the adm elements with the specified type.

template<typename **Element**>
ElementRange<*Element*> **getReferences**()
Get references to ADM elements template.

Templated get method with the ADM parameter type as template argument. Returns a set of all references to the adm elements with the specified type.

void **removeReference**(std::shared_ptr<*AudioObject*> object)
Remove reference to an *AudioObject*.

template<typename **Element**>
void **clearReferences**()
Clear references to Elements template.

Templated clear method with the ADM parameter type as template argument. Removes all references to the adm elements with the specified type.

void **print**(std::ostream &os) const
Print overview to ostream.

const std::weak_ptr<*Document*> &**getParent**() const
Get *adm::Document* this element belongs to.

Public Static Functions

template<typename ...**Parameters**>
static std::shared_ptr<*AudioContent*> **create**(*AudioContentName* name, *Parameters*... optionalNamedArgs)
Static create function template.

Templated static create function which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters. The actual constructor is private. This way it is ensured, that an *AudioContent* object can only be created as a `std::shared_ptr`.

class `adm::`**AudioContentId**
Representation of an *AudioContentId*.

Unnamed Group

bool **operator==**(const *AudioContentId* &other) const
Operator overload.

Compares the string representation of the *AudioContentId*.

bool **operator!=**(const *AudioContentId* &other) const

bool **operator<**(const *AudioContentId* &other) const

Public Types

typedef AudioContentIdTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **AudioContentId**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

AudioContentId(AudioContentIdValue)

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(AudioContentIdValue value)
Set value.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**AudioContentName** = detail::NamedType<std::string, AudioContentNameTag>
NamedType for the audioContentName attribute.

using adm::**AudioContentLanguage** = detail::NamedType<std::string, AudioContentLanguageTag>
NamedType for the language attribute of the audioContent element.

using adm::**DialogueId** = detail::NamedType<unsigned int, DialogueIdTag, detail::RangeValidator<0, 2>>
NamedType for the dialogueId attribute.

Valid values are in the range [0, 2]

namespace adm::**Dialogue**
DialogueId definitions.

Variables

const *DialogueId* **NON_DIALOGUE** = *DialogueId*(0)

const *DialogueId* **DIALOGUE** = *DialogueId*(1)
DialogueId for music.

const *DialogueId* **MIXED** = *DialogueId*(2)
DialogueId for effects.

typedef boost::variant<*NonDialogueContentKind*, *DialogueContentKind*, *MixedContentKind*> adm::**ContentKind**
Type to hold a NonDialogueContentKind, DialogueContentKind or MixedContentKind.

using adm::**NonDialogueContentKind** = detail::NamedType<unsigned int, NonDialogueContentKindTag, detail::RangeValidator<0, 2>>
NamedType for the nonDialogueContentKind type.

Valid values are in the range [0, 2]

namespace adm::**NonDialogueContent**
NonDialogueContentKind definitions.

Variables

const *NonDialogueContentKind* **UNDEFINED** = *NonDialogueContentKind*(0)

const *NonDialogueContentKind* **MUSIC** = *NonDialogueContentKind*(1)
NonDialogueContentKind for music.

const *NonDialogueContentKind* **EFFECT** = *NonDialogueContentKind*(2)
NonDialogueContentKind for effects.

using adm::**DialogueContentKind** = detail::NamedType<unsigned int, DialogueContentKindTag, detail::RangeValidator<0, 6>>
NamedType for the dialogueContentKind type.

Valid values are in the range [0, 6]

namespace adm::**DialogueContent**
DialogueContentKind definitions.

Variables

const *DialogueContentKind* **UNDEFINED** = *DialogueContentKind*(0)

const *DialogueContentKind* **DIALOGUE** = *DialogueContentKind*(1)
DialogueContentKind for (storyline) dialogue.

const *DialogueContentKind* **VOICEOVER** = *DialogueContentKind*(2)
DialogueContentKind for voiceover.

const *DialogueContentKind* **SPOKEN_SUBTITLE** = *DialogueContentKind*(3)
DialogueContentKind for spoken subtitle.

const *DialogueContentKind* **AUDIO_DESCRIPTION** = *DialogueContentKind*(4)
DialogueContentKind for audio description/visually impaired.

const *DialogueContentKind* **COMMENTARY** = *DialogueContentKind*(5)
DialogueContentKind for commentary.

const *DialogueContentKind* **EMERGENCY** = *DialogueContentKind*(6)
DialogueContentKind for emergency.

using adm::**MixedContentKind** = detail::NamedType<unsigned int, MixedContentKindTag, detail::RangeValidator<0, 3>>
NamedType for the mixedContentKind type.

Valid values are in the range [0, 3]

namespace adm::**MixedContent**
MixedContent definitions.

Variables

const *MixedContentKind* **UNDEFINED** = *MixedContentKind*(0)

const *MixedContentKind* **COMPLETE_MAIN** = *MixedContentKind*(1)
MixedContentKind for complete main.

const *MixedContentKind* **MIXED** = *MixedContentKind*(2)
MixedContentKind for mixed.

const *MixedContentKind* **HEARING_IMPAIRED** = *MixedContentKind*(3)
MixedContentKind for hearing impaired.

7.3 AudioObject

class adm::**AudioObject** : public std::enable_shared_from_this<*AudioObject*>, private AudioObjectBase
Class representation of the audioObject ADM element.

ADM Parameter	Parameter Type	Pattern Type
audioObjectID	*AudioObjectId*	*RequiredParameter*
audioObjectName	*AudioObjectName*	*RequiredParameter*
start	*Start*	*DefaultParameter*
duration	*Duration*	*OptionalParameter*
dialogue	*DialogueId*	*OptionalParameter*
importance	*Importance*	*OptionalParameter*
interact	*Interact*	*OptionalParameter*
disableDucking	*DisableDucking*	*OptionalParameter*
audioObjectLabel	*Labels*	*VectorParameter*
audioComplementaryObject-GroupLabel	*ACOGL*	*VectorParameter*
audioObjectInteraction	*AudioObjectInteraction*	*OptionalParameter*
gain	*Gain*	*DefaultParameter*
headLocked	*HeadLocked*	*DefaultParameter*
positionOffset	*PositionOffset*	• *VariantParameter* • *OptionalParameter*
mute	*Mute*	*DefaultParameter*

Note that:

- dialogue is defined in BS.2076-2 as having a default of 2.
- importance is defined in BS.2076-2 as having a default of 10.
- interact is defined in BS.2076-2 as having a default of 0 (false).
- disableDucking is defined in BS.2076-2 as having a default of 0 (false).

Public Types

typedef AudioObjectTag **tag**

typedef *AudioObjectId* **id_type**
Type that holds the id for this element;.

Public Functions

std::shared_ptr<*AudioObject*> **copy**() const
Copy *AudioObject*.

The actual copy constructor is private to ensure that an *AudioObject* can only be created as a `std::shared_ptr`. This is not a deep copy! All referenced objects will be disconnected.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*AudioObjectId* id)
AudioObjectId setter.

void **set**(*AudioObjectName* name)
AudioObjectName setter.

void **set**(*Start* start)
Start setter.

void **set**(*Duration* duration)
Duration setter.

void **set**(*DialogueId* id)
DialogueId setter.

void **set**(*Importance* importance)
Importance setter.

void **set**(*Interact* interact)
Interact setter.

void **set**(*DisableDucking* disableDucking)
DisableDucking setter.

void **set**(*AudioObjectInteraction* objectInteraction)
AudioObjectInteraction setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

bool **addReference**(std::shared_ptr<*AudioObject*> object)
Add reference to another *AudioObject*.

bool **addReference**(std::shared_ptr<*AudioPackFormat*> packFormat)
Add reference to an *AudioPackFormat*.

bool **addReference**(std::shared_ptr<*AudioTrackUid*> trackUid)
Add reference to an *AudioTrackUid*.

template<typename **Element**>
ElementRange<const *Element*> **getReferences**() const
Get references to ADM elements template.

Templated get method with the ADM parameter type as template argument. Returns a set of all references to the ADM elements with the specified type.

template<typename **Element**>
ElementRange<*Element*> **getReferences**()
Get references to ADM elements template.

Templated get method with the ADM parameter type as template argument. Returns a set of all references to the ADM elements with the specified type.

void **removeReference**(std::shared_ptr<*AudioObject*> object)
Remove reference to an *AudioObject*.

void **removeReference**(std::shared_ptr<*AudioPackFormat*> packFormat)
Remove reference to an *AudioPackFormat*.

void **removeReference**(std::shared_ptr<*AudioTrackUid*> trackUid)
Remove reference to an *AudioTrackUid*.

template<typename **Element**>
void **clearReferences**()
Clear references to Elements template.

Templated clear method with the ADM parameter type as template argument. Removes all references to the ADM elements with the specified type.

bool **addComplementary**(std::shared_ptr<*AudioObject*> object)
Add reference to a complementary *AudioObject*.

const std::vector<std::shared_ptr<*AudioObject*>> &**getComplementaryObjects**() const
Get references to complementary AudioObjects.

void **removeComplementary**(const std::shared_ptr<*AudioObject*> &object)
Remove reference to a complementary *AudioObject*.

void **clearComplementaryObjects**()
Remove all references to complementary AudioObjects.

void **print**(std::ostream &os) const
Print overview to ostream.

const std::weak_ptr<*Document*> &**getParent**() const
Get *adm::Document* this element belongs to.

Public Static Functions

template<typename ...**Parameters**>
static std::shared_ptr<*AudioObject*> **create**(*AudioObjectName* name, *Parameters*... optionalNamedArgs)
Static create function template.

Templated static create function which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters. The actual constructor is private. This way it is ensured, that an *AudioObject* object can only be created as a `std::shared_ptr`.

class `adm::`**AudioObjectId**
Representation of an *AudioObjectId*.

Unnamed Group

bool **operator==**(const *AudioObjectId* &other) const
Operator overload.

Compares the string representation of the *AudioObjectId*.

bool **operator!=**(const *AudioObjectId* &other) const

bool **operator<**(const *AudioObjectId* &other) const

Public Types

typedef AudioObjectIdTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **AudioObjectId**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

AudioObjectId(AudioObjectIdValue value)

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>

bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(AudioObjectIdValue value)
Set value.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**AudioObjectName** = detail::NamedType<std::string, AudioObjectNameTag>
NamedType for the audioObjectName attribute.

using adm::**Interact** = detail::NamedType<bool, InteractTag>
NamedType for the interact attribute.

using adm::**DisableDucking** = detail::NamedType<bool, DisableDuckingTag>
NamedType for the disableDucking attribute.

class adm::**AudioObjectInteraction**
ADM parameter class to specify a channel lock.

Public Types

typedef AudioObjectInteractionTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **AudioObjectInteraction**(*OnOffInteract* onOffInteract, *Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*OnOffInteract*)
OnOffInteract setter.

void **set**(*GainInteract*)
GainInteract setter.

void **set**(*PositionInteract*)
PositionInteract setter.

void **set**(*GainInteractionRange*)
GainInteractionRange setter.

void **set**(*PositionInteractionRange*)
PositionInteractionRange setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**OnOffInteract** = detail::NamedType<bool, OnOffInteractTag>
NamedType for channelLockFlag parameter.

using adm::**GainInteract** = detail::NamedType<bool, GainInteractTag>
NamedType for channelLockFlag parameter.

using adm::**PositionInteract** = detail::NamedType<bool, PositionInteractTag>
NamedType for channelLockFlag parameter.

class adm::**GainInteractionRange**
ADM parameter class to specify a gainInteractionRange.

Public Types

typedef GainInteractionRangeTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **GainInteractionRange**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(GainInteractionMin)
GainInteractionMin setter.

void **set**(GainInteractionMax)
GainInteractionMax setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

class adm::**PositionInteractionRange**
ADM parameter class to specify a positionInteractionRange.

Public Types

typedef PositionInteractionRangeTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **PositionInteractionRange**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(AzimuthInteractionMin)
AzimuthInteractionMin setter.

void **set**(AzimuthInteractionMax)
AzimuthInteractionMax setter.

void **set**(ElevationInteractionMin)
ElevationInteractionMin setter.

void **set**(ElevationInteractionMax)
ElevationInteractionMax setter.

void **set**(DistanceInteractionMin)
DistanceInteractionMin setter.

void **set**(DistanceInteractionMax)
DistanceInteractionMax setter.

void **set**(XInteractionMin)
XInteractionMin setter.

void **set**(XInteractionMax)
XInteractionMax setter.

void **set**(YInteractionMin)
YInteractionMin setter.

void **set**(YInteractionMax)
YInteractionMax setter.

void **set**(ZInteractionMin)
ZInteractionMin setter.

void **set**(ZInteractionMax)
ZInteractionMax setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**AudioComplementaryObjectGroupLabel** = detail::NamedType<*Label*, AudioComplementaryObjectGroupLabelTag>

using adm::**AudioComplementaryObjectGroupLabels** = std::vector<*AudioComplementaryObjectGroupLabel*>

using adm::**AzimuthOffset** = detail::NamedType<float, AzimuthOffsetTag, detail::RangeValidator<-360, 360>>
NamedType for the azimuth parameter of the positionOffset element.

Valid values are in the range [-180, 180]

using adm::**ElevationOffset** = detail::NamedType<float, ElevationOffsetTag, detail::RangeValidator<-180, 180>>
NamedType for the elevation parameter of the positionOffset element.

Valid values are in the range [-90, 90]

using adm::**DistanceOffset** = detail::NamedType<float, DistanceOffsetTag>
NamedType for the distance parameter of the positionOffset offset element.

Valid values are in the range [-1, 1]

class adm::**SphericalPositionOffset** : private SphericalPositionOffsetBase, private detail::AddWrapperMethods<*SphericalPositionOffset*>
ADM parameter class to specify a spherical position offset.

Supported parameters are as follows:

ADM Parameter	Parameter Type	Pattern Type
positionOffset, coordinate="azimuth"	*AzimuthOffset*	*RequiredParameter*
positionOffset, coordinate="elevation"	*ElevationOffset*	*RequiredParameter*
positionOffset, coordinate="distance"	*DistanceOffset*	*RequiredParameter*

Public Types

typedef SphericalPositionOffsetTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **SphericalPositionOffset**(*Parameters*&&... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

void **print**(std::ostream &os) const
Print overview to ostream.

Friends

friend class detail::AddWrapperMethods< SphericalPositionOffset >

using adm::**XOffset** = detail::NamedType<float, XOffsetTag>
NamedType for the X parameter of the positionOffset element.

using adm::**YOffset** = detail::NamedType<float, YOffsetTag>
NamedType for the Y parameter of the positionOffset element.

using adm::**ZOffset** = detail::NamedType<float, ZOffsetTag>
NamedType for the Z parameter of the positionOffset element.

class adm::**CartesianPositionOffset** : private CartesianPositionOffsetBase, private detail::AddWrapperMethods<*CartesianPositionOffset*>
ADM parameter class to specify a cartesian position offset.

Supported parameters are as follows:

ADM Parameter	Parameter Type	Pattern Type
positionOffset, coordinate="X"	*XOffset*	*RequiredParameter*
positionOffset, coordinate="Y"	*YOffset*	*RequiredParameter*
positionOffset, coordinate="Z"	*ZOffset*	*RequiredParameter*

Public Types

typedef CartesianPositionOffsetTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **CartesianPositionOffset**(*Parameters*&&... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

void **print**(std::ostream &os) const
Print overview to ostream.

Friends

friend class detail::AddWrapperMethods< CartesianPositionOffset >

typedef boost::variant<*SphericalPositionOffset*, *CartesianPositionOffset*> **adm::PositionOffset**
Type to hold a *SphericalPositionOffset* or *CartesianPositionOffset*.

using **adm::Mute** = detail::NamedType<bool, MuteTag>
NamedType for the mute attribute.

7.4 AudioTrackUid

class **adm::AudioTrackUid** : public std::enable_shared_from_this<*AudioTrackUid*>
Class representation of the audioTrackUID ADM element.

> **Warning:** This class has unsupported parameters:
>
> - audioMXFLookUp

Public Types

typedef AudioTrackUidTag **tag**

typedef *AudioTrackUidId* **id_type**
Type that holds the id for this element;.

Public Functions

std::shared_ptr<*AudioTrackUid*> **copy**() const
Copy *AudioTrackUid*.

The actual copy constructor is private to ensure that an *AudioTrackUid* can only be created as a `std::shared_ptr`. This is not a deep copy! All referenced objects will be disconnected.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*AudioTrackUidId* id)
AudioTrackUidId setter.

void **set**(*SampleRate* sampleRate)
SampleRate setter.

void **set**(*BitDepth* bitDepth)
BitDepth setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **setReference**(std::shared_ptr<*AudioTrackFormat*> trackFormat)
Set reference to an *AudioTrackFormat*.

A pending unresolved reference will be removed.

void **setReference**(std::shared_ptr<*AudioChannelFormat*> channelFormat)
Set reference to an *AudioChannelFormat*.

A pending unresolved reference will be removed.

void **setReference**(std::shared_ptr<*AudioPackFormat*> packFormat)
Set reference to an *AudioPackFormat*.

A pending unresolved reference will be removed.

template<typename **Element**>
std::shared_ptr<const *Element*> **getReference**() const

template<typename **Element**>
std::shared_ptr<*Element*> **getReference**()
Get reference to ADM element template.

Templated get method with the ADM parameter type as template argument. Returns the reference to the ADM element with the specified type. If it is not set a nullptr will be returned.

template<typename **Element**>
void **removeReference**()
Remove reference to an Element template.

Templated remove method with the ADM parameter type as template argument. Removes the reference to the ADM elements with the specified type.

bool **isSilent**() const
true if this element has an ID of 0

These elements do not appear in adm XML, but are referenced from audioObjects to indicate that a channel has no corresponding audio track. If this is true, no parameters or references may be set and getReference will return null pointers.

void **print**(std::ostream &os) const
Print overview to ostream.

const std::weak_ptr<*Document*> &**getParent**() const
Get *adm::Document* this element belongs to.

Public Static Functions

template<typename ...**Parameters**>
static std::shared_ptr<*AudioTrackUid*> **create**(*Parameters*... optionalNamedArgs)
Static create function template.

Templated static create function which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters. The actual constructor is private. This way it is ensured, that an *AudioTrackUid* object can only be created as a `std::shared_ptr`.

static std::shared_ptr<*AudioTrackUid*> **getSilent**(std::shared_ptr<*Document*> &document)
get a silent *AudioTrackUid* which can be referenced from AudioObjects

this will get an existing silent track from the document, or create a new one of none exists

static std::shared_ptr<*AudioTrackUid*> **getSilent**()
get a silent *AudioTrackUid* which can be referenced from AudioObjects

using this can lead to having multiple silent AudioTrackUids in a *Document*

class `adm::`**AudioTrackUidId**
Representation of an *AudioTrackUidId*.

Unnamed Group

bool **operator==**(const *AudioTrackUidId* &other) const
Operator overload.

Compares the string representation of the *AudioTrackUidId*.

bool **operator!=**(const *AudioTrackUidId* &other) const

bool **operator<**(const *AudioTrackUidId* &other) const

Public Types

typedef AudioTrackUidIdTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **AudioTrackUidId**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

AudioTrackUidId(AudioTrackUidIdValue)

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(AudioTrackUidIdValue value)
Set value.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**SampleRate** = detail::NamedType<unsigned int, SampleRateTag>
NamedType for the sampleRate element.

using adm::**BitDepth** = detail::NamedType<unsigned int, BitDepthTag>
NamedType for the bitDepth element.

7.5 AudioPackFormat

class adm::**AudioPackFormat** : public std::enable_shared_from_this<*AudioPackFormat*>
Class representation of the audioPackFormat ADM element.

Supported parameters are as follows:

ADM Parameter	Parameter Type	Pattern Type
audioPackFormatID	*AudioPackFormatId*	*RequiredParameter*
audioPackFormatName	*AudioPackFormatName*	*RequiredParameter*
importance	*Importance*	*OptionalParameter*
absoluteDistance	*AbsoluteDistance*	*OptionalParameter*
• typeDefinition • typeLabel	*TypeDescriptor*	*RequiredParameter*

Note that TypeDescriptor can only be set in the constructor.

Subclassed by *adm::AudioPackFormatHoa*

Public Types

typedef AudioPackFormatTag **tag**

typedef *AudioPackFormatId* **id_type**
Type that holds the id for this element;.

Public Functions

virtual **~AudioPackFormat**() = default

std::shared_ptr<*AudioPackFormat*> **copy**() const
Copy *AudioPackFormat*.

The actual copy constructor is private to ensure that an *AudioPackFormat* can only be created as a `std::shared_ptr`. This is not a deep copy! All referenced objects will be disconnected.

template<typename **Parameter**>

Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*AudioPackFormatId* id)
AudioPackFormatId setter.

void **set**(*AudioPackFormatName* name)
AudioPackFormatName setter.

void **set**(*Importance* importance)
Importance setter.

void **set**(*AbsoluteDistance* distance)
AbsoluteDistance setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

bool **addReference**(std::shared_ptr<*AudioChannelFormat*> channelFormat)
Add reference to an *AudioChannelFormat*.

bool **addReference**(std::shared_ptr<*AudioPackFormat*> packFormat)
Add reference to another *AudioPackFormat*.

template<typename **Element**>
ElementRange<const *Element*> **getReferences**() const
Get references to ADM elements template.

Templated get method with the ADM parameter type as template argument. Returns a set of all references to the ADM elements with the specified type.

template<typename **Element**>
ElementRange<*Element*> **getReferences**()
Get references to ADM elements template.

Templated get method with the ADM parameter type as template argument. Returns a set of all references to the ADM elements with the specified type.

void **removeReference**(std::shared_ptr<*AudioChannelFormat*> channelFormat)
Remove reference to an *AudioChannelFormat*.

void **removeReference**(std::shared_ptr<*AudioPackFormat*> packFormat)
Remove reference to an *AudioPackFormat*.

template<typename **Element**>
void **clearReferences**()
Clear references to Elements template.

Templated clear method with the ADM parameter type as template argument. Removes all references to the ADM elements with the specified type.

void **print**(std::ostream &os) const
Print overview to ostream.

const std::weak_ptr<*Document*> &**getParent**() const
Get *adm::Document* this element belongs to.

Public Static Functions

template<typename ...**Parameters**>
static std::shared_ptr<*AudioPackFormat*> **create**(*AudioPackFormatName* name, *TypeDescriptor* channelType, *Parameters*... optionalNamedArgs)
Static create function template.

Templated static create function which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters. The actual constructor is private. This way it is ensured, that an *AudioPackFormat* object can only be created as a `std::shared_ptr`.

class `adm::`**AudioPackFormatHoa** : public adm::*AudioPackFormat*, private AudioPackFormatHoaBase
Class representation of the audioPackFormat ADM element for HOA.

Supported parameters are those from *AudioPackFormat*, as well as:

ADM Parameter	Parameter Type	Pattern Type
normalization	*Normalization*	*DefaultParameter*
nfcRefDist	*NfcRefDist*	*DefaultParameter*
screenRef	*ScreenRef*	*DefaultParameter*

Public Functions

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>

bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

Public Static Functions

template<typename ...**Parameters**>
static std::shared_ptr<*AudioPackFormatHoa*> **create**(*AudioPackFormatName* name, *Parameters*... optionalNamedArgs)

class adm::**AudioPackFormatId**
Representation of an *AudioPackFormatId*.

Unnamed Group

bool **operator==**(const *AudioPackFormatId* &other) const
Operator overload.

Compares the string representation of the *AudioPackFormatId*.

bool **operator!=**(const *AudioPackFormatId* &other) const

bool **operator<**(const *AudioPackFormatId* &other) const

Public Types

typedef AudioPackFormatIdTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **AudioPackFormatId**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>

Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*TypeDescriptor* channelType)
Set channel type.

void **set**(AudioPackFormatIdValue value)
Set value.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**AudioPackFormatName** = detail::NamedType<std::string, AudioPackFormatNameTag>
NamedType for the audioPackFormatName attribute.

using adm::**AbsoluteDistance** = detail::NamedType<float, AbsoluteDistanceTag>
NamedType for the absoluteDistance attribute.

7.6 AudioChannelFormat

class adm::**AudioChannelFormat** : public std::enable_shared_from_this<*AudioChannelFormat*>
Class representation of the audioChannelFormat ADM element.

Supported parameters are as follows:

ADM Parameter	Parameter Type	Pattern Type
audioChannelFormatID	*AudioChannelFormatId*	*RequiredParameter*
audioChannelFormatName	*AudioChannelFormatName*	*RequiredParameter*
• typeDefinition • typeLabel	*TypeDescriptor*	*RequiredParameter*
frequency	*Frequency*	*OptionalParameter*

Note that TypeDescriptor can only be set in the constructor.

Public Types

typedef AudioChannelFormatTag **tag**

typedef *AudioChannelFormatId* **id_type**
Type that holds the id for this element;.

Public Functions

std::shared_ptr<*AudioChannelFormat*> **copy**() const
Copy *AudioChannelFormat*.

The actual copy constructor is private to ensure that an *AudioChannelFormat* can only be created as a `std::shared_ptr`. Added AudioBlockFormats will be copied too.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*AudioChannelFormatId* id)
AudioChannelFormatId setter.

void **set**(*AudioChannelFormatName* name)
AudioChannelFormatName setter.

void **set**(*Frequency* frequency)
Frequency setter.

template<typename **Parameter**>

void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **add**(*AudioBlockFormatDirectSpeakers* blockFormat)
Add AudioBlockFormats.

The AudioBlockFormat has to be of the correct type. Otherwise an exception is thrown.

void **add**(*AudioBlockFormatMatrix* blockFormat)

void **add**(*AudioBlockFormatObjects* blockFormat)

void **add**(*AudioBlockFormatHoa* blockFormat)

void **add**(*AudioBlockFormatBinaural* blockFormat)

template<typename **AudioBlockFormat**>
BlockFormatsConstRange<*AudioBlockFormat*> **getElements**() const
AudioBlockFormat elements getter template.

Templated getter with the wanted audioBlockFormat type as template argument.

Returns ContainerProxy containing all audioBlockFormats of the given type.

template<typename **AudioBlockFormat**>
BlockFormatsRange<*AudioBlockFormat*> **getElements**()
AudioBlockFormat elements getter template.

Templated getter with the wanted audioBlockFormat type as template argument.

Returns ContainerProxy containing all audioBlockFormats of the given type.

void **clearAudioBlockFormats**()
Clear AudioBlockFormats.

Removes all audioBlockFormats from the *AudioChannelFormat*

void **print**(std::ostream &os) const
Print overview to ostream.

const std::weak_ptr<*Document*> &**getParent**() const
Get *adm::Document* this element belongs to.

Public Static Functions

template<typename ...**Parameters**>
static std::shared_ptr<*AudioChannelFormat*> **create**(*AudioChannelFormatName* name, *TypeDescriptor* channelType, *Parameters*... optionalNamedArgs)
Static create function template.

Templated static create function which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters. The actual constructor is private. This way it is ensured, that an *AudioChannelFormat* object can only be created as a `std::shared_ptr`.

class adm::**AudioChannelFormatId**
Representation of an *AudioChannelFormatId*.

Unnamed Group

bool **operator==**(const *AudioChannelFormatId* &other) const
Operator overload.

Compares the string representation of the *AudioChannelFormatId*.

bool **operator!=**(const *AudioChannelFormatId* &other) const

bool **operator<**(const *AudioChannelFormatId* &other) const

Public Types

typedef AudioChannelFormatIdTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **AudioChannelFormatId**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*TypeDescriptor* channelType)
Set channel type.

void **set**(AudioChannelFormatIdValue value)
Set value.

template<typename **Parameter**>

void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**AudioChannelFormatName** = detail::NamedType<std::string, AudioChannelFormatNameTag>
NamedType for the audioChannelFormatName attribute.

class adm::**Frequency**
ADM parameter class to specify a frequency element in an audioChannelFormat.

Public Types

typedef FrequencyTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **Frequency**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*HighPass* frequency)
HighPass setter.

void **set**(*LowPass* frequency)
LowPass setter.

template<typename **Parameter**>

void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using **adm::LowPass** = detail::NamedType<float, LowPassTag, detail::MinValidator<0>>
NamedType for the low pass frequency.

Valid values are at least 0

using **adm::HighPass** = detail::NamedType<float, HighPassTag, detail::MinValidator<0>>
NamedType for the high pass frequency.

Valid values are at least 0

7.7 AudioStreamFormat

class **adm::AudioStreamFormat** : public std::enable_shared_from_this<*AudioStreamFormat*>
Class representation of the audioStreamFormat ADM element.

Notes on std::weak_ptr usage

Please note that the references to *AudioTrackFormat*s are represented using `std::weak_ptr`. Consequently, the method to retrieve those references is *AudioStreamFormat::getAudioTrackFormatReferences()* to make the difference to other `getReferences<Element>()` explicit.

The reason for using `std::weak_ptr` in the first place is because there's a cyclic reference between *AudioStreamFormat* and *AudioTrackFormat*. As it turns out, using the `std::weak_ptr` for *AudioTrackFormat* is much more convenient than the other way around. This is mostly due to the fact that otherwise there's no direct connection from e.g. *AudioTrackUid* to *AudioStreamFormat*, which would cause the latter to be removed if we would use the `std::weak_ptr` connection the other way around.

Furthermore, from the code we wrote so far, we can see that the *AudioTrackFormat* -> *AudioStreamFormat* connection is used quite often, while the *AudioStreamFormat* -> *AudioTrackFormat* has been virtually unused so far. Thus it was the obvious choice to make the first use case more convenient for users of the library.

Public Types

enum **ReferenceSyncOption**
Options to change AudioStreamFormat/AudioTrackFormat reference sync behaviour.

These options are to be used with `setReference` and `removeReference` and controls if *this AudioTrackFormat* is automatically added/removed as a reference to the *AudioStreamFormat* if it is referenced by *this*.

The default (and only) behaviour is to keep *AudioTrackFormat* and *AudioStreamFormat* in sync.

Future applications might want to use a more relaxed policy by providing (and implementing, and testing!) other options.

Values:

enumerator **sync_with_track_format**
commit all changes to *AudioStreamFormat*

typedef AudioStreamFormatTag **tag**

typedef *AudioStreamFormatId* **id_type**
Type that holds the id for this element;.

Public Functions

std::shared_ptr<*AudioStreamFormat*> **copy**() const
Copy *AudioStreamFormat*.

The actual copy constructor is private to ensure that an *AudioStreamFormat* can only be created as a `std::shared_ptr`. This is not deep copy! All referenced objects will be disconnected.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*AudioStreamFormatId* id)
AudioStreamFormatId setter.

void **set**(*AudioStreamFormatName* name)
AudioStreamFormatName setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **setReference**(std::shared_ptr<*AudioChannelFormat*> channelFormat)
Set reference to an *AudioChannelFormat*.

A pending unresolved reference will be removed.

void **setReference**(std::shared_ptr<*AudioPackFormat*> packFormat)
Set reference to an *AudioPackFormat*.

A pending unresolved reference will be removed.

bool **addReference**(std::weak_ptr<*AudioTrackFormat*> trackFormat, *ReferenceSyncOption* sync = *ReferenceSyncOption*::*sync_with_track_format*)

Add reference to an *AudioTrackFormat*.

Parameters

- **trackFormat** – which should be added.
- **sync** – controls if *this* will be automatically added to `trackFormat` as a reference as well.

template<typename **Element**>
std::shared_ptr<const *Element*> **getReference**() const

Get reference to ADM element template.

Templated get method with the ADM parameter type as template argument. Returns the reference to the ADM element with the specified type. If it is not set a nullptr will be returned.

template<typename **Element**>
std::shared_ptr<*Element*> **getReference**()

Get reference to ADM element template.

Templated get method with the ADM parameter type as template argument. Returns the reference to the ADM element with the specified type. If it is not set a nullptr will be returned.

ElementWeakRange<const *AudioTrackFormat*> **getAudioTrackFormatReferences**() const

Get references to AudioTrackFormats.

Returns a range of std::weak_ptr<const AudioTrackFormats>.

Note: Please read the class documentation of *adm::AudioStreamFormat* for the rationale behind using std::weak_ptr.

ElementWeakRange<*AudioTrackFormat*> **getAudioTrackFormatReferences**()

Get references to AudioTrackFormats.

Returns a range of std::weak_ptr<AudioTrackFormats>.

Note: Please read the class documentation of *adm::AudioStreamFormat* for the rationale behind using std::weak_ptr.

---

void **removeReference**(std::weak_ptr<*AudioTrackFormat*> trackFormat, *ReferenceSyncOption* sync = *ReferenceSyncOption*::*sync_with_track_format*)

Remove reference to an *AudioTrackFormat*.

Parameters

- **trackFormat** – reference which should be removed.
- **sync** – controls if *this* will be automatically added to `trackFormat` as a reference as well.

template<typename **Element**>
void **removeReference**()

Remove reference to an Element template.

Templated remove method with the ADM parameter type as template argument. Removes the reference to the ADM elements with the specified type.

template<typename **Element**>

void **clearReferences**()
Clear references to Elements template.

Templated clear method with the ADM parameter type as template argument. Removes all references to the ADM elements with the specified type.

void **print**(std::ostream &os) const
Print overview to ostream.

const std::weak_ptr<*Document*> &**getParent**() const
Get *adm::Document* this element belongs to.

Public Static Functions

template<typename ...**Parameters**>
static std::shared_ptr<*AudioStreamFormat*> **create**(*AudioStreamFormatName* name, FormatDescriptor format, *Parameters*... optionalNamedArgs)
Static create function template.

Templated static create function which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters. The actual constructor is private. This way it is ensured, that an *AudioStreamFormat* object can only be created as a `std::shared_ptr`.

class `adm::`**AudioStreamFormatId**
Representation of an *AudioStreamFormatId*.

Unnamed Group

bool **operator==**(const *AudioStreamFormatId* &other) const
Operator overload.

Compares the string representation of the *AudioStreamFormatId*.

bool **operator!=**(const *AudioStreamFormatId* &other) const

bool **operator<**(const *AudioStreamFormatId* &other) const

Public Types

typedef AudioStreamFormatIdTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **AudioStreamFormatId**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>

Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*TypeDescriptor* channelType)
Set channel type.

void **set**(AudioStreamFormatIdValue value)
Set value.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**AudioStreamFormatName** = detail::NamedType<std::string, AudioStreamFormatNameTag>
NamedType for the audioStreamFormatName attribute.

7.8 AudioTrackFormat

class adm::**AudioTrackFormat** : public std::enable_shared_from_this<*AudioTrackFormat*>
Class representation of the audioTrackFormat ADM element.

Public Types

enum **ReferenceSyncOption**
Options to change AudioStreamFormat/AudioTrackFormat reference sync behaviour.

These options are to be used with `setReference` and `removeReference` and controls if *this* *AudioTrackFormat* is automatically added/removed as a reference to the *AudioStreamFormat* if it is referenced by *this*.

The default (and only) behaviour is to keep *AudioTrackFormat* and *AudioStreamFormat* in sync.

Future applications might want to use a more relaxed policy by providing (and implementing, and testing!) other options.

Values:

enumerator **sync_with_stream_format**
commit all changes to *AudioStreamFormat*

typedef AudioTrackFormatTag **tag**

typedef *AudioTrackFormatId* **id_type**
Type that holds the id for this element;.

Public Functions

std::shared_ptr<*AudioTrackFormat*> **copy**() const
Copy *AudioTrackFormat*.

The actual copy constructor is private to ensure that an *AudioTrackFormat* can only be created as a `std::shared_ptr`. This is not deep copy! All referenced objects will be disconnected.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*AudioTrackFormatId* id)
AudioTrackFormatId setter.

void **set**(*AudioTrackFormatName* name)
AudioTrackFormatName setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **setReference**(std::shared_ptr<*AudioStreamFormat*> streamFormat, *ReferenceSyncOption* sync = *ReferenceSyncOption*::*sync_with_stream_format*)
Set reference to an *AudioStreamFormat*.

A pending unresolved reference will be removed.

Parameters

- **streamFormat** – which should be referenced.

- **sync** – controls if *this* will be automatically added to `streamFormat` as a reference as well.

template<typename **Element**>
std::shared_ptr<const *Element*> **getReference**() const
Get reference to ADM element template.

Templated get method with the ADM parameter type as template argument. Returns the reference to the ADM element with the specified type. If it is not set a nullptr will be returned.

template<typename **Element**>
std::shared_ptr<*Element*> **getReference**()
Get reference to ADM element template.

Templated get method with the ADM parameter type as template argument. Returns the reference to the ADM element with the specified type. If it is not set a nullptr will be returned.

template<typename **Element**>
void **removeReference**(*ReferenceSyncOption* sync = *ReferenceSyncOption*::*sync_with_stream_format*)
Remove reference to an Element template.

Templated remove method with the ADM parameter type as template argument. Removes the reference to the ADM elements with the specified type.

Parameters **sync** – controls if any reference to *this* will be automatically removed from any referenced `streamFormat` as well, **if is** . For other `Element` types (which do not exist), this option is ignored does nothing

void **print**(std::ostream &os) const
Print overview to ostream.

const std::weak_ptr<*Document*> &**getParent**() const
Get *adm::Document* this element belongs to.

Public Static Functions

template<typename ...**Parameters**>
static std::shared_ptr<*AudioTrackFormat*> **create**(*AudioTrackFormatName* name, FormatDescriptor format, *Parameters*... optionalNamedArgs)
Static create function template.

Templated static create function which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters. The actual constructor is private. This way it is ensured, that an *AudioTrackFormat* object can only be created as a `std::shared_ptr`.

class `adm::`**AudioTrackFormatId**
Representation of an *AudioTrackFormatId*.

Unnamed Group

bool **operator==**(const *AudioTrackFormatId* &other) const
Operator overload.

Compares the string representation of the *AudioTrackFormatId*.

bool **operator!=**(const *AudioTrackFormatId* &other) const

bool **operator<**(const *AudioTrackFormatId* &other) const

Public Types

typedef AudioTrackFormatIdTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **AudioTrackFormatId**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*TypeDescriptor* channelType)
Set channel type.

void **set**(AudioTrackFormatIdValue value)
Set value.

void **set**(AudioTrackFormatIdCounter counter)
Set counter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**AudioTrackFormatName** = detail::NamedType<std::string, AudioTrackFormatNameTag>
NamedType for the audioTrackFormatName attribute.

7.9 AudioBlockFormat

As the audioBlockFormat ADM elements are quite different for each typeDefinition there are five different `AudioBlockFormat` classes.

> **Warning:** The `Matrix` typeDefinition is not completely supported yet.

7.9.1 DirectSpeakers

class `adm::`**`AudioBlockFormatDirectSpeakers`** : private AudioBlockFormatDirectSpeakersBase

Class representation for ADM element audioBlockFormat if audioChannelFormat.typeDefinition == "DirectSpeakers".

Supported parameters are as follows:

ADM Parameter	Parameter Type	Pattern Type
audioBlockFormatId	*AudioBlockFormatId*	*RequiredParameter*
rtime	*Rtime*	*DefaultParameter*
duration	*Duration*	*OptionalParameter*
position	• *SpeakerPosition* • *SphericalSpeakerPosition* • *CartesianSpeakerPosition*	*VariantParameter* *RequiredParameter*
gain	*Gain*	*DefaultParameter*
importance	*Importance*	*DefaultParameter*
headLocked	*HeadLocked*	*DefaultParameter*
headphoneVirtualise	*HeadphoneVirtualise*	*DefaultParameter*
speakerLabel	*SpeakerLabels*	*VectorParameter*

Warning: not all methods are implemented for speakerLabel

Public Types

typedef AudioBlockFormatDirectSpeakersTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **AudioBlockFormatDirectSpeakers**(*Parameters*... optionalNamedArgs)

AudioBlockFormatDirectSpeakers(const *AudioBlockFormatDirectSpeakers*&) = default

AudioBlockFormatDirectSpeakers(*AudioBlockFormatDirectSpeakers*&&) = default

AudioBlockFormatDirectSpeakers &**operator=**(const *AudioBlockFormatDirectSpeakers*&) = default

AudioBlockFormatDirectSpeakers &**operator=**(*AudioBlockFormatDirectSpeakers*&&) = default

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

bool **isDefault**(detail::ParameterTraits<*Rtime*>::tag) const

void **set**(*AudioBlockFormatId* id)
AudioBlockFormatId setter.

void **set**(*Rtime* rtime)
Rtime setter.

void **set**(*Duration* duration)
Duration setter.

void **set**(*CartesianSpeakerPosition* speakerPosition)
CartesianSpeakerPosition setter.

void **set**(*SphericalSpeakerPosition* speakerPosition)
SphericalSpeakerPosition setter.

void **set**(*SpeakerPosition* speakerPosition)
SpeakerPosition setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

bool **add**(*SpeakerLabel* label)
Add a SpeakerLabel.

void **remove**(const *SpeakerLabel* &label)
remove a SpeakerLabel

using adm::**SpeakerLabel** = detail::NamedType<std::string, SpeakerLabelTag>
NamedType for a speaker label.

using adm::**SpeakerLabels** = std::vector<*SpeakerLabel*>
NamedType for all speaker labels of an AudioBlockFormat.

using adm::**SpeakerPosition** = boost::variant<*SphericalSpeakerPosition*, *CartesianSpeakerPosition*>
NamedType for speaker position in an AudioBlockFormat.

class adm::**CartesianSpeakerPosition**
ADM parameter class to specify a cartesian speaker position.

Public Types

typedef CartesianSpeakerPositionTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **CartesianSpeakerPosition**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>

bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(X x)
X setter.

void **set**(XMin xMin)
XMin setter.

void **set**(XMax xMax)
XMax setter.

void **set**(Y y)
Y setter.

void **set**(YMin yMin)
Y minimum setter.

void **set**(YMax yMax)
YMax setter.

void **set**(Z z)
Z setter.

void **set**(ZMin zMin)
ZMin setter.

void **set**(ZMax zMax)
ZMax setter.

void **set**(*ScreenEdgeLock* screenEdgeLock)
ScreenEdgeLock setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

class adm::**SphericalSpeakerPosition**
ADM parameter class to specify a spherical speaker position.

Public Types

typedef SphericalSpeakerPositionTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **SphericalSpeakerPosition**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(Azimuth azimuth)
Azimuth setter.

void **set**(AzimuthMin azimuthMin)
AzimuthMin setter.

void **set**(AzimuthMax azimuthMax)
AzimuthMax setter.

void **set**(Elevation elevation)
Elevation setter.

void **set**(ElevationMin elevationMin)
Elevation minimum setter.

void **set**(ElevationMax elevationMax)
ElevationMax setter.

void **set**(Distance distance)
Distance setter.

void **set**(DistanceMin distanceMin)
DistanceMin setter.

void **set**(DistanceMax distanceMax)
DistanceMax setter.

void **set**(*ScreenEdgeLock* screenEdgeLock)
ScreenEdgeLock setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

7.9.2 Matrix

class adm::**AudioBlockFormatMatrix** : private AudioBlockFormatMatrixBase
Class representation for ADM element audioBlockFormat if audioChannelFormat.typeDefinition == "Matrix".

Warning: This class has unsupported parameters

- encodePackFormatIDRef
- decodePackFormatIDRef
- inputPackFormatIDRef
- outputPackFormatIDRef

Public Types

typedef AudioBlockFormatMatrixTag **tag**

typedef *AudioBlockFormatId* **id_type**
Type that holds the id for this element;.

Public Functions

template<typename ...**Parameters**>
explicit **AudioBlockFormatMatrix**(*Parameters*... optionalNamedArgs)

AudioBlockFormatMatrix(const *AudioBlockFormatMatrix*&) = default

AudioBlockFormatMatrix(*AudioBlockFormatMatrix*&&) = default

AudioBlockFormatMatrix &**operator=**(const *AudioBlockFormatMatrix*&) = default

AudioBlockFormatMatrix &**operator=**(*AudioBlockFormatMatrix*&&) = default

template<typename **Parameter**>

Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

bool **isDefault**(detail::ParameterTraits<*Rtime*>::tag) const

void **set**(*AudioBlockFormatId* id)
AudioBlockFormatId setter.

void **set**(*Rtime* rtime)
Rtime setter.

void **set**(*Duration* duration)
Duration setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

7.9.3 Objects

class adm::**AudioBlockFormatObjects** : private AudioBlockFormatObjectsBase
Class representation for ADM element audioBlockFormat if audioChannelFormat.typeDefinition == "Objects".

Supported parameters are as follows:

ADM Parameter	Parameter Type	Pattern Type
audioBlockFormatId	*AudioBlockFormatId*	*RequiredParameter*
rtime	*Rtime*	*DefaultParameter*
duration	*Duration*	*OptionalParameter*
cartesian	*Cartesian*	custom, see below
position	• *Position* • *SphericalPosition* • *CartesianPosition*	
width	*Width*	*DefaultParameter*
height	*Height*	*DefaultParameter*
depth	*Depth*	*DefaultParameter*
diffuse	*Diffuse*	*DefaultParameter*
gain	*Gain*	*DefaultParameter*
importance	*Importance*	*DefaultParameter*
headLocked	*HeadLocked*	*DefaultParameter*
headphoneVirtualise	*HeadphoneVirtualise*	*DefaultParameter*
screenEdgeLock	*ScreenEdgeLock*	*OptionalParameter*
channelLock	*ChannelLock*	*DefaultParameter*
objectDivergence	*ObjectDivergence*	*DefaultParameter*
jumpPosition	*JumpPosition*	*DefaultParameter*
screenRef	*ScreenRef*	*DefaultParameter*

`cartesian` and `position` attributes are linked; see *void set(Cartesian)*, *void set(Position)*, *void set(CartesianPosition)* and *void set(SphericalPosition)*.

Warning: This class has unsupported parameters

- ZoneExclusion

Public Types

typedef AudioBlockFormatObjectsTag **tag**

typedef *AudioBlockFormatId* **id_type**
Type that holds the id for this element;.

Public Functions

template<typename ...**Parameters**>
explicit **AudioBlockFormatObjects**(*CartesianPosition* position, *Parameters*... optionalNamedArgs)

template<typename ...**Parameters**>
explicit **AudioBlockFormatObjects**(*SphericalPosition* position, *Parameters*... optionalNamedArgs)

AudioBlockFormatObjects(const *AudioBlockFormatObjects*&) = default

AudioBlockFormatObjects(*AudioBlockFormatObjects*&&) = default

AudioBlockFormatObjects &**operator=**(const *AudioBlockFormatObjects*&) = default

AudioBlockFormatObjects &**operator=**(*AudioBlockFormatObjects*&&) = default

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*Cartesian* cartesian)
Cartesian setter.

Note: If setting the ADM parameter cartesian will change the coordinate system the corresponding position will be set to a default value.

---

void **set**(*Position* position)
Position setter.

Note: Setting a *SphericalPosition* will automatically unset the *CartesianPosition* and the other way around. If a *CartesianPosition* is set the Cartesian flag will be set too. If a *SphericalPosition* is set the Cartesian flag will only be set if has already been set before (either directly or automatically).

---

void **set**(*SphericalPosition* position)
SphericalPosition setter.

Note: Setting a *SphericalPosition* will automatically unset the *CartesianPosition*. The Cartesian flag will only be set if it has already been set before (either directly or automatically).

---

void **set**(*CartesianPosition* position)
CartesianPosition setter.

Note: Setting a *CartesianPosition* will automatically unset the *SphericalPosition*. Also the Cartesian flag will be set automatically.

---

void **set**(*ScreenEdgeLock* screenEdgeLock)
ScreenEdgeLock setter.

void **set**(*ChannelLock* channelLock)
ChannelLock setter.

void **set**(*ObjectDivergence* objectDivergence)
ObjectDivergence setter.

void **set**(*JumpPosition* jumpPosition)
JumpPosition setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

using adm::**Cartesian** = detail::NamedType<bool, CartesianTag>
NamedType for cartesian parameter.

typedef boost::variant<*SphericalPosition*, *CartesianPosition*> adm::**Position**
Type to hold a *SphericalPosition* or *CartesianPosition*.

class adm::**SphericalPosition**
ADM parameter class to specify a spherical position.

Public Types

typedef SphericalPositionTag **tag**

Public Functions

explicit **SphericalPosition**(Azimuth azimuth = Azimuth(0.f), Elevation elevation = Elevation(0.f))
Constructor without optional parameters.

template<typename ...**Parameters**>
SphericalPosition(Azimuth azimuth, Elevation elevation, *Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(Azimuth azimuth)
Azimuth setter.

void **set**(Elevation elevation)
Elevation setter.

void **set**(Distance distance)
Distance setter.

void **set**(*ScreenEdgeLock* screenEdgeLock)
ScreenEdgeLock setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

class adm::**CartesianPosition**
ADM parameter class to specify a cartesian position.

Public Types

typedef CartesianPositionTag **tag**

Public Functions

explicit **CartesianPosition**(X x = X(0.f), Y y = Y(1.f))
Constructor without optional parameters.

template<typename ...**Parameters**>
CartesianPosition(X x, Y y, *Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>

bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(X x)
X setter.

void **set**(Y y)
Y setter.

void **set**(Z z)
Z setter.

void **set**(*ScreenEdgeLock* screenEdgeLock)
ScreenEdgeLock setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**Width** = detail::NamedType<float, WidthTag>
NamedType for width parameter.

using adm::**Height** = detail::NamedType<float, HeightTag>
NamedType for height parameter.

using adm::**Depth** = detail::NamedType<float, DepthTag>
NamedType for depth parameter.

class adm::**ScreenEdgeLock**
ADM parameter class to specify a screen edge lock.

Public Types

typedef ScreenEdgeLockTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **ScreenEdgeLock**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*ScreenEdge* screenEdge)
ScreenEdge setter.

void **set**(*HorizontalEdge* horizontalEdge)
HorizontalEdge setter.

void **set**(*VerticalEdge* verticalEdge)
VerticalEdge setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**ScreenEdge** = detail::NamedType<std::string, ScreenEdgeTag, detail::ScreenEdgeValidator>
NamedType for the screen edge.

Valid values are “left”, “right”, “top” and “bottom”

using adm::**HorizontalEdge** = detail::NamedType<std::string, HorizontalEdgeTag,
detail::HorizontalEdgeValidator>
NamedType for the horizontal screen edge.

Valid values are “left” and “right”

using adm::**VerticalEdge** = detail::NamedType<std::string, VerticalEdgeTag, detail::VerticalEdgeValidator>
NamedType for the vertical screen edge.

Valid values are “top” and “bottom”

using adm::**Diffuse** = detail::NamedType<float, DiffuseTag, detail::RangeValidator<0, 1>>
NamedType for diffuse parameter.

class adm::**ChannelLock**
ADM parameter class to specify a channel lock.

Public Types

typedef ChannelLockTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **ChannelLock**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*ChannelLockFlag* channelLockFlag)
ChannelLockFlag setter.

void **set**(*MaxDistance* maxDistance)
MaxDistance setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**ChannelLockFlag** = detail::NamedType<bool, ChannelLockFlagTag>
NamedType for channelLockFlag parameter.

using adm::**MaxDistance** = detail::NamedType<float, MaxDistanceTag, detail::RangeValidator<0, 2>>
NamedType for the maxDistance attribute.

Valid values are in the range [0, 2].

class adm::**ObjectDivergence**
ADM parameter class to specify the object divergence.

Public Types

typedef ObjectDivergenceTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **ObjectDivergence**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(Divergence divergence)
Divergence setter.

void **set**(AzimuthRange azimuthRange)
AzimuthRange setter.

void **set**(PositionRange positionRange)
PositionRange setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

class adm::**JumpPosition**
ADM parameter class to specify a jump position.

Public Types

typedef JumpPositionTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **JumpPosition**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*JumpPositionFlag* objectDivergenceFlag)
JumpPositionFlag setter.

void **set**(*InterpolationLength* interpolationLength)
InterpolationLength setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**JumpPositionFlag** = detail::NamedType<bool, JumpPositionFlagTag>
NamedType for jumpPositionFlag parameter.

using adm::**InterpolationLength** = detail::NamedType<std::chrono::nanoseconds, interpolationLengthTag>
NamedType for interpolationLength parameter.

using adm::**ScreenRef** = detail::NamedType<bool, ScreenRefTag>
NamedType for screenref parameter.

7.9.4 HOA

class adm::**AudioBlockFormatHoa** : private AudioBlockFormatHoaBase
Class representation for ADM element audioBlockFormat if audioChannelFormat.typeDefinition == "HOA".

Supported parameters are as follows:

ADM Parameter	Parameter Type	Pattern Type
audioBlockFormatId	*AudioBlockFormatId*	*RequiredParameter*
rtime	*Rtime*	*DefaultParameter*
duration	*Duration*	*OptionalParameter*
order	*Order*	*RequiredParameter*
degree	*Degree*	*RequiredParameter*
normalization	*Normalization*	*DefaultParameter*
nfcRefDist	*NfcRefDist*	*DefaultParameter*
gain	*Gain*	*DefaultParameter*
importance	*Importance*	*DefaultParameter*
headLocked	*HeadLocked*	*DefaultParameter*
headphoneVirtualise	*HeadphoneVirtualise*	*DefaultParameter*
screenRef	*ScreenRef*	*DefaultParameter*
equation	*Equation*	*OptionalParameter*

Public Types

typedef AudioBlockFormatHoaTag **tag**

typedef *AudioBlockFormatId* **id_type**
Type that holds the id for this element;.

Public Functions

template<typename ...**Parameters**>
AudioBlockFormatHoa(*Order* order, *Degree* degree, *Parameters*... optionalNamedArgs)

AudioBlockFormatHoa(const *AudioBlockFormatHoa*&) = default

AudioBlockFormatHoa(*AudioBlockFormatHoa*&&) = default

AudioBlockFormatHoa &**operator=**(const *AudioBlockFormatHoa*&) = default

AudioBlockFormatHoa &**operator=**(*AudioBlockFormatHoa*&&) = default

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

using **adm::Order** = detail::NamedType<int, OrderTag>
NamedType for order parameter.

using **adm::Degree** = detail::NamedType<int, DegreeTag>
NamedType for degree parameter.

using **adm::Normalization** = detail::NamedType<std::string, NormalizationTag>
NamedType for a normalization parameter, defaulting to SN3D.

using **adm::NfcRefDist** = detail::NamedType<float, NfcRefDistTag>
NamedType for degree parameter.

using adm::**Equation** = detail::NamedType<std::string, EquationTag>
NamedType for a equation parameter.

7.9.5 Binaural

class adm::**AudioBlockFormatBinaural** : private AudioBlockFormatBinauralBase
Class representation for ADM element audioBlockFormat if audioChannelFormat.typeDefinition == "Binaural".

Supported parameters are as follows:

ADM Parameter	Parameter Type	Pattern Type
audioBlockFormatId	*AudioBlockFormatId*	*RequiredParameter*
rtime	*Rtime*	*DefaultParameter*
duration	*Duration*	*OptionalParameter*
gain	*Gain*	*DefaultParameter*
importance	*Importance*	*DefaultParameter*

Public Types

typedef AudioBlockFormatBinauralTag **tag**

typedef *AudioBlockFormatId* **id_type**
Type that holds the id for this element;.

Public Functions

template<typename ...**Parameters**>
explicit **AudioBlockFormatBinaural**(*Parameters*... optionalNamedArgs)

AudioBlockFormatBinaural(const *AudioBlockFormatBinaural*&) = default

AudioBlockFormatBinaural(*AudioBlockFormatBinaural*&&) = default

AudioBlockFormatBinaural &**operator=**(const *AudioBlockFormatBinaural*&) = default

AudioBlockFormatBinaural &**operator=**(*AudioBlockFormatBinaural*&&) = default

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>

bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*AudioBlockFormatId* id)
AudioBlockFormatId setter.

void **set**(*Rtime* rtime)
Rtime setter.

void **set**(*Duration* duration)
Duration setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

7.10 Shared Elements

using adm::**Start** = detail::NamedType<Time, StartTag>
NamedType for the start attribute.

class adm::**LoudnessMetadata**

Public Types

typedef LoudnessMetadataTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **LoudnessMetadata**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(LoudnessMethod loudnessMethod)
LoudnessMethod setter.

void **set**(LoudnessRecType loudnessRecType)
LoudnessRecType setter.

void **set**(LoudnessCorrectionType loudnessLoudnessCorrectionType)
LoudnessCorrectionType setter.

void **set**(IntegratedLoudness integratedLoudness)
IntegratedLoudness setter.

void **set**(LoudnessRange loudnessRange)
LoudnessRange setter.

void **set**(MaxTruePeak maxTruePeak)
MaxTruePeak setter.

void **set**(MaxMomentary maxMomentary)
MaxMomentary setter.

void **set**(MaxShortTerm maxShortTerm)
MaxShortTerm setter.

void **set**(DialogueLoudness dialogueLoudness)
DialogueLoudness setter.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**LoudnessMetadatas** = std::vector<*LoudnessMetadata*>

using adm::**Importance** = detail::NamedType<int, ImportanceTag, detail::RangeValidator<0, 10>>
NamedType for the importance parameter.

Valid values are in the range [0, 10].

class adm::**AudioBlockFormatId**
Representation of an *AudioBlockFormatId*.

Unnamed Group

bool **operator==**(const *AudioBlockFormatId* &other) const
Operator overload.

Compares the string representation of the *AudioBlockFormatId*.

bool **operator!=**(const *AudioBlockFormatId* &other) const

bool **operator<**(const *AudioBlockFormatId* &other) const

Public Types

typedef AudioBlockFormatIdTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **AudioBlockFormatId**(*Parameters*... optionalNamedArgs)
Constructor template.

Templated constructor which accepts a variable number of ADM parameters in random order after the mandatory ADM parameters.

template<typename **Parameter**>
Parameter **get**() const
ADM parameter getter template.

Templated getter with the wanted ADM parameter type as template argument. If currently no value is available trying to get the adm parameter will result in an exception. Check with the has method before

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the ADM parameter is set or has a default value.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the ADM parameter is the default value.

void **set**(*TypeDescriptor* channelType)
Set channel type.

void **set**(AudioBlockFormatIdValue value)
Set value.

void **set**(AudioBlockFormatIdCounter counter)
Set counter.

template<typename **Parameter**>

void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

void **print**(std::ostream &os) const
Print overview to ostream.

using adm::**Rtime** = detail::NamedType<Time, RtimeTag>
NamedType for the rtime attribute.

using adm::**Duration** = detail::NamedType<Time, DurationTag>
NamedType for the duration attribute.

using adm::**HeadLocked** = detail::NamedType<bool, HeadLockedTag>
NamedType for the headLocked parameter.

class adm::**HeadphoneVirtualise** : private HeadphoneVirtualiseBase

Public Types

typedef HeadphoneVirtualiseTag **tag**

Public Functions

template<typename ...**Parameters**>
explicit **HeadphoneVirtualise**(*Parameters*... optionalNamedArgs)

template<typename **Parameter**>
Parameter **get**() const
ADM parameter get template.

template<typename **Parameter**>
bool **has**() const
ADM parameter has template.

Templated has method with the ADM parameter type as template argument. Returns true if the parameter has been set or is optional.

template<typename **Parameter**>
bool **isDefault**() const
ADM parameter isDefault template.

Templated isDefault method with the ADM parameter type as template argument. Returns true if the optional parameter has been changed from its default value.

template<typename **Parameter**>
void **unset**()
ADM parameter unset template.

Templated unset method with the ADM parameter type as template argument. Removes an ADM parameter if it is optional or resets it to the default value if there is one.

class adm::**Gain**
Storage for gain parameters, which can either be a linear or dB value.

Gain objects should be constructed using *fromLinear()* or *fromDb()*.

Public Types

typedef GainTag **tag**

Public Functions

inline explicit **Gain**(double linearGain)
alias for *Gain::fromLinear* for compatibility with existing code

inline double **asLinear**() const
Get the gain as a linear value, converting if necessary.

inline double **asDb**() const
Get the gain as a value in dB, converting if necessary.

inline bool **isLinear**() const
Is this gain stored as a linear value?

inline bool **isDb**() const
Is this gain stored in dB?

inline double **get**() const
alias for asLinear for compatibility with existing code

inline double **operator***() const
alias for asLinear for compatibility with existing code

Public Static Functions

static inline *Gain* **fromLinear**(double linearGain)
Make a linear *Gain*.

static inline *Gain* **fromDb**(double dbGain)
Make a *Gain* in dB.

class adm::**Label** : private LabelBase, private detail::AddWrapperMethods<*Label*>
Represents an ADM label element, with optional language attribute.

Public Types

using **tag** = LabelTag

Public Functions

template<typename ...**Parameters**>
inline explicit **Label**(*Parameters*... namedArgs)

inline explicit **Label**(std::string str)

explicit **Label**(const char *s)

void **print**(std::ostream &os) const

Friends

friend class detail::AddWrapperMethods< Label >

using adm::**Labels** = std::vector<*Label*>
Represents multiple labels with different languages.

using adm::**TypeDescriptor** = detail::NamedType<int, TypeDescriptorTag, detail::RangeValidator<0, 5>>
Combined ADM typeLabel and typeDefinition.

Valid values are in the range [0, 5]

See *TypeDefinition* A list of pre-defined type definitions

namespace adm::**TypeDefinition**
BS.2076 type definitions.

Variables

const *TypeDescriptor* **UNDEFINED**

const *TypeDescriptor* **DIRECT_SPEAKERS**
TypeDefinition for Direct Speakers.

const *TypeDescriptor* **MATRIX**
TypeDefinition for Matrix.

const *TypeDescriptor* **OBJECTS**
TypeDefinition for Objects.

const *TypeDescriptor* **HOA**
TypeDefinition for Higher Order Ambisonics.

const *TypeDescriptor* **BINAURAL**
TypeDefinition for Binaural.

CHAPTER

EIGHT

HELPERS

class adm::**Route**

Route along ADM elements within an ADM document.

Describes route along ADM elements within an ADM document, i.e. which references were followed to reach a certain element from a given start point.

This is the same purpose (and basic interface) as *adm::Path*, but currently the implementation and constraints are different.

Thus, the name *adm::Route* has been chosen to distinguish between those two.

The main/essential difference is that the *adm::Route* stores pointers to the ADM elements, to make it easy to access the elements after a *adm::Route* has been created, while the *adm::Path* does not, as they might become invalid. Instead the *adm::Path* stores the (coded) element ids, which are also used to check if *adm::Path*s are equal or not.

To easily create an *adm::Route* you may use the *adm::RouteTracer*.

Routes are ordered first by the hash of their elements, then by the elements themselves.

Public Types

typedef std::size_t **hash_type**

typedef std::vector<adm::ElementConstVariant>::iterator **iterator**

typedef std::vector<adm::ElementConstVariant>::const_iterator **const_iterator**

typedef adm::ElementConstVariant **value_type**

Public Functions

Route() = default

template<typename **Element**>
inline void **add**(std::shared_ptr<*Element*> element)

inline void **add**(adm::ElementConstVariant element)

template<typename **Element**>

std::shared_ptr<const *Element*> **getFirstOf**() const

template<typename **Element**>
std::shared_ptr<const *Element*> **getLastOf**() const

template<typename **Element**>
const_iterator **findFirstOf**() const

template<typename **Element**>
const_iterator **findLastOf**() const

inline *const_iterator* **begin**() const

inline *const_iterator* **end**() const

inline const adm::ElementConstVariant &**front**() const

inline const adm::ElementConstVariant &**back**() const

inline std::size_t **size**() const

inline *hash_type* **hash**() const
Get hash value.

class **adm::Path**
Path along ADM elements within an ADM document.

Describes route along ADM elements within an ADM document, i.e. which references were followed to reach a certain element from a given start point.

This is the same purpose (and basic interface) as *adm::Route*, but currently the implementation and constraints are different.

Thus, the name *adm::Path* has been chosen to distinguish between those two.

The main/essential difference is that the *adm::Route* stores pointers to the ADM elements, to make it easy to access the elements after a *adm::Route* has been created, while the *adm::Path* does not, as they might become invalid. Instead the *adm::Path* stores the (coded) element ids, which are also used to check if *adm::Path*s are equal or not.

Public Types

typedef std::size_t **hash_type**

typedef std::vector<ElementIdVariant>::iterator **iterator**

typedef std::vector<ElementIdVariant>::const_iterator **const_iterator**

Public Functions

template<typename **AdmId**>
inline void **add**(*AdmId* id)

template<typename **Element**>
inline void **add**(std::shared_ptr<const *Element*> element)

void **add**(ElementConstVariant elementVariant)

void **add**(ElementIdVariant elementIdVariant)

inline *hash_type* **hash**() const

inline *const_iterator* **begin**() const

inline *const_iterator* **end**() const

inline ElementIdVariant **front**() const

inline ElementIdVariant **back**() const

inline std::size_t **size**() const

template<typename **AdmElementId**>
AdmElementId **getFirstOf**() const

template<typename **AdmElementId**>
AdmElementId **getLastOf**() const

template<typename **AdmElementId**>
bool **hasIdType**() const

using **adm::RouteTracer** = detail::GenericRouteTracer<*Route*, detail::DefaultFullDepthStrategy>
Creates *adm::Route*s.

This implementation traces the following route:

- audioProgramme
- audioContent
- audioObject
- audioPackFormat
- audioChannelFormat

Complementary AudioObjects are not interpreted as such. Hence for every complementary audioObject an *Route* will be returned.

> **Warning:** If the ADM structure contains a reference cycle, trace will get stuck in an infinite loop.

template<typename **Property**, typename **ElementPtr**>
Property adm::**getPropertyOr**(*ElementPtr* element, *Property* defaultValue)

Return value of a property or a given defaultValue if the property is not available.

This is just a more expressive shorthand equivalent to

```
element->has<Property>() ? element->get<Property>() : defaultValue;
```

CHAPTER

NINE

UTILITIES

9.1 Object Creation

struct adm::**SimpleObjectHolder**
Simple holder used as return type for *createSimpleObject()* and *addSimpleObjectTo()*.

Gives access to all elements created by *createSimpleObject()*. This exists basically to give quick and convenient direct access to the elements after creation.

Public Members

std::shared_ptr<*AudioObject*> **audioObject**

std::shared_ptr<*AudioPackFormat*> **audioPackFormat**

std::shared_ptr<*AudioChannelFormat*> **audioChannelFormat**

std::shared_ptr<*AudioStreamFormat*> **audioStreamFormat**

std::shared_ptr<*AudioTrackFormat*> **audioTrackFormat**

std::shared_ptr<*AudioTrackUid*> **audioTrackUid**

SimpleObjectHolder adm::**createSimpleObject**(const std::string &name)
Create *AudioObject* hierarchie for single *TypeDefinition::OBJECTS*-type element.

Creates an *AudioObject* including referenced *AudioPackFormat* and *AudioChannelFormat* of type *TypeDefinition::OBJECTS*, as well an *AudioTrackUid*, the referenced *AudioTrackFormat* and *AudioStreamFormat* of type FormatDefinition::PCM.

Parameters **name** – Name that will be used for the created Audio{Object,PackFormat, ChannelFormat}.

SimpleObjectHolder adm::**addSimpleObjectTo**(std::shared_ptr<*Document*> document, const std::string &name)
Create and add *AudioObject* hierarchie for single *TypeDefinition::OBJECTS*-type element.

same as createSimpleObject, but the elements are automatically added to the given document

struct adm::**SimpleCommonDefinitionsObjectHolder**
Simple holder used as return type for *addSimpleCommonDefinitionsObjectTo()*.

Gives access to all elements created by *addSimpleCommonDefinitionsObjectTo()*. This exists basically to give quick and convenient direct access to the elements after creation.

Public Members

std::shared_ptr<*AudioObject*> **audioObject**

std::map<std::string, std::shared_ptr<*AudioTrackUid*>> **audioTrackUids**
a mapping from the short speaker label (e.g. M+000) to the audioTrackUid

SimpleCommonDefinitionsObjectHolder adm::**addSimpleCommonDefinitionsObjectTo**(std::shared_ptr<*Document*> document, const std::string &name, const std::string &speakerLayout)

Create and add *AudioObject* with common definitions direct speakers channel bed to document.

Creates an *AudioObject* and corresponding AudioTrackUids and connects it to the common definition ADM elements for the given speaker layout. The created ADM elements are added to the given document.

See adm::audioPackFormatLookupTable

Note: The document must already have the common definition elements added.

Parameters

- **document** – The document where the *AudioObject* and the AudioTrackUids should be added to and whose common definition ADM elements should be used.
- **name** – Name that will be used for the created *AudioObject*.
- **speakerLayout** – Speaker layout which will be created. For possible values

SimpleCommonDefinitionsObjectHolder adm::**addTailoredCommonDefinitionsObjectTo**(std::shared_ptr<*Document*> document, const std::string &name, const adm::*AudioPackFormatId* packFormatId, const std::vector<adm::*AudioTrackFormatId*> &trackFormatIds, const std::vector<std::string> &speakerLabels)

Create and add *AudioObject* with common definitions direct speakers channel bed to document with a given channel order.

Creates an *AudioObject* and corresponding AudioTrackUids and connects it to the common definition ADM elements for the given speaker layout. The created ADM elements are added to the given document.

Note: The document must already have the common definition elements added.

Parameters

- **document** – The document where the *AudioObject* and the AudioTrackUids should be added to and whose common definition ADM elements should be used.

- **name** – Name that will be used for the created *AudioObject*.
- **packFormatId** – *AudioPackFormatId* of the given layout.
- **trackFormatIds** – AudioTrackFormatIds of all the speakers in the layout.
- **speakerLabels** – Labels of all the speakers in the layout.

9.2 audioBlockFormat timing fixes

void adm::**updateBlockFormatDurations**(std::shared_ptr<*Document*> document)
Set or update durations of all AudioBlockFormats

This function provides essentially the same functionality as *adm::updateBlockFormatDurations*(std::shared_ptr<Document>, std::chrono::nanoseconds), with the only difference that the duration of the *AudioProgramme* will be to determine the lifetime of *AudioObject*s.

See void *updateBlockFormatDurations*(std::shared_ptr<Document>, std::chrono::nanoseconds)

Parameters **document** – The document to update, durations will be adapted in-place.

void adm::**updateBlockFormatDurations**(std::shared_ptr<*Document*> document, const Time &fileLength)
Set or update durations of all AudioBlockFormats

If an *AudioChannelFormat* has multiple AudioBlockFormats, all of them should have an rtime and a duration.

As these durations might be linked to the duration of referencing *AudioObject*s, the length of the parent *AudioProgramme* and/or the length of a BW64 file, it is hard or impossible to set the correct duration during AudioBlockFormat creation.

This utility function will update the AudioBlockFormat durations to match the lifetime of the referencing *AudioObject*(s) or, if not set, the length of the audio file given by fileLength.

An exception will be raised if there's any ambiguity in the resulting duration, for example due to multiple *AudioObject*s with different durations referencing the same *AudioChannelFormat*. Differences between the duration of an *AudioProgramme* and fileLength will also be considered an error. If one of those error conditions is met (and an exception is raised), the *adm::Document* will remain unchanged.

See void *updateBlockFormatDurations(std::shared_ptr<Document>)*

Parameters

- **document** – The document to update, durations will be adapted in-place.
- **fileLength** – The length of the BW64 audio file

CHAPTER

TEN

READ AND WRITE XML

enum **ParserOptions**
Representation of available options to influence the behaviour of the XML parser.

`ParserOptions` satisfies the requirements of BitmaskType.

This means that the bitwise operators (e.g. `operator|` or `operator&`) are defined for this type. Thus options may be combined by `OR`-ing the respective values.

Note: No options have been implemented so far. As soon as this is done, provide an usage example, a list describing the member constants (options) and which options can be combined. Refer to std::filesystem::copy_options for an example.

Values:

enumerator **none**
default behaviour

enumerator **recursive_node_search**
recursively search whole xml for audioFormatExtended node

enum **WriterOptions**
Representation of available options to influence the behaviour of the XML writer.

`WriterOptions` satisfies the requirements of BitmaskType.

This means that the bitwise operators (e.g. `operator|` or `operator&`) are defined for this type. Thus options may be combined by `OR`-ing the respective values.

Available options

At most one writer option in each of the following options groups may be present, otherwise the behavior is undefined.

Constant	Meaning
	options controlling the XML envelope
none	use <ebuCoreMain> envelope (default)
itu_structure	use <ituADM> to contain the ADM elements
	options controlling default values
none	use <ebuCoreMain> envelope (default)
write_default_values	use <ebuCoreMain> envelope (default)

Values:

enumerator **none**
default behaviour

enumerator **itu_structure**
use ITU xml structure

enumerator **write_default_values**
write default values

std::shared_ptr<Document> **parseXml**(const std::string &filename, xml::ParserOptions options = xml::ParserOptions::none)
Parse an XML representation of the Audio Definition Model.

Convenience wrapper for files using `parseXml(std::istream&)`

Parameters

- **filename** – XML file to read and parse
- **options** – Options to influence the XML parser behaviour

std::shared_ptr<Document> **parseXml**(std::istream &stream, xml::ParserOptions options = xml::ParserOptions::none)
Parse an XML representation of the Audio Definition Model.

Parse adm data from an `std::istream`.

Parameters

- **stream** – input stream to parse XML data
- **options** – Options to influence the XML parser behaviour

void **writeXml**(const std::string &filename, std::shared_ptr<const Document> admDocument, xml::WriterOptions options = xml::WriterOptions::none)
Write an *Document*.

Convenience wrapper for files using `writeXml(std::ostream&, std::shared_ptr<const Document>)`

Parameters

- **filename** – XML file to write to
- **admDocument** – ADM document that should be transformed into XML
- **options** – Options to influence the XML generator behaviour

std::ostream &**writeXml**(std::ostream &stream, std::shared_ptr<const Document> admDocument,
xml::WriterOptions options = xml::WriterOptions::none)

Write an *Document* to an output stream.

Parameters

- **stream** – output stream to write XML data
- **admDocument** – ADM document that should be transformed into XML
- **options** – Options to influence the XML generator behaviour

The libadm library is a modern C++11 library to parse, modify, create and write Recommendation ITU-R BS.2076-1 conform XML document. It works well with the header-only library libbw64 to write ADM related applications with minimal dependencies.

CHAPTER

ELEVEN

FEATURES

- minimal dependencies
- expressive syntax
- easy access to referenced ADM elements
- common definitions support

CHAPTER

TWELVE

ACKNOWLEDGEMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687645.

INDEX

A

P

W