ISO/IEC JTC 1/SC 29/WG 03 N0446

**ISO/IEC JTC 1/SC 29/WG 03  
MPEG Systems   
Convenorship: KATS (Korea, Republic of)**

**Document type:** Output Document

**Title:** Procedures for standard development and software of ISO/IEC 23090-13

**Status:** Approved

**Date of document:** 2022-01-21

**Source:** ISO/IEC JTC 1/SC 29/WG 03

**No. of pages:** 11 (with cover page)

**Email of Convenor:** young.L @ samsung . com

**Committee URL:** <https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg3>

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 03 MPEG SYSTEMS**

**ISO/IEC JTC 1/SC 29/WG 03 N0446**

**January 20222, Virtual**

|  |  |
| --- | --- |
| **Title** | **Procedures for standard development and software of ISO/IEC 23090-13** |
| **Source** | **WG 03, MPEG Systems** |
| **Status** | **Approved** |
| **Serial Number** | **21162** |

# Scope

This document provides information to support the development of ISO/IEC 23090-13.

Contents

[1 Scope 1](#_Toc93670716)

[2 Reference software 1](#_Toc93670717)

[2.1 The libvdi library 2](#_Toc93670718)

[2.1.1 Architecture 2](#_Toc93670719)

[2.1.2 Implementation 3](#_Toc93670720)

[2.1.3 Logistics 3](#_Toc93670721)

[2.2 Sample software 3](#_Toc93670722)

[2.2.1 Operations of input formatting 3](#_Toc93670723)

[2.2.2 Logistics 3](#_Toc93670724)

[3 Conformance software 3](#_Toc93670725)

[3.1 Operations of input formatting 4](#_Toc93670726)

[3.2 Logistics 4](#_Toc93670727)

[4 Test vectors 4](#_Toc93670728)

[4.1 Elementary streams constraints 4](#_Toc93670729)

[4.2 Logistics 4](#_Toc93670730)

[5 Standardisation process 4](#_Toc93670731)

[6 Summary logistics 5](#_Toc93670732)

# Reference software

## The libvdi library

### Architecture

As a reminder, the scope of VDI is as follows:



**Key**

|  |  |  |  |
| --- | --- | --- | --- |
| MDS | media stream |  |  |
| ES | elementary stream |  |  |
| MTS | metadata stream |  |  |
| DS | decoded sequence |  |  |

NOTE Multiple elementary streams that are output of the input formatting function may be fed to a single video decoder instance.

Figure 1 - Video Decoding Engine and interfaces

The reference software comprises several libraries based on this diagram. Each module inside in the Video Decoding Engine translates to a sub library, i.e.:

* Input formatting 🡪 libbeam
* Lime locking 🡪 libtiming
* Output formatting 🡪 liboutput

In addition, since many ongoing projects in the video standardisation space is developed in the C++ language, this language is used to write the reference libraries as depicted in the figure below:

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure 2 - VDI reference library organization

For each sub library, the functions defined in the normative specification are implemented in the library with template parameters to accommodate different underlying types.

### Implementation

In addition to use the C++ language, the feature of C++ 20 called concepts are used. Concepts allow to define requirement on template types. This allows the libvdi to define once the important types and their requirements and then application can use any underlying library as long as the constraint on the types are fulfilled.

For instance, one can define a template type ElementaryStream without providing a concrete implementation but with providing constraints on the implementation of the concrete type that will be used by the application using the libvdi library. For instance, an application could use the VTM, ETM or HM libraries in conjunction with the libBEAM where each may have a different class defining access units.

Note that C++ 20 concepts is a new feature of the C++ language with support in the latest compiling tooling **Error! Reference source not found.**:

* gcc 10
* Visual Studio 2019
* Clang 10

The decision of using concepts may be revised at a later stage if support in compiler is not broad enough to allow the timely development of the standard. One possible alternative would be to define the template types without the constraints.

### Logistics

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Language** | **Description** | **Hosting** |
| libBEAM | C++ | Reference libraries | <https://gitlab.com/mpeg-i/video-decoding-interface/libbeam> |

The other libraries are not yet created.

## Sample software

### Operations of input formatting

A command-line tool performing the normative operations is developed. The implementation of the operation itself being normative.

The goal of implementing this software are the following:

* Verify the ability to implement a proposed operation
* Identify the normative constraints on the input elementary streams to execute the operation
* Assess the complexity of an operation

### Logistics

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Language** | **Description** | **Hosting** |
| BEAMOp | C++ | Implements input formatting operations on elementary streams | <https://gitlab.com/mpeg-i/video-decoding-interface/beamop> |

# Conformance software

## Operations of input formatting

A project for validating the constrains on the input and output elementary streams is also developed. This way, external implementations of the specifications can be validated but also the sample software can be verified to behave accordingly to the specification.

## Logistics

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Language** | **Description** | **Hosting** |
| BEAMConf | C++ | Validate conformance of input and output elementary streams passed on or output by the input formatting operations | <https://gitlab.com/mpeg-i/video-decoding-interface/beamconf> |

# Test vectors

## Elementary streams constraints

In order to execute the normative functions, the VDI expects certain constraints on the elementary streams.

Test vectors will be collected during the development of the specification to illustrate the defined constraints.

## Logistics

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Hosting** |
| Test Vectors | Elementary streams | Test vectors of elementary streams conforming the normative constraints defined in the different codec bindings | <https://gitlab.com/mpeg-i/video-decoding-interface/test-vectors> |

# Standardisation process

Proponents submitting a contribution for a new operation or a new binding to a certain elementary stream codec needs to provide the implementation in the libBEAM library as well as the calling of this function in the BEAMOp command line.

As long as this is not provided, the contribution cannot be considered for adoption.

The study of this software will be used to assess the technical merits of the technical contribution in terms of application complexity. Without the implementation, this does not allow a correct evaluation by the experts and therefore prevents the group to decide.

When a software contribution is tentatively accepted at a meeting, the changes are merged into the ‘staging’ branch. Experts are invited to study the staging branch in preparation to the next to come AhG call on VDI. At the call, the staging branch is further merged to the master branch provided no objection were raised by experts during the review period between the end of the previous meeting and the AhG call.

The links to the staging branches are provided below:

|  |  |
| --- | --- |
| **Name** | **Staging branch** |
| libbeam | <https://gitlab.com/mpeg-i/video-decoding-interface/libbeam/-/tree/staging/> |

# Summary logistics

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Language** | **Description** | **Hosting** |
| libbeam | C++ | Reference libraries | <https://gitlab.com/mpeg-i/video-decoding-interface/libbeam> |
| BEAMOp | C++ | Implements input formatting operations on elementary streams | <https://gitlab.com/mpeg-i/video-decoding-interface/beamop> |
| BEAMConf | C++ | Validate conformance of input and output elementary streams passed on or output by the input formatting operations | <https://gitlab.com/mpeg-i/video-decoding-interface/beamconf> |
| Test Vectors | Elementary streams | Test vectors of elementary streams conforming the normative constraints defined in the different codec bindings | <https://gitlab.com/mpeg-i/video-decoding-interface/test-vectors> |

The project is hosted at: <https://gitlab.com/mpeg-i/video-decoding-interface>

For access to the project, please register an account on GitLab.com at <https://gitlab.com/users/sign_in> and collect the following information:

* GitLab.com username
* GitLab.com email address

Please then send an email containing this information to the VDI GitLab managers:

Emmanuel Thomas (thomase@xiaomi.com)

1. - **Collected information on OpenMAX implementations**
   1. **OpenMAX**

As reminder, OpenMAX provides three layers of interfaces: application layer (AL), integration layer (IL) and development layer (DL). The VDI speciation refers to and integrates with the IL interface.

Specification of OpenMAX IL and the source file of the API can be found at:

<https://github.com/KhronosGroup/OpenMAX-IL-Registry>

The header files for each specification version of the API are provided here:

<https://github.com/KhronosGroup/OpenMAX-IL-Registry/tree/master/api>

The extension of the API are collected under each vendor code here:

<https://github.com/KhronosGroup/OpenMAX-IL-Registry/tree/master/extensions>

Note that only Khronos extensions are present at the moment, namely:

* OpenMAX\_IL\_1\_1\_2\_Extension Deferred Commit.pdf
* OpenMAX\_IL\_1\_1\_2\_Extension NAL Unit Packaging.pdf
  1. **OpenMAX Integration Layer (IL) (source: Wikipedia)**

The following paragraph is a quote from:

Wikipedia contributors. (2020, April 25). OpenMAX. In *Wikipedia, The Free Encyclopedia*. Retrieved 21:02, May 30, 2020, from <https://en.wikipedia.org/w/index.php?title=OpenMAX&oldid=953004646>

The OpenMAX IL API strives to give media components portability across an array of platforms using the [C-language](https://en.wikipedia.org/wiki/C_(programming_language)). In the OpenMAX IL, components represent individual blocks of functionality. Components can be sources, sinks, codecs, filters, splitters, mixers, or any other data operator. Depending on the implementation, a component could possibly represent a piece of hardware, a software codec, another processor, or a combination thereof.

The interface abstracts the hardware and software architecture in the system. The OpenMAX IL API allows the user to load, control, connect, and unload the individual components. This flexible core architecture allows the Integration Layer to easily implement almost any media use case and mesh with existing graph-based media frameworks. The key focus of the OpenMAX IL API is portability of media components.

The OpenMAX IL API design devotes particular attention to use case flexibility and optimized data transfers between components.

The OpenMAX IL API has been chosen as the base for the API to integrate Audio and Video codecs on Android, this results in most SoC vendors shipping a minimal implementation that only supports the subset required by Android. Applications do not use those OpenMAX IL components directly, but only through the Android MediaCodec API. Android's subset of OpenMAX IL with its extensions is now the de facto standard.

In 2011 the provisional version 1.2.0 was released.



Figure 3 OpenMAX IL Architecture

* 1. **OpenMAX IL Open Source Implementations**

Open source OpenMAX IL implementations are available,

* [Bellagio](http://omxil.sourceforge.net/), is maintained by STMicroelectronics
  + Shared library with IL core and a reference OpenMAX component
  + An implementation of a number of OpenMAX components
  + Language: C
* [LIM OpenMAX](http://limoa.sourceforge.net/), an implementation that has both AL and IL.
  + *limoi-core*: component loader and all OpenMAX IL APIs
  + *limoi-base*: base implementation of OpenMAX IL
  + *limoi-components*: implementations of OpemMAX components, including an FFmpeg component
  + *limoa*: implementation of OpenMAX AL based in OpenMAX IL components
  + Language: C
* [omxil\_core](https://github.com/intel/omx_comp) and [omx\_comp](https://github.com/intel/omx_comp), maintained by Intel
  + *omxil\_core*: an OpenMAX IL implementation for Intel Vaapi, including a core library and a component base framework
  + *omx\_comp*: implementation of some OpenMAX components
  + Language: C/C++
* [Android StageFright](https://android.googlesource.com/platform/frameworks/av/+/nougat-mr2.1-release/media/libstagefright/omx/), a partial implementation of IL that is the de facto standard
  + Language: C/C++
* [tizonia-openmax-il](https://github.com/tizonia/tizonia-openmax-il), part of the Tizonia Project. First open-source implementation of OpenMAX IL 1.2
  + Tizonia is a command-line cloud music player for Linux with support for Spotify, Google Play Music, YouTube, etc.
  + Maintained by Aratelia Limited
  + Tizonia OpenMAX IL Core (libtizcore)
    - implements the base OpenMAX IL Core infrastructure
  + Tizonia OpenMAX IL API implementation (libtizonia)
    - implements the base OpenMAX IL component infrastructure, which includes support for the standard OpenMAX IL state machine, port management, and buffer processing
  + Tizonia OpenMAX Components (plugins)
  + Skema: A Test execution framework to build and test arbitrary OpenMAX IL component graphs/pipelines using XML
    - <https://github.com/tizonia/tizonia-openmax-il/wiki/Skema>
  + Language: C/C++



Figure 4 - Media architecture of StageFright Android

* 1. **Available implementations**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Library Name** | **Website** | **Code** | **License** | **Code last modified** |
| Bellagio | <http://omxil.sourceforge.net/> | [link](https://sourceforge.net/p/omxil/_list/git) | [LGPL-2.1](https://sourceforge.net/p/omxil/omxil/ci/master/tree/COPYING) | 2012-07-25 |
| LIM OpenMAX | <http://limoa.sourceforge.net/> | [link](https://sourceforge.net/p/limoa/_list/git) | [LGPL-2.1](https://sourceforge.net/p/limoa/limoa/ci/master/tree/COPYING) | 2012-09-25 |
| omxil\_core | <https://github.com/intel/omxil_core> | [link](https://github.com/intel/omxil_core) | Apache-2.0 | 2014-12-05 |
| Android StageFright | <https://source.android.com/devices/media> | [link](https://android.googlesource.com/platform/frameworks/av/+/refs/heads/master/media/libstagefright) | [Apache-2.0](https://android.googlesource.com/platform/frameworks/av/+/refs/heads/master/media/libstagefright/NOTICE) | 2020-05-28 |
| The Tizonia Project | <https://tizonia.org/> | [link](https://github.com/tizonia/tizonia-openmax-il) | [LGPL-3](https://github.com/tizonia/tizonia-openmax-il/blob/master/COPYING.LESSER.md) | 2020-06-12 |

1. - **Implementation of a VDI-based decoding platform**

The VDI specification provides two main normative aspects:

1. Extension of the OpenMAX IL interface
2. Operations for input formatting, time-locking and output formatting

Most of the implementations of OpenMAX IL listed above are to some extent dependent on hardware decoders. However, the VDI specification does bind with video coding standard that are not published yet, e.g., VVC. As a result, these open source implementations would need to be extended in such a way that new SW decoders (e.g. VTM decoder) would be added. In addition, the fact that these implementations assume a certain HW architecture adds a layer a complexity for extending these implementations since we rather operate on a pure SW basis.

Therefore, it could be advantageous, and overall of lower effort, to build a standalone VDI-based decoding platform that integrates the extended OpenMAX IL interface (item 1), implements the newly defined functions (item 2), and links with existing SW decoders for simulating the operation of the platform, e.g. VTM, HTM, ETM, …. This simulated platform could be arbitrarily configured to support a given set of video profiles, concurrent number of decoders, etc…

Decode

Simulated Decoding Engine (incl. new functions, e.g. beam)

ETM?

VTM

Decode

Decode

Config

Allocate

Open MAX IL (with VDI extension)

Sample App