Moving Picture, Audio and Data Coding
by Artificial Intelligence
www.mpai.community

MPAI Technical Specification

AI Framework
MPAI-AIF

V1.1

AI Framework
Version 1.1

1   Introduction (Informative)

In recent years, Artificial Intelligence (AI) and related technologies have been applied to a broad range of applications, have started affecting the lives of millions of people and are expected to do so even more in the future. As digital media standards have positively influenced industry and billions of people, AI-based data coding standards are expected to have a similar positive impact. Indeed, research has shown that data coding with AI-based technologies is generally *more efficient* than with existing technologies for, e.g., compression and feature-based description.

However, some AI technologies may carry inherent risks, e.g., in terms of bias toward some classes of users. Therefore, the need for standardisation is more important and urgent than ever.

The international, unaffiliated, not-for-profit MPAI – Moving Picture, Audio and Data Coding by Artificial Intelligence Standards Developing Organisation has the mission to develop *AI-enabled data coding standards*. MPAI Application Standards enable the development of AI-based products, applications and services.

As a part of its mission, MPAI has developed standard and operating procedures to enable users of MPAI implementations to make informed decisions about their applicability. Central to this is the notion of Performance, defined as a set of attributes characterising a reliable and trustworthy implementation.

For the aforementioned reasons, to fully achieve the MPAI mission, Technical Specifications will be complemented by an ecosystem designed, created and managed to underpin the life cycle of MPAI standards through the steps of specification, technical testing, assessment of product safety and security, and distribution.

In the following, Terms beginning with a capital letter are defined in *Table 1* if they are specific to this Standard and in *Table 2* if they are common to all MPAI Standards.

The MPAI Ecosystem, fully specified in [1], is composed of:
- MPAI as a provider of Technical, Conformance and Performance Specifications.

- Implementers of MPAI standards.
- MPAI-appointed Performance Assessors.
- The MPAI Store which takes care of the secure distribution of validated Implementations.

The common infrastructure enabling the implementation of MPAI Application Standards and access to the MPAI Store is the AI Framework (AIF) Standard (MPAI-AIF), specified in this document.

Figure 1 depicts the MPAI-AIF Reference Model under which Implementations of MPAI Application Standards and user-defined MPAI-AIF conforming applications operate.

An AIF Implementation allows the execution of AI Workflows (AIW), composed of basic processing elements called AI Modules (AIM).
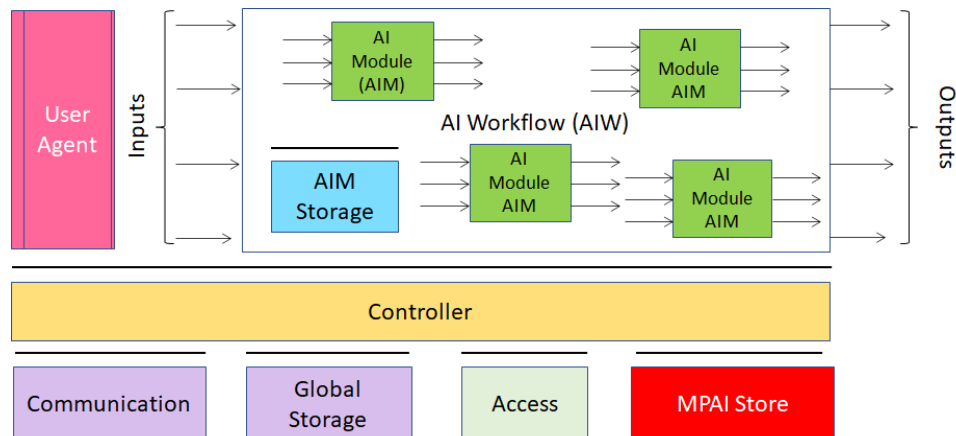
Figure 1 – The AI Framework (AIF) Reference Model and its Components

MPAI Application Standards normatively specify the Syntax and Semantics of the input and output data and the Function of the AIW and the AIMs, and the Connections between and among the AIMs of an AIW.

An AIM is defined by its Function and data, but not by its internal architecture, which may be based on AI or data processing, and implemented in software, hardware or hybrid software and hardware technologies.

MPAI defines Interoperability as the ability to replace an AIW or an AIM Implementation with a functionally equivalent Implementation. MPAI also defines 3 Interoperability Levels of an AIW that executes an AIW. The AIW may have 3 Levels:

Level 1 – Implementer-specific and satisfying the MPAI-AIF Standard.

Level 2 – Specified by an MPAI Application Standard.

Level 3 – Specified by an MPAI Application Standard and certified by a Performance Assessor.

MPAI offers Users access to the promised benefits of AI with a guarantee of increased transparency, trust and reliability as the Interoperability Level of an Implementation moves from 1 to 3. Additional information on Interoperability Levels is provided in Annex 3.

2   Scope (Normative)

The MPAI *AI Framework* (MPAI-AIF) Technical Specification specifies the architecture, interfaces, protocols and Application Programming Interfaces (API) of an AI Framework (AIspeciallyally designed for execution of AI-based implementations, but also suitable for mixed AI and traditional data processing workflows.

MPAI-AIF possesses the following main features:

- Independent of the Operating System.
- Component-based modular architecture with specified interfaces.

- Interfaces encapsulate Components to abstract them from the development environment.
- Interface with the MPAI Store enables access to validated Components.
- Component can be Implemented as:
  - Software only, from MCUs to HPC.
  - Hardware only.
  - Hybrid hardware-software.
- Component system features are:
  - Execution in local and distributed Zero-Trust architectures [14].
  - Possibility to interact with other Implementations operating in proximity.
  - Direct support to Machine Learning functionalities.

The current version of the MPAI-AIF Technical Specification has been developed by the MPAI AI Framework Development Committee (AIF-DC). Future Versions may revise and/or extend the Scope of the Standard.

3 Terms and definitions (Normative)

The Terms used in this standard whose first letter is capital are defined in *Table 1*. The Terms of MPAI-wide applicability are defined in *Table 1*.

Table 1 – MPAI-AIF Terms

Term	Definition
Access	Static or slowly changing data that are required by an application such as domain knowledge data, data models, etc.
AI Framework (AIF)	The environment where AIWs are executed.
AI Module (AIM)	A processing element receiving AIM-specific Inputs and producing AIM-specific Outputs according to according to its Function. An AIM may be an aggregation of AIMs. AIMs operate in the Trusted Zone.
AI Workflow (AIW)	A structured aggregation of AIMs implementing a Use Case receiving AIM-specific inputs and producing AIM-specific outputs according to its Function. AIWs operate in the Trusted Zone.
AIF Metadata	The data set describing the capabilities of an AIF set by the AIF Implementer.
AIM Metadata	The data set describing the capabilities of an AIM set by the AIM Implementer.
AIM Storage	A Component to store data of individual AIMs. An AIM may only access its own data. The AIM Storage is part of the Trusted Zone.
AIW Metadata	The data set describing the capabilities of an AIW set by the AIW Implementer.
Channel	A physical or logical connection between an output Port of an AIM and an input Port of an AIM. The term "connection" is also used as synonymous. Channels are part of the Trusted Zone.
Communication	The infrastructure that implements message passing between AIMs. Communication operates in the Trusted Zone.
Component	One of the 9 AIF elements: Access, AI Module, AI Workflow, Communication, Controller, AIM Storage, Shared Storage, MPAI Store, and User Agent.
Controller	A Component that manages and controls the AIMs in the AIWs, so that

	they execute in the correct order and at the time when they are needed. The Controller operates in the Trusted Zone.
Data Type	An instance of the Data Types defined by 6.1.1.
Device	A hardware and/or software entity running at least one instance of an AIF.
Event	An occurrence acted on by an Implementation.
External Port	An input or output Port simulating communication with an external Controller.
Knowledge Base	Structured and/or unstructured information made accessible to AIMs via MPAI-specified interfaces.
Message	A sequence of Records.
MPAI Ontology	A dynamic collection of terms with a defined semantics managed by MPAI.
MPAI Server	A remote machine executing one or more AIMs.
MPAI Store	The repository of Implementations.
Port	A physical or logical communication interface of an AIM.
Record	Data with a specified Format.
Resource policy	The set of conditions under which specific actions may be applied.
Shared Storage	A Component to store data shared among AIMs. The Shared Storage is part of the Trusted Zone.
Status	The set of parameters characterising a Component.
Structure	A composition of Records
Swarm Element	An AIF in a in a proximity-based scenario.
Time Base	The protocol specifying how Components can access timing information. The \Time Base is part of the Trusted Zone.
Topology	The set of Channels connecting AIMs in an AIW.
Trusted Zone	An environment that contains only trusted objects, i.e., object that do not require further authentication.
User Agent	The Component interfacing the user with an AIF through the Controller
Zero Trust	A cybersecurity model primarily focused on data and service protection that assumes no implicit trust [14].

4 References

4.1 Normative references

MPAI-AIF normatively references the following documents:

1. Technical Specification: The Governance of the MPAI Ecosystem V1, 2021; https://mpai.community/standards/mpai-gme/
2. GIT protocol, https://git-scm.com/book/en/v2/Git-on-the-Server-The-Protocols.
3. ZIP format, https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT.
4. Date and Time in the Internet: Timestamps; IETF RFC 3339; July 2002.
5. Uniform Resource Identifiers (URI): Generic Syntax, IETF RFC 2396, August 1998.
6. The JavaScript Object Notation (JSON) Data Interchange Format; https://datatracker.ietf.org/doc/html/rfc8259; IETF rfc8259; December 2017
7. JSON Schema; https://json-schema.org/.
8. BNF Notation for syntax; https://www.w3.org/Notation.html
9. MPAI; The MPAI Ontology; https://mpai.community/standards/mpai-aif/mpai-ontology/
10. MPAI; The MPAI Statutes; https://mpai.community/statutes/
11. MPAI; The MPAI Patent Policy; https://mpai.community/about/the-mpai-patent-policy/.

12. Framework Licence of the Artificial Intelligence Framework Technical Specification (MPAI-AIF); https://mpai.community/standards/mpai-aif/framework-licence/

4.2 Informative references

13. Message Passing Interface (MPI), https://www.mcs.anl.gov/research/projects/mpi/
14. Rose, Scott; Borchert, Oliver; Mitchell, Stu; Connelly, Sean; "Zero Trust Architecture"; https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf
15. MPAI Technical Specification; Context-based Audio Enhancement (MPAI-CAE) V1; https://mpai.community/standards/resources/.
16. MPAI Technical Specification; Compression and Understanding of Industrial Data (MPAI-CUI) V1; https://mpai.community/standards/resources/.
17. MPAI Technical Specification; Multimodal Conversation (MPAI-MMC) V1; https://mpai.community/standards/resources/.
18. W. Wang, J. Gao, M. Zhang, S. Wang, G. Chen, T. K. Ng, B. C. Ooi, J. Shao, and M. Reyad, "Rafiki: machine learning as an analytics service system," Proceedings of the VLDB Endowment, vol. 12, no. 2, pp. 128–140, 2018.
19. Y. Lee, A. Scolari, B.-G. Chun, M. D. Santambrogio, M. Weimer, and M. Interlandi; PRETZEL: Opening the black box of machine learning prediction serving systems; in 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI18), pp. 611–626, 2018.
20. ML.NET [ONLINE]; https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet.
21. D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica; Clipper: A low-latency online prediction serving system; in NSDI, pp. 613–627, 2017.
22. S. Zhao, M. Talasila, G. Jacobson, C. Borcea, S. A. Aftab, and J. F. Murray; Packaging and sharing machine learning models via the acumos ai open platform; in 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 841–846, IEEE, 2018.
23. Apache Prediction I/O; https://predictionio.apache.org/.
24. D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. Crespo, D. Dennison; Hidden technical debt in Machine learning systems Share; on NIPS'15: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2; December 2015 Pages 2503–2511

5 Architecture of the AI Framework (Normative)

5.1 AI Framework Components

MPAI-AIF normatively specifies the Components of *Figure 1*. All Components, except Access, operate in a Trusted Zone. This is graphically depicted with a read frame around the Components. The specific functions of the Components are:

1. **Controller**:
   a. Provides basic functionalities such as scheduling, communication between AIMs and with AIF Components such as Internal and Shared Storage.
   b. Acts as a resource manager, according to instructions given by the User.
   c. Is connected by default to all the AIMs in a given AIF.
   d. Activates/suspends/resumes/deactivates AIWs based on User's or other inputs.
   e. Supports complex application scenarios by balancing load and resources.
   f. Exposes three APIs:

       i.   *AIM APIs* enable AIMs to communicate with it (register themselves, communicate and access the rest of the AIF environment). An AIW is an AIM with additional metadata. Therefore, an AIW uses the same AIM API.

     ii.   *User APIs* enable User or other Controllers to perform high-level tasks (e.g., switch the Controller on and off, give inputs to the AIW through the Controller).

   iii.   *Controller-to-Controller API* enables interactions among Controllers.

  g. Accesses the *MPAI Store APIs* enabling communication between the AIF and the Store.

  h. May run one or more AIWs and on a different computing platform than the AIW.

  i. May communicate with other Controllers. When different Controllers running on separate agents (Swarm Elements) interact with one another, they cooperate by requesting one or more Controllers in range to open Remote Ports. The Controllers on which the Remote Ports are opened can then react to information sent by other Controllers in range through the Remote Ports and implement a collective behaviour of choice. For instance: there is a main Controller and the other Controllers in the swarm react to the information it sends; or there is no main Controller and all Controllers in the swarm behave according to a collective logic specified in the programming of all Controllers.

2. **Communication**: connects the AIF Components via Events or Channels connecting an output Port of an AIM with an input Port of another AIM. Communication has the following characteristics:

  a. The Communication Component is turned on jointly with the Controller.

  b. Communication needs not be persistent.

  c. Channels are unicast and may be physical or logical.

  d. Messages are transmitted via Channels. They are composed of sequences of Records and may be of two types:

       i.   High-Priority Messages expressed as up to 16-bit integers.

     ii.   Normal-Priority Messages expressed as MPAI-AIF defined types (6.1.1).

  Messages may be communicated through Channels or Events.

3. **AI Module** (AIM): a data processing element with a specified Function receiving AIM-specific inputs and producing AIM-specific outputs having the following characteristics:

  a. Communicates with other Components through Ports or Events.

  b. Includes at least one AIM with an input Port and one AIM with an output Port.

  c. May incorporate other AIMs.

  d. May be hot-pluggable, and dynamically register and disconnect itself on the fly.

  e. May be executed:

       i.   Locally, i.e., it encapsulates hardware physically accessible to the Controller.

     ii.   On different computing platforms, e.g., in the cloud or on swarms of drones, and encapsulates communication with a remote Controller.

4. **AI Workflow** (AIW): an organised aggregation of AIMs receiving AIM-specific inputs and producing AIM-specific outputs according to its Function implementing a use case that is proprietary or specified by an MPAI Application Standard.

5. **Shared Storage**: stores data shared by AIMs.

6. **AIM Storage**: stores data of individual AIMs.

7. **User Agent**: interfaces the User with an AIF through the Controller.

8. **Access**: provides access to static or slowly changing data that is required by AIMs such as domain knowledge data, data models, etc.

9. **MPAI Store**: stores Implementations for users to download by secure protocols.

5.2 AI Framework Implementations

MPAI-AIF enables a wide variety of Implementations:

1. AIF Implementations can be tailored to different execution environments, e.g., High-Performance Computing systems or resource-constrained computing boards. For instance, the Controller might be a process on a HPC system or a library function on a computing board.
2. There is always a Controller even if the AIF is a lightweight Implementation.
3. The API may have different MPAI-defined Profiles to allow for Implementations:
   a. To run on different computing platforms and different programming languages.
   b. To be based on different hardware and resources available.
4. AIMs may be Implemented in hardware, software and mixed-hardware and software.
5. Interoperability between AIMs is ensured by the way communication between AIMs is defined, irrespective of whether they are implemented in hardware or software.
6. Use of Ports and Channels ensures that compatible AIM Ports may be connected together irrespective of the AIMs' implementation technology.
7. Message generation and Event management is implementation independent.

5.3   AIMs

5.3.1   Implementation types

AIMs can be implemented in either hardware or software keeping the same interfaces independent of the implementation technology. However, the nature of the AIM might impose constraints on the specific values of certain API parameters and different Profiles may impose different constraints. For instance, Events (easy to accommodate in software but less so in hardware); and persistent Channels (easy to make in hardware, less so in software).

While software-software and hardware-hardware connections are homogeneous, a hybrid hardware-software scenario is inherently heterogeneous and requires the specification of additional communication protocols, which are used to wrap the hardware part and connect it to software. A list of such protocols is provided by the MPAI Ontology [9].

Examples of supported architectures are:

- *CPU-based devices* running an operating system.
- *Memory-mapped devices* (FPGAs, GPUs, TPUs) which are presented as accelerators.
- *Cloud-based frameworks*.
- *Naked hardware devices* (i.e., IP in FPGAs) that communicate through hardware Ports.
- *Encapsulated blocks of a hardware design* (i.e., IP in FPGAs) that communicate through a memory-mapped bus. In this case, the Metadata associated with the AIM (see 6.3) shall also specify the low-level communication protocol used by the Ports.

5.3.2   Combination

MPAI-AIF supports the following ways of combining AIMs:

- *Software AIMs* connected to other software AIMs resulting in a software AIM
- *Non-encapsulated hardware blocks* connected to other non-encapsulated hardware blocks, resulting in a larger, non-encapsulated hardware AIM
- *Encapsulated hardware blocks* connected to either other encapsulated hardware blocks or other software blocks, resulting in a larger software AIM.

Connection between a non-encapsulated hardware AIM and a software AIM is not supported as in such a case direct communication between the AIMs cannot be defined in any meaningful way.

5.3.3   Hardware-software compatibility

To achieve communication among AIMs irrespective of their implementation technology, the requirements considered in the following two cases should be satisfied:

1. *Hardware AIM to Hardware AIM*: Each named type in a Structure is transmitted as a separate channel. Vector types are implemented as two channels, one transmitting the size and the second transmitting the data.
2. *All other combinations*: Fill out a Structure by recursively traversing the definition (breadth-first). Sub-fields are laid down according to their type, in little-endian order.

5.3.4 Actual implementations

5.3.4.1 Hardware

Metadata ensures that hardware blocks can be directly connected to other hardware/software blocks, provided the specification platforms for the two blocks have compatible interfaces, i.e., they have compatible Ports and Channels.

5.3.4.2 Software

Software Implementations shall ensure that Communication among different constituent AIMs, and with other AIMs outside the block, is performed correctly.

In addition, AIM software Implementations shall contain a number of well-defined steps so as to ensure that the Controller is correctly initialised and remains in a consistent internal state, i.e.:
1. **Code registering the different AIMs** used by the AIW. The registration operation specifies where the AIMs will be executed, either locally or remotely. The AIM Implementations are archives downloaded from the MPAI Store containing source code, binary code and hardware designs executed on a local machine/HPC cluster/MPC machine or a remote machine.
2. **Code starting/stopping** the AIMs.
3. **Code registering the input/output Ports** for the AIM.
4. **Code instantiating unicast channels** between AIM Ports belonging to AIMs used by the AIW, and connections from/to the AIM being defined to/from remote AIMs.
5. **Registering Ports** and connecting them may result in a number of steps performed by the Controller – some suitable data structure (including, for instance, data buffers) will be allocated for each Port or Channel, in order to support the functions specified by the Controller API called by the AIM (7.5).
6. **Explicitly write/read data** to/from, any of the existing Ports.
7. In general, arbitrary functionality can be added to a software AIM. For instance, depending on the AIM Function, one would typically link libraries that allow a GPU or FPGA to be managed through Direct Memory Access (DMA), or link and use high-level libraries (e.g., TensorFlow) that implement AI-related functionality.
8. The API implementation depends on the architecture the Implementation is designed for.

6 Metadata

Metadata specifies static properties pertaining to the interaction between:
1. A Controller and its hosting hardware.
2. An AIW and its hosting Controller.
3. An AIM and the AIW it belongs to.
Metadata specified in the following Sections is represented in JSON Schema.

6.1 Communication channels and their data types

This Section specifies how Metadata pertaining to a communication Channel is defined.

6.1.1 Type system

The data interchange happening through buffers involves the exchange of structured data.

Message data types exchanged through Ports and communication Channels are defined by the following Backus–Naur Form (BNF) specification [8]. Words in bold typeface are keywords; capitalised words such as NAME are tokens.

```
fifo_type :=
        | /* The empty type */
        | base_type NAME
recursive_type :=
        | recursive_base_type NAME
base_type :=
        | toplevel_base_type
        | recursive_base_type
        | ( base_type )
toplevel_base_type :=
        | array_type
        | toplevel_struct_type
        | toplevel_variant_type
array_type :=
        | recursive_base_type []
toplevel_struct_type :=
        | { one_or_more_fifo_types_struct }
one_or_more_fifo_types_struct :=
        | fifo_type
        | fifo_type ; one_or_more_fifo_types_struct
toplevel_variant_type :=
        | { one_or_more_fifo_types_variant }
one_or_more_fifo_types_variant :=
        | fifo_type | fifo_type
        | fifo_type | one_or_more_fifo_types_variant
recursive_base_type :=
        | signed_type
        | unsigned_type
        | float_type
        | struct_type
        | variant_type
signed_type :=
        | int8
        | int16
        | int32
        | int64
unsigned_type :=
        | uint8 | byte
        | uint16
        | uint32
        | uint64
float_type :=
        | float32
        | float64
struct_type :=
        | { one_or_more_recursive_types_struct }
one_or_more_recursive_types_struct :=
        | recursive_type
        | recursive_type ; one_or_more_recursive_types_struct
variant_type :=
        | { one_or_more_recursive_types_variant }
one_or_more_recursive_types_variant :=
        | recursive_type | recursive_type
        | recursive_type | one_or_more_recursive_types_variant
```

Valid types for FIFOs are those defined by the production `fifo_type`.

Although this syntax allows to specify types having a fixed length, the general record type written to, or read from, the Port will not have a fixed length. If an AIM implemented in hardware receives data from an AIM implemented in software the data format should be harmonised with the limitations of the hardware AIM.

6.1.2 Mapping the type to buffer contents

The Type definition allows to derive an automated way of filling and transmitting buffers both for hardware and software implementations. Data structures are turned into low-level memory buffers, filled out by recursively traversing the definition (breadth-first). Sub-fields are laid down according to their type, in little-endian order.

For instance, a definition for transmitting a video frame through a FIFO might be:

```
{int32 frameNumber; int16 x; int16 y; byte[] frame} frame_t
```

and the corresponding memory layout would be

```
[32 bits: frameNumber | 16 bits: x | 16 bits: y | 32 bits: size(frame) | 8*size(frame)
bits: frame].
```

API functions are provided to parse the content of raw memory buffers in a platform- and implementation-independent fashion (see Subsection 7.5.7).

6.2 AIF Metadata

AIF Metadata is specified in terms of JSON Schema [7] definition.

```json
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://mpai.community/standards/MPAI-AIF/V1/AIF-metadata.schema.json",
  "title": "MPAI-AIF V1 AIF metadata",
  "type": "object",
  "properties": {
    "ImplementerID": {
      "description": "A numeric ID identifying the Implementer. Provided by MPAI
Store",
      "type": "integer"
    },
    "Version": {
      "description": "Provided by the Implementer. Replaced by '*' in technical
specifications",
      "type": "string"
    },
    "APIProfile": {
      "description": "Provided by MPAI. Selected by the Implementer",
      "type": "string",
      "enum": [ "Base", "Main", "High" ]
    },
    "ResourcePolicies": {
      "description": "A set of policies describing computing resources made available
to AIWs",
      "type": "array",
      "items": {
        "description": "A policy describing computing resources made available to
AIWs",
        "type": "object",
        "properties": {
          "Name": {
            "description": "An entry in the MPAI-specified Ontology",
```

```
            "type": "string"
          },
          "Minimum": {
            "description": "An entry in the MPAI-specified Ontology",
            "type": "string"
          },
          "Maximum": {
            "description": "An entry in the MPAI-specified Ontology",
            "type": "string"
          }
        },
        "required": [ "Name" ]
      }
    },
    "Authentication": {
      "description": "An entry in the MPAI-specified Ontology",
      "type": "string"      },
    "TimeBase": {
      "description": "A protocol providing a time base. If absent, timestamps are in-
terpreted according to the host time clock (absolute time with the appropriate time-
scale conversion)",
      "type": "string",
      "enum": [ "NTP", "RTP", "RTCP" ]
    }
  },
  "required": [ "ImplementerID", "Version", "Authentication" ]
}
```

6.3   AIW/AIM Metadata

AIM Metadata specifies static, abstract properties pertaining to one or more AIM implementations, and how the AIM will interact with the Controller.

AIW/AIM Metadata is specified in terms of JSON Schema [7] definition.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://mpai.community/standards/MPAI-AIF/V1/AIW-AIM-metadata.schema.json",
  "id": "#root",
  "title": "MPAI-AIF V1 AIW/AIM metadata",
  "type": "object",
  "properties": {
    "Identifier": {
      "id": "#identifier",
      "description": "Information uniquely identifying an AIW/AIM implementation",
      "type": "object",
      "properties": {
        "ImplementerID": {
          "description": "A numeric ID identifying the Implementer. Provided by MPAI
Store",
          "type": "integer"
        },
        "Specification": {
          "oneOf": [
            {
              "description": "An AIW/AIM defined by an MPAI standard",
              "type": "object",
              "properties": {
                "Standard": {
                  "description": "Defined by the Standard",
```

```json
              "type": "string"
            },
            "AIW": {
              "description": "Defined by the Standard",
              "type": "string"
            },
            "AIM": {
              "description": "Same as AIW when the Metadata being defined de-
scribes the AIW, otherwise the name of the AIM as defined by the Standard",
              "type": "string"
            },
            "Version": {
              "description": "Defined by the Standard",
              "type": "string"
            },
            "Profile": {
              "description": "Provided by MPAI. Selected by the Implementer",
              "type": "array",
              "items": {
                "type": "string",
                "enum": [ "Base", "Main", "High" ]
              }
            }
          },
          "required": [ "Standard", "AIW", "AIM", "Version" ]
        },
        {
          "description": "An AIW/AIM defined by an Implementer",
          "type": "object",
          "properties": {
            "Name": {
              "description": "Provided by the Implementer",
              "type": "string"
            },
            "Version": {
              "description": "Provided by the Implementer",
              "type": "string"
            }
          },
          "required": [ "Name", "Version" ]
        }
      ]
    }
  },
  "required": [ "ImplementerID", "Specification" ]
},
"APIProfile": {
  "description": "Provided by MPAI. Selected by the Implementer",
  "type": "string",
  "enum": [ "Base", "Main", "High" ]
},
"Description": {
  "description": "Free text describing the AIM",
  "type": "string"
},
"Types": {
  "description": "A list of shorthands for Channel data types, defined according
to 6.1.1",
  "type": "array",
```

```json
      "items": {
        "description": "A shorthand for a Channel data type, defined according to
6.1.1",
        "type": "object",
        "properties": {
          "Name": {
            "description": "The unique shorthand used for a Channel data type",
            "type": "string"
          },
          "Type": {
            "description": "A Channel data type, defined according to 6.1.1",
            "type": "string"
          }
        },
        "required": [ "Name", "Type" ]
      }
    },
    "Ports": {
      "description": "A list of AIM Ports",
      "type": "array",
      "items": {
        "description": "A Port, i.e., a physical or logical interface through which
the AIM communicates",
        "type": "object",
        "properties": {
          "Name": {
            "description": "Implementer-defined name",
            "type": "string"
          },
          "Direction": {
            "description": "The direction of the communication flow",
            "type": "string",
            "enum": [ "OutputInput", "InputOutput" ]
          },
          "RecordType": {
            "description": "Port data type defined either in the dictionary Types, or
according to Section 6.1.1",
            "type": "string"
          },
          "Technology": {
            "description": "Whether the Port is implemented in hardware or software",
            "type": "string",
            "enum": [ "Hardware", "Software" ]
          },
          "Protocol": {
            "description": "An entry in the MPAI-specified Ontology",
            "type": "string"          },
          "IsRemote": {
            "description": "Boolean specifying whether the port is remote",
            "type": "boolean"
          }
        },
        "required": [ "Name", "Direction", "RecordType", "Technology", "Protocol",
"IsRemote" ]
      }
    },
    "SubAIMs": {
      "description": "A list of AIMs in terms of which the current AIM is defined",
      "type": "array",
```

```
      "items": {
        "description": "One of the AIMs in terms of which the current AIM is
defined",
        "type": "object",
        "properties": {
          "Name": {
            "description": "A unique shorthand for the AIM in terms of which the cur-
rent AIM is defined",
            "type": "string"
          },
          "Identifier": {
            "$ref": "#identifier"
          }
        },
        "required": [ "Name", "Identifier" ]
      }
    },
    "Topology": {
      "description": "A list of Channels connecting one Output to one Input Port",
      "type": "array",
      "items": {
        "description": "A Channel connecting one Output to one Input Port",
        "type": "object",
        "properties": {
          "Output": {
            "id": "#portID",
            "description": "A Port identifier",
            "type": "object",
            "properties": {
              "AIMName": {
                "description": "The unique shorthand for a SubAIM",
                "type": "string"
              },
              "PortName": {
                "description": "The unique shorthand for one of the SubAIM Ports",
                "type": "string"
              }
            },
            "required": [ "AIMName", "PortName" ]
          },
          "Input": {
            "$ref": "#portID"
          }
        },
        "required": [ "Output", "Input" ]
      }
    },
    "Implementations": {
      "description": "A list of Implementations for the AIM being defined",
      "type": "array",
      "items": {
        "description": "An Implementation for the AIM being defined",
        "type": "object",
        "properties": {
          "BinaryName": {
            "description": "Specifies an entry in the archive containing the
Implementation down-loaded from the MPAI store",
            "type": "string"
          },
```

```
            "Architecture": {
              "description": "An entry in the MPAI-specified Ontology",
              "type": "string"
            },
            "OperatingSystem": {
              "description": "An entry in the MPAI-specified Ontology",
              "type": "string"
            },
            "Version": {
              "description": "An entry in the MPAI-specified Ontology",
              "type": "string"
            },              "Source": {
              "description": "Where the AIM Implementation should be found",
              "type": "string",
              "enum": [ "AIMStorage", "MPAIStore" ]
            },
            "Destination": {
              "description": "If empty, the Implementation is executed locally. Other-
wise, the string shall be a valid URI of an MPAI Server",
              "type": "string"
            }
          },
          "required": [ "BinaryName", "Architecture", "OperatingSystem", "Version",
"Source", "Destination" ]
        }
      },
      "ResourcePolicies": {
        "description": "A set of policies describing computing resources needed by the
AIW/AIF being defined",
        "type": "array",
        "items": {
          "description": "A policy describing computing resources needed by the AIW/AIF
being defined",
          "type": "object",
          "properties": {
            "Name": {
              "description": "An entry in the MPAI-specified Ontology",
              "type": "string"
            },
            "Minimum": {
              "description": "An entry in the MPAI-specified Ontology",
              "type": "string"
            },
            "Maximum": {
              "description": "An entry in the MPAI-specified Ontology",
              "type": "string"
            },
            "Request": {
              "description": "An entry in the MPAI-specified Ontology",
              "type": "string"              }
          },
          "required": [ "Name" ]
        }
      },
      "Documentation": {
        "definition": "A list of references to documents specifying information
relevant to the design, implementation and usage of the AIM being defined",
        "type": "array",
        "items": {
```

```
        "description": "A reference to a document specifying information relevant to
the design, implementation and usage of the AIM being defined",
        "type": "object",
        "properties": {
          "Type": {
            "description": "The type of the document",
            "type": "string",
            "enum": [ "Specification", "Manual", "Tutorial", "Video" ]
          },
          "URI": {
            "description": "A valid URI for the document",
            "type": "string"
          }
        }
      }
    }
  },
  "required": [ "Identifier", "Ports", "SubAIMs", "Topology", "Implementations" ]
}
```

7   API

7.1   General

This Chapter specifies the API of the software library supporting this Technical Specification. MPAI-AIF specifies the following API:
1. MPAI Store API called by a Controller
2. Controller API called by a User Agent.
3. Controller API called by an AIM.
4. Controller API called by other Controllers.

7.2   Conventions

The API is written in a C-like fashion. However, the specification should be meant as a definition for a general programming language.

Note that namespaces for modules, ports and communication channels (strings belonging to which are indicated in the next sections with names such as *module_name*, *port_name*, and *channel_name*, respectively) are all independent.

7.2.1   API types

We assume that the implementation defines several types, as follows:

message_t   the type of messages being passed through communication ports and channels
parser_t   the type of parsed message datatypes (a.k.a. "the high-level protocol")
error_t   the type of return code defined in 7.2.2.

The actual types are opaque, and their exact definition is left to the Implementer. The only meaningful way to operate on library types with defined results is by using library functions.

On the other hand, the type of AIM Implementations, module_t, is always defined as:

```
typedef error_t *(module_t)()
```

across all implementations, in order to ensure cross-compatibility.

Types such as void, size_t, char, int, float are regular C types.

7.2.2   Return codes

Valid return codes are:

Code	Numeric value
MPAI_AIM_ALIVE	1
MPAI_AIM_DEAD	2
MPAI_AIF_OK	0

Valid error codes are:

Code	Semantic value
MPAI_ERROR	A generic error code
MPAI_ERROR_MEM_ALLOC	Memory allocation error
MPAI_ERROR_MODULE_NOT_FOUND	The operation requested of a module cannot be executed since the module has not been found
MPAI_ERROR_INIT	The AIW cannot be initialied
MPAI_ERROR_TERM	The AIW cannot be properly terminated
MPAI_ERROR_MODULE_CREATION_FAILED	A new AIM cannot be created
MPAI_ERROR_PORT_CREATION_FAILED	A new AIM Port cannot be created
MPAI_ERROR_CHANNEL_CREATION_FAILED	A new Channel between AIMs could not be created.
MPAI_ERROR_WRITE	A generic message writing error
MPAI_ERROR_TOO_MANY_PENDING_MESSAGES	A message writing operation failed because there are too many pending messages waiting to be delivered
MPAI_ERROR_PORT_NOT_FOUND	One or both ports of a connection has (or have) been removed
MPAI_ERROR_READ	A generic message reading error
MPAI_ERROR_OP_FAILED	The requested operation failed
MPAI_ERROR_EXTERNAL_CHANNEL_CREATION_FAILED	A new Channel between Controllers could not be created.

7.2.3 High-priority Messages

Code	Numeric value
MPAI_AIM_SIGNAL_START	1
MPAI_AIM_SIGNAL_STOP	2
MPAI_AIM_SIGNAL_RESUME	3
MPAI_AIM_SIGNAL_PAUSE	4

7.3 MPAI Store API called by Controller

It is assumed that all the communication between the Controller and the MPAI Store occur via https protocol. Thus, the APIs reported refer to the http secure protocol functions (i.e. GET, POST, etc). The MPAI Store supports the GIT protocol [2].
The Controller implements the functions relative to the file retrieval as described in 7.3.1.

7.3.1 Get and parse archive

Get and parse an archive from the MPAI Store.

7.3.1.1 MPAI_AIFS_GetAndParseArchive

```
error_t MPAI_AIFS_GetAndParseArchive(const char* filename)
```

The default file format is tar.gz. Options are tar.gz, tar.bz2, tbz, tbz2, tb2, bz2, tar, and zip. For example, specifying archive.zip would send an archive in ZIP format [3]. The archive shall include one AIW Metadata file and one or more binary files. The parsing of JSON Metadata and the creation of the corresponding data structure is left to the Implementer.
All archives downloaded from the MPAI Store shall not leave the Trusted Zone.

7.4 Controller API called by User Agent

7.4.1 General

This section specifies functions executed by the User Agent when interacting with the Controller. In particular:
1. Initialise all the Components of the AIF.
2. Start/Stop/Suspend/Resume AIWs.
3. Manage Resource Allocation.

7.4.1.1 MPAI_AIFU_Controller_Initialize

```
error_t MPAI_AIFU_Controller_Initialize()
```

This function, called by the User Agent, switches on and initialies the Controller, in particular the Communication Component.

7.4.1.2 MPAI_AIFU_Controller_Destroy

```
error_t MPAI_AIFU_Controller_Destroy()
```

This function, called by the User Agent, switches off the Controller, after data structures related to running AIWs have been disposed of.

7.4.2 Start/Pause/Resume/Stop Messages to other AIWs

These functions can be used by the User Agent to send messages from the Controller to AIWs. Errors encountered while transmitting/receiving these Messages are non-recoverable – i.e., they terminate the entire AIW. AIWs can communicate with other AIWs and the Controller uses this API to Start/Pause/Resume/Stop the AIWs.

7.4.2.1 MPAI_AIFU_AIW_Start

```
error_t MPAI_AIFU_AIW_Start(const char* name, int* AIW_ID)
```

This function, called by the User Agent, registers with the Controller and starts an instance of the AIW named *name*. The AIW Metadata for *name* shall have been previously parsed. The AIW ID is returned in the variable *AIW_ID*. If the operation succeeds, it has immediate effect.

7.4.2.2 MPAI_AIFU_AIW_Pause

```
error_t MPAI_AIFU_AIW_Pause(int AIW_ID)
```

With this function the User Agent asks the Controller to pause the AIW with ID *AIW_ID*. If the operation succeeds, it has immediate effect.

7.4.2.3 *MPAI_AIFU_AIW_Resume*

```
error_t MPAI_AIFU_AIW_Resume(int AIW_ID)
```

With this function the User Agent asks the Controller to resume the AIW with ID *AIW_ID*. If the operation succeeds, it has immediate effect.

7.4.2.4 *MPAI_AIFU_AIW_Stop*

```
error_t MPAI_AIFU_AIW_Stop(int AIW_ID)
```

This function, called by the User Agent, deregisters and stops the AIW with ID *AIW_ID* from the Controller. If the operation succeeds, it has immediate effect.

7.4.3 Inquire about state of AIWs and AIMs

7.4.3.1 *MPAI_AIFU_AIM_GetStatus*

```
error_t MPAI_AIFU_AIM_GetStatus(int AIW_ID, const char* name, int* status)
```

With this function the User Agent inquires about the current status of the AIM named *name* belonging to AIW with ID *AIW_ID*. The status is returned in *status*. Admissible values are: `MPAI_AIM_ALIVE`, `MPAI_AIM_DEAD`.

7.4.4 Management of Shared and AIM Storage for AIWs

7.4.4.1 *MPAI_AIFU_SharedStorage_Init*

```
error_t MPAI_AIFU_SharedStorage_init(int AIW_ID)
```

With this function the User Agent initialises the Shared Storage interface for the AIW with ID *AIW_ID*.

7.4.4.2 *MPAI_AIFU_ AIMStorage_Init*

```
error_t MPAI_AIFU_ AIMStorage_init(int AIM_ID)
```

With this function the User Agent initialises the AIM Storage interface for the AIW with ID *AIW_ID*.

7.4.5 Communication management

Communication takes place with Messages communicated via Events or Ports and Channels. Their actual implementation and signal type depends on the MPAI-AIF Implementation (and hence on the specific platform, operating system and programming language the Implementation is developed for). Events are defined AIF wide while Ports, Channels and Messages are specific to the AIM and thus part of the AIM API.

7.4.5.1 *MPAI_AIFU_Communication_Event*

```
error_t MPAI_AIFU_Communication_Event(const char* event)
```

With this function the User Agent initialises the event handling for Event named *event*.

7.4.6 Resource allocation management

7.4.6.1 *MPAI_AIFU_Resource_GetGlobal*

```
error_t MPAI_AIFU_Resource_GetGlobal(const char* key, const char* min_value, const
```

```
char* max_value, const char* requested_value)
```

With this function the User Agent interrogates the resource allocation for one AIF Metadata entry.
why not numerical types for min max requested value?

7.4.6.2 MPAI_AIFU_Resource_SetGlobal

```
error_t MPAI_AIFU_Resource_SetGlobal(const char* key, const char* min_value, const
char* max_value, const char* requested_value)
```

With this function the User Agent initialises the resource allocation for one AIF Metadata entry.

7.4.6.3 MPAI_AIFU_Resource_GetAIW

```
error_t MPAI_AIFU_Resource_GetAIW(int AIW_ID, const char* key, const char*
min_value, const char* max_value, const char* requested_value)
```

With this function the User Agent interrogates the resource allocation for one AIM Metadata entry
for the AIW with AIW ID AIW_ID.

7.4.6.4 MPAI_AIFU_Resource_SetAIW

```
error_t MPAI_AIFU_Resource_SetAIW(int AIW_ID, const char* key, const char*
min_value, const char* max_value, const char* requested_value)
```

With this function the User Agent interrogates the resource allocation for one AIM Metadata entry
for the AIW with AIW ID AIW_ID.

7.5   Controller API called by AIMs

7.5.1   General

The following set of functions specifies how AIWs:
1. Define the topology and connections of AIMs in the AIW.
2. Define the Time base.
3. Define the Resource Policy.

7.5.2   Resource allocation management

7.5.2.1 MPAI_AIFM_Resource_GetGlobal

```
error_t MPAI_AIFM_Resource_GetGlobal(const char* key, const char* min_value, const
char* max_value, const char* requested_value)
```

With this function the AIM interrogates the resource allocation for one AIF Metadata entry.

7.5.2.2 MPAI_AIFM_Resource_SetGlobal

```
error_t MPAI_AIFM_Resource_SetGlobal(const char* key, const char* min_value, const
char* max_value, const char* requested_value)
```

With this function the AIM initialises the resource allocation for one AIF Metadata entry.

7.5.2.3 MPAI_AIFM_Resource_GetAIW

```
error_t MPAI_AIFM_Resource_GetAIW(int AIW_ID, const char* key, const char*
min_value, const char* max_value, const char* requested_value)
```

With this function the AIM interrogates the resource allocation for one AIM Metadata entry for
the AIW with AIW ID AIW_ID.

7.5.2.4 MPAI_AIFM_Resource_SetAIW

```
error_t MPAI_AIFM_Resource_SetAIW(int AIW_ID, const char* key, const char*
min_value, const char* max_value, const char* requested_value)
```

With this function the AIM interrogates the resource allocation for one AIM Metadata entry for the AIW with AIW ID `AIW_ID`.

7.5.3 Register/deregister AIMs with the Controller

7.5.3.1 MPAI_AIFM_AIM_Register_Local

```
error_t MPAI_AIFM_AIM_Register_Local(const char* name)
```

With this function the AIM registers the AIM named `name` with the Controller. The AIM shall be defined in the AIM Metadata. An Implementation that can be run on the Controller shall have been downloaded from the MPAI Store together with the Metadata or be available in the AIM Storage after having been downloaded from the MPAI Store together with the Metadata.

7.5.3.2 MPAI_AIFM_AIM_Register_Remote

```
error_t MPAI_AIFM_AIM_Register_Remote(const char* name, const char* uri)
```

With this function the AIM registers the AIM named `name` with the Controller. The AIM shall be defined in the AIM Metadata. An implementation that can be run on the Controller shall have been downloaded from the MPAI Store together with the Metadata or be available locally. The AIM will be run remotely on the MPAI Server identified by `uri`.

7.5.3.3 MPAI_AIFM_AIM_Deregister

```
error_t MPAI_AIFM_AIM_Deregister(const char* name)
```

The AIW deregisters the AIM named `name` from the Controller.

7.5.4 Send Start/Pause/Resume/Stop Messages to other AIMs

AIMs can send Messages to AIMs defined in its Metadata.
Errors encountered while transmitting/receiving these Messages are non-recoverable – i.e., they terminate the entire AIM. AIMs can communicate with other AIMs and the Controller uses this API to Start/Pause/Resume/Stop the AIMs.

7.5.4.1 MPAI_AIFM_AIM_Start

```
error_t MPAI_AIFM_AIM_Start(const char* name)
```

With this function the AIM asks the Controller to start the AIM named `name`. If the operation succeeds, it has immediate effect.

7.5.4.2 MPAI_AIFM_AIM_Pause

```
error_t MPAI_AIFM_AIM_Pause(const char* name)
```

With this function the AIM asks the Controller to pause the AIM named `name`. If the operation succeeds, it has immediate effect.

7.5.4.3 MPAI_AIFM_AIM_Resume

```
error_t MPAI_AIFM_AIM_Resume(const char* name)
```

With this function the AIM asks the Controller to resume the AIM named `name`. If the operation succeeds, it has immediate effect.

7.5.4.4 MPAI_AIFM_AIM_Stop

```
error_t MPAI_AIFM_AIM_Stop(const char* name)
```

With this function the AIM asks the Controller to stop the AIM named `name`. If the operation succeeds, it has immediate effect.

7.5.4.5 MPAI_AIFM_AIM_EventHandler

```
error_t MPAI_AIFM_AIM_EventHandler(const char* name)
```

The AIF creates EventHandler for the AIW with given name `name`. If the operation succeeds, it has immediate effect.

7.5.5 Register Connections between AIMs

7.5.5.1 MPAI_AIFM_Channel_Create

```
error_t
  MPAI_AIFM_Channel_Create(const char* name, const char* out_AIM_name, const char*
out_port_name, const char* in_AIM_name, const char* in_port_name)
```

With this function the AIM asks the Controller to create a new interconnecting channel between an output port and an input port. AIM and port names are specified with the name used when constructed.

7.5.5.2 MPAI_AIFM_Channel_Destroy

```
error_t MPAI_AIFM_Channel_Destroy(const char* name)
```

With this function the AIM asks the Controller to destroy the channel with name *name*. This API Call closes all Ports related to the Channel.

7.5.6 Using Ports

7.5.6.1 MPAI_AIFM_Port_Output_Read

```
message_t* MPAI_AIFM_Port_Output_Read(
                         const char* AIM_name, const char* port_name)
```

This function reads a message from the Port identified by (*AIM_name*,*port_name*). The read is blocking. Hence, in order to avoid deadlocks, the Implementation should first probe the Port with `MPAI_AIF_Port_Probe`. It returns a copy of the original Message.

7.5.6.2 MPAI_AIFM_Port_Input_Write

```
error_t MPAI_AIFM_Port_Input_Write(
              const char* AIM_name, const char* port_name, message_t* message)
```

This function writes a message *message* to the Port identified by (*AIM_name*,*port_name*). The write is blocking. Hence, in order to avoid deadlocks the Implementation should first probe the Port with `MPAI_AIF_Port_Probe`. The Message being transmitted shall remain available until the function returns, or the behaviour will be undefined.

7.5.6.3 MPAI_AIFM_Port_Reset

```
error_t MPAI_AIFM_Port_Reset(const char* AIM_name, const char* port_name)
```

This function resets an input or output Port identified by (`AIM_name`,`port_name`) by deleting all the pending Messages associated with it.

7.5.6.4 MPAI_AIFM_Port_CountPendingMessages

```
size_t MPAI_AIFM_Port_CountPendingMessages(
                const char* AIM_name, const char* port_name)
```

This function returns the number of pending messages on a input or output Port identified by (`AIM_name`,`port_name`).

7.5.6.5 MPAI_AIFM_Port_Probe

```
error_t MPAI_AIFM_Port_Probe(const char* port_name, message_t* message)
```

This function returns `MPAI_AIF_OK` if either the Port is a FIFO input port and an AIM can write to it, or the Port is a FIFO output Port and data is available to be read from it.

7.5.6.6 MPAI_AIFM_Port_Select

```
int MPAI_AIFM_Port_Output_Select(
        const char* AIM_name_1,const char* port_name_1,...)
```

Given a list of output Ports, this function returns the index of one Port for which data has become available in the meantime. The call is blocking to address potential race conditions.

7.5.7 Operations on messages

All implementations shall provide a common Message passing functionality which is abstracted by the following functions.

7.5.7.1 MPAI_AIFM_Message_Copy

```
message_t* MPAI_AIFM_Message_Copy(message_t* message)
```

This function makes a copy of a Message structure *message*.

7.5.7.2 MPAI_AIFM_Message_Delete

```
message_t* MPAI_AIFM_Message_Delete(message_t* message)
```

This function deletes a Message *message* and its allocated memory. The format of each Message passing through a Channel is defined by the Metadata for that Channel.

7.5.7.3 MPAI_AIFM_Message_GetBuffer

```
void* MPAI_AIFM_Message_GetBuffer(message_t* message)
```

This function gets access to the low-level memory buffer associated with a message structure *message*.

7.5.7.4 MPAI_AIFM_Message_GetBufferLength

```
size_t MPAI_AIFM_Message_GetBufferLength(message_t* message)
```

This function gets the size in bits of the low-level memory buffer associated with a message structure *message*.

7.5.7.5 MPAI_AIFM_Message_Parse

```
parser_t* MPAI_AIFM_Message_Parse (const char* type)
```

This function creates a parsed representation of the data type defined in `type` according to the Metadata syntax defined in Subsection 6.1.1 Type system, to facilitate the successive parsing of raw memory buffers associated with message structures (see functions below).

7.5.7.6 MPAI_AIFM_Message_Parse_Get_StructField

```
void* MPAI_AIFM_Message_Parse_Get_StructField(
          parser_t* parser, void* buffer, const char* field_name)
```

This function assumes that the low-level memory buffer `buffer` contains data of type `struct_type` whose complete parsed type definition (specified according to the metadata syntax defined in Subsection 6.1.1 Type system) can be found in `parser`. This function fetches the element of the `struct_type` named `field_name`, and return it in a freshly allocated low-level memory buffer. If a element with such name does not exist, return NULL.

7.5.7.7 MPAI_AIFM_Message_Parse_Get_VariantType

```
void* MPAI_AIFM_Message_Parse_Get_VariantType(
          parser_t* parser, void* buffer, const char* type_name)
```

This function assumes that the low-level memory buffer `buffer` contains data of type `variant_type` whose complete parsed type definition (specified according to the Metadata syntax defined in Chapter 6, Type system) can be found in `parser`. Fetch the member of the `variant_type` named `field_name`, and return it in a freshly allocated low-level memory buffer. If a element with such name does not exist, return NULL.

7.5.7.8 MPAI_AIFM_Message_Parse_Get_ArrayLength

```
int MPAI_AIFM_Message_Parse_Get_ArrayLength(parser_t* parser, void* buffer)
```

This function assumes that the low-level memory buffer `buffer` contains data of type `array_type` whose complete parsed type definition (specified according to the Metadata syntax defined in Chapter 6, Type system) can be found in `parser`. Retrieve the length of such an array. If the buffer does not contain an array, return -1.

7.5.7.9 MPAI_AIFM_Message_Parse_Get_ArrayField

```
void* MPAI_AIFM_Message_Parse_Get_ArrayField(
          parser_t* parser, void* buffer, const int field_num)
```

This function assumes that the low-level memory buffer `buffer` contains data of type `array_type` whose complete parsed type definition (specified according to the metadata syntax defined in Chapter 6, Type system) can be found in `parser`. Fetch the element of the `array_type` named `field_num`, and return it in a freshly allocated low-level memory buffer. If such element does not exist, return NULL.

7.5.7.10 MPAI_AIFM_Message_Parse_Delete

```
void MPAI_AIFM_Message_Parse_Delete(parser_t* parser)
```

This function deletes the parsed representation of a data type defined by `parser`, and deallocates all memory associated to it.

7.5.8　Functions specific to machine learning

The two key functionalities supported by the Framework are reliable update of AIMs with Machine Learning functionality and hooks for Explainability.

7.5.8.1　*Support for model update*

The following API supports AIM ML model update. Such update occurs via the MPAI Store by using the MPAI Store specific APIs or via Shared (SharedStorage) or AIM-specific (AIMStorage) storage by using the specified APIs.

```
error* MPAI_AIFM_Model_Update(const char* model_name)
```

The URI *model_name* points to the updated model. In some cases, such update needs to happen in highly available way so as not to impact the operation of the system. How this is effected is left to the Implementer.

7.5.8.2　*Support for model drift*

With this function the Controller detects possible degradation in ML operation caused by the characteristics of input data being significantly different from those used in training.

```
float MPAI_AIFM_Model_Drift(const char* name)
```

7.5.9　Controller API called by Controller

This Section specifies functions used by an AIM to Communicate through a Remote Port with an AIM running on another Controller. The local and remote AIMs shall belong to the same type of AIW.

7.5.9.1　*MPAI_AIFM_External_List*

```
error_t MPAI_AIFM_External_List(int* num_in_range, const char**
controllers_metadata)
```

This function returns the number *num_in_range* of in-range Controllers with which it is possible to establish communication and running the same type of AIW, and a vector *controllers_metadata* containing AIW Metadata for each reachable Controller specified according to the JSON format defined in Section 6.3. In case more than one AIW of the same type is running on the same remote Controller, each such AIW is presented as a separate vector element.

7.5.9.2　*MPAI_AIFM_External_Output_Read*

```
message_t* MPAI_AIFM_External_Output_Read(int controllerID, const char* AIM_name,
const char* port_name)
```

This function attempts to read a message from the External Port identified by (*controllerID, AIM_name,port_name*). The read is blocking. Hence, in order to avoid deadlocks, the Implementation should first probe the Port with `MPAI_AIF_Port_Probe`. It returns a copy of the original Message. This function attempts to establish a connection between the Controller and the external in-range Controller identified with a previous call to `MPAI_AIFM_Communication_List`. The call might fail due to the Controller not being in range anymore or other communication-related issues.

7.5.9.3　*MPAI_AIFM_External_Input_Write*

```
error_t MPAI_AIFM_External_Input_Write(int controllerID, const char* AIM_name,
const char* port_name, message_t* message)
```

This function attempts to write a message *message* to the External Port identified by (`controllerID,`
`AIM_name`, `port_name`). The write is blocking. Hence, in order to avoid deadlocks the
Implementation should first probe the Port with `MPAI_AIF_Port_Probe`. The Message being
transmitted shall remain available until the function returns, or the behaviour will be undefined.
This function attempts to establish a connection between the Controller and the external in-range
Controller identified with a previous call to `MPAI_AIFM_Communication_List.` The call might fail
due to the Controller not being in range anymore or other communication-related issues.

8   Implementation Guidelines (Informative)

This informative Chapter is to be developed in the next version. It will provide guidelines for
implementation of:
- Message queues.
- Control structures.
- Messages vs Events.
- Support for hardware and software.
- Support for local and remote AIMs.
- Scope of programming language dependence.
- Scheduling of AIMs and AIWs.

9   Examples (Informative)

9.1   AIF Implementations

This Chapter contains informative examples of high-level descriptions of possible AIF operations.
This Chapter will continue to be developed in subsequent Version of this Technical Specification
by adding more examples.

9.1.1   Resource-constrained implementation

1. Controller is a single process that implements the AIW and operates based on interrupts call-backs
2. AIF is instantiated via a secure communication interface
3. AIMs can be local or has been instantiated through a secure communication interface
4. Controller initialises the AIF
5. AIF asks the AIMs to be instantiated
6. Controller manages the Events and Messages
7. User Agent can act on the AIWs at the request of the user.

9.1.2   Non-resource-constrained implementation

1. Controller and AIW are two independent processes
8. Controller manages the Events and Messages
2. AIW contacts Controller on Communication and authenticates itself
3. Controller requests AIW configuration metadata
4. AIW sends Controller the configuration metadata
5. The implementation of the AIW can be local or can be downloaded from the MPAI Store
6. Controller authenticates itself with the MPAI Store and requests implementations for the needed AIMs listed in the metadata from the MPAI Store
7. MPAI Store sends the requested AIM implementations and the configuration metadata
8. Controller

a. Instantiates the AIMs specified in the AIW metadata
b. Manages their communication and resources by sending Messages to AIMs.
9. User Agent can gain control of AIWs running on the Controller via a specific Controller API, e.g., User Agent can test conformance of a AIW with an MPAI standard through a dedicated API call.

9.2 Examples of types

```
byte[] bitstream_t
```

An array of `bytes`, with variable length.

```
{int32 frameNumber; int16 x; int16 y; byte[] frame} frame_t
```

A `struct_type` with 4 members named `frameNumber`, `x`, `y`, and `frame` — they are an `int32`, an `int16`, an `int16`, and an array of `bytes` with variable length, respectively.

```
{int32 i32 | int64 i64} variant_t
```

A `variant_type` that can be either an `int32` or an `int64`.

9.3 Examples of Metadata

This section contains the AIF, AIW and AIM Metadata of the Enhanced Audioconference Experience Use Case.

9.3.1 Metadata of Enhanced Audioconference Experience AIF

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://mpai.community/standards/resources/MPAI-AIF/V1/AIF-
metadata.schema.json",
  "title": "MPAI-AIF V1 AIF metadata",
    "ImplementerID": 100,
    "Version": "v0.1",
    "APIProfile": "Main",
    "ResourcePolicies": [
        {
            "Name": "Memory",
            "Minimum": "50000",
            "Maximum": "100000",
            "Request": "75000"
      },
        {
            "Name": "CPUNumber",
            "Minimum": "1",
            "Maximum": "2",
            "Request": "1"
      },
        {
            "Name": "CPU:Class",
            "Minimum": "Low",
            "Maximum": "High",
            "Request": "Medium"
      },
        {
            "Name": "GPU:CUDA:FrameBuffer",
            "Minimum": "11GB_GDDR5X",
            "Maximum": "8GB_GDDR6X",
            "Request": "11GB_GDDR6"
```

```
        },
          {
                "Name": "GPU:CUDA:MemorySpeed",
                "Minimum": "1.60GHz",
                "Maximum": "1.77GHz",
                "Request": "1.71GHz"
        },
          {

                "Name": "GPU:CUDA:Class",
                "Minimum": "SM61",
                "Maximum": "SM86",
                "Request": "SM75"
        },
          {

                "Name": "GPU:Number",
                "Minimum": "1",
                "Maximum": "1",
                "Request": "1"
        }
    ],
    "Authentication": "admin",
    "TimeBase": "NTP"
}
```

9.3.2   Metadata of Enhanced Audioconference Experience AIW

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://mpai.community/standards/resources/MPAI-AIF/V1/AIW-AIM-
metadata.schema.json",
  "title": "EAE AIF v1 AIW/AIM metadata",
    "Identifier": {
      "ImplementerID": 100,
      "Specification": {
        "Standard": "MPAI-CAE",
        "AIW": "CAE-EAE",
        "AIM": "CAE-EAE",
        "Version": "1"
    }
  },
    "APIProfile": "Main",
    "Description": "This AIF is used to call the AIW of EAE",
     "Types": [
      {
                        "Name":"Audio_t",
                        "Type":"uint16[]",
                },
          {
                        "Name":"Array_Audio_t",
                        "Type":"Audio_t[]",
                },
          {
                        "Name":"TransformArray_Audio_t",
                        "Type":"Array_Audio_t[]",
                },
          {
                        "Name":"Text_t",
                        "Type":"uint8[]",
                }
```

```
        ],
    "Ports": [
      {
                        "Name":"MicrophoneArrayAudio",
                        "Direction":"InputOutput",
                        "RecordType":"Array_Audio_t",
                        "Technology":"Software",
                        "Protocol":"",
            "IsRemote": false
              },
              {
                        "Name":"TransformMultichannelAudio",
                        "Direction":"OutputInput",
                        "RecordType":"TransformArray_Audio_t",
                        "Technology":"Software",
                        "Protocol":"",
            "IsRemote": false
              },
              {
                        "Name":"TransformMultichannelAudio",
                        "Direction":"InputOutput",
                        "RecordType":"TransformArray_Audio_t",
                        "Technology":"Software",
                        "Protocol":"",
            "IsRemote": false
              },
          {
                        "Name":"MicrophoneArrayGeometry",
                        "Direction":"InputOutput",
                        "RecordType":"Text_t",
                        "Technology":"Software",
                        "Protocol":"",
            "IsRemote": false
              },
              {
                        "Name":"SphericalHarmonicsDecomposition",
                        "Direction":"OutputInput",
                        "RecordType":"TransformArray_Audio_t",
                        "Technology":"Software",
                        "Protocol":"",
            "IsRemote": false
              },
              {
                        "Name":"SphericalHarmonicsDecomposition",
                        "Direction":"InputOutput",
                        "RecordType":"TransformArray_Audio_t",
                        "Technology":"Software",
                        "Protocol":"",
            "IsRemote": false
              },
          {
                        "Name":"TransformSpeech",
                        "Direction":"OutputInput",
                        "RecordType":"TransformArray_Audio_t",
                        "Technology":"Software",
                        "Protocol":"",
            "IsRemote": false
              },
              {
```

```
                    "Name":"AudioSceneGeometry",
                    "Direction":"OutputInput",
                    "RecordType":"Text_t",
                    "Technology":"Software",
                    "Protocol":"",
            "IsRemote": false
                },
                {
                    "Name":"SphericalHarmonicsDecomposition",
                    "Direction":"InputOutput",
                    "RecordType":"TransformArray_Audio_t",
                    "Technology":"Software",
                    "Protocol":"",
            "IsRemote": false
                },
                {
                    "Name":"TransformSpeech",
                    "Direction":"InputOutput",
                    "RecordType":"TransformArray_Audio_t",
                    "Technology":"Software",
                    "Protocol":"",
            "IsRemote": false
                },
                {
                    "Name":"AudioSceneGeometry",
                    "Direction":"InputOutput",
                    "RecordType":"Text_t",
                    "Technology":"Software",
                    "Protocol":"",
            "IsRemote": false
                },
                {
                    "Name":"DenoisedTransformSpeech",
                    "Direction":"OutputInput",
                    "RecordType":"TransformArray_Audio_t",
                    "Technology":"Software",
                    "Protocol":"",
            "IsRemote": false
                },
                {
                    "Name":"DenoisedTransformSpeech",
                    "Direction":"InputOutput",
                    "RecordType":"TransformArray_Audio_t",
                    "Technology":"Software",
                    "Protocol":"",
            "IsRemote": false
                },
                {
                    "Name":"DenoisedSpeech",
                    "Direction":"OutputInput",
                    "RecordType":"Array_Audio_t",
                    "Technology":"Software",
                    "Protocol":"",
            "IsRemote": false
                }
        ],
    "SubAIMs": [
        {
            "Name": "AnalysisTransform",
```

```json
      "Identifier": {
        "ImplementerID": 100,
        "Specification": {
          "Standard": "MPAI-CAE",
          "AIW": "CAE-EAE",
          "AIM": "AnalysisTransform",
          "Version": "1"
        }
      }
    },
    {
      "Name": "SoundFieldDescription",
      "Identifier": {
        "ImplementerID": 100,
        "Specification": {
          "Standard": "MPAI-CAE",
          "AIW": "CAE-EAE",
          "AIM": "SoundFieldDescription",
          "Version": "1"
        }
      }
    },
    {
      "Name": "SpeechDetectionandSeparation",
      "Identifier": {
        "ImplementerID": 100,
        "Specification": {
          "Standard": "MPAI-CAE",
          "AIW": "CAE-EAE",
          "AIM": "SpeechDetectionandSeparation",
          "Version": "1"
        }
      }
    },
    {
      "Name": "NoiseCancellation",
      "Identifier": {
        "ImplementerID": 100,
        "Specification": {
          "Standard": "MPAI-CAE",
          "AIW": "CAE-EAE",
          "AIM": "NoiseCancellation",
          "Version": "1"
        }
      }
    },
    {
      "Name": "SynthesisTransform",
      "Identifier": {
        "ImplementerID": 100,
        "Specification": {
          "Standard": "MPAI-CAE",
          "AIW": "CAE-EAE",
          "AIM": "SynthesisTransform",
          "Version": "1"
        }
      }
    },
    {
```

```json
          "Name": "Packager",
          "Identifier": {
            "ImplementerID": 100,
            "Specification": {
              "Standard": "MPAI-CAE",
              "AIW": "CAE-EAE",
              "AIM": "Packager",
              "Version": "1"
            }
          }
        }
    ],
    "Topology": [
            {
                            "Output":{
                                    "AIMName":"",
                                    "PortName":"MicrophoneArrayAudio"
                            },
                            "Input":{
                                    "AIMName":"AnalysisTransform",
                                    "PortName":"MicrophoneArrayAudio"
                            }
            },
    {
                            "Output":{
                                    "AIMName":"",
                                    "PortName":"MicrophoneArrayGeometry_1"
                            },
                            "Input":{
                                    "AIMName":"SoundFieldDescription",
                                    "PortName":" MicrophoneArrayGeometry_1"
                            }
            },
    {
                            "Output":{
                                    "AIMName":"",
                                    "PortName":"MicrophoneArrayGeometry_2"
                            },
                            "Input":{
                                    "AIMName":"Packager",
                                    "PortName":" MicrophoneArrayGeometry_2"
                            }
            },
    {
                            "Output":{
                                    "AIMName":"AnalysisTransform",
                                    "PortName":"TransformMultiChannelAudio"
                            },
                            "Input":{
                                    "AIMName":"SoundFieldDescription",
                                    "PortName":"TransformMultiChannelAudio"
                            }
            },
    {
                            "Output":{
                                    "AIMName":"SoundFieldDescription",

   "PortName":"SphericalHarmonicsDecomposition_1"
                            },
```

```
                                "Input":{
                                        "AIMName":"SpeechDetectionandSeparation",

 "PortName":"SphericalHarmonicsDecomposition_1"
                                }
                        },
{
                                "Output":{
                                        "AIMName":"SoundFieldDescription",

 "PortName":"SphericalHarmonicsDecomposition_2"
                                },
                                "Input":{
                                        "AIMName":"SpeechDetectionandSeparation",

 "PortName":"SphericalHarmonicsDecomposition_2"
                                }
                        },
{
                                "Output":{
                                        "AIMName":"SpeechDetectionandSeparation",
                                        "PortName":"TransformSpeech"
                                },
                                "Input":{
                                        "AIMName":"NoiseCancellation",
                                        "PortName":"TransformSpeech"
                                }
                        },
{
                                "Output":{
                                        "AIMName":"SpeechDetectionandSeparation",
                                        "PortName":"AudioSceneGeometry_1"
                                },
                                "Input":{
                                        "AIMName":"NoiseCancellation",
                                        "PortName":"AudioSceneGeometry_1"
                                }
                        },
{
                                "Output":{
                                        "AIMName":"SpeechDetectionandSeparation",
                                        "PortName":"AudioSceneGeometry_2"
                                },
                                "Input":{
                                        "AIMName":"Packager",
                                        "PortName":"AudioSceneGeometry_2"
                                }
                        },
{
                                "Output":{
                                        "AIMName":"NoiseCancellation",
                                        "PortName":"DenoisedTransformSpeech"
                                },
                                "Input":{
                                        "AIMName":"SynthesisTransform",
                                        "PortName":"DenoisedTransformSpeech"
                                }
                        },
{
```

```json
                                "Output":{
                                        "AIMName":"SynthesisTransform",
                                        "PortName":"DenoisedSpeech"
                                },
                                "Input":{
                                        "AIMName":"Packager",
                                        "PortName":"DenoisedSpeech"
                                }
                        }
                ],
        "Implementations": [{
                "BinaryName": "eae.exe",
                "Architecture": "x64",
                "OperatingSystem": "Windows",
                "Version": "v0.1",
                "Source": "AIMStorage",
                "Destination": ""
        }
        ],
        "ResourcePolicies": [
            {
                "Name": "Memory",
                "Minimum": "50000",
                "Maximum": "100000",
                "Request": "75000"
            },
            {
                "Name": "CPUNumber",
                "Minimum": "1",
                "Maximum": "2",
                "Request": "1"
            },
            {
                "Name": "CPU:Class",
                "Minimum": "Low",
                "Maximum": "High",
                "Request": "Medium"
            },
            {
                "Name": "GPU:CUDA:FrameBuffer",
                "Minimum": "11GB_GDDR5X",
                "Maximum": "8GB_GDDR6X",
                "Request": "11GB_GDDR6"
            },
            {
                "Name": "GPU:CUDA:MemorySpeed",
                "Minimum": "1.60GHz",
                "Maximum": "1.77GHz",
                "Request": "1.71GHz"
            },
            {
                "Name": "GPU:CUDA:Class",
                "Minimum": "SM61",
                "Maximum": "SM86",
                "Request": "SM75"
            },
            {
                "Name": "GPU:Number",
                "Minimum": "1",
```

```
            "Maximum": "1",
            "Request": "1"
      }
  ],
            "Documentation":[
                    {
                            "Type":"Tutorial",
                            "URI":"https://mpai.community/standards/mpai-cae/"
                    }
            ]
}
```

9.3.3   Metadata of CAE-EAE Analysis Transform AIM

```
{
      "Identifier":{
            "ImplementerID":100,
            "Specification":{
                    "Name": "CAE",
                    "AIW": "EAE",
                    "AIM": "AnalysisTransform",
                    "Version":"1"
            }
    },
      "Description":"This AIM implements analysis transform function for CAE-EAE
that converts microphone array audio into transform multichannel audio.",
      "Types":[
            {
                    "Name": "Audio_t",
            "Type": "uint16[]"
        },
            {
                    "Name": "Array_Audio_t",
            "Type": "Audio_t[]"
        },
            {
                    "Name": "Transform_Array_Audio_t",
            "Type": "Array_Audio_t[]"
        }
      ],
      "Ports":[
            {
                    "Name":"MicrophoneArrayAudio",
                    "Direction":"InputOutput",
                    "RecordType":"Array_Audio_t",
                    "Technology":"Software",
                    "Protocol":"",
            "IsRemote": false
            },
            {
                    "Name":"TransformMultichannelAudio",
                    "Direction":"OutputInput",
                    "RecordType":"TransformArray_Audio_t",
                    "Technology":"Software",
                    "Protocol":"",
            "IsRemote": false
        }
      ],
      "SubAIMs":[],
```

```
        "Topology":[],
    "Implementations": [],
      "Documentation":[
            {
                "Type":"Tutorial",
                "URI":"https://mpai.community/standards/mpai-cae/"
            }
        ]
}
```

9.3.4 Metadata of Enhanced Audioconference Experience AIW

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://mpai.community/standards/resources/MPAI-AIF/V1/AIW-AIM-
metadata.schema.json",
  "title": "EAE AIF v1 AIW/AIM metadata",
    "Identifier": {
      "ImplementerID": 100,
      "Specification": {
        "Standard": "MPAI-CAE",
        "AIW": "CAE-EAE",
        "AIM": "CAE-EAE",
        "Version": "1"
      }
    },
    "APIProfile": "Main",
    "Description": "This AIF is used to call the AIW of EAE",
     "Types": [
      {
                        "Name":"Audio_t",
                        "Type":"uint16[]",
                },
        {
                        "Name":"Array_Audio_t",
                        "Type":"Audio_t[]",
                },
        {
                        "Name":"TransformArray_Audio_t",
                        "Type":"Array_Audio_t[]",
                },
        {
                        "Name":"Text_t",
                        "Type":"uint8[]",
                }
    ],
    "Ports": [
      {
                        "Name":"MicrophoneArrayAudio",
                        "Direction":"InputOutput",
                        "RecordType":"Array_Audio_t",
                        "Technology":"Software",
                        "Protocol":"",
                "IsRemote": false
                },
                {
                        "Name":"TransformMultichannelAudio",
                        "Direction":"OutputInput",
                        "RecordType":"TransformArray_Audio_t",
```

```
                    "Technology":"Software",
                    "Protocol":"",
          "IsRemote": false
            },
            {
                    "Name":"TransformMultichannelAudio",
                    "Direction":"InputOutput",
                    "RecordType":"TransformArray_Audio_t",
                    "Technology":"Software",
                    "Protocol":"",
          "IsRemote": false
            },
    {
                    "Name":"MicrophoneArrayGeometry",
                    "Direction":"InputOutput",
                    "RecordType":"Text_t",
                    "Technology":"Software",
                    "Protocol":"",
          "IsRemote": false
            },
            {
                    "Name":"SphericalHarmonicsDecomposition",
                    "Direction":"OutputInput",
                    "RecordType":"TransformArray_Audio_t",
                    "Technology":"Software",
                    "Protocol":"",
          "IsRemote": false
            },
            {
                    "Name":"SphericalHarmonicsDecomposition",
                    "Direction":"InputOutput",
                    "RecordType":"TransformArray_Audio_t",
                    "Technology":"Software",
                    "Protocol":"",
          "IsRemote": false
            },
    {
                    "Name":"TransformSpeech",
                    "Direction":"OutputInput",
                    "RecordType":"TransformArray_Audio_t",
                    "Technology":"Software",
                    "Protocol":"",
          "IsRemote": false
            },
            {
                    "Name":"AudioSceneGeometry",
                    "Direction":"OutputInput",
                    "RecordType":"Text_t",
                    "Technology":"Software",
                    "Protocol":"",
          "IsRemote": false
            },
            {
                    "Name":"SphericalHarmonicsDecomposition",
                    "Direction":"InputOutput",
                    "RecordType":"TransformArray_Audio_t",
                    "Technology":"Software",
                    "Protocol":"",
          "IsRemote": false
```

```json
            },
            {
                "Name":"TransformSpeech",
                "Direction":"InputOutput",
                "RecordType":"TransformArray_Audio_t",
                "Technology":"Software",
                "Protocol":"",
        "IsRemote": false
            },
            {
                "Name":"AudioSceneGeometry",
                "Direction":"InputOutput",
                "RecordType":"Text_t",
                "Technology":"Software",
                "Protocol":"",
        "IsRemote": false
            },
            {
                "Name":"DenoisedTransformSpeech",
                "Direction":"OutputInput",
                "RecordType":"TransformArray_Audio_t",
                "Technology":"Software",
                "Protocol":"",
        "IsRemote": false
            },
            {
                "Name":"DenoisedTransformSpeech",
                "Direction":"InputOutput",
                "RecordType":"TransformArray_Audio_t",
                "Technology":"Software",
                "Protocol":"",
        "IsRemote": false
            },
            {
                "Name":"DenoisedSpeech",
                "Direction":"OutputInput",
                "RecordType":"Array_Audio_t",
                "Technology":"Software",
                "Protocol":"",
        "IsRemote": false
            }
    ],
"SubAIMs": [
    {
        "Name": "AnalysisTransform",
        "Identifier": {
          "ImplementerID": 100,
          "Specification": {
            "Standard": "MPAI-CAE",
            "AIW": "CAE-EAE",
            "AIM": "AnalysisTransform",
            "Version": "1"
          }
        }
    },
    {
        "Name": "SoundFieldDescription",
        "Identifier": {
          "ImplementerID": 100,
```

```json
        "Specification": {
          "Standard": "MPAI-CAE",
          "AIW": "CAE-EAE",
          "AIM": "SoundFieldDescription",
          "Version": "1"
        }
      }
    },
    {
        "Name": "SpeechDetectionandSeparation",
        "Identifier": {
          "ImplementerID": 100,
          "Specification": {
            "Standard": "MPAI-CAE",
            "AIW": "CAE-EAE",
            "AIM": "SpeechDetectionandSeparation",
            "Version": "1"
          }
        }
    },
    {
        "Name": "NoiseCancellation",
        "Identifier": {
          "ImplementerID": 100,
          "Specification": {
            "Standard": "MPAI-CAE",
            "AIW": "CAE-EAE",
            "AIM": "NoiseCancellation",
            "Version": "1"
          }
        }
    },
    {
        "Name": "SynthesisTransform",
        "Identifier": {
          "ImplementerID": 100,
          "Specification": {
            "Standard": "MPAI-CAE",
            "AIW": "CAE-EAE",
            "AIM": "SynthesisTransform",
            "Version": "1"
          }
        }
    },
    {
        "Name": "Packager",
        "Identifier": {
          "ImplementerID": 100,
          "Specification": {
            "Standard": "MPAI-CAE",
            "AIW": "CAE-EAE",
            "AIM": "Packager",
            "Version": "1"
          }
        }
    }
  ],
  "Topology": [
            {
```

```
                        "Output":{
                                "AIMName":"",
                                "PortName":"MicrophoneArrayAudio"
                        },
                        "Input":{
                                "AIMName":"AnalysisTransform",
                                "PortName":"MicrophoneArrayAudio"
                        }
                },
{
                        "Output":{
                                "AIMName":"",
                                "PortName":"MicrophoneArrayGeometry_1"
                        },
                        "Input":{
                                "AIMName":"SoundFieldDescription",
                                "PortName":" MicrophoneArrayGeometry_1"
                        }
                },
{
                        "Output":{
                                "AIMName":"",
                                "PortName":"MicrophoneArrayGeometry_2"
                        },
                        "Input":{
                                "AIMName":"Packager",
                                "PortName":" MicrophoneArrayGeometry_2"
                        }
                },
{
                        "Output":{
                                "AIMName":"AnalysisTransform",
                                "PortName":"TransformMultiChannelAudio"
                        },
                        "Input":{
                                "AIMName":"SoundFieldDescription",
                                "PortName":"TransformMultiChannelAudio"
                        }
                },
{
                        "Output":{
                                "AIMName":"SoundFieldDescription",

 "PortName":"SphericalHarmonicsDecomposition_1"
                        },
                        "Input":{
                                "AIMName":"SpeechDetectionandSeparation",

 "PortName":"SphericalHarmonicsDecomposition_1"
                        }
                },
{
                        "Output":{
                                "AIMName":"SoundFieldDescription",

 "PortName":"SphericalHarmonicsDecomposition_2"
                        },
                        "Input":{
                                "AIMName":"SpeechDetectionandSeparation",
```

```
                "PortName":"SphericalHarmonicsDecomposition_2"
                                }
                        },
        {
                                "Output":{
                                        "AIMName":"SpeechDetectionandSeparation",
                                        "PortName":"TransformSpeech"
                                },
                                "Input":{
                                        "AIMName":"NoiseCancellation",
                                        "PortName":"TransformSpeech"
                                }
                        },
        {
                                "Output":{
                                        "AIMName":"SpeechDetectionandSeparation",
                                        "PortName":"AudioSceneGeometry_1"
                                },
                                "Input":{
                                        "AIMName":"NoiseCancellation",
                                        "PortName":"AudioSceneGeometry_1"
                                }
                        },
        {
                                "Output":{
                                        "AIMName":"SpeechDetectionandSeparation",
                                        "PortName":"AudioSceneGeometry_2"
                                },
                                "Input":{
                                        "AIMName":"Packager",
                                        "PortName":"AudioSceneGeometry_2"
                                }
                        },
        {
                                "Output":{
                                        "AIMName":"NoiseCancellation",
                                        "PortName":"DenoisedTransformSpeech"
                                },
                                "Input":{
                                        "AIMName":"SynthesisTransform",
                                        "PortName":"DenoisedTransformSpeech"
                                }
                        },
        {
                                "Output":{
                                        "AIMName":"SynthesisTransform",
                                        "PortName":"DenoisedSpeech"
                                },
                                "Input":{
                                        "AIMName":"Packager",
                                        "PortName":"DenoisedSpeech"
                                }
                        }
                ],
        "Implementations": [{
                "BinaryName": "eae.exe",
                "Architecture": "x64",
                "OperatingSystem": "Windows",
```

```json
            "Version": "v0.1",
            "Source": "AIMStorage",
            "Destination": ""
        }
    ],
    "ResourcePolicies": [
        {
            "Name": "Memory",
            "Minimum": "50000",
            "Maximum": "100000",
            "Request": "75000"
        },
        {

            "Name": "CPUNumber",
            "Minimum": "1",
            "Maximum": "2",
            "Request": "1"
        },
        {

            "Name": "CPU:Class",
            "Minimum": "Low",
            "Maximum": "High",
            "Request": "Medium"
        },
        {

            "Name": "GPU:CUDA:FrameBuffer",
            "Minimum": "11GB_GDDR5X",
            "Maximum": "8GB_GDDR6X",
            "Request": "11GB_GDDR6"
        },
        {

            "Name": "GPU:CUDA:MemorySpeed",
            "Minimum": "1.60GHz",
            "Maximum": "1.77GHz",
            "Request": "1.71GHz"
        },
        {

            "Name": "GPU:CUDA:Class",
            "Minimum": "SM61",
            "Maximum": "SM86",
            "Request": "SM75"
        },
        {

            "Name": "GPU:Number",
            "Minimum": "1",
            "Maximum": "1",
            "Request": "1"
        }
    ],
            "Documentation":[
                    {
                            "Type":"Tutorial",
                            "URI":"https://mpai.community/standards/mpai-cae/"
                    }
                ]
}
```

9.3.5 Metadata of CAE-EAE Analysis Transform AIM

```
{
    "Identifier":{
        "ImplementerID":100,
        "Specification":{
            "Name": "CAE",
            "AIW": "EAE",
            "AIM": "AnalysisTransform",
            "Version":"1"
        }
    },
    "Description":"This AIM implements analysis transform function for CAE-EAE
that converts microphone array audio into transform multichannel audio.",
    "Types":[
        {
            "Name": "Audio_t",
        "Type": "uint16[]"
        },
        {
            "Name": "Array_Audio_t",
        "Type": "Audio_t[]"
        },
        {
            "Name": "Transform_Array_Audio_t",
        "Type": "Array_Audio_t[]"
        }
    ],
    "Ports":[
        {
            "Name":"MicrophoneArrayAudio",
            "Direction":"InputOutput",
            "RecordType":"Array_Audio_t",
            "Technology":"Software",
            "Protocol":"",
        "IsRemote": false
        },
        {
            "Name":"TransformMultichannelAudio",
            "Direction":"OutputInput",
            "RecordType":"TransformArray_Audio_t",
            "Technology":"Software",
            "Protocol":"",
        "IsRemote": false
        }
    ],
    "SubAIMs":[],
    "Topology":[],
  "Implementations": [],
    "Documentation":[
        {
            "Type":"Tutorial",
            "URI":"https://mpai.community/standards/mpai-cae/"
        }
    ]
}
```

9.3.6 Metadata of CAE-EAE Sound Field Description AIM

```
{
```

```
        "AIM":{
                "ImplementerID": 100,
                "Standard":{
                        "Name": "CAE",
                        "AIW": "EAE",
                        "AIM": "SoundFieldDescription",
                        "Version":"1"
                },
                "Description":"This AIM implements sound field description function for
CAE-EAE that converts transform multichannel audio into spherical harmonics
decomposition.",
                "Types":[
{
                        "Name": "Text_t",
                        "Type": "uint8[]"
                },
                        {
                        "Name": "Audio_t",
                        "Type": "uint16[]"
                },
                        {
                        "Name": "Array_Audio_t",
                        "Type": "Audio_t[]"
                },
                        {
                        "Name": "Transform_Array_Audio_t",
                        "Type": "Array_Audio_t[]"
                }
                ],
                "Ports":[
                        {
                                "Name":"TransformMultichannelAudio",
                                "Direction":"InputOutput",
                                "RecordType":"TransformArray_Audio_t",
                                "Technology":"Software",
                                "Protocol":"",
                                "IsRemote": false
                        },
{
                                "Name":"MicrophoneArrayGeometry",
                                "Direction":"InputOutput",
                                "RecordType":"Text_t",
                                "Technology":"Software",
                                "Protocol":"",
                                "IsRemote": false
                        },
                        {
                                "Name":"SphericalHarmonicsDecomposition",
                                "Direction":"OutputInput",
                                "RecordType":"TransformArray_Audio_t",
                                "Technology":"Software",
                                "Protocol":"",
                                "IsRemote": false
                        }
                ],
                "SubAIMs":[],
                "Topology":[],
                "Documentation":[
                        {
```

```
                        "Type":"tutorial",
                        "URI":"https://mpai.community/standards/mpai-cae/"
                }
        ]
    }
}
```

9.3.7 Metadata of CAE-EAE Speech Detection and Separation AIM

```
{
    "AIM":{
            "ImplementerID": 100,
            "Standard":{
                    "Name": "CAE",
                    "AIW": "EAE",
                    "AIM": "SpeechDetectionandSeparation",
                    "Version":"1"
},
            "Description":"This AIM implements speech detection and separation
function for CAE-EAE that converts spherical harmonics coefficients into transform
speech and Audio Scene Geometry.",
"Types":[
{
                    "Name": "Text_t",
                    "Type": "uint8[]"
            },
{
                    "Name": "Audio_t",
                    "Type": "uint16[]"
            },
            {
                    "Name": "Array_Audio_t",
                    "Type": "Audio_t[]"
            },
            {
                    "Name": "Transform_Array_Audio_t",
                    "Type": "Array_Audio_t[]"
            }
            ],
            "Ports":[
                    {
                            "Name":"SphericalHarmonicsDecomposition",
                            "Direction":"InputOutput",
                            "RecordType":"TransformArray_Audio_t",
                            "Technology":"Software",
                            "Protocol":"",
                            "IsRemote": false

                    },
{
                            "Name":"TransformSpeech",
                            "Direction":"OutputInput",
                            "RecordType":"TransformArray_Audio_t",
                            "Technology":"Software",
                            "Protocol":"",
                            "IsRemote": false

                    },
```

```
                {
                        "Name":"AudioSceneGeometry",
                        "Direction":"OutputInput",
                        "RecordType":"Text_t",
                        "Technology":"Software",
                        "Protocol":"",
                        "IsRemote": false

                }
        ],
        "AIMs":[],
        "Topology":[],
        "Documentation":[
                {
                        "Type":"tutorial",
                        "URI":"https://mpai.community/standards/mpai-cae/"
                }
        ]
    }
}
```

9.3.8 Metadata of CAE-EAE Noise Cancellation AIM

```
{
        "AIM":{
                "ImplementerID": 100,
                "Standard":{
                        "Name": "CAE",
                        "AIW": "EAE",
                        "AIM": "NoiseCancellation",
                        "Version":"1"
                },
                "Description":"This AIM implements noise cancellation function for CAE-
EAE that converts transform speech into denoised transform speech.",
"Types":[
{
                        "Name": "Text_t",
                        "Type": "uint8[]"
                },
{
                        "Name": "Audio_t",
                        "Type": "uint16[]"
                },
                {
                        "Name": "Array_Audio_t",
                        "Type": "Audio_t[]"
                },
                {
                        "Name": "Transform_Array_Audio_t",
                        "Type": "Array_Audio_t[]"
                }
                ],
                "Ports":[
                        {
                                "Name":"SphericalHarmonicsDecomposition",
                                "Direction":"InputOutput",
                                "RecordType":"TransformArray_Audio_t",
                                "Technology":"Software",
                                "Protocol":"",
```

```
                                "IsRemote": false

                        },
{
                                "Name":"TransformSpeech",
                                "Direction":"InputOutput",
                                "RecordType":"TransformArray_Audio_t",
                                "Technology":"Software",
                                "Protocol":"",
                                "IsRemote": false

                        },
                        {
                                "Name":"AudioSceneGeometry",
                                "Direction":"InputOutput",
                                "RecordType":"Text_t",
                                "Technology":"Software",
                                "Protocol":"",
                                "IsRemote": false

                        },
{
                                "Name":"DenoisedTransformSpeech",
                                "Direction":"OutputInput",
                                "RecordType":"TransformArray_Audio_t",
                                "Technology":"Software",
                                "Protocol":"",
                                "IsRemote": false

                        }

                ],
                "AIMs":[

                ],
                "Topology":[

                ],

                "Documentation":[
                        {
                                "Type":"tutorial",
                                "URI":"https://mpai.community/standards/mpai-cae/"
                        }
                ]
        }
}
```

9.3.9   Metadata of CAE-EAE Synthesis Transform AIM

```
{
        "AIM":{
                "ImplementerID": 100,
                "Standard":{
                        "Name": "CAE",
                        "AIW": "EAE",
                        "AIM": "SynthesisTransform",
                        "Version":"1"
```

```
            },
            "Description":"This AIM implements synthesis transform function for
CAE-EAE that converts denoised transform speech into denoised speech.",
            "Types":[
{
                    "Name": "Audio_t",
                    "Type": "uint16[]"
            },
            {
                    "Name": "Array_Audio_t",
                    "Type": "Audio_t[]"
            },
            {
                    "Name": "Transform_Array_Audio_t",
                    "Type": "Array_Audio_t[]"
            }

            ],

            "Ports":[
                {
                        "Name":"DenoisedTransformSpeech",
                        "Direction":"InputOutput",
                        "RecordType":"TransformArray_Audio_t",
                        "Technology":"Software",
                        "Protocol":"",
"IsRemote": false

                },
{
                        "Name":"DenoisedSpeech",
                        "Direction":"OutputInput",
                        "RecordType":"Array_Audio_t",
                        "Technology":"Software",
                        "Protocol":"",
"IsRemote": false

                }
            ],
            "AIMs":[

            ],
            "Topology":[

            ],

            "Documentation":[
                {
                        "Type":"tutorial",
                        "URI":"https://mpai.community/standards/mpai-cae/"
                }
            ]
    }
}
```

9.3.10 Metadata of CAE-EAE Packager AIM

```
{
      "AIM":{
            "ImplementerID": 100,
            "Standard":{
                  "Name": "CAE",
                  "AIW": "EAE",
                  "AIM": "Packager",
                  "Version":"1"
            },
            "Description":"This AIM implements packager function for CAE-EAE that
converts denoised speech into Multichannel Audio + Audio Scene Geometry.",
"Types":[
{
                  "Name": "Text_t",
                  "Type": "uint8[]"
            },
{
                  "Name": "Audio_t",
                  "Type": "uint16[]"
            },
            {
                  "Name": "Array_Audio_t",
                  "Type": "Audio_t[]"
            }
            ],
            "Ports":[
                  {
                        "Name":"DenoisedSpeech",
                        "Direction":"InputOutput",
                        "RecordType":"Array_Audio_t",
                        "Technology":"Software",
                        "Protocol":"",
                        "IsRemote": false

                  },
{
                        "Name":"AudioSceneGeometry",
                        "Direction":"InputOutput",
                        "RecordType":"Text_t",
                        "Technology":"Software",
                        "Protocol":"",
                        "IsRemote": false

                  },
{
                        "Name":"MultichannelAudioandAudioSceneGeometry",
                        "Direction":"OutputInput",
                        "RecordType":"Array_Audio_t",
                        "Technology":"Software",
                        "Protocol":"",
                        "IsRemote": false
                  }
            ],
            "AIMs":[],
            "Topology":[ ],
            "Documentation":[
```

```
                {
                        "Type":"tutorial",
                        "URI":"https://mpai.community/standards/mpai-cae/"
                }
        ]
    }
}
```

Annex 1 – MPAI-wide terms and definitions (Normative)

The Terms used in this standard whose first letter is capital and are not already included in *Table 1* are defined in *Table 2*.

Table 2 – MPAI-wide Terms

Term	Definition
Access	Static or slowly changing data that are required by an application such as domain knowledge data, data models, etc.
AI Framework (AIF)	The environment where AIWs are executed.
AI Workflow (AIW)	An organised aggregation of AIMs implementing a Use Case receiving AIM-specific Inputs and producing AIM-specific Outputs according to its Function.
AI Module (AIM)	A processing element receiving AIM-specific Inputs and producing AIM-specific Outputs according to according to its Function. AIMs operate in the Trusted Zone
Application	A usage domain target of an Application Standard
Application Programming Interface (API)	A software interface that allows two applications to talk to each other
Application Standard	An MPAI Standard specifying AIWs, AIMs, Topologies and Formats suitable for a particular application domain.
Channel	A connection between an output port of an AIM and an input port of an AIM. The term "connection" is also used as synonymous. Channels are parts of the Trusted Zone.
Communication	The infrastructure that implements message passing between AIMs. Communication operates in the Trusted Zone
Component	One of the 7 AIF elements: Access, Communication, Controller, AIM Storage, Shared Storage, MPAI Store, and User Agent
Conformance	The attribute of an Implementation of being a correct technical Implementation of a Technical Specification.
Conformance Tester	An entity authorised by MPAI to Test the Conformance of an Implementation.
Conformance Testing	The normative document specifying the Means to Test the Conformance of an Implementation.
Conformance Testing Means	Procedures, tools, data sets and/or data set characteristics to Test the Conformance of an Implementation.
Connection	A Channel connecting an output port of an AIM and an input port of an AIM.
Controller	A Component that manages and controls the AIMs in the AIF, so that they execute in the correct order and at the time when they are needed. The Controller operates in the Trusted Zone.
Data format	The standard digital representation of data and their semantics.
Ecosystem	The ensemble of the following actors: MPAI, MPAI Store, Implementers, Conformance Testers, Performance Testers and Users of MPAI-AIF Implementations as needed to enable an Interoperability Level.
Explainability	The ability to trace the output of an Implementation back to the inputs that have produced it.

Fairness	The attribute of an Implementation whose extent of applicability can be assessed by making the training set and/or network open to testing for bias and unanticipated results.
Function	The operations effected by an AIW or an AIM on input data.
Shared Storage	A Component to store data shared by AIMs. The Shared Storage is part of the Trusted Zone.
AIM Storage	A Component to store data of the individual AIMs. The AIM Storage is part of the Trusted Zone.
Identifier	A name that uniquely identifies an Implementation.
Implementation	1. An embodiment of the MPAI-AIF Technical Specification, or 2. An AIW or AIM of a particular Level (1-2-3) conforming with a Use Case of an MPAI Application Standard.
Interoperability	The ability to functionally replace an AIM with another AIM having the same Interoperability Level
Interoperability Level	The attribute of an AIW and its AIMs to be executable in an AIF Implementation and to be proprietary (Level 1) or to pass the Conformance Testing (Level 2) or the Performance Testing (Level 3) of an MPAI Application Standard.
Knowledge Base	Structured and/or unstructured information made accessible to AIMs via MPAI-specified interfaces
Message	A sequence of Records transported by Communication through Channels.
Normativity	The set of attributes of a technology or a set of technologies specified by the applicable parts of an MPAI standard.
Performance	The attribute of an Implementation of being Reliable, Robust, Fair and Replicable.
Performance Assessment	The normative document specifying the procedures, the tools, the data sets and/or the data set characteristics to Assess the Grade of Performance of an Implementation.
Performance Assessment Means	Procedures, tools, data sets and/or data set characteristics to Assess the Performance of an Implementation.
Performance Assessor	An entity authorised by MPAI to Assess the Performance of an Implementation in a given Application domain
Profile	A particular subset of the technologies used in MPAI-AIF or an AIW of an Application Standard and, where applicable, the classes, other subsets, options and parameters relevant to that subset.
Record	A data structure with a specified structure
Reference Software	A technically correct software implementation of a Technical Specification containing source code, or source and compiled code.
Reliability	The attribute of an Implementation that performs as specified by the Application Standard, profile and version the Implementation refers to, e.g., within the application scope, stated limitations, and for the period of time specified by the Implementer.
Replicability	The attribute of an Implementation whose Performance, as Assessed by a Performance Assessor, can be replicated, within an agreed level, by another Performance Assessor.
Robustness	The attribute of an Implementation that copes with data outside of the stated application scope with an estimated degree of confidence.
Service Provider	An entrepreneur who offers an Implementation as a service (e.g., a

	recommendation service) to Users.
Standard	The ensemble of Technical Specification, Reference Software, Conformance Testing and Performance Assessment of an MPAI application Standard.
Technical Specification	(Framework) the normative specification of the AI Framework. (Application) the normative specification of the set of Use Cases belonging to an Application Domain along with the AIMs required to Implement the Use Cases. the collection of Use Cases relevant to the Application Domain that include: 1. The formats of the Input/Output data of the AIWs implementing the Use Cases. 2. The Topology of the AIMs of the AIWs. 3. The formats of the Input/Output data of the AIMs belonging the AIW.
Time Base	The protocol specifying how Components can access timing information. The Time Base is part of the Trusted Zone.
Topology	The set of AIM Connections of an AIW.
Use Case	A particular instance of the Application domain target of an Application Standard.
User	A user of an Implementation.
User Agent	The Component interfacing the user with an AIF through the Controller.
Version	A revision or extension of a Standard or of one of its elements.

Annex 2 - Notices and Disclaimers Concerning MPAI Standards (Informative)

The notices and legal disclaimers given below shall be borne in mind when downloading and using approved MPAI Standards.

In the following, "Standard" means the collection of four MPAI-approved and published documents: "Technical Specification", "Reference Software" and "Conformance Testing" and, where applicable, "Performance Testing".

Life cycle of MPAI Standards
MPAI Standards are developed in accordance with the MPAI Statutes. An MPAI Standard may only be developed when a Framework Licence has been adopted. MPAI Standards are developed by especially established MPAI Development Committees who operate on the basis of consensus, as specified in Annex 1 of the MPAI Statutes. While the MPAI General Assembly and the Board of Directors administer the process of the said Annex 1, MPAI does not independently evaluate, test, or verify the accuracy of any of the information or the suitability of any of the technology choices made in its Standards.

MPAI Standards may be modified at any time by corrigenda or new editions. A new edition, however, may not necessarily replace an existing MPAI standard. Visit the web page to determine the status of any given published MPAI Standard.

Comments on MPAI Standards are welcome from any interested parties, whether MPAI members or not. Comments shall mandatorily include the name and the version of the MPAI Standard and, if applicable, the specific page or line the comment applies to. Comments should be sent to the MPAI Secretariat. Comments will be reviewed by the appropriate committee for their technical relevance. However, MPAI does not provide interpretation, consulting information, or advice on MPAI Standards. Interested parties are invited to join MPAI so that they can attend the relevant Development Committees.

Coverage and Applicability of MPAI Standards
MPAI makes no warranties or representations of any kind concerning its Standards, and expressly disclaims all warranties, expressed or implied, concerning any of its Standards, including but not limited to the warranties of merchantability, fitness for a particular purpose, non-infringement etc. MPAI Standards are supplied "AS IS".

The existence of an MPAI Standard does not imply that there are no other ways to produce and distribute products and services in the scope of the Standard. Technical progress may render the technologies included in the MPAI Standard obsolete by the time the Standard is used, especially in a field as dynamic as AI. Therefore, those looking for standards in the Data Compression by Artificial Intelligence area should carefully assess the suitability of MPAI Standards for their needs.

IN NO EVENT SHALL MPAI BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: THE NEED TO PROCURE SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF

THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

MPAI alerts users that practicing its Standards may infringe patents and other rights of third parties. Submitters of technologies to this standard have agreed to licence their Intellectual Property according to their respective Framework Licences.

Users of MPAI Standards should consider all applicable laws and regulations when using an MPAI Standard. The validity of Conformance Testing is strictly technical and refers to the correct implementation of the MPAI Standard. Moreover, positive Performance Assessment of an implementation applies exclusively in the context of the MPAI Governance and does not imply compliance with any regulatory requirements in the context of any jurisdiction. Therefore, it is the responsibility of the MPAI Standard implementer to observe or refer to the applicable regulatory requirements. By publishing an MPAI Standard, MPAI does not intend to promote actions that are not in compliance with applicable laws, and the Standard shall not be construed as doing so. In particular, users should evaluate MPAI Standards from the viewpoint of data privacy and data ownership in the context of their jurisdictions.

Implementers and users of MPAI Standards documents are responsible for determining and complying with all appropriate safety, security, environmental and health and all applicable laws and regulations.

Copyright
MPAI draft and approved standards, whether they are in the form of documents or as web pages or otherwise, are copyrighted by MPAI under Swiss and international copyright laws. MPAI Standards are made available and may be used for a wide variety of public and private uses, e.g., implementation, use and reference, in laws and regulations and standardisation. By making these documents available for these and other uses, however, MPAI does not waive any rights in copyright to its Standards. For inquiries regarding the copyright of MPAI standards, please contact the MPAI Secretariat.

The Reference Software of an MPAI Standard is released with the MPAI Modified Berkeley Software Distribution licence. However, implementers should be aware that the Reference Software of an MPAI Standard may reference some third-party software that may have a different licence.

Annex 3 – The Governance of the MPAI Ecosystem (Informative)

Level 1 Interoperability

With reference to *Figure 1*, MPAI issues and maintains a Technical Specification – called MPAI-AIF – whose components are:

1. An environment called AI Framework (AIF) running AI Workflows (AIW) composed of inter-connected AI Modules (AIM) exposing standard interfaces.
2. A distribution system of AIW and AIM Implementation called MPAI Store from which an AIF Implementation can download AIWs and AIMs.

A Level 1 Implementation shall be an Implementation of the MPAI-AIF Technical Specification executing AIWs composed of AIMs able to call the MPAI-AIF APIs.

Implementers' benefits	Upload to the MPAI Store and have globally distributed Implementations of - AIFs conforming to MPAI-AIF. - AIWs and AIMs performing proprietary functions executable in AIF.
Users' benefits	Rely on Implementations that have been tested for security.
MPAI Store's role	- Tests the Conformance of Implementations to MPAI-AIF. - Verifies Implementations' security, e.g., absence of malware. - Indicates unambiguously that Implementations are Level 1.

Level 2 Interoperability

In a Level 2 Implementation, the AIW shall be an Implementation of an MPAI Use Case and the AIMs shall conform with an MPAI Application Standard.

Implementers' benefits	Upload to the MPAI Store and have globally distributed Implementations of - AIFs conforming to MPAI-AIF. - AIWs and AIMs conforming to MPAI Application Standards.
Users' benefits	- Rely on Implementations of AIWs and AIMs whose Functions have been reviewed during standardisation. - Have a degree of Explainability of the AIW operation because the AIM Functions and the data Formats are known.
Market's benefits	- Open AIW and AIM markets foster competition leading to better products. - Competition of AIW and AIM Implementations fosters AI innovation.
MPAI Store's role	- Tests Conformance of Implementations with the relevant MPAI Standard. - Verifies Implementations' security. - Indicates unambiguously that Implementations are Level 2.

Level 3 Interoperability

MPAI does not generally set standards on how and with what data an AIM should be trained. This is an important differentiator that promotes competition leading to better solutions. However, the performance of an AIM is typically higher if the data used for training are in greater quantity and more in tune with the scope. Training data that have large variety and cover the spectrum of all cases of interest in breadth and depth typically lead to Implementations of higher "quality".

For Level 3, MPAI normatively specifies the process, the tools and the data or the characteristics of the data to be used to Assess the Grade of Performance of an AIM or an AIW.

| Implementers' benefits	May claim their Implementations have passed Performance Assessment.
Users' benefits	Get assurance that the Implementation being used performs correctly, e.g., it has been properly trained.

Market's benefits	Implementations' Performance Grades stimulate the development of more Performing AIM and AIW Implementations.
MPAI Store's role	- Verifies the Implementations' security - Indicates unambiguously that Implementations are Level 3.

The MPAI ecosystem

The following *Figure 2* is a high-level description of the MPAI ecosystem operation applicable to fully conforming MPAI implementations as specified in the Governance of the MPAI Ecosystem Specification [1]:

1. MPAI establishes and controls the not-for-profit MPAI Store.
2. MPAI appoints Performance Assessors.
3. MPAI publishes Standards.
4. Implementers submit Implementations to Performance Assessors.
5. If the Implementation Performance is acceptable, Performance Assessors inform Implementers and MPAI Store.
6. Implementers submit Implementations to the MPAI Store
7. MPAI Store verifies security and Tests Conformance of Implementation.
8. Users download Implementations and report their experience to MPAI.
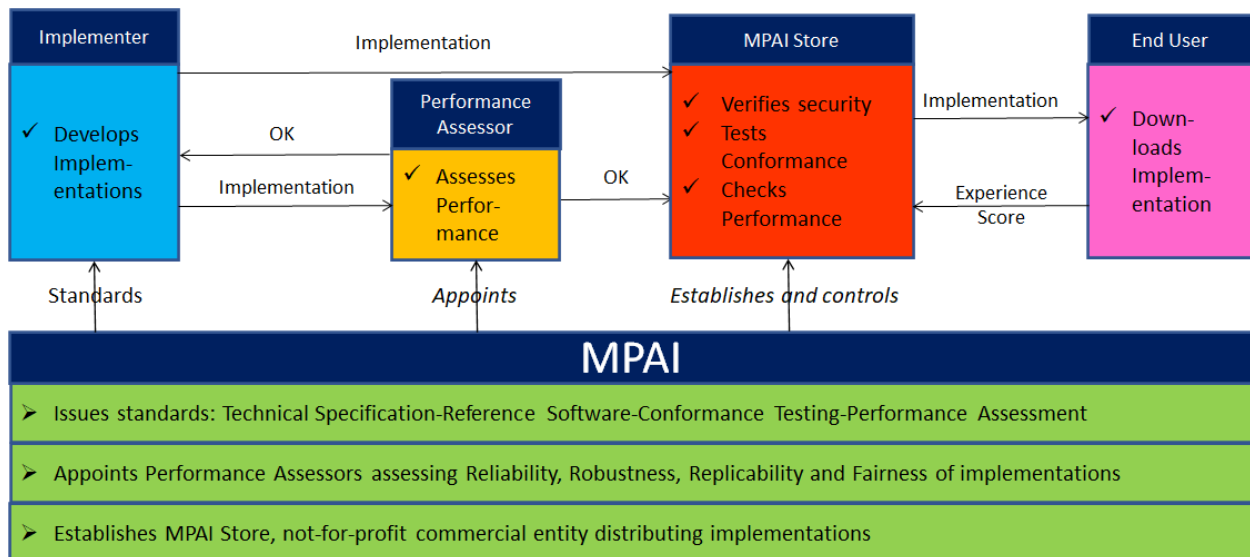
Figure 2 – The MPAI ecosystem operation

Annex 4 – Patent declarations

The MPAI Artificial Intelligence Framework (MPAI-AIF) Technical Specification has been developed according to the process outlined in the MPAI Statutes [10] and the MPAI Patent Policy [11].
The following entities have agreed to licence their standard essential patents reading on the MPAI Artificial Intelligence Framework (MPAI-AIF) Technical Specification according to the MPAI-AIF Framework Licence [12]:

Entity	Name	email address
Speech Morphing Inc.	Fathy Yassa	fathy@speechmorphing.com