INTERNATIONAL ORGANISATION FOR STANDARDISATION

ORGANISATION INTERNATIONALE DE NORMALISATION

**ISO/IEC JTC 1/SC 29/WG 11**

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC 1/SC 29/WG 11/ N16715**

**Jan. 2017, Geneva, Switzerland**

|  |  |
| --- | --- |
| **Source:** | **3DG** |
| **Title:** | **Collection of IoMTW related ideas** |
| **Author:** | **Sungmoon Chun, Anna Yang, Jae-Gon Kim, Miran Choi, Sang-Kyun Kim, Mihai Mitrea, Sunghan Kim** |
| **Contact :** | **Marius Preda** |

Note: this document is a collection of various contributions presented in previous meetings. It should not be interpreted as a starting point for IoMTW standardisation. This documentation has no impact on the evaluation of answers to the CfP, whatever technology presented in this document should follow the evaluation process.

[1 Introduction 4](#_Toc453243594)

[2 Data formats for IoMTW 4](#_Toc453243595)

[2.1 Root elements 4](#_Toc453243596)

[2.1.1 Introduction 4](#_Toc453243597)

[2.1.2 XML Schema 5](#_Toc453243598)

[2.2 Base types 6](#_Toc453243599)

[2.2.1 Introduction 6](#_Toc453243600)

[2.3 IoMW elements 8](#_Toc453243601)

[2.3.1 *IoMWDescriptionType* 8](#_Toc453243602)

[2.3.1.1 Introduction 8](#_Toc453243603)

[2.3.1.2 XML Schema 9](#_Toc453243604)

[2.3.1.2.1 Syntax 9](#_Toc453243605)

[2.3.1.2.2 Semantics 9](#_Toc453243606)

[2.3.2 *WearableDeviceType* 9](#_Toc453243607)

[2.3.2.1 Syntax 9](#_Toc453243608)

[2.3.2.2 Semantics 10](#_Toc453243609)

[2.3.3 *SensorType* 10](#_Toc453243610)

[2.3.3.1 Syntax 10](#_Toc453243611)

[2.3.3.2 Semantics 11](#_Toc453243612)

[2.3.4 *IndicatorType* 11](#_Toc453243613)

[2.3.4.1 Syntax 11](#_Toc453243614)

[2.3.4.2 Semantics 12](#_Toc453243615)

[2.3.5 *MWImageAnalysisType* 12](#_Toc453243616)

[2.3.5.1 Syntax 12](#_Toc453243618)

[2.3.5.2 Semantics 13](#_Toc453243619)

[2.4 IoMT elements 14](#_Toc453243620)

[2.4.1 *IoMTDescriptionType* 14](#_Toc453243621)

[2.4.1.1 Introduction 14](#_Toc453243622)

[2.4.1.2 XML Schema 14](#_Toc453243623)

[2.4.1.2.1 *Syntax* 14](#_Toc453243624)

[2.4.1.2.2 *Semantics* 14](#_Toc453243625)

[2.4.2 *IoTDeviceType* 15](#_Toc453243626)

[2.4.3 *SensorType* 15](#_Toc453243627)

[2.4.4 *IndicatorType* 15](#_Toc453243628)

[2.5 Data elements 15](#_Toc453243629)

[2.5.1 *DataDescriptionType* 15](#_Toc453243630)

[2.5.1.1 *Introduction* 15](#_Toc453243631)

[2.5.1.2 *XML Schema* 15](#_Toc453243632)

[2.5.1.2.1 *Syntax* 15](#_Toc453243633)

[2.5.1.2.2 *Semantics* 16](#_Toc453243634)

[2.5.1.3 *ProcessingDataType* 16](#_Toc453243635)

[2.5.1.3.1 *Syntax* 16](#_Toc453243636)

[2.5.1.3.2 *Semantics* 16](#_Toc453243637)

[2.5.2 *IntermediateDataType* 17](#_Toc453243638)

[2.5.2.1 *Syntax* 17](#_Toc453243639)

[2.5.2.2 *Semantics* 17](#_Toc453243640)

[2.5.3 *HandGestureType* 18](#_Toc453243641)

[2.5.3.1 *Syntax* 18](#_Toc453243642)

[2.5.3.2 </complexType> *Semantics* 18](#_Toc453243643)

[2.5.4 *HandContourType* 19](#_Toc453243644)

[2.5.4.1 *Syntax* 19](#_Toc453243645)

[2.5.4.2 Semantics 19](#_Toc453243646)

[2.6 GroupBezierCurveType 20](#_Toc453243647)

[2.6.1 Syntax 20](#_Toc453243648)

[2.6.2 Semantics 20](#_Toc453243649)

[2.7 BezierCurveType 21](#_Toc453243650)

[2.7.1 Syntax 21](#_Toc453243651)

[2.7.2 Semantics 21](#_Toc453243652)

[2.8 HandTrajectoryType 22](#_Toc453243653)

[2.8.1 Syntax 22](#_Toc453243654)

[2.8.2 Semantics 22](#_Toc453243655)

[2.9 Coordination2DType 22](#_Toc453243656)

[2.9.1 Syntax 22](#_Toc453243657)

[2.9.2 Semantics 23](#_Toc453243658)

[2.10 *MediaDataType* 23](#_Toc453243659)

[2.10.1.1 *Syntax* 23](#_Toc453243660)

[2.10.1.2 *Semantics* 23](#_Toc453243661)

[2.11 PUnit elements 24](#_Toc453243662)

[2.11.1 *PUintDescriptionType* 24](#_Toc453243663)

[2.11.1.1 *Introduction* 24](#_Toc453243664)

[2.11.1.2 *XML Schema* 24](#_Toc453243665)

[2.11.1.2.1 *Syntax* 24](#_Toc453243666)

[2.11.1.2.2 *Semantics* 24](#_Toc453243667)

[2.11.2 *GestureRecognitionType* 25](#_Toc453243668)

[2.11.2.1 *Syntax* 25](#_Toc453243669)

[2.11.2.2 *Semantics* 25](#_Toc453243670)

[2.11.3 *PUImageAnalysisType* 26](#_Toc453243671)

[2.11.3.1 *Syntax* 26](#_Toc453243672)

[2.11.3.2 *Semantics* 26](#_Toc453243673)

[2.11.4 *SpeechRecognitionType* 27](#_Toc453243674)

[2.11.4.1 *Syntax* 27](#_Toc453243675)

[2.11.4.2 *Semantics* 28](#_Toc453243676)

[2.11.5 *SpeechSynthesisType* 28](#_Toc453243677)

[2.11.5.1 *Syntax* 28](#_Toc453243678)

[2.11.5.2 *Semantics* 29](#_Toc453243679)

[2.12 Cmmd elements 29](#_Toc453243680)

[2.12.1 *CommandDescriptionType* 29](#_Toc453243681)

[2.12.1.1 *Introduction* 29](#_Toc453243682)

[2.12.1.2 *XML Schema* 29](#_Toc453243683)

[2.12.1.2.1 *Syntax* 29](#_Toc453243684)

[2.12.1.2.2 *Semantics* 30](#_Toc453243685)

[2.12.2 *CommandInteractionType* 30](#_Toc453243686)

[2.12.2.1 *Syntax* 30](#_Toc453243687)

[2.12.2.2 *Semantics* 30](#_Toc453243688)

[3 APIs for IoMT Camera System 31](#_Toc453243689)

[3.1 Introduction 31](#_Toc453243690)

[3.2 IoMT APIs and Descriptions 31](#_Toc453243691)

[4 API Instances for Wearable Applications for IoMTW 37](#_Toc453243692)

[4.1 Gesture detection API 37](#_Toc453243693)

[4.2 Gesture recognition API 38](#_Toc453243694)

[4.3 Self-adaptive multimedia API 40](#_Toc453243695)

[4.4 Blind Person 42](#_Toc453243696)

[4.4.1 Blind Person 1 42](#_Toc453243697)

[4.4.2 Blind Person 2 49](#_Toc453243698)

[4.4.3 Blind Person 3 53](#_Toc453243699)

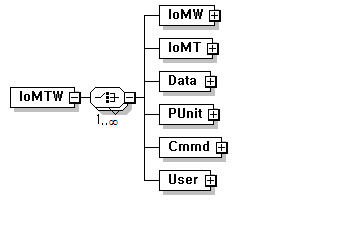
[5 References 54](#_Toc453243700)

# Introduction

This document includes data formats and APIs contributed in 115h Geneva Meeting.

# Data formats for IoMTW

The proposed root elements are designed based on a conceptual model of Internet of Media related Thing and Wearable (IoMTW) [1] to encompass whole metadata that would be needed to support diverse use cases in IoMTW.



< Root elements of IoMTW >

# Root elements

## Introduction

This sub clause specifies the root elements which shall be used as the topmost element in a description. The root element, IoMTW, provides any combination of elements of Data, Cmmd, IoMT, IoMW, PUnit, and User for providing description of both applications of Internet of Media-centric Things and Wearable. The combination also provides individual description by using each of the elements depending on a given application.

Data is used to describe data to be delivered and stored in IoMTW, which can be classified into processing data and media data. The processing data include the incoming data from input devices equipped in IoT/wearable devices such as stereo image sequence, voice, etc., and intermediate data that are usually generated during the processing of incoming data. On the other hand, the audiovisual data to be rendered and presented to a user can be described as media data.

The element of Cmmd is instantiated to describe commands to control IoT/wearable devices, which consist of interaction commands and acting commands. The interaction commands are used for interaction between a user and IoT/Wearable devices such as application SW control, media play control, I/O devices control, etc. The interaction commands can be invoked as diverse types of user inputs such as hand gesture, voice, touch pad, etc. The acting commands are used to control sensors equipped in/connected to IoT/Wearable devices.

The description about media-centric devices called MThing that can be IoT devices or wearable devices, which may include the type information and equipped I/O devices information of MThing, is created by using the elements of IoMT and IoMW, respectively.

The element of PUnit is used to describe information related to the processing to be done in processing units that can be located in MThing or remotely connected by networks. The processing may include gesture recognition, voice recognition, image analysis, speech synthesis, video rendering, etc. In general, a set of interaction commands or media data to be presented to a user can be generated by the results of such processing in processing units.

## XML Schema

This sub clause presents the specification of the root elements in terms of syntax and semantics.

### Syntax

<!-- ################################################ -->

<!-- Definition of IoMTW root elements -->

<!-- ################################################ -->

<element name="IoMTW">

<complexType>

<choice maxOccurs="unbounded">

<element name="IoMT" type="IoMTW:IoMTDescriptionType"/>

<element name="IoMW" type="IoMTW:IoMWDescriptionType"/>

<element name="Data" type="IoMTW:DataDescriptionType"/>

<element name="PUnit" type="IoMTW:ProcessingUnitDescriptionType"/>

<element name="Cmmd" type="IoMTW:CommandDescriptionType"/>

<element name="User" type="IoMTW:UserDescriptionType"/>

</choice>

</complexType>

</element>

<element name="IoMT" type="IoMTW:IoMTDescriptionType"/>

<element name="IoMW" type="IoMTW:IoMWDescriptionType"/>

<element name="Data" type="IoMTW:DataDescriptionType"/>

<element name="PUnit" type="IoMTW:ProcessingUnitDescriptionType"/>

<element name="Cmmd" type="IoMTW:CommandDescriptionType"/>

<element name="User" type="IoMTW:UserDescriptionType"/>

### Semantics

| *Name* | *Definition* |
| --- | --- |
| IoMTW | Serves as the root element of IoMTW description. IoMTW shall be used as the topmost element in a description. |
| IoMT | Describes an instance of one IoMT description. An IoMT description can be used to represent information of the IoMT device itself, which includes the type and equipped I/O devices information. |
| IoMW | Describes an instance of one IoMW description. An IoMW description can be used to represent information of the IoMW device itself, which includes the type and equipped I/O devices information. |
| Data | Describes an instance of data description. A data description can be used to represent information of data, which includes processing data incoming from input devices or intermediate data generated during the processing, and media data to be presented to a user. |
| Cmmd | Describes an instance of command description. A command description can be used to represent commands, which includes interaction commands for interaction between a user and IoT/Wearable devices, and acting commands for controlling sensors equipped in/connected to IoT/Wearable devices. |
| PUnit | Describes an instance of information of processing unit description. A processing unit description can be used to represent information of processing to be done in a processing unit. |
| User | Describes an instance of user information description. A user description can be used to represent information related to a user. |

# Base types

## Introduction

This sub clause specifies the base types of the description tools. Each description tool extends its own base type.

### Syntax

<!-- ################################################ -->

<!-- Definition of IoMTW base types -->

<!-- ################################################ -->

<!-- Definition of IoMT description base type -->

<complexType name="BaseIoMTType" abstract="true"/>

<!-- Definition of IoMW description base type -->

<complexType name="BaseIoMWType" abstract="true"/>

<!-- Definition of data description base type -->

<complexType name="BaseDataType" abstract="true"/>

<!-- Definition of processing unit description base type -->

<complexType name="BasePUnitType" abstract="true"/>

<!-- Definition of command description base type -->

<complexType name="BaseCmmdType" abstract="true">

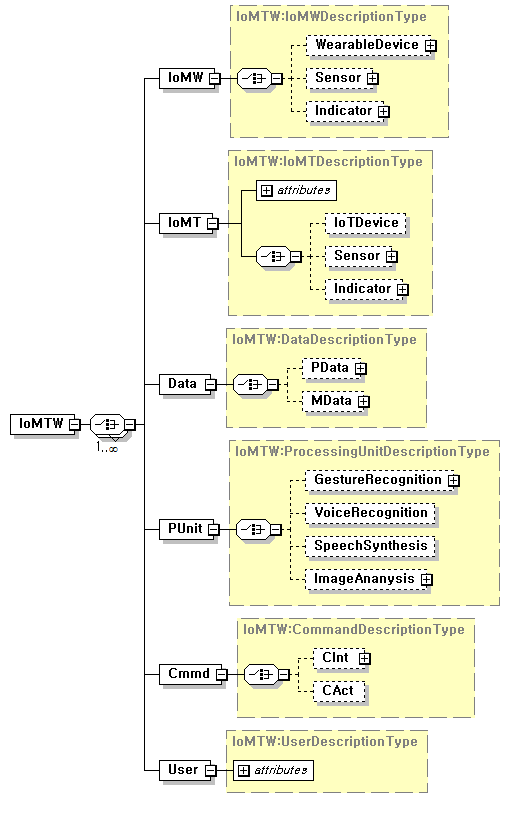
<!-- Definition of user description base type -->

<complexType name="BaseUserType" abstract="true"/>

### Semantics

| *Name* | *Definition* |
| --- | --- |
| BaseIoMTType | Provides a base abstract type for one of element of IoMTType to provide Internet of media-centric Thing specific properties. |
| BaseIoMWType | Provides a base abstract type for one of element of IoMWType to provide media-centric wearable specific properties. |
| BaseDataType | Provides a base abstract type for one of element of DataType to provide data specific properties. |
| BasePUnitType | Provides a base abstract type for one of element of PUnitType to provide processing unit specific properties. |
| BaseCmmdType | Provides a base abstract type for one of element of CmdType to provide processing unit specific properties. |
| BaseUserType | Provides a base abstract type for one of element of User to provide user specific properties. |

# IoMW elements



< common elemets of IoMTW >

## *IoMWDescriptionType*

## Introduction

IoMWDescriptionType: The description about a media-centric wearable device, which may include the type information and equipped I/O devices (sensor, indicator) information of MThing, is created by using the element of IoMW.

## XML Schema

This sub clause presents the specification of the IoMW description in terms of syntax and semantics.

## Syntax

<!-- ################################################ -->

<!-- Definition of Media-centric Wearable Description -->

<!-- ################################################ -->

<complexType name="IoMWDescriptionType">

<choice>

<element name="WearableDevice" type="IoMTW:WearableDeviceType"

minOccurs="0"/>

<element name="Sensor" type="IoMTW:SensorType" minOccurs="0"/>

<element name="Indicator" type="IoMTW:IndicatorType" minOccurs="0"/>

<element name=”MWImageAnalysis” type=”IoMTW:MWImageAnalysisType” minOccurs=”0”/>

</choice>

</complexType>

## Semantics

| *Name* | *Definition* |
| --- | --- |
| IoMWDescription Type | Provides an abstract of IoMW description. An IoMW description represents information of the IoMW device itself, which includes the type and equipped I/O devices information. |
| WearableDevice | Describes information of the IoMW device itself, which includes the type and equipped I/O devices information. |
| Sensor | Describes information of the sensors equipped in the IoMW device. |
| Indicator | Describes information of indicators equipped in the IoMW device. |
| MWImageAnalysis | Describes information of Image Analysis in the IoMW device. |

## *WearableDeviceType*

The WearableDeviceType is used to represent information of the IoMW device itself including type information.

## Syntax

<!-- ################################################ -->

<!-- Definition of Wearable Device -->

<!-- ################################################ -->

<complexType name="WearableDeviceType">

<choice>

<element name="SmartWatch" type=" IoMTW:SmartWatchType"

minOccurs="0"/>

<element name="SmartGlasses" type="IoMTW:SmartGlassesType"

minOccurs="0"/>

<element name="SmartShirt" type="IoMTW:SmartShirtType"

minOccurs="0"/>

<element name=”SmartShoes" type="IoMTW:SmartShoesType"

minOccurs="0"/>

<element name=”SmartHeadphone" type="IoMTW:SmartHeadphoneType"

minOccurs="0"/>

</choice>

</complexType>

## Semantics

| *Name* | *Definition* |
| --- | --- |
| WearableDeviceType | Provides an abstract of wearable device description. A wearable device description represents information of the M- Wearable device itself including type information. |
| SmartWatch | Describes information of a smart watch which may equip with camera. |
| SmartGlasses | Describes information of smart glasses which may equip with mono camera and/or stereo camera. |
| SmartShirt | Describes information of a smart shirt or D-shirt which may equip with healthcare sensors. |
| SmartShoes | Describes information of smart shoes which may equip with healthcare sensors. |
| SmartHeadphone | Describes information of smart headphone which may be equipped with regular and in-ear microphones. |

## *SensorType*

The SensorType is used to represent information of sensors equipped in the media-centric devices of IoT/wearable devices, which may be input and/or output devices.

## Syntax

<!-- ################################################ -->

<!-- Definition of Sensor -->

<!-- ################################################ -->

<complexType name="SensorType">

<choice>

<element name="MonoCamera" type="IoMTW:MonoCameraType"

minOccurs="0"/>

<element name="StereoCamera" type=" IoMTW:StereoCameraType"

minOccurs="0"/>

<element name="TouchPad" type=" IoMTW:TouchPadType" minOccurs="0"/>

<element name=”Gyro" type=" IoMTW:GyroType" minOccurs="0"/>

<element name=”InfraredCamera" type="IoMTW: InfraredCameraType"

minOccurs="0"/>

<element name=”Accelerometer" type=" IoMTW:AccelerometerType"

minOccurs="0"/>

<element name=”RegularMic" type=" IoMTW:RegularMicType"

minOccurs="0"/>

<element name=”InearMic" type=" IoMTW:InearMicType"

minOccurs="0"/>

</choice>

<attribute name="input" type="boolean" default="false"/>

</complexType>

## Semantics

| *Name* | *Definition* |
| --- | --- |
| SensorType | Provides an abstract of sensor description. Describes information of the sensors equipped in the media-centric devices, which may be input and/or output devices. |
| MonoCamera | Describes information of a mono camera equipped in the media-centric devices. |
| StereoCamera | Describes information of a stereo camera equipped in the media-centric devices. |
| TouchPad | Describes information of a touch pad equipped in the media-centric devices. |
| Gyro | Describes information of a gyro equipped in the media-centric devices. |
| InfraredCamera | Describes information of an infrared camera equipped in the media-centric devices. |
| Accelerometer | Describes information of an accelerometer equipped in the media-centric devices. |
| RegularMic | Describes information of a microphone equipped in the media-centric devices. |
| InearMic | Describes information of an in-ear microphone equipped in the media-centric devices. |

## *IndicatorType*

The IndicatorType is used to represent information of the indicators equipped in the media centric devices of IoT/wearable devices, which may be input and/or output devices.

## Syntax

<!-- ################################################ -->

<!-- Definition of Indicator -->

<!-- ################################################ -->

<complexType name="IndicatorType">

<choice>

<element name="MonoDisplay" type="IoMTW:MonoDisplayType"

minOccurs="0"/>

<element name="StereoDisplay" type="IoMTW:StereoDisplayType"

minOccurs="0"/>

<element name="DualDisplay" type="IoMTW:DualDisplayType"

minOccurs="0"/>

<element name="Voice" type="IoMTW:VoiceType" minOccurs="0"/>

<element name="Sound" type="IoMTW:SoundType" minOccurs="0"/>

<element name="Vibration" type="IoMTW:VibrationType" minOccurs="0"/>

<element name="Marker" type="IoMTW:MarkerType" minOccurs="0"/>

</choice>

<attribute name="input" type="boolean" default="false"/>

</complexType>

## Semantics

| *Name* | *Definition* |
| --- | --- |
| IndicatorType | Provides an abstract of indicator description. Describes information of the indicators equipped in the media-centric devices, which may be input and/or output devices. |
| MonoDisplay | Describes information of a mono display equipped in the media-centric devices. |
| StereoDisplay | Describes information of a stereo display equipped in the media-centric devices. |
| DualDisplay | Describes information of a dual display equipped in the media-centric devices. |
| Voice | Describes information of a voice indicator equipped in the media-centric devices. |
| Sound | Describes information of a sound indicator equipped in the media-centric devices. |
| Vibration | Describes information of a vibration indicator equipped in the media-centric devices. |
| Marker | Describes information of a marker equipped in the media-centric devices. |

## *MWImageAnalysisType*

The MWImageAnalysisType is used to represent information of the image analysis equipped in IoT/wearable devices, which may be input and/or output devices. Image analysis may include object detection, motion detection, pattern detection, segmentation, enhancement, edge and shape extraction, etc.



## Syntax

<!-- ################################################ -->

<!-- Definition of MWImageAnalysis -->

<!-- ################################################ -->

<complexType name=”MWImageAnalysisType”>

<choice>

<element name=”FaceDetection” type=”IoMTW:FaceDetectionType” minOccurs=”0”/>

<element name=”LaneDetection” type=”IoMTW:LaneDetectionType” minOccurs=”0”/>

<element name=”PedestrianDetection” type=”IoMTW:PedestrianDetectionType” minOccurs=”0”/>

<element name=”ObjectDetection” type=”IoMTW:ObjectDetectionType” minOccurs=”0”/>

<element name=”MotionDetection” type=”IoMTW:MotionDetectionType” minOccurs=”0”/>

<element name=”PatternDetection” type=”IoMTW:PetternDetectionType” minOccurs=”0”/>

<element name=”EdgeDetection” type=”IoMTW:EdgeDetectionType” minOccurs=”0”/>

<element name=”Segmentation” type=”IoMTW:SegmentationType” minOccurs=”0”/>

<element name=”Enhancement” type=”IoMTW:EnhancementType” minOccurs=”0”/>

</choice>

<attribute name=”mode” type=”string”/>

<attribute name=”colorspace” type=”string”/>

</complexType>

## Semantics

| *Name* | *Definition* |
| --- | --- |
| MWImageAnalysisType | Provides an abstract of indicator description. Describes information of the Image Analysis in the IoT/wearable Device, which may be input and/or output devices. |
| FaceDetection | Describes information of a Face Detection in the Image Analysis. |
| LaneDetection | Describes information of a Lane Detection in the Image Analysis. |
| PedestrianDetection | Describes information of a Pedestrian Detection in the Image Analysis. |
| ObjectDetection | Describes information of a Object Detection in the Image Analysis. |
| MotionDetection | Describes information of a Motion Detection in the Image Analysis. |
| PatternDetection | Describes information of a Pattern Detection in the Image Analysis. |
| EdgeDetection | Describes information of a Edge Detection in the Image Analysis. |
| Segmentation | Describes information of a Segmentation in the Image Analysis. |
| Enhancement | Describes information of a Enhancement in the Image Analysis. |

# IoMT elements

## *IoMTDescriptionType*

## Introduction

IoMTDescriptionType: The description about a Internet of media-centric Thing device, which may include the type information and equipped I/O devices (sensor, indicator) information of MThing, is created by using the element of IoMT.

## XML Schema

This sub clause presents the specification of the IoMT description in terms of syntax and semantics.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Internet of media-centric Thing Description -->

<!-- ################################################ -->

<complexType name="IoMTDescriptionType">

<choice>

<element name="IoTDevice" type="IoMTW:IoTDeviceType"

minOccurs="0"/>

<element name="Sensor" type="IoMTW:SensorType" minOccurs="0"/>

<element name="Indicator" type="IoMTW:IndicatorType" minOccurs="0"/>

</choice>

</complexType>

## *Semantics*

| *Name* | *Definition* |
| --- | --- |
| IoMTDescription Type | Provides an abstract of IoMT description. An IoMT description represents information of the IoMT device itself, which includes the type and equipped I/O devices information. |
| IoTDevice | Describes information of the IoMT device itself, which includes the type and equipped I/O devices information. |
| Sensor | Describes information of the sensors equipped in the media-centric device. |
| Indicator | Describes information of indicators equipped in the media-centric device. |

## *IoTDeviceType*

The IoTDeviceType is used to represent information of the IoMT device itself including type information.

The specifications of IoTDeviceType in terms of syntax and semantics are to be defined.

## *SensorType*

The SensorType is used to represent information of sensors equipped in the media-centric devices of IoT/wearable devices, which may be input and/or output devices.

The specifications of SensorType in terms of syntax and semantics are presented in sub clauses of 2.3.1 and 2.3.2, respectively.

## *IndicatorType*

The IndicatorType is used to represent information of the indicators equipped in the media centric devices of IoT/wearable devices, which may be input and/or output devices.

The specifications of IndicatorType in terms of syntax and semantics are presented in sub clauses of 2.4.1 and 2.4.2, respectively.

# Data elements

## *DataDescriptionType*

## *Introduction*

DataDescriptionType is used to describe the data to be delivered and stored in the IoMTW framework, which can be classified into processing data and media data. The processing data includes the incoming data from input devices of media-centric devices such as stereo image sequence, voice, etc., and the intermediate data generated during the processing of incoming data. On the other hand, multimedia data to be rendered and presented to a user can be described as media data.

## *XML Schema*

This sub clause presents the specification of the Data Description in terms of syntax and semantics.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Data Description -->

<!-- ################################################ -->

<complexType name="DataDescriptionType">

<choice>

<element name="PData" type="IoMTW:ProcessingDataType" minOccurs="0"/>

<element name="MData" type="IoMTW:MediaDataType" minOccurs="0"/>

</choice>

</complexType>

## *Semantics*

| *Name* | *Definition* |
| --- | --- |
| DataDescriptionType | Provides an abstract of data description. A data description can be used to represent information of data, which includes the processing data incoming from input devices or the generated intermediate data during the processing, and media data to be presented to a user. |
| PData | Describes an instance of processing data. A description of processing data can be used to represent information of processing data, which includes the data incoming from input devices or the generated intermediate data during the processing. |
| MData | Describes an instance of media data. A media data can be used to represent information of media data to be presented to a user. |

## *ProcessingDataType*

The ProcessingDataType is used to represent information of processing data, which includes the data incoming from input devices or the intermediate data generated during the processing.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Processing Data -->

<!-- ################################################ -->

<complexType name="ProcessingDataType">

<choice>

<element name="ImageSequence" type="IoMTW:ImageSequenceType"

minOccurs="0"/>

<element name="StereoImageSequence"

type="IoMTW:StereoImageSequenceType" minOccurs="0"/>

<element name="Voice" type="IoMTW:VoiceType" minOccurs="0"/>

<element name="Speech/audio" type="IoMTW:SpeechType" minOccurs="0"/>

<element name="Text" type="IoMTW:TextType" minOccurs="0"/>

<element name="IntermediateData" type="IoMTW:IntermediateDataType"

minOccurs="0"/>

</choice>

<attribute name="input" type="boolean" default="false"/>

</complexType>

## *Semantics*

| *Name* | *Definition* |
| --- | --- |
| ProcessingDataType | Provides an abstract of processing data description. A processing data description can be used to represent information of processing data, which include the data incoming from input devices or the generated intermediate data during the processing. |
| ImageSequence | Describes image sequences incoming to an input device of media-centric device. The image sequence data is generally raw data format. |
| StereoImageSequence | Describes stereo image sequences incoming to an input device of media-centric device. The stereo image sequence data is generally raw data format. |
| IntermediateData | Describes an intermediate data generated as a result of the processing of incoming data. |
| Voice | Describes voice data incoming to an input device of media-centric device. The voice data is generally raw data format. |
| Speech | Describes speech data incoming to an input device of media-centric device. The speech data is generally raw data format. |
| Text | Describes text incoming to an input device of media-centric device. The text data is generally raw data format. |

## *IntermediateDataType*

The IntermediateDataType is used to represent intermediate data generated as a result of processing of the incoming data, which may include hand shape, object shape, etc.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Intermediate Data -->

<!-- ################################################ -->

<complexType name="IntermediateDataType">

<choice>

<element name="HandGesture" type="IoMTW:HandGestureType" minOccurs="0" maxOccurs="unbounded"/>

<element name="ObjectShape" type="IoMTW:ObjectShapeType"

minOccurs="0"/>

<element name="TextResult" type="IoMTW:TextType"

minOccurs="0"/>

</complexType>

## *Semantics*

| *Name* | *Definition* |
| --- | --- |
| IntermediateDataType | Provides an abstract of intermediate data description. A intermediate data description can be used to represent data generated as a result of the processing of incoming data, which may include hand shape, object shape, and etc. |
| HandGesture | Describes hand contour and/or hand trajectory extracted from the incoming image sequences by the processing of hand gesture detection. Hand gesture description can be used to generate hand gesture-based command for user interaction. |
| ObjectShape | Describes object shape extracted from the incoming image sequences by the processing of object detection. Object shape description can be used to generate object based command for user interaction. |
| TextResult | Describes text results which are output of speech recognition process. Text results can be used to generate speech based command for user interaction. |

## *HandGestureType*

The HandGestureType is used to general hand gesture in forms of hand contour and/or hand trajectory which are extracted from the incoming image sequences by the processing of hand gesture detection. Hand gesture description can be used to generate hand gesture-based command for user interaction.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Hand Gesture -->

<!-- ################################################ -->

<complexType name="HandGestureType">

<choice>

<element name="HandContour" type="IoMTW:HandContourType" minOccurs="0" maxOccurs="unbounded"/>

<element name="HandTrajectory" type="IoMTW:HandTrajectoryType" minOccurs="0"/>

minOccurs="0"/>

</choice>

<attribute name="dynamic" type="boolean" default="false"/>

<attribute name="bothHands" type="boolean" default="false"/>

<attribute name="threeDspace" type="boolean" default="false"/>

<attribute name="frameRate" type="decimal" use="optional" default="30"/>

</complexType>

## *Semantics*

| *Name* | *Definition* |
| --- | --- |
| HandContour | Describes hand contour extracted from the incoming image sequences by the processing of hand gesture detection. Hand contour description can be used to generate hand gesture based command for user interaction by the processing of hand gesture recognition. |
| HandTrajectory | Describes hand trajectory extracted from the incoming image sequences by the processing of hand trajectory extraction. Hand trajectory description can be used to generate hand gesture based command for user interaction by the processing of hand gesture recognition. |
| ObjectShape | Describes object shape extracted from the incoming image sequences by the processing of object detection. Object shape description can be used to exploit object shape in diverse IoMTW applications. |
| dynamic | Describe the indication whether the gesture is time varying or static. |
| bothHands | Describe the indication whether the gesture is generated by using either both hands or single hand. |
| threeDspace | Describe the indication whether the gesture is represented in 3-D space or 2-D space. For example, hand trajectory can be represented as a 3-D trajectory or 2-D trajectory that is the projection of 3-D trajectory the into 2-D space. |
| frameRate | Describe the frame rate of the incoming image sequence. framerate is used to calculate the time interval between consecutive hand contours or the speed of hand motion, etc. |

## *HandContourType*

The HandContourType is used to represent hand contour, from which gesture-based interaction commands can be generated as a result of gesture recognition.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Hand Contour -->

<!-- ################################################ -->

<complexType name="HandContourType">

<sequence>

<choice>

<element name="Coordinate" type="IoMTW:CoordinateType" minOccurs="0"/>

<element name="GroupBezierCurve" type="IoMTW:GroupBezierCurveType" minOccurs="0"/>

</choice>

<element name="CenterPosition" type="IoMTW:Coordination2DType" minOccurs="0"/>

</sequence>

<attribute name="useLastContour" type="boolean" default="false"/>

</complexType>

## Semantics

| *Name* | *Definition* |
| --- | --- |
| HandContourType | Provides abstracts of hand contour description. A hand contour description can be used to represent general hand contour, from which gesture-based interaction commands can be generated as a result of the gesture recognition. |
| Coordinate | Describes hand contour in a form of a set of coordinates in 2-D or 3-D space. |
| GroupBezierCurve | Describes hand contour or hand trajectory in a form of a set of curves in 2-D space. A curve can be represented in a form of a set of 2-D coordinates. |
| CenterPosition | Describes the center position of the detected hand contour in a form of a 2-D coordinates. Hand trajectory can be generated from a set of center positions of hands later. |
| useLastContour | Describes the indication whether the current contour description is available or the last available contour is used. When we don’t need to generate hand contour description of the detected hand for each frame or the valid hand contour is not detected for a certain frame, the last available contour description can be used instead as the current frame’s contour description. |

## GroupBezierCurveType

The GroupBezierCurvetype is used to describe a set of Bezier curves for hand contour or hand trajectory.

## Syntax

<!-- ################################################ -->

<!-- Definition of Group Bezier Curve -->

<!-- ################################################ -->

<complexType name="GroupBezierCurveType">

<sequence>

<element name="InitialStartPoint" type="IoMTW:InitialStartPointType"/>

<sequence maxOccurs="unbounded">

<element name="BezierCurve" type="IoMTW:BezierCurveType"/>

</sequence>

</sequence>

<attribute name="OrderOfBezierCurve" type="decimal" default="3"/>

<attribute name="FittingError" type="double", use="optional"/>

</complexType>

## Semantics

| *Name* | *Definition* |
| --- | --- |
| GroupBezierCurveType | Provides an abstract of a set of Bezier curves description. A group of Bezier curve description can be used to represent hand gesture or trajectory. |
| InitialStartPoint | Describes the initial point for the first Bezier curve of a set of Bezier curves representing the whole hand contour. By describing the initial point, the overlapping point between the consecutive curves can be described once instead of describing the same point redundantly. |
| BezierCurve | Describes a part of hand contour in a form of Bezier curve. For each Bezier curve, (the order of Bezier curve n + 1) points are used to fitting the given curve by polynomial. By using the initial point of the first Bezier curve, n-points are used to represent each curve of consecutive curves representing hand contour instead of (n + 1) points. Each point is described as a form of coordinate in 2-D space. |
| OrderOfBezierCurve | Describes the order of Bezier curve, which is the order of polynomial fitting the given curve. For n-th Bezier curve, (n + 1) points in 2-D space, which consists of the starting point, end point, and (n-1) control points, are used to represent the given curve. |
| FittingError | Describes the fitting error of the Bezier curve. The fitting error is used to represent the accuracy of Bezier curve in optional. |

## BezierCurveType

The BezierCurveType is used to represent one Bezier curve for hand contour or hand trajectory.

## Syntax

<!-- ################################################ -->

<!-- Definition of Bezier Curve -->

<!-- ################################################ -->

<complexType name="BezierCurveType">

<sequence>

<element name="ControlPoint" type="IoMTW:Coordination2DType" maxOccurs="unbounded"/>

<element name="StartEndPoint" type="IoMTW:Coordination2DType"/>

</sequence>

</complexType>

## Semantics

| *Name* | *Definition* |
| --- | --- |
| BezierCurveType | Provides an abstract of a Bezier curve description. A Bezier curve use (the order of Bezier curve n + 1) points to fit the given curve by polynomial. By using the initial point of the first Bezier curve, n-points are used to represent each curve of consecutive curves representing hand contour instead of (n + 1) points. Each point is described as a form of coordinate in 2-D space. |
| ControlPoint | Describes a control point for Bezier curve in a form of 2-D coordinate. |
| StartEndPoint | Describes a start and an end point of a Bezier curve in a form of a 2-D coordinate. Start and end point is the overlapping point between consecutive curves representing hand contour. |

## HandTrajectoryType

The HandTrajectoryType is used to describe hand trajectory, from which gesture-based interaction commands can be generated as a result of gesture recognition.

## Syntax

<!-- ################################################ -->

<!-- Definition of Hand Trajectory -->

<!-- ################################################ -->

<complexType name="HandTrajectoryType">

<choice>

<element name="GroupBezierCurve" type="IoMTW:GroupBezierCurveType" minOccurs="0"/>

<sequence maxOccurs="unbounded">

<element name="CenterPosition" type="IoMTW:Coordination2DType"/>

</sequence>

</choice>

<attribute name="dynamic" type="boolean" default="false"/>

<attribute name="trajectoryType" type="string" default="motionInterval"/>

</complexType>

## Semantics

| *Name* | *Definition* |
| --- | --- |
| HandTrajectoryType | Provides an abstract of hand trajectory description. A hand trajectory description can be used to represent general hand trajectory, from which gesture-based interaction commands can be generated as a result of the gesture recognition.  It can be represented by a group of Bezier curves or a sequence of center position of hand. |
| dynamic | Describes the indication whether the hand contour is combined with hand trajectory in the given user gesture. |
| trajectoryType | Describes the indication whether the trajectory is defined and generated in the unit of predefined time interval or in the moving duration. The type of motion interval (motionInterval) means the trajectory is generated during hand moving duration that is identified by motion detection. The type of time interval (timeInterval) indicates the trajectory is generated in the unit of predefined time interval. |

## Coordination2DType

The Coordination2Dtype is used to describe a point in 2-D space in a form of a 2-D coordinate.

## Syntax

<!-- ################################################ -->

<!-- Definition of 2D Coordination Data -->

<!-- ################################################ -->

<complexType name="Coordination2DType">

<attribute name="x" type="decimal"/>

<attribute name="y" type="decimal"/>

</complexType>

## Semantics

| *Name* | *Definition* |
| --- | --- |
| Coordination2DType | Provides an abstract of 2D coordinate description. A coordination 2D description can be used to represent general point of Bezier curve and trajectory using center point. |

## *MediaDataType*

The MediaDataType is used to represent information of media data to be presented to a user, which include video, audio, image, and/or image.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Media Data -->

<!-- ################################################ -->

<complexType name="MediaDataType">

<choice>

<element name="Video" type="IoMTW:VideoType" minOccurs="0"/>

<element name="Audio" type="IoMTW:AudioType" minOccurs="0"/>

<element name="Image" type="IoMTW:ImageType" minOccurs="0"/>

<element name="Text" type="IoMTW:TextType" minOccurs="0"/>

</choice>

<attribute name="compressed" type="boolean" default="false"/>

</complexType>

## *Semantics*

| *Name* | *Definition* |
| --- | --- |
| MediaDataType | Provides an abstract of media data description. A media data description can be used to represent information of media data to be presented to a user, which include video, audio, image, and/or image. |
| Video | Describes video data to be presented to a user. The video data can be a compressed format or a raw data format. |
| Audio | Describes audio data to be presented to a user. The audio data can be a compressed format or a raw data format. |
| Image | Describes image data to be presented to a user. The image data can be a compressed format or a raw data format. |
| Text | Describes text data to be presented to a user. The text data can be a compressed format or a raw data format. |

# PUnit elements

## *PUintDescriptionType*

## *Introduction*

PUnitDescriptionType is used to describe information related to the processing to be done in processing units that can be located in MThing or remotely connected by networks. The processing may include gesture recognition, voice recognition, image analysis, speech synthesis, video rendering, etc. In general, a set of interaction commands or media data to be presented to a user can be generated by the results of such processing.

## *XML Schema*

This sub clause presents the specification of the Processing Unit Description in terms of syntax and semantics.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Processing Unit Description -->

<!-- ################################################ -->

<complexType name="PUnitDescriptionType">

<choice>

<element name="GestureRecognition" type="IoMTW:GestureRecognitionType"

minOccurs="0"/>

<element name="VoiceRecognition" type="IoMTW:VoiceRecognitionType"

minOccurs="0"/>

<element name="SpeechSynthesis" type="IoMTW:SpeechSynthesisType"

minOccurs="0"/>

<element name="ImageAnalysis" type="IoMTW:PUImageAnalysisType"

minOccurs="0"/>

</choice>

</complexType>

## *Semantics*

| *Name* | *Definition* |
| --- | --- |
| PUnitDescriptionType | Provides an abstract of processing unit description. A processing unit description can be used to represent information of processing to be done in a processing unit. |
| GestureRecognition | Describes information of the processing of gesture recognition, which may process the incoming image or image sequences. |
| VoiceRecognition | Describes information of the processing voice recognition, which may process the incoming voice. |
| SpeechSynthesis | Describes information related to the processing of speech synthesis. |
| PUImageAnalysis | Describes information of the processing of image analysis, which may process the incoming image or image sequences. |

## *GestureRecognitionType*

The GestureRecognitionType is used to represent information of gesture recognition, which may include hand gesture, face gesture, head motion, and body gesture. Gesture can be used as an input of use for generating user interaction commands.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Gesture Recognition -->

<!-- ################################################ -->

<complexType name="GestureRecognitionType">

<choice>

<element name="HandGesture" type="IoMTW:HandGestureType"

minOccurs="0"/>

<element name="FaceGesture" type="IoMTW:FaceGestureType"

minOccurs="0"/>

<element name="HeadMotion" type="IoMTW:HeadMotionType" minOccurs="0"/>

<element name="BodyGesture" type="IoMTW:BodyGestureType"

minOccurs="0"/>

</choice>

</complexType>

## *Semantics*

| *Name* | *Definition* |
| --- | --- |
| GestureRecognitionType | Provide an abstract of description of gesture recognition, which can be done in a processing unit. |
| HandGesture | Describes information of hand gesture recognition, which can be done in a processing unit. |
| FaceGesture | Describes information of face gesture recognition, which can be done in a processing unit. |
| HeadMotion | Describes information of head motion recognition, which can be done in a processing unit. |
| BodyGesture | Describes information of body gesture recognition, which can be done in a processing unit. |

## *PUImageAnalysisType*

The PUImageAnalysisType represents information of image analysis, which can be done in a processing unit. Image analysis may include object detection, motion detection, pattern detection, segmentation, enhancement, edge and shape extraction, etc.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Image Analysis -->

<!-- ################################################ -->

<complexType name="PUImageAnalysisType">

<choice>

<element name="FaceDetection" type="IoMTW:FaceDetectionType"

minOccurs="0"/>

<element name="LaneDetection" type="IoMTW:LaneDetectionType"

minOccurs="0"/>

<element name="PedestrianDetection" type="IoMTW:PedestrianDetectionType" minOccurs="0"/>

<element name="ObjectDetection" type="IoMTW:ObjectDetectionType"

minOccurs="0"/>

<element name="MotionDetection" type="IoMTW:MotionDetectionType"

minOccurs="0"/>

<element name="PatternDetection" type="IoMTW:PatternDetectionType"

minOccurs="0"/>

<element name="EdgeDetection"

type="IoMTW:EdgeDetectionType" minOccurs="0"/>

<element name="Segmentation" type="IoMTW:SegmentationType"

minOccurs="0"/>

<element name="Enhancement" type="IoMTW:EnhancementType"

minOccurs="0"/>

<element name="FaceRecognition" type="IoMTW:FaceRecognitionType"

minOccurs="0"/>

<element name="ObjectRecognition" type="IoMTW:ObjectRecognitionType"

minOccurs="0"/>

<element name="MotionRecognition" type="IoMTW:MotionRecognitionType"

minOccurs="0"/>

<element name="VehiclePrediction" type="IoMTW:VehiclePredictionType"

minOccurs="0"/>

<element name="PedestrianPrediction" type="IoMTW:PedestrianPredictionType" minOccurs="0"/>

</choice>

</complexType>

## *Semantics*

| *Name* | *Definition* |
| --- | --- |
| PUImageAnalysisType | Provides an abstract of description of image analysis, which can be done in a processing unit. |
| FaceDetection | Describes information of face detection in an image or image sequences, which can be done in a processing unit. |
| LaneDetection | Describes information of lane detection in an image or image sequences, which can be done in a processing unit. |
| PedestrianDetection | Describes information of pedestrian detection in an image or image sequences, which can be done in a processing unit. |
| ObjectDetection | Describes information of object detection in an image or image sequences, which can be done in a processing unit. |
| MotionDetection | Describes information of motion detection in an image or image sequences, which can be done in a processing unit. |
| PatternDetection | Describes information of pattern detection in an image or image sequences, which can be done in a processing unit. |
| EdgeDetection | Describes information of edge detection in an image or image sequences, which can be done in a processing unit. |
| Segmentation | Describes information of segmentation in an image or image sequences, which can be done in a processing unit. |
| Enhancement | Describes information of enhancement in an image or image sequences, which can be done in a processing unit. |
| FaceRecognition | Describes information of face recognition in an image or image sequences, which can be done in a processing unit. |
| ObjectRecognition | Describes information of object recognition in an image or image sequences, which can be done in a processing unit. |
| MotionRecognition | Describes information of motion recognition in an image or image sequences, which can be done in a processing unit. |
| VehiclePrediction | Describes information of vehicle prediction in an image or image sequences, which can be done in a processing unit. |
| PedestrianPrediction | Describes information of pedestrian prediction in an image or image sequences, which can be done in a processing unit. |

## *SpeechRecognitionType*

The SpeechRecognitionType is used to represent information of speech recognition, which may include speech and speech features. The result of speech recognition can be used as an input of use for generating user interaction commands.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Speech Recognition -->

<!-- ################################################ -->

<complexType name="SpeechRecognitionType">

<sequence>

<choice>

<element name="SpeechInput" type="uri"

minOccurs="0"/>

<element name="SpeechFeature" type="IoMTW:SpeechFeatureType"

minOccurs="0"/>

</choice>

<element name="Language" type="language" minOccurs="0"/>

</sequence>

</complexType>

## *Semantics*

| *Name* | *Definition* |
| --- | --- |
| SpeechRecognitionType | Provide an abstract of description of speech recognition, which can be done in a processing unit. |
| SpeechInput | Describes Speech signal spoken by the user. |
| SpeechFeature | Compressed speech features extracted from the speech signal. |
| Language | Indicates the language of the input speech.  NOTE If present, the Language element shall take precedence over other language indications present within the speech input. |

## *SpeechSynthesisType*

The SpeechSynthesisType is used to represent information of speech synthesis, which may include speech and speech features. The result of speech synthesis can be used as an output to the user.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Speech Synthesis -->

<!-- ################################################ -->

<complexType name="SpeechSynthesisType">

<sequence>

<element name="TextInput" type=String

minOccurs="0"/>

<element name="SynthesizedSpeechFeature" type="IoMTW:SynthesizedSpeechFeatureType"

minOccurs="0"/>

<element name="Language" type="language" minOccurs="0"/>

</sequence>

</complexType>

## *Semantics*

| *Name* | *Definition* |
| --- | --- |
| SpeechSynthesisType | Provides an abstract description of speech synthesis, which can be done in a processing unit. |
| TextInput | Describes text input to be synthesized by the process of speech synthesis. |
| SynthesizedSpeechFeature | Output of synthesized speech features such as gender, tones and voice speed to be reflected in speech output. |
| Language | Indicates the language of the input speech.  NOTE If present, the Language element shall take precedence over other language indications present within the speech input. |

# Cmmd elements

## *CommandDescriptionType*

## *Introduction*

CommandDescriptionType is used to describe commands for controlling media-centric devices of IoT/wearable devices, which consists of interaction commands and acting commands. The interaction commands are used for interaction between a user and IoT/Wearable devices such as application SW control, media play control, I/O devices control, etc. The interaction commands can be invoked as diverse types of user inputs such as hand gesture, voice, touch pad, etc. The acting commands are used to control sensors equipped in/connected to IoT/Wearable devices.

## *XML Schema*

This sub clause presents the specification of the Command Description in terms of syntax and semantics.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Command Description -->

<!-- ################################################ -->

<complexType name="CommandDescriptionType">

<choice>

<element name="CInt" type="IoMTW:CommandInteractionType"

minOccurs="0"/>

<element name="CAct" type="IoMTW:CommandActingType" minOccurs="0"/>

</choice>

</complexType>

## *Semantics*

| *Name* | *Definition* |
| --- | --- |
| CommandDescriptionType | Provides an abstract of command description. A command description can be used to represent commands, which includes interaction commands for interaction between a user and IoT/Wearable devices, and acting commands for controlling sensors equipped in/connected to IoT/Wearable devices. |
| CInt | Describes interaction commands. An interaction command description can be used to represent interaction commands for interaction between a user and IoT/Wearable devices, which include application control, media play control, I/O devices control, and etc. |
| CAct | Describes acting commands. An acting command description can be used to represent acting commands, which is used to control sensors equipped in/connected to IoT/Wearable devices. |

## *CommandInteractionType*

The CommandInteractionType can be used to represent to represent interaction commands for interaction between a user and IoT/Wearable devices, which include application SW control, media play control, I/O devices control, etc.

## *Syntax*

<!-- ################################################ -->

<!-- Definition of Command Interaction -->

<!-- ################################################ -->

<complexType name="CommandInteractionType">

<choice>

<element name="ApplicationControl" type="IoMTW:ApplicationControlType"

minOccurs="0"/>

<element name="MediaPlayControl" type="IoMTW:MediaPlayControlType"

minOccurs="0"/>

<element name="DeviceControl" type="IoMTW:DeviceControlType"

minOccurs="0"/>

</choice>

</complexType>

## *Semantics*

| *Name* | *Definition* |
| --- | --- |
| CommandInteractionType | Provides an abstract of command description. An interaction command description can be used to represent interaction commands for interaction between a user and IoT/Wearable devices, which include application control, media play control, I/O devices control, and etc. |
| ApplicationControl | Describes application SW control commands. A description of application control commands is used to represent commands for controlling applications provided in a wearable device. |
| MediaPlayControl | Describes media play control commands. A description of media play control commands is used to represent commands for controlling play of media in a wearable device, which may include paly, pause, fast forward, rewind, stop, and etc. |
| DeviceControl | Describes device control commands. A description of device control commands is used to represent commands for controlling a wearable device. |

# APIs for IoMT Camera System

# Introduction

In contribution M37018 [6] of the 113rd Geneva meeting, we presented data exchange examples of a IoMT(Internet of Media Things) Camera System. As a result of the discussion during the meeting, the example of the IoMT human detection and tracking system was included in the output document N15728 [1]. The modified APIs were presented in the 115th Geneva meeting [3].

In this document, we further present modified version of basic APIs to implement such an IoMT camera system. Figure 1 shows a component diagram of the proposed IoMT human detection and tracking system

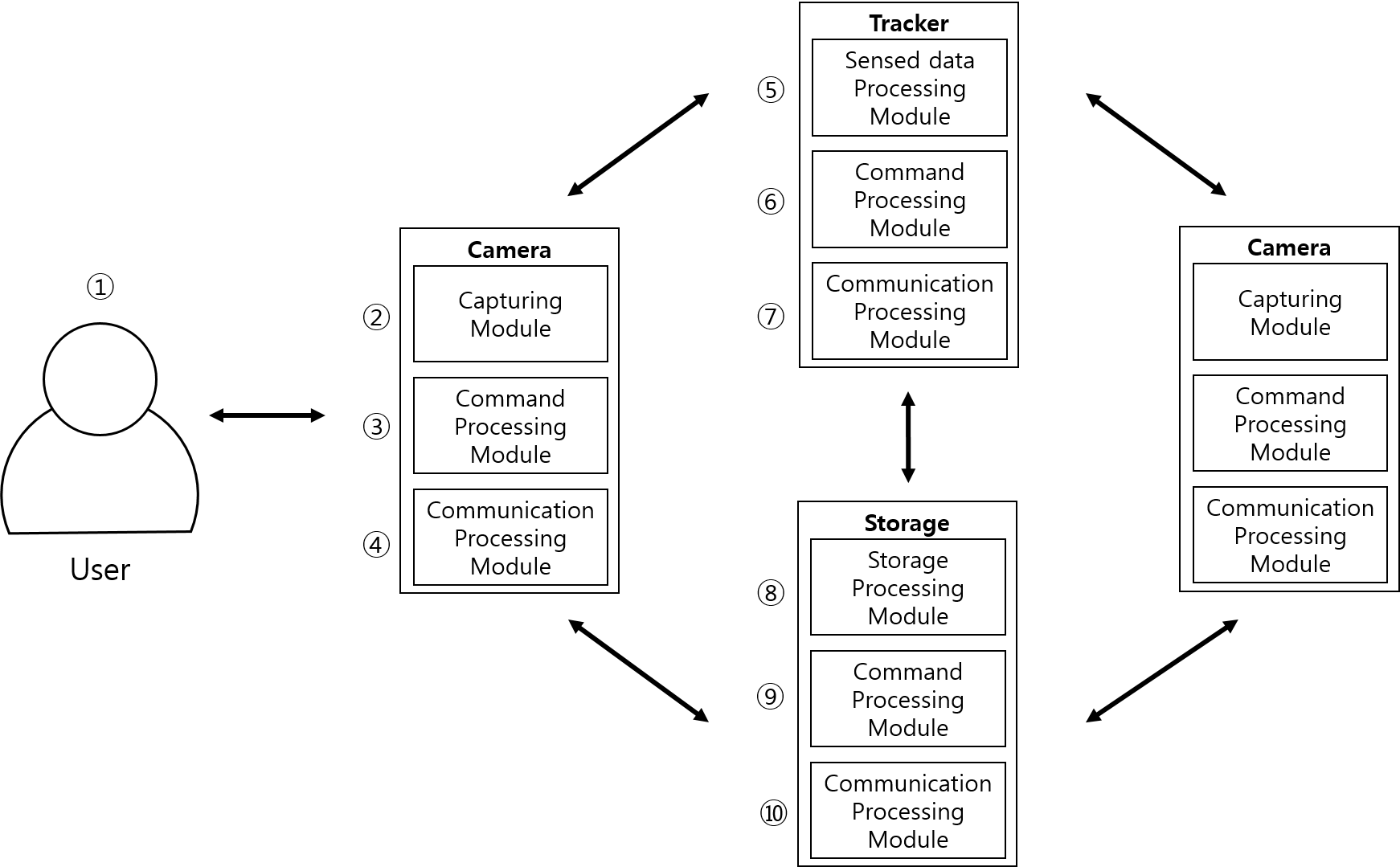


Figure 1 IoMT human detection and tracking system [1]

# IoMT APIs and Descriptions



Figure 2. Entire API Map.



Figure 3. Class diagram for IoMT Camera APIs.

**Class MThingBaseType extends MThingListenerProvider**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| MThingBaseType() | |
| Default constructor. | |
|  | |
| MThingBaseType(String id) | |
|  | |
| MThingBaseType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| String | id |
|  |  |
| String | characteristics |
|  | MThing capability instances created by MPEG-V Part 2 standard. |
|  |  |
| Boolean | isBusy |
|  |  |
| String | ipAddress |
|  |  |
| Integer | port |
|  |  |
| List<MThingBaseType> | mThingList |
|  | List of all the neighboring MThings. |
|  |  |
| List<MThingBaseType> | bindMThingList |
|  | List of all MThings, which perform a task. |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |
| Integer | connect() |
|  | Connect to server using IP Address and port. |
|  |  |
| Integer | connect(String ipAddress, Integer port) |
|  | Connect to server manually. |
|  |  |
| Boolean | registerMThingInfo(MThingBaseType mThing) |
|  | Register received MThing information into the mThingList. |
|  |  |
| Boolean | unregisterMThingInfo(MThingBaseType mThing) |
|  | Unregister a designated MThing from the list. |
|  |  |
| Integer | broadcastMThingInfo() |
|  | Broadcast my MThing information. |
|  |  |
| Integer | sendbackMThingInfo(MThingBaseType targetMThing) |
|  | Send back my MThing information to specific MThing device. |
|  |  |
| Integer | sendCollaborationRequestMessage(MThingBaseType targetMThing) |
|  | Send a request message to a specific MThing to collaborate and to accomplish a designated task. |
|  |  |
| Integer | sendCollaborationRequestMessage(List<MThingBaseType> targetMThings) |
|  | Send a request message to specific MThings to collaborate and to accomplish a designated task. |
|  |  |
| Integer | sendACKMessage(MThingBaseType targetMThing, Boolean isBusy) |
|  | Send an acknowledge massage whether the MThing is available or not. |
|  |  |
| Integer | sendIoMTDataMessage() |
|  | Send data (e.g., compressed video, audio, feature metadata, url). |
|  |  |
| Integer | sendCommandMessage() |
|  | Send a command message to control (or actuate) MThings. |
|  |  |
| List<MThingBaseType> | taskAnalysis(List<MThingBaseType> mThingList) |
|  | Analyze the given task and select MThings capable of accomplishing the task. |
|  |  |
| Boolean | bindMThings() |
|  | Bind MThings, which perform the task. |
|  |  |
| Boolean | unbindMThings() |
|  | Unbind (release) MThings. |
|  |  |
| void | setupInformation() |
|  | Assign a task (or a mission) to an MThing. |
|  |  |

**Class SensorBaseType extends MThingBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| SensorBaseType() | |
| Default constructor. | |
|  | |
| SensorBaseType(String id) | |
|  | |
| SensorBaseType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| enum | unit |
|  | Sensor value unit. (e.g., cm, mm, km, sec, kg, etc.) |
|  |  |
| Float | maxValue |
|  | The maximum value. |
|  |  |
| Float | minValue |
|  | The minimum value. |
|  |  |
| Float | sensitivity |
|  | The sensitivity of the value. |
|  |  |
| Float | SNR |
|  | Signal to noise ratio. |
|  |  |
| BigInteger | timeStamp |
|  | The timestamp of a sensed data |
|  |  |
| Boolean | activate |
|  | Sensor activated. |
|  |  |
| Float | value |
|  | Sensor value. |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |

**Class CameraType extends SensorBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| CameraType() | |
| Default constructor. | |
|  | |
| CameraType(String id) | |
|  | |
| CameraType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Integer | rotateCamera(float x, float y, float z) |
|  | Move a camera direction. |
|  |  |
| Integer | changeCaptureMode() |
|  | Change capture modes (Video, Image, Burst, etc.). |
|  |  |
| void | changeZoom(float value) |
|  | Change the zoom. According to camera capabilities from MPEG-V Part 2 descriptions. |
|  |  |
| Integer | streamVideo() |
|  | API for streaming a video. |
|  |  |

**Class MediaAnalyzerBaseType extends MThingBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| MediaAnalyzerBaseType() | |
| Default constructor. | |
|  | |
| MediaAnalyzerBaseType(String id) | |
|  | |
| MediaAnalyzerBaseType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| URL | mediaSource |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| void | changeMediaSource() |
|  | Change the media source (i.e., camera, audio). |
|  |  |

**Class TrackerType extends MediaAnalyzerBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| TrackerType() | |
| Default constructor. | |
|  | |
| TrackerType(String id) | |
|  | |
| TrackerType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| void | extractObjectColor()  extractObjectTexture()  extractObjectShape() |
|  | Extract visual features from region of interest in the scene. |
|  |  |
| void | trackObject() |
|  | Trace an object in the scene. |
|  |  |
| Boolean | sendAlertMessage(MThingBaseType mThing) |
|  | Generate an alert message when an object of interest gets out of the camera’s visual angle. |
|  |  |

**Class StorageType extends MThingBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| StorageType() | |
| Default constructor. | |
|  | |
| StorageType(String id) | |
|  | |
| StorageType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| BigInteger | checkStorageSpace() |
|  | Check the space of the storage. |
|  |  |
| Integer | sendStorageSpaceFullAlertMessage() |
|  | Generate an alert message when the storage becomes full. |
|  |  |

**Class EventRootType**

The parent class, which contains event objects.

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| List<EventBaseType> | events |
|  | Children element any event. |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| static Marshaller | getMarshallerInstance() |
|  |  |
| static Unmarshaller | getUnmarshallerInstance() |
|  |  |
| String | marshal() |
|  | Convert classes to xml instances. |
|  |  |
| static EventRootType | unmarshal(String xmlInstance) |
|  | Convert xml instances to classes. |
|  |  |
| List<EventBaseType> | getEvents() |
|  | Get all event elements. |
|  |  |
| void | addEvent(EventBaseType event) |
|  | Add event element. |
|  |  |

**Class EventBaseType**

The highest class wrapping the data when communicating between MThings.

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
|  |  |

**Class BroadcastEventType extends EventBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| BroadcastEventType() | |
| Default constructor. | |
|  | |
| BroadcastEventType(MThingBaseType sender) | |
| A sender is generated when a BroadcastMThingInfoType is created. | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| MThingBaseType | sender |
|  | Broadcast sender. |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |

**Class SendbackEventType extends EventBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| SendbackEventType() | |
| Default constructor. | |
|  | |
| SendbackEventType(MThingBaseType sender, MThingBaseType receiver) | |
| A sender and a receiver are generated when an MThingInfoType is created. | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| MThingBaseType | sender |
|  | Request sender. |
|  |  |
| MThingBaseType | receiver |
|  | Request receiver. |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |

**Class MThingListenerProvider**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| List<OnMessageReceivedListener> | onMessageReceivedListeners |
|  |  |
| List<OnNewMThingDetectedListener> | onNewMThingDetectedListeners |
|  |  |
| List<OnMThingPresenceChangedListener> | onMThingPresenceChangedListeners |
|  |  |
| List<OnMThingAttributeChangedListener> | onMThingAttributeChangedListeners |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Boolean | addOnMessageReceivedListener(OnMessageReceivedListener listener) |
|  |  |
| Boolean | addOnNewMThingDetectedListener(OnNewMThingDetectedListenerlistener) |
|  |  |
| Boolean | addOnMThingPresenceChangedListener(OnMThingPresenceChangedListener listener) |
|  |  |
| Boolean | addOnMThingAttributeChangedListener(OnMThingAttributeChangedListener listener) |
|  |  |
| void | triggerOnRequestMessageReceivedListener() |
|  | Execute all listeners of OnMessageReceivedListener. |
|  |  |
| void | triggerOnACKMessageReceivedListener(MThingBaseType mThing) |
|  | Execute all listeners of OnMessageReceivedListener. |
|  |  |
| void | triggerOnIoMTDataMessageReceivedListener() |
|  | Execute all listeners of OnMessageReceivedListener. |
|  |  |
| void | triggerOnCommandMessageReceivedListener() |
|  | Execute all listeners of OnMessageReceivedListener. |
|  |  |
| void | triggerOnNewMThingDetectedListener(MThingBaseType newMThing) |
|  | Execute all listeners of OnNewMThingDetectedListener. |
|  |  |
| void | triggerOnMThingPresenceOnListener() |
|  | Execute all listeners of OnMThingPresenceChangedListener. |
|  |  |
| void | triggerOnMThingPresenceOffListener() |
|  | Execute all listeners of OnMThingPresenceChangedListener. |
|  |  |
| void | triggerOnMThingPresenceOnListener() |
|  | Execute all listeners of OnMThingPresenceChangedListener. |
|  |  |

**Class OnNewMThingDetectedListener**

|  |  |
| --- | --- |
| Methods | |
| Modifier and Type | Method and Description |
| void | OnNewMThingDetected(MThingBaseType newMThing) |
|  | A process when a new MThing is detected. ;If a MThing is newly included in  ;the system, do this and that … |

**Class OnMessageRecievedListener**

|  |  |
| --- | --- |
| Methods | |
| Modifier and Type | Method and Description |
| void | OnRequestMessageRecieved() |
|  | A process when a request message is received. |
|  |  |
| void | OnACKMessageRecieved(MThingBaseType mThing) |
|  | A process when an acknowledge message is received. |
|  |  |
| void | OnIoMTDataMessageRecieved() |
|  | A process when data are received. |
|  |  |
| void | OnCommandMessageRecieved() |
|  | A process when a command message is received. |

**Class OnMThingAttributeChangedListener**

|  |  |
| --- | --- |
| Methods | |
| Modifier and Type | Method and Description |
| void | OnMThingAttributeChanged() |
|  | A process when attributes of an MThing in the list are changed. |
|  |  |

**Class OnMThingPresenceChangedListener**

|  |  |
| --- | --- |
| Methods | |
| Modifier and Type | Method and Description |
| void | OnMThingPresenceOn() |
|  | A process when the MThing is visible. |
|  |  |
| void | OnMThingPresenceOff() |
|  | A process when the MThing is invisible (or becomes unavailable). |
|  |  |

# IoMT APIs and Descriptions

**Class IoMT**

This class is mainly composed of Listeners.

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Class and Description |
| static interface | IoMT.OnNewMThingDetectedListener |
|  | Activate a process when detecting a new Mthing. |
|  |  |
| static interface | IoMT.OnMessageRecievedListener |
|  | Activate a process when receiving a message. |
|  |  |
| static interface | IoMT.OnMThingAttributeChangedListener |
|  | Activate a process when attributes of an MThing are changed. |
|  |  |
| static interface | IoMT.OnMThingPresenceChangedListener |
|  | Activate a process when an MThing is visible or invisible. |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
|  |  |

**Class MThing**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| MThing(String ID, String ipAddress, int port) | |
| ID, IP address, and a port number is generated when an MThing is created. | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| String | ID |
| Bool | isBusy |
| List<MThing> | mThingList ;list of all the neighboring MThings |
| List<MThing> | bindMThingList ;list of all MThings, which perform a task |
| String | ipAddress |
| Int | port |
| String | Capability ; MThing capability instances created by  ; MPEG-V Part 2 standard. |
| Methods | |
| Modifier and Type | Method and Description |
| int | broadcastMThingInfo(MThing MThing) |
|  | Broadcast MThing information. |
|  |  |
| Int | registerMThingInfo(MThing mThing) |
|  | Register received MThing information into the mThingList. |
|  |  |
| Int | unregisterMThingInfo(MThing mThing) |
|  | Unregister a designated MThing from the list. |
|  |  |
| int | sendCollaborationRequestMessage(MThing targetMThing) |
|  | Send a request message to a specific MThing to collaborate and to accomplish a designated task. |
|  |  |
| int | sendCollaborationRequestMessage(List<MThing> targetMThings) |
|  | Send a request message to specific MThings to collaborate and to accomplish a designated task. |
|  |  |
| int | sendACKMessage(MThing targetMThing, bool isBusy) |
|  | Send an acknowledge massage whether the MThing is available or not. |
|  |  |
| int | sendIoMTDataMessage() |
|  | Send data (e.g., compressed video, audio, feature metadata, url). |
|  |  |
| int | sendCommandMessage() |
|  | Send a command message to control (or actuate) MThings. |
|  |  |
| Bool | isBusy() |
|  | Available (0) or unavailable (1). |
|  |  |
| List<MThing> | taskAnalysis(List<MThing> mThingList) |
|  | Analyze the given task and select MThings capable of accomplishing the task. |
|  |  |
| Int | bindMThings() |
|  | Bind MThings which perform the task. |
|  |  |
| Int | unBindMThings() |
|  | Unbind (release) MThings. |
|  |  |
| String | getID() |
|  |  |
| String | getCommunicationInfo() |
|  |  |
| Int | getPort() |
|  |  |
| void | setUpInformation() ;need to investigate how to specify  ;setup information (Standard?) |
|  | Assign a task (or a mission) to an MThing. |
|  |  |

**Class OtherPlatform**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
|  |  |

**Class Camera, which extends MThing**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| Camera() | |
| Fields | |
| Modifier and Type | Field and Description |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Int | rotateCamera(float x, float y, float z) |
|  | Move a camera direction. |
|  |  |
| Int | changeCaptureMode() |
|  | Change capture modes (Video, Image, Burst, etc.) |
|  |  |
| void | changeZoom(float value) ; According to camera capabilities from  ; MPEG-V Part 2 descriptions |
|  | Change the zoom |
|  | … (and more) |
|  |  |
| Int | streamVideo() |
|  | API for streaming a video |

**Class Tracker, which extends MThing**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| Tracker() | |
| Fields | |
| Modifier and Type | Field and Description |
| URL | videoSource |
| Methods | |
| Modifier and Type | Method and Description |
| void | extractObjectColor ()  extractObjectTexture()  extractObjectShape() |
|  | Extract visual features from region of interest in the scene. |
|  |  |
| void | trackObject() |
|  | Trace an object in the scene. |
|  |  |
| void | changeVideoSource() |
|  | Change the video source (i.e., camera). |
|  |  |
| void | sendAlertMessage(MThing mThing) |
|  | Generate an alert message when an object of interest gets out of the camera’s visual angle. |
|  |  |

**Class Storage, which extends MThing**

|  |  |
| --- | --- |
|  | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| Storage() | |
| Fields | |
| Modifier and Type | Field and Description |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Bool | checkStorageSpace() |
|  | Check the space of the storage. |
|  |  |
| Void | sendStorageSpaceFullAlertMessage() |
|  | Generate an alert message when the storage becomes full |
|  |  |

**Class OnNewMThingDetectedListener**

|  |  |
| --- | --- |
| Methods | |
| Modifier and Type | Method and Description |
| void | OnNewMThingDetected(MThing newMThing) |
|  | A process when a new MThing is detected.  ;If a MThing is newly included in  ;the system, do this and that … |

**Class OnMessageRecievedListener**

|  |  |
| --- | --- |
| Methods | |
| Modifier and Type | Method and Description |
| void | OnRequestMessageRecieved() |
|  | A process when a request message is received. |
|  |  |
| void | OnACKMessageRecieved(MThing mThing) |
|  | A process when an acknowledge message is received. |
|  |  |
| void | OnIoMTDataMessageRecieved() |
|  | A process when data are received. |
|  |  |
| Void | OnCommandMessageRecieved() |
|  | A process when a command message is received. |

**Class OnMThingAttributeChangedListener**

|  |  |
| --- | --- |
| Methods | |
| Modifier and Type | Method and Description |
| void | OnMThingAttributeChanged() |
|  | A process when attributes of an MThing in the list are changed. |
|  |  |

**Class OnMThingPresenceChangedListener**

|  |  |
| --- | --- |
| Methods | |
| Modifier and Type | Method and Description |
| void | OnMThingPresenceOn() |
|  | A process when the MThing is visible. |
|  |  |
| void | OnMThingPresenceOff() |
|  | A process when the MThing is invisible (or becomes unavailable). |
|  |  |

# API Instances for Wearable Applications for IoMTW

In contribution M38527 [7] of the 115rd Geneva meeting, instance od APIs for gesture-based wearable applications in IoMTW were presented. In this document, we present some APIs for gesture-based wearable applications. Figure 2 shows the . Conceptual model of gesture-based smart glasses applications in IoMTW.

## Gesture detection API

Class GestureDetection which extends MThing, is specified below.

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| GestureDetection(int StereoImageSequence[ ]) | |
| Fields | |
| Modifier and Type | Field and Description |
| Iplimage | StereoImageSequence  ;input stereo image sequence acquisitioned by stereo camera equipped in a wearable device (e.g., smart glasses) |
|  |  |
| File | Description  ;output gesture description metadata file |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| void | ExtractDepthImage( ) |
|  | Generate the depth Image for hand detection from the incoming stereo image sequence |
|  |  |
| void | ExtractObjectColor( ) |
|  | Extract color from the detected objects form the incoming stereo image sequence |
|  |  |
| void | HandShapeDetection( ) |
|  | Extract hand shape by processing the color image and depth image |
|  |  |
| void | EncoderBezierCurve( ) |
|  | Generate the Bezier curve from the extracted hand contour or hand trajectory |
|  |  |
| void | HandContourDescription( ) |
|  | Generate metadata describing the detected hand contour based on the predefined schema in an interoperable way  The generated description is stored and/or delivered to the processing module of gesture recognition |
|  |  |
| void | ListHandCenterPoint( ) |
|  | Keep the hand centroid points at table or buffer |
|  |  |
| void | HandTrajectoryDescription( ) |
|  | Generate metadata describing the detected hand trajectory over time based on the predefined schema in an interoperable way  The generated description is stored and/or delivered to the processing module of gesture recognition |
|  |  |
| FILE | GestureDescription |
|  |  |

## Gesture recognition API

Class GestureRecognition which extends MThing, is specified below.

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| GestureRecognition(FILE \*GestureDescription, FILE \*PredefinedGestureCommand) | |
| ;The GestureDescription include the HandContourDescription and HandTrajectoryDescription. | |
| Fields | |
| Modifier and Type | Field and Description |
| FILE | GestureDescription  ;input the generated gesture description metadata that describes general hand gesture including hand contour and/or hand trajectory |
|  |  |
| FILE | PredefinedGestureCommand  ;input a set of predefined gesture commands that specifies the mapping relationship between a specific hand gesture and the associated gesture-based command |
|  |  |
| unsigned int | Command  ;output a command recognized from the gesture generated from a user, which is used for controlling a wearable device or an available application |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| FILE | ParingGestureDescription( ) |
|  | Parse gesture description and extract hand contour and/or trajectory information from the inputted gesture description (e.g., the raw data of Bezier curve that fits a hand contour and/or hand trajectory) |
|  |  |
| void | DecoderBezierCurve( ) |
|  | Generate the hand contour or trajectory from the raw data of Bezier curve |
|  |  |
| void | GestureRecognition( ) |
|  | Identify the type of the given gesture by mapping the inputted gesture information and the predefined gesture |
|  |  |
| unsigned int | GestureCommandGeneration( ) |
|  | Generate a specific command identified by the inputted gesture based on the result of gesture recognition |
|  |  |
| unsigned int | Command() |
|  | Output a command recognized from the gesture generated from a user, which is used for controlling a wearable device or an available application |

## Self-adaptive multimedia API

This shows an example of parameter information to support self-adaptive application in Wearable MPEG environment.

### User Interaction (User to Wearable)

User can initially set up the parameters of wearable according to user’s requirement. The parameter’s value are application’s type, performance expectation value, monitoring setup value and logging on/off information, etc. And below is one example of parameters for user interaction.

<PROGRAM>

<SA\_TYPE> 1 </SA\_TYPE> // Integer Type: 0: Internal, 1: External for Self-adaptive Application

<SA\_TARGET\_QOS> 1.00 </SA\_TARGET\_QOS> // Double Type: Program Heartbeat (refer to SA\_MAX\_QOS and SA\_MIN\_QOS)

<SA\_MAX\_QOS> 2.00 </SA\_MAX\_QOS> // Double Type: Heartbeat Max,Min value

<SA\_MIN\_QOS> 0.00 </SA\_MIN\_QOS>

</PROGRAM>

<MONITORING>

<SA\_WINDOW\_SIZE> 10 </SA\_WINDOW\_SIZE> // Long Type: Heartbeat Window Size

<SA\_BUFFER\_DEPTH> 1 </SA\_BUFFER\_DEPTH> // Long Type: Heartbeat Buffer Depth

<SA\_SAMPLING\_TIME> 100 </SA\_SAMPLING\_TIME> // Long Type: Heartbeat Timestamp

<SA\_HB\_NUMBER> 1 </SA\_HB\_NUMBER> // Long Type: Heartbeat

<SA\_LOGNAME> log.txt </SA\_LOGNAME> // String Type: Log Name (Default = NULL)

</MONITORING>

### Wearable Output (Wearable to Processing Unit)

This is an interface to access application for monitoring, which is necessary static/dynamic information for application PID, application type, QoS, performance’s average/maximum/total value. Also, there is system information for wearable device such as CPU type and process status.

* Self-adaptive Application Type

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Description** |
| Self-adaptive application type | 0x01 | Internal mode (TYPE\_SA\_APP\_INTERNAL) |
| 0x02 | External mode  (TYPE\_SA\_APP\_EXTERNAL) |

* Application Status Information

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| Application Status Information | int | Application Identifier (id) |
| double | Average in total (heartbeat\_global) |
| double | Average in window (heartbeat \_average ) |
| double | Heartbeat value (heartbeat \_instant ) |

* Application Performance Requirement Condition

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | | **Description** |
| Application Performance Requirement | | int | Application Identifier (id) |
| int | INTERNAL/EXTERNAL (type) |
| double | Maximum quality (max\_qos) |
| double | Steady-state quality (target\_qos) |
| double | Minimum quality (min\_qos) |

* Heartbeat Information Type

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Description** |
| Heartbeat Information Type | 0x11 | Average in total  (TYPE\_HB\_GLOBAL) |
| 0x12 | Average in window  (TYPE\_HB\_AVERAGE) |
| 0x13 | Heartbeat value  (TYPE\_HB\_INSTANT) |

* CPU Type and Operation Mode

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Description** |
| CPU Type | **0x01** | **CPU Master (TYPE\_CLUSTER\_MASTER)** |
| **0x02** | **CPU Slave**  **(TYPE\_CLUSTER\_SLAVE)** |
| CPU Operation Mode | **0x00** | **CPU Running OFF**  **(TYPE\_OPERATION\_DISABLED)** |
| **0x01** | **CPU Running ON**  **(TYPE\_OPERATION\_ENABLED)** |

* Process Status Information

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Description** |
| Process Status Information | **0x00** | **TYPE\_PS\_NONE** |
| **0x01** | **Running State (TYPE\_PS\_RUNNING)** |
| **0x02** | **TYPE\_PS\_INTERRUPT\_SLEEP** |
| **0x03** | **TYPE\_PS\_DISK\_SLEEP** |
| **0x04** | **TYPE\_PS\_TRACED** |
| **0x05** | **TYPE\_PS\_PAGING** |
| **0x06** | **TYPE\_PS\_ZOMBIE** |

### Wearable Input (Processing Unit to Wearable )

This is an interface to control system resource in wearable , e.g. CPU, GPU, Power, etc.

* CPU Control Type

|  |  |  |
| --- | --- | --- |
| **Name** | **Value** | **Description** |
| CPU Control Type | **0x01** | **CPU Master**  **(TYPE\_CLUSTER\_MASTER)** |
| **0x02** | **CPU Slave**  **(TYPE\_CLUSTER\_SLAVE)** |

### Wearable Acting (Wearable to User)

This is an interface to support general audio/video streaming data to user.

## Blind Person

## Blind Person 1

**Class SensorBaseType extends MThingBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| SensorBaseType() | |
| Default constructor. | |
|  | |
| SensorBaseType(String id) | |
|  | |
| SensorBaseType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| enum | unit |
|  | Sensor value unit. (e.g., cm, mm, km, sec, kg, etc.) |
|  |  |
| Float | maxValue |
|  | The maximum value. |
|  |  |
| Float | minValue |
|  | The minimum value. |
|  |  |
| Float | sensitivity |
|  | The sensitivity of the value. |
|  |  |
| Float | SNR |
|  | Signal to noise ratio. |
|  |  |
| BigInteger | timeStamp |
|  | The timestamp of a sensed data |
|  |  |
| Boolean | activate |
|  | Sensor activated. |
|  |  |
| Float | value |
|  | Sensor value. |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |

**Class DistanceSensorType extends SensorBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| DistanceSensorType() | |
| Default constructor. | |
|  | |
| DistanceSensorType(String id) | |
|  | |
| DistanceSensorType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Float | getDistance() |
|  | Convert sensor values (e.g., Tick values for distance) to distance values with metric. |
|  |  |

**Class UltrasonicSensorType extends DistanceSensorType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| UltrasonicSensorType() | |
| Default constructor. | |
|  | |
| UltrasonicSensorType(String id) | |
|  | |
| UltrasonicSensorType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
|  |  |

**ActuatorBaseType extends MThingBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| ActuatorBaseType() | |
| Default constructor. | |
|  | |
| ActuatorBaseType(String id, Float intensity) | |
|  | |
| ActuatorBaseType(String id, Float intensity, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| Float | intensity |
|  | Actuation intensity. |
|  |  |
| Float | maxIntensity |
|  | Controlled by users or other MThings to configure the actuators. (<= MAX\_INTENSITY) |
|  |  |
| Float | minIntensity |
|  | Controlled by users or other MThings to configure the actuators. (>= MIN\_INTENSITY) |
|  |  |
| BigInteger | timeStamp |
|  | The timestamp of an activation. |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |

**Class SpeakerType extends ActuatorBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| SpeakerType() | |
| Default constructor. | |
|  | |
| SpeakerType(String id, Float volume) | |
|  | |
| SpeakerType(String id, Float volume, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| Float | volume |
|  | Speaker volume. |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |

**Class HeadphoneType extends SpeakerType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| HeadphoneType() | |
| Default constructor. | |
|  | |
| HeadphoneType(String id) | |
|  | |
| HeadphoneType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| URL | path |
|  | A play file path. |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| void | playFromURL(URL path) |
|  |  |

**Class VibratorType extends** **ActuatorBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| VibratorType() | |
| Default constructor. | |
|  | |
| VibratorType(String id, Float intensity) | |
|  | |
| VibratorType(String id, Float intensity, Integer count, Long[] pattern) | |
|  | |
| VibratorType(String id, Float intensity, String ipAddress, Integer port) | |
|  | |
| VibratorType(String id, Float intensity, Integer count, Long[] pattern, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| Integer | count |
|  | Execution count of vibrator pattern. |
|  |  |
| Long[] | pattern |
|  | Vibration pattern. |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |
| void | vibration() |
|  | Execute vibration. |
|  |  |
| void | vibration(Integer count, Long[] pattern) |
|  | Execute vibration manually. |
|  |  |
| void | stopVibraton() |
|  |  |

**Class TextToSpeechType extends MediaAnalyzerBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| TextToSpeechType() | |
| Default constructor. | |
|  | |
| TextToSpeechType(String id) | |
|  | |
| TextToSpeechType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| String | text |
|  | The text to speech. |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| URL | getSpeechSource(String text) |
|  | Get a path of a speech source file. |
|  |  |
| void | speech() |
|  | Execute a text-to-speech operation. |
|  |  |

**Class CollisionCoordinatorType extends MediaAnalyzerBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| CollisionCoordinatorType() | |
| Default constructor. | |
|  | |
| CollisionCoordinatorType (String id, Float volume) | |
|  | |
| CollisionCoordinatorType (String id, Float volume, String ipAddress, Integer port) | |
|  | |
| Modifier and Type | Field and Description |
| Void | detectCollision() |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |

## Blind Person 2

**Class RFIDReaderType extends SensorBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| RFIDSensorType() | |
| Default constructor. | |
|  | |
| RFIDReaderType(String id) | |
|  | |
| RFIDReaderType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| String | tagInformation |
|  | RFID tag. |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |

**Class OrientationSensorType extends** **SensorBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| OrientationSensorType() | |
| Default constructor. | |
|  | |
| OrientationSensorType(String id) | |
|  | |
| OrientationSensorType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| Float | yawMin |
|  |  |
| Float | yawMax |
|  |  |
| Float | pitchMin |
|  |  |
| Float | pitchMax |
|  |  |
| Float | rollMin |
|  |  |
| Float | rollMax |
|  |  |
| Float | yaw |
|  |  |
| Float | pitch |
|  |  |
| Float | roll |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |

**Class CompassType extends OrientationSensorType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| CompassType() | |
| Default constructor. | |
|  | |
| CompassType(String id) | |
|  | |
| CompassType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| enum | direction |
|  | (e.g., N, NE, E, SE, S, SW, W, NW) |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |

**Class PositionSensorType extends SensorBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| PositionSensorType() | |
| Default constructor. | |
|  | |
| PositionSensorType(String id, Float x, Float y, Float z) | |
|  | |
| PositionSensorType(String id, Float x, Float y, Float z, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| Float | x |
|  |  |
| Float | y |
|  |  |
| Float | z |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |

**Class GPSType extends PositionSensorType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| GPSType() | |
| Default constructor. | |
|  | |
| GPSType(String id, Float longitude, Float latitude, Float altitude) | |
|  | |
| GPSType(String id, Float longitude, Float latitude, Float altitude, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| Float | longitude |
|  |  |
| Float | latitude |
|  |  |
| Float | altitude |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |
| Boolean | detect() |
|  | Detect GPS location. |
|  |  |
| Boolean | detect(String provType) |
|  | Detect GPS location using a specific location provider (e.g., GPS, WiFi, LTE). |
|  |  |

**Class** **DirectionGuiderType extends MediaAnalyzerBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| DirectionGuideType() | |
| Default constructor. | |
|  | |
| DirectionGuiderType(String id) | |
|  | |
| DirectionGuiderType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| Float[2] | sourceLocation |
|  | e.g., Person’s current position. (0: longitude, 1: latitude) |
|  |  |
| Float[2] | destinationLocation |
|  | e.g., Next RFID tag location. (0: longitude, 1: latitude) |
|  |  |
| enum | destinationDirection |
|  | (N, NE, E, SE, S, SW, W, NW) |
|  |  |
| Float | currentAzimuth |
|  |  |
| Enum | rotation |
|  | (LEFT, RIGHT) |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Type | Get() |
|  |  |
| void | Set(Type) |
|  |  |
| Float | calculateCurrentDirection() |
|  |  |
| Float | calculateRotationalAngle() |
|  |  |

## Blind Person 3

**Class VisualFeatureExtractorType extends MediaAnalyzerBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| VisualFeatureExtractorType() | |
| Default constructor. | |
|  | |
| VisualFeatureExtractorType(String id) | |
|  | |
| VisualFeatureExtractorType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| URL | videoSource |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| void | extractObjectColor() |
|  |  |
| void | extractObjectTexture() |
|  |  |
| void | extractObjectShape() |
|  |  |
| void | extractCDVS() |
|  |  |
| void | extractSURF() |
|  |  |
| void | extractSIFT() |
|  |  |

**Class LandmarkFinderType extends MediaAnalyzerBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| LandmarkFinderType () | |
| Default constructor. | |
|  | |
| LandmarkFinderType (String id) | |
|  | |
| LandmarkFinderType (String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
| URL | imageSource |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| String | findLandmark() |
|  | Return landmark name |

**Class CameraType extends SensorBaseType**

|  |  |
| --- | --- |
| Nested Classes | |
| Modifier and Type | Method and Description |
|  |  |
| Constructor | |
| Constructor and Description | |
| CameraType() | |
| Default constructor. | |
|  | |
| CameraType(String id) | |
|  | |
| CameraType(String id, String ipAddress, Integer port) | |
|  | |
| Fields | |
| Modifier and Type | Field and Description |
|  |  |
| Methods | |
| Modifier and Type | Method and Description |
| Integer | rotateCamera(float x, float y, float z) |
|  | Move a camera direction. |
|  |  |
| Integer | changeCaptureMode() |
|  | Change capture modes (Video, Image, Burst, etc.). |
|  |  |
| void | changeZoom(float value) |
|  | Change the zoom. According to camera capabilities from MPEG-V Part 2 descriptions. |
|  |  |
| Integer | streamVideo() |
|  | API for streaming a video. |

## References

1. M. Mitrea, S.-K. Kim, S. Chun, “Conceptual model, architecture and use cases for Media-centric IoTs and Wearable” ISO/IEC JTC1/SC29/WG11 N15728, Geneva, Swiss, Oct. 2015.
2. A. Yang, J. G. Kim, S. Chun, and H. Ko, “Root Elements for Media-centric IoT and Wearable (IoMTW)” ISO/IEC JTC1/SC29/WG11 m37763, San Diego, USA, Feb. 2015.
3. S. Chun, H. Ko, A. Yang, and J. G. Kim, “ Common and Gesture-Based Wearable Description for IoMTW ” ISO/IEC JTC1/SC29/WG11 m37764, San Diego, USA, Feb. 2015.
4. D. Kim, B. Koo, “ Image Analysis description for Media-centric IoT and Wearable (IoMTW) ” ISO/IEC JTC1/SC29/WG11 m 37911, San Diego, USA, Feb. 2015.
5. M. Choi, H. Kim, “Speech-Related Wearable Element Description for IoMTW,” ISO/IEC JTC1/SC29/WG11 m38113, San Diego, USA, Feb. 2015.
6. S.-K. Kim, ISO/IEC JTC1/SC29/WG11 m37108, “Progress on camera system,” 113rd MPEG Meeting, Oct. 2015, Geneva, Switzerland.
7. A. Yang, J. G. Kim, S. Chun, and H. Ko , ISO/IEC JTC1/SC29/WG11 m38526, “ Hand Gesture Description for Wearable Applications in IoMTW, ” 115rd MPEG Meeting, Jun. 2016, Geneva, Switzerland
8. S. Chun, H. Ko, A. Yang, and J. G. Kim, ISO/IEC JTC1/SC29/WG11 m38527, “API Instances for Gesture-Based Wearable Applications for IoMTW, ” 115rd MPEG Meeting, Jun. 2016, Geneva, Switzerland.
9. ISO/IEC JTC1/SC29/WG11 m37975, “MIoT API instances” 114th MPEG Meeting, Feb. 2016, San Diego, USA.
10. ISO/IEC JTC1/SC29/WG11 m38790, “MIoT use case and APIs assisting a blind person to avoid obstacles,” 115th MPEG Meeting, May. 2016, Geneva, Switzerland.
11. ISO/IEC JTC1/SC29/WG11 w16345, “Use cases for Internet of Media-Things and Wearables,” 115th MPEG Meeting, May. 2016, Geneva, Switzerland.

**Appendix A**

The sensors of interests for the Wearable are listed here including a notice if they are already specified or missed considering the use cases submitted at MPEG 111th.

From ISO/IEC 23005-5 latest edition, these are the sensors of interest for Wearable as per a preliminary check. An in depth check has to be performed during the next Ad Hoc Period.

1. Light sensor
2. Ambient noise sensor
3. Temperature sensor
4. Humidity sensor
5. Distance sensor
6. Atmospheric pressure sensor
7. Position sensor
8. Velocity sensor
9. Acceleration sensor
10. Orientation sensor
11. Angular velocity sensor
12. Angular acceleration sensor
13. Force sensor
14. Torque sensor
15. Pressure sensor
16. Motion sensor
17. Intelligent camera
18. Multi Interaction point sensor
19. Gaze tracking sensor
20. Wind sensor
21. Global position sensor
22. Altitude sensor type
23. Bend sensor type
24. Gas sensor type
25. Dust sensor type
26. Body height sensor
27. Body weight sensor
28. Body temperature sensor
29. Body fat sensor
30. Blood type sensor
31. Blood pressure sensor type
32. Blood sugar sensor
33. Blood oxygen sensor
34. Heart rate sensor
35. Electrograph sensor
36. EEG sensor type
37. ECG sensor type
38. EMG sensor
39. EOG sensor
40. GSR sensor
41. Bio sensor
42. Weather sensor
43. Facial expression sensor
44. Facial morphology sensor
45. Facial expression characteristics
46. Geomagnetic sensor
47. Proximity sensor
48. Camera Sensor
49. Color Camera Sensor
50. Depth Camera Sensor
51. Stereo Camera Sensor
52. Infrared Camera Sensor
53. Thermographic camera sensor

**Appendix B**

The actions from the Wearable to the User and the data input to the Wearable from the Processing Unit) correspond to technologies already existing in MPEG listed here.

1. ISO Base Media File Format
2. Video : MPEG4 Simple profile based stereoscopic codec
3. Video: MPEG4 AVC based stereoscopic codec
4. HEVC
5. Still Image : JPEG based stereoscopic still image
6. Audio: MPEG-1 Layer-2, MPEG-2 AAC+, MPEG-4 ER-BSAC MPEG-4 HE-AAC
7. System: IOD/OD, BIFS, MPEG-4 File format, AVC File format, MMT.
8. User metadata: MPEG-7, TV-Anytime, etc
9. Stereoscopic VAF
10. MPEG-UD

**Appendix C**

**C. Reference IoT architectures promoted by organizations outside MPEG**

This section illustrates reference and/or functional models related to IoT or wearable devices emerged from the efforts carried out outside MPEG.

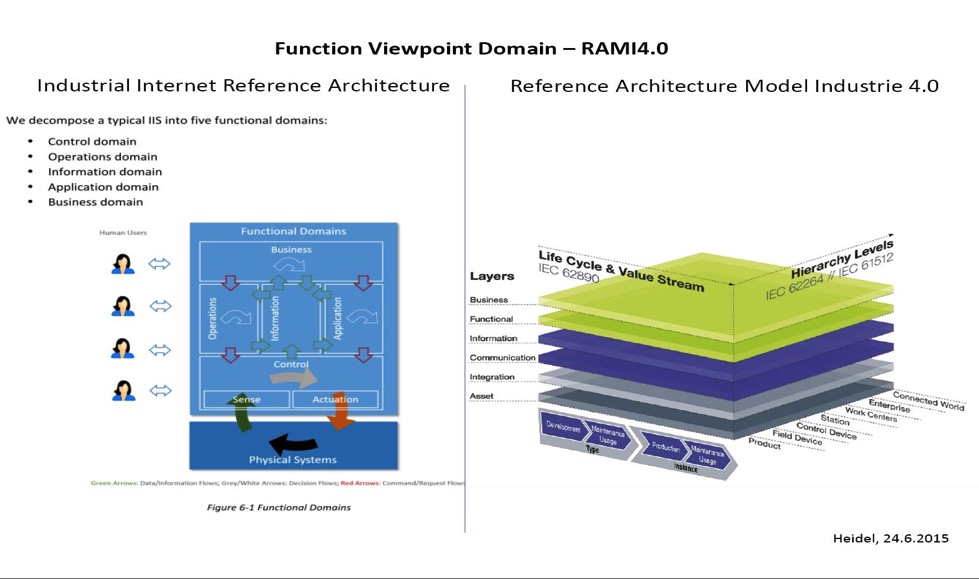


Fig. 2 Internet Industrial Consortium (IIC) view point on the reference architecture for IOT

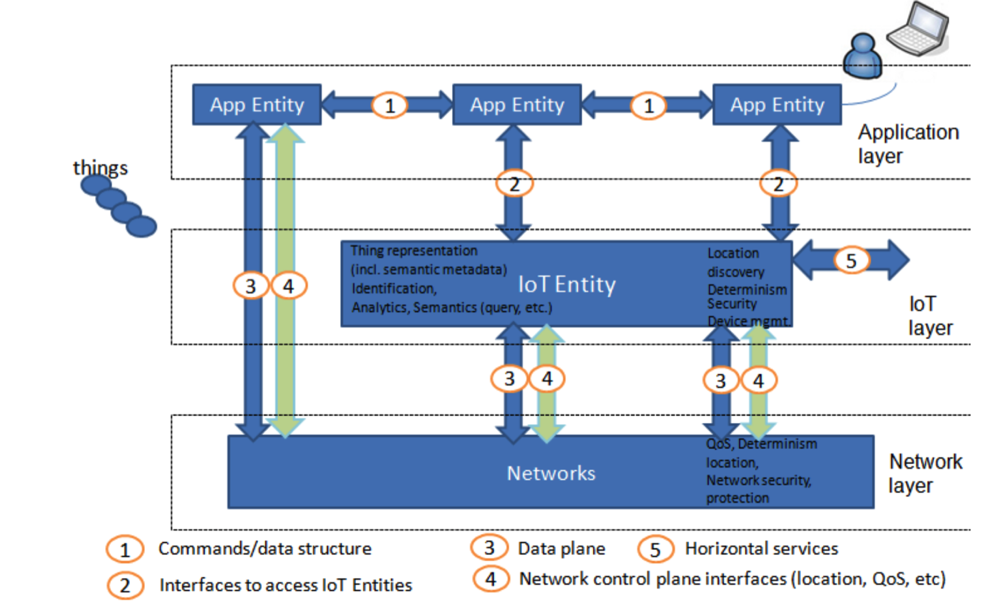


Fig. 3 Alliance for Internet of Things Innovation (AIOTI) on Functional model, a three layer approach.

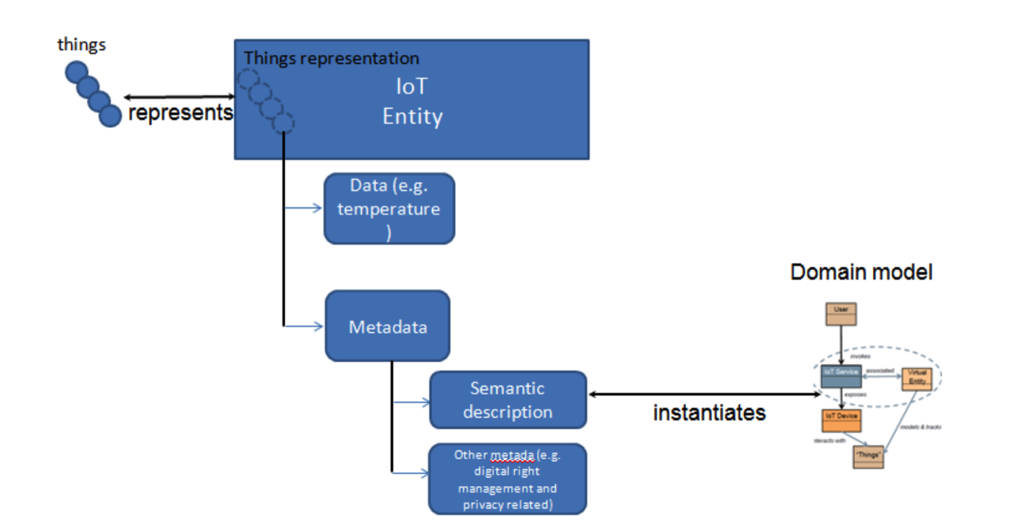


Fig. 4 Alliance for Internet of Things Innovation (AIOTI) : Relationship to semantics metadata

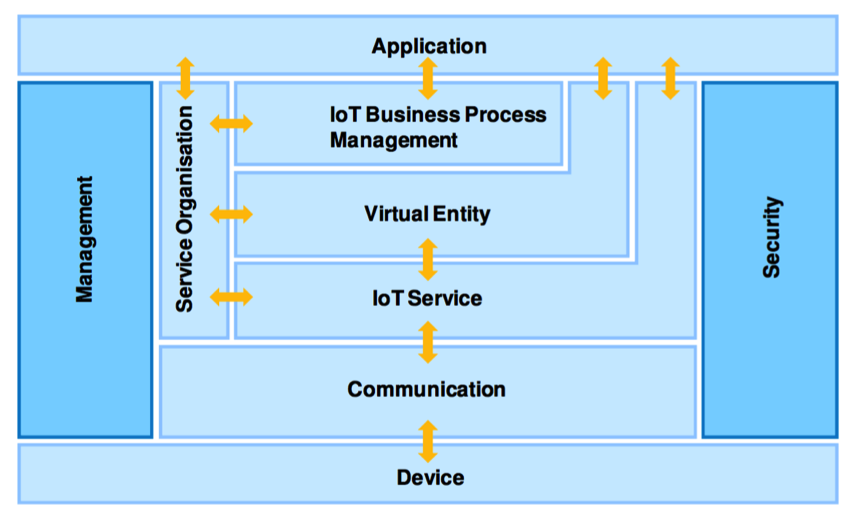


Fig. 5 Internet of Things Architecture (IoTA) FP7 project: Functional Model

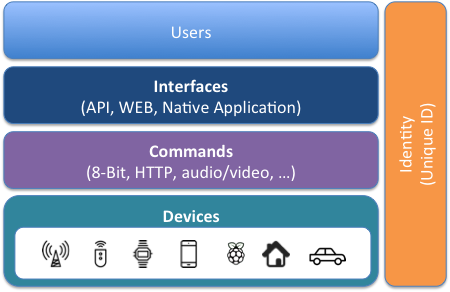


Fig. 6: WSO2 reference architecture for the IoT

The table below presents an analysis on the possible alignment among present day architectures outside MPEG and the conceptual IOMTW model.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **IOMTW** | **Interface 1** | **Interface 2** | **Interface 2’** | **Interface 3** |
| data provided by the system designer (set-up, commands, services reference, …) | raw or high level (descriptions) AV data within Mthing | a wrapped and transmission friendly version of Interface 2 | IoMT capabilities, discovery, configuration data |
| **IIC** | = as IOMTW | = Information | = Operation | = Application |
| **AIOTI** | = Interface 2 : Interface to access IoT Entities | NOT presented | = Interfaces 3 & 4 : Data plane & network control plane | = Application Layer |
| **IOTA** | = Communication : between Device & IoT Service | =Virtual entity of the IoT | = IoT Business process management | = Application |
| **WS02** | = Commands : considering unique ID of the device | = Unique ID of the IoT | = Interfaces (API/Application) | = Interfaces (API/Application) |

**Appendix D**

**D. Interoperability between Internet of Media-Related Things and Wearables and oneM2m and OIC**

MThing is defined as a Thing with at least one of audio/vidual sensing and actuating capabilties. It means that MThing shall be interoperable with existing Things. To provide an valuable IoMT service to the user, MThing or MThings are not enough. There must be a way to interact MThings to existing IoT Things. To do so, it is required to make MThings and IoT compatible. Though MThing is defined as a kind of Thing with at least one of audio/vidual sensing and actuating capabilities, it is not yet clear how to interact Thing and MThing.

To this purpose, there shall be a guideline to expose MThing as a Thing. A single IoMTW device or service can be implemeted diffrently with respect to the compatible IoT protocols. For example, in OneM2M, IoMTW can be exposed as an Application Entity which is identified by an App-ID. Other OneM2M things can find MThing device by querying Registrar Service with specific App-ID. In OIC, IoMTW can be expressed as a device or resource. It shall have name with rt:x.<ICANNName).<resource identification>. Other OIC devices will use that name for discovery. Moreover, even on a same base protocol, implementation of IoMTW can be different – some company implement it as an Application Entity while the other company implement it as a custon CSE - and it can make it hard to communicate between IoMTW device and IoT device.

In this section, guidelines to map MThing with existing IoT standards are described as well as brief information of related IoT standards.

* 1. **Interoperability with OneM2M**
     1. **Brief Introduction of OneM2M standards**

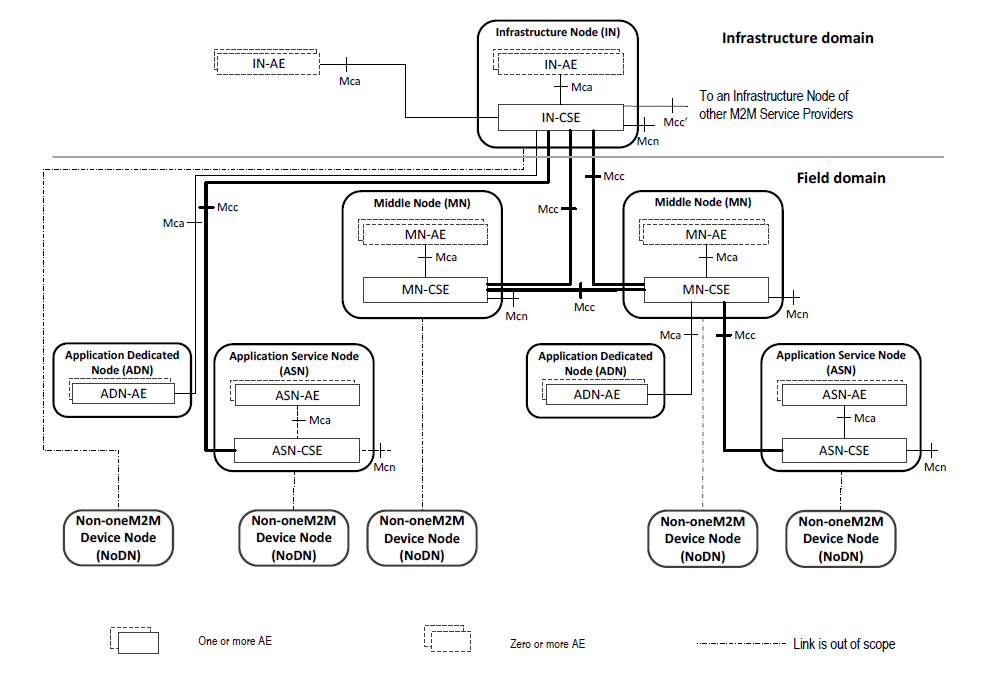
****

Figure 2 Exmaple Architecture of OneM2M

Figure 2 shows an example of OneM2M architecture. Presenting full description of OneM2M standards is not appropriate here, however, it is worth mention about their characteristics. In oneM2M, each instance is a node, and node consists of Application Entity, Common Service Entity, and/or Network Service Entity. By combination of entities in itself and its role, node is categorized one of these five: Application Service Node (ASN), Application Dedicated Node (ADN), Middle Node (MN), Infrastructure Node (IF), and Non-OneM2M Node (NoDN). Characteristics of OneM2M standards are worth mentioning and as follows:

* Server – Client Architecture
* 3 Entities for a Node (AE, CSE, NSE)
* 5 Types of Node (IN, MN, ASD, AND, NoDN)
* CSE provides OneM2M service to Application Entity or other CSE
  + 1. **Mapping of MThing into OneM2M standards**

To make an MThing compatible with OneM2M device, MThing shall be mapped to OneM2M by following rules:

* All MThings are defined as AE(Application Entity) for OneM2M.
* No additional CSE are defined for IoMTW.
* MThing is identified by OnM2M App-ID.
  1. **Interoperability with OIC**
     1. **Brief Introduction of OIC standards**

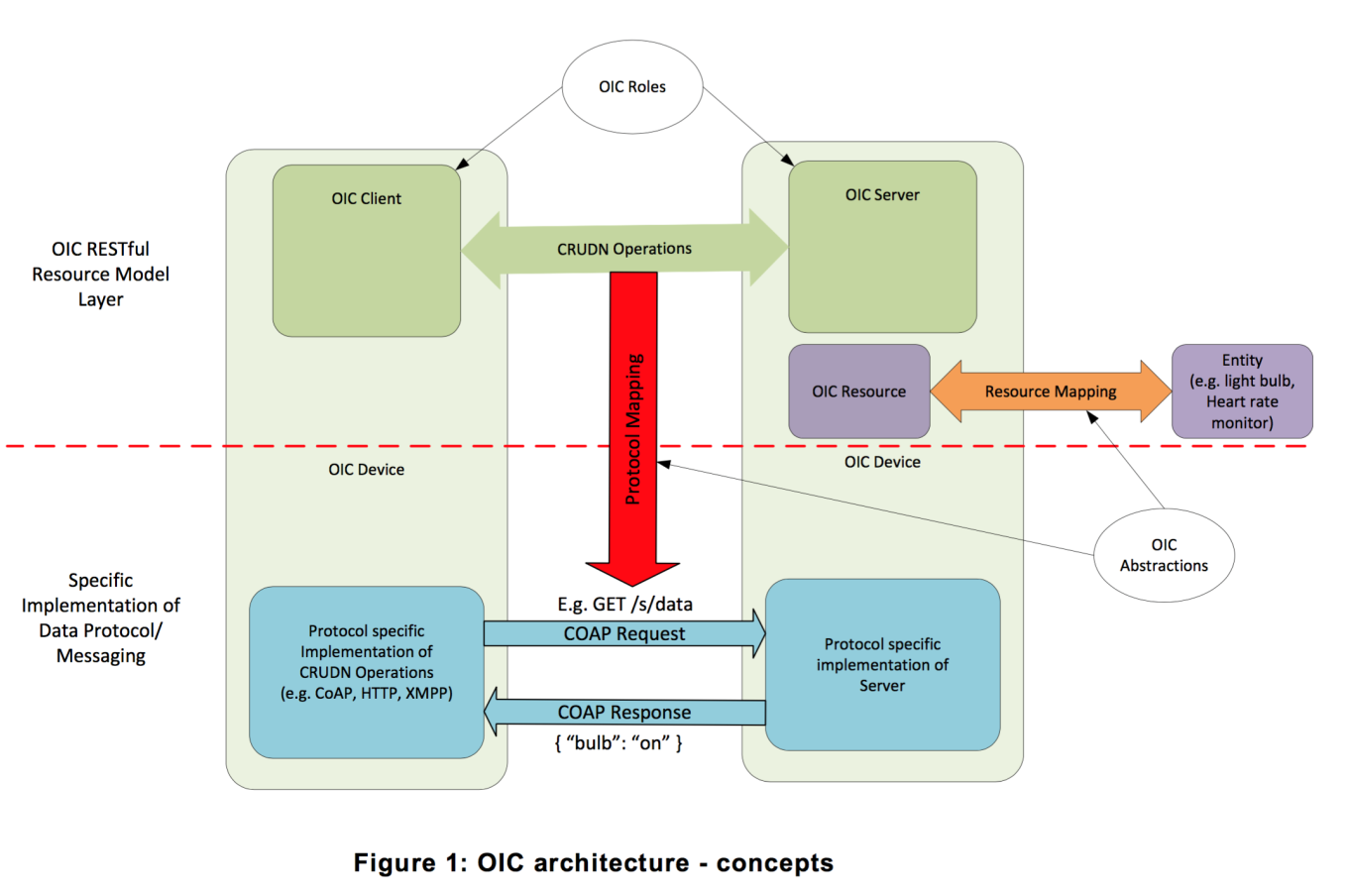
****

Figure 3 OIC Architecture – Concepts

Figure 3 shows conceptual diagram of OIC architecture. In OIC, a device or entity becomes an OIC Resource, and works as a OIC Server. Application is an OIC Client, and OIC Server and OIC Client talk to each other, like peer-to-peer service.

Characteristic of OIC is as follows:

* Server-Client Architecture
* Peer-to-Peer between Server and Client
* Device is a Resource
  + 1. **Mapping of MThing into OIC standards**

To make an MThing compatible with OIC device, MThing shall be mapped to OneM2M by following rules:

* All fMThings are represented as a OIC resource and have a unique resource name.
* Resource name starts with“x.mpeg-IoMTw.“, followed by resource name.

**Appendix E**

1. **Use case for Smart Glass**
   * 1. **Information**

Multimedia consumption applications which may include image/video acquisition/presentation, visual information rendering, visual monitoring, video call, video chatting, etc. would be one of the most important functions to be supported in smart glasses. Therefore, it is highly demanded to control multimedia consumption in an efficient way. Hand gesture is regarded as an attractive and promising user interface to allow such efficient control of multimedia consumption as well as wearable device manipulation.

In this document, we present use cases of multimedia consumption utilizing hand gestures as a main user input in smart glasses. In order to control multimedia consumption, hand gestures is recognized in real time from input information incoming from input devices, and then the recognized gesture information is converted into gesture commands. Such overall procedure of use cases scenarios and possible configuration of input devices providing information to be used for gesture recognition are briefly presented.

Monoscopic and stereoscopic content can be created from a mono and stereoscopic camera on smart glasses. This content may contain mono/stereoscopic still images and/or motion pictures. A user will be able to upload/download this content via a wireless or wired network. Existing networks allow real-time transmission of content. The stereoscopic content player on smart glasses will be able to play the transmitted real-time files or stored files as well as play monoscopic content. In addition, the functions of video call, voice call, and multimedia communication can be supported between smart glasses and smart phones. Such multimedia communications on smart glasses can be compatible with different OSs (Androidwear/Google, iOS/Apple, TizenOS/Samsung).

* + 1. **Use case of multimedia application adaptation Multimedia Consumption for smart glass**

Figure 21 shows an example of smart glasses with which Samsung wearable AP and OLED display are equipped. The underlying table shows the various functions of multimedia consumption available in the Insignal’s smart glasses. This smart glasses support the recoding/playing of HD resolution video. Gesture recognition and 3D conversion can be processed in the Insignal’s engine.

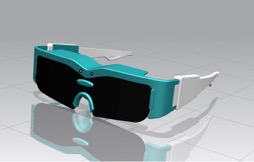
 

Figure 21 An example of smart glasses (Insignal)

*Multimedia consumption functions of Insignal’s smart glasses*

|  |  |  |
| --- | --- | --- |
| Multimedia function | Target Specifications | Codec/Engine |
| Video Recoding | 1280x720, 20~30fps, stereoscopic | MPEG4 AVC |
| Video Playing | 1024x768,20~30fps,stereoscopic | MPEG4 AVC |
| Voice recognition | Real-time(Wearable or Processing Unit) | AMR/Google |
| Gesture recognition | Real-time(Wearable or Processing Unit) | Insignal’s engine |
| Auto-convergence | When stereoscopic recode | Insignal’s engine |
| 2D/3D conversion | Mono to stereoscopic image conversion | Insignal’s engine |
| 3D format conversion | Side by side, Frame sequential,.. | Insignal’s engine |

* + 1. **Input information**

Figure 22 shows the configuration of input devices to be available for smart glasses, which includes mono camera, stereo camera, depth camera, and/or physical sensors (9-axis sensor). Although all of these input devices can be available, serval different combinations of these devices can also be provided. Stereo camera allows gesture recognition in real time by using the depth map information that can be generated from stereo views. In addition, 9-axis physical sensor and/or depth camera would be useful in the recognition of more accurate gestures. The underlying table summarizes the information that can be obtained by each input device and necessary information in gesture recognition. Information incoming from input devices is processed and converted into gesture commands through gesture recognition in a processing module placed in a wearable device or server. Then, gesture command is used to control the basic functions of wearable device or multimedia content consumption.

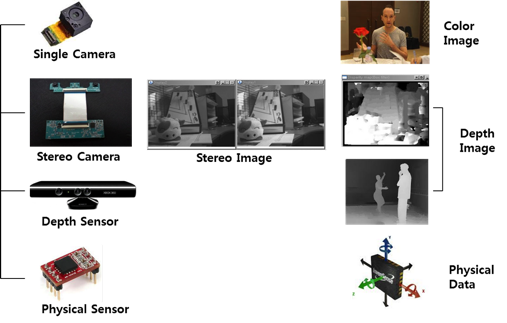


Figure 22. Input devices of smart glasses

*Input information of smart glasses*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input Devices** | **HW Component** | **Information** | **Analysis Information** | HW/SW |
| Image  Camera | Single Camera | Color Image | Gesture | SW |
| Stereo Camera | Stereo Color Image | Gesture | HW, SW |
| Depth Camera | TOF Sensor  Kinect Module | Depth Image | Gesture | HW |
| Physical  Sensor | Acceleration Sensor  Position Sensor | Physical Data | Moving, Speed  Subside Parameters | HW |
| Voice | Mic, Audio Codec | Voice Raw Data | Text (voice to text) | SW |
| Touch | Panel | Coordinate | Position / Point | HW, SW |
| Server |  |  |  | HW, SW |

* + 1. **Use case of Gesture command**

Gestures can originate from any bodily motion or state but commonly originate from the face or hand. Gesture recognition enables humans to communicate with the machine and interact naturally without any mechanical devices. For example, using the concept of gesture recognition, it is possible to point a finger at the computer screen so that the cursor will move accordingly. This could potentially make conventional input devices such as mouse, keyboards and even touch-screens redundant.

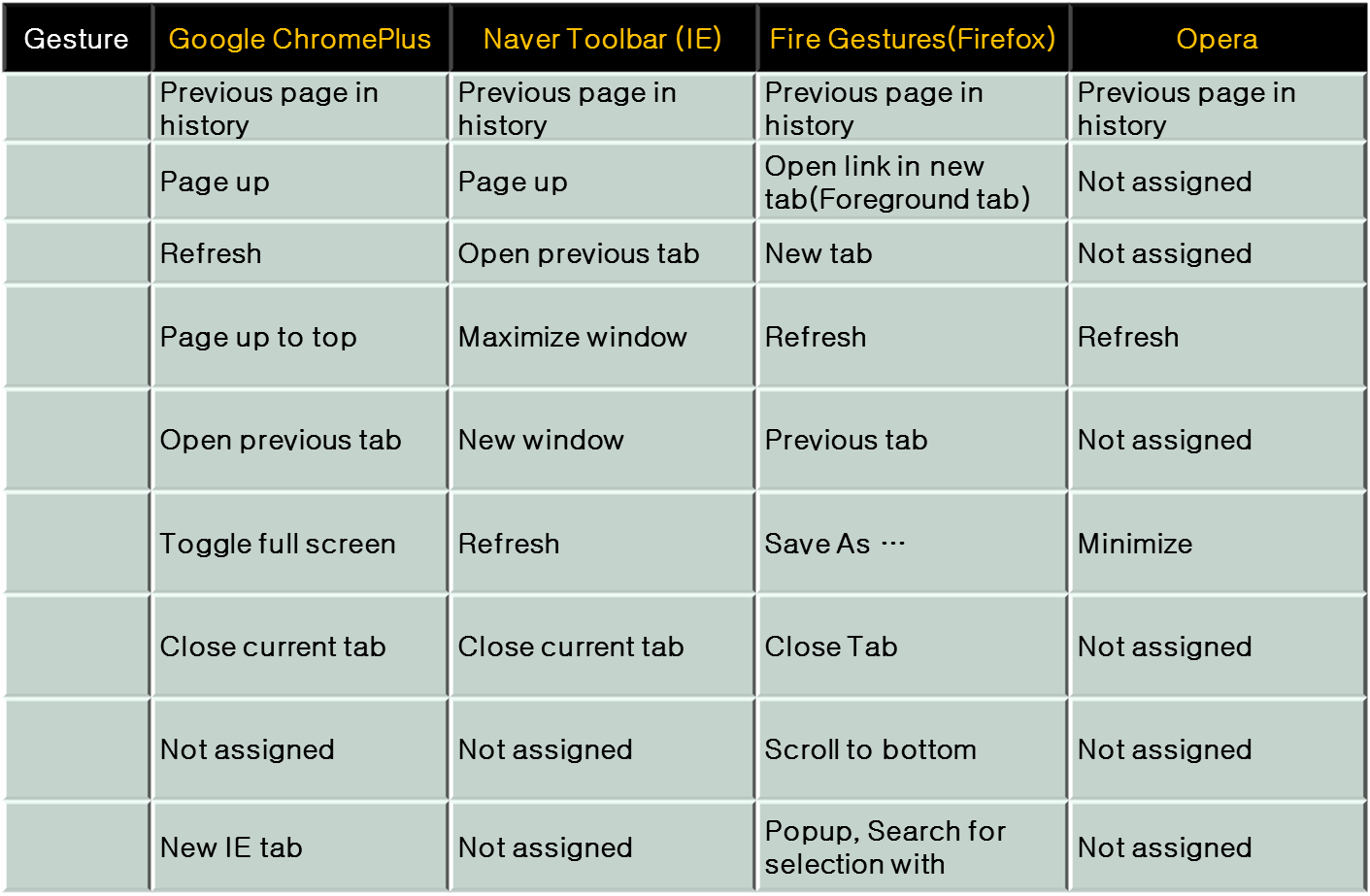
Smart glasses are wearable devices that add information onto reality or actually help people see better. Modern smart glasses are effectively wearable computers which can run self-contained mobile apps. Some are handsfree that can communicate with the Internet via gesture commands and natural language voice commands, while other uses touch buttons. Gesture recognition through which commands to control wearable devices can be generated can be conducted with techniques of computer vision and image processing [1].

This document presents use cases of gesture-based control of multimedia applications in smart glasses and their requirements in aspect of wearable MPEG [2]. Some conventional gesture commands mainly used in smart TV and/or PC environments are briefly introduced. Possible gesture commands in smart glasses environment are presented.

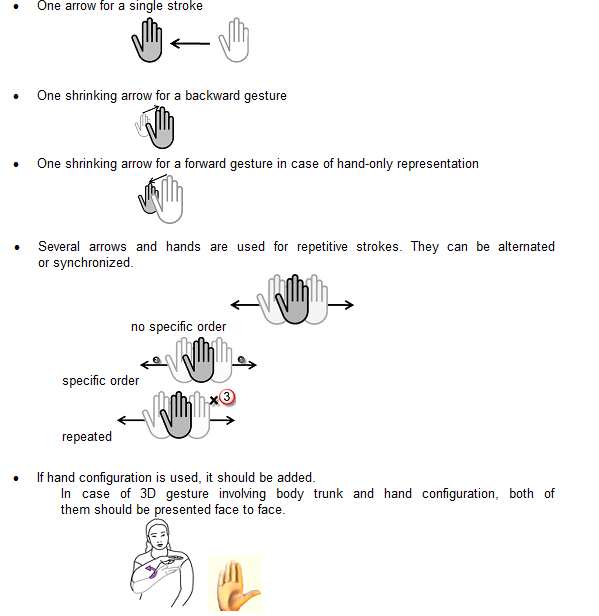
**Conventional Gesture Commands**

In general, a standard 2D camera is used for the gesture recognition in diverse image recognition applications. However, single camera may not be as effective as stereo camera or depth camera since it does not give accurate depth information. But, some companies are providing solutions of gesture recognition in a single camera environment. The standardization of general gesture command is currently underway in the ISO/IEC JTC 1/SC 35.

*Mouse gesture commands defined for commercial browsers*



*Gesture commend defined in ISO/IEC JTC/1 SC35*



* + 1. **Use Cases of Multimedia Applications by Using Gesture Commands**

Diverse multimedia applications such as audio/video presentation/capturing, chatting, game, etc. can be available in smart glasses environment. Then, gesture-based control/manipulation of such multimedia applications would be highly demanded to allow efficient use of smart glasses. Gesture commands presented in the Table 3 can be used for multimedia applications. Such gesture commands can be categorized in terms of the types of applications as follows.

*Common gesture commands*

• Application selection/start/end

• Application basic control

– window control

– select, delete, stop, close

– numbering

– current status information

• Audio/video/image presentation control

– play, pause, zoom-in/-out

– view scaling

– forward, backward

*Application specific commands*

• Application way finding control

– Location

– Map control

• Application website control

• Game

* Control navigation bar
* Chatting
  + 1. **Hand Gesture Metadata of Wearable device (smart glasses)**
* The use cases of gesture-based control of multimedia applications in smart glasses and their requirements in aspect of wearable MPEG was presented in the Warsaw meeting. In a smart glasses environment, hand gesture is considered as a promising natural user interface since it may not be easy to attach mechanical input devices to the smart glasses. In the use case, gesture commands to be used for controlling smart glasses can be generated through the process of gesture recognition which can be performed with techniques of [computer vision](http://en.wikipedia.org/wiki/Computer_vision) and [image processing](http://en.wikipedia.org/wiki/Image_processing). In general, the processing in smart glass is limited to the basic processing of detection and analysis since a small-sized, low-power application process (AP) is embedded. On the other hand, intensive computation such as gesture recognition can be conducted in a remote server or a peripheral device such as smartphone depending on the computation power available in smart glasses.
* In this document, we present the overall process of gesture command generation which consists of two parts: 1) gesture detection and representation in which hand gestures are detected and the detected gesture contours are represented; 2) gesture recognition in which the represented gesture contours are classified into corresponding commands. In order to support efficient multimedia consumption control for devise applications in a wearable environment, we propose that a normative representation way of hand gesture contours as intermediate gesture metadata from which gesture commands defined by a specific application can be extracted should be provided by wearable MPEG.

**Gesture Detection and Representation**

**Overall procedure**

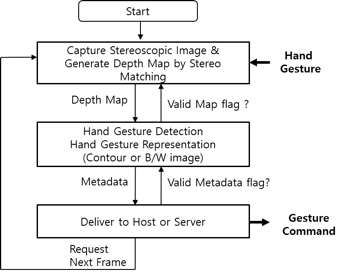
* Figure 23 illustrates the overall procedure of gesture detection and recognition, which consists of stereo image sequences acquisition, hand detection and representation, and gesture recognition as follows:
* Depth map is extracted from stereo images captured by a stereo camera by using stereo matching algorithm in a frame-by-frame basis.
* Hand is detected from depth image by separating the hand region from the background based on a depth histogram. Then a hand contour is obtained by boundary filtering on the detected hand region.
* The hand contour is represented as a metadata of which schema should be specified in wearable MPEG.
* The represented hand gesture contour is delivered to host or server, in which hand gesture is recognized and mapped into a corresponding command through gesture recognition.
* 

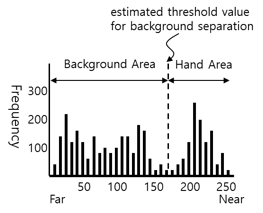
Figure 23. Overall procedure of hand gesture detection and recognition

**Input image acquisition**

* A stereo camera is mainly considered as an input device to capture image sequences for the gesture detection. Depth image extracted from the stereo images by stereo-matching algorithm allows efficient hand detection. While depth image is directly obtained by a depth camera, a depth camera may not be appropriate for smart glasses since it requires LED for IR light source and causes power consumption of light source.

**Detection of hand**

Figure 24 shows an example of simple hand detection procedure, which is processed by a built-in AP in smart glass.

(a) (b)

(c) (d)

(a) Depth map generated by stereo matching

(b) Depth histogram from which a threshold value for separating hand region from background is estimated

(c) Hand region detected from depth map by using the threshold value estimated from the depth histogram

(d) Boundary filtered hand region which is the final result to be used for gesture contour representation

Figure 24. Hand detection procedure

* + 1. **A Review of the Existing MPEG-U and MPEG-7 Tools for Hand Gesture Description in IoMTWReview of Related Tools in MPEG - 7 and MPEG –U**

we present the review of the existing tools in MPEG which might be used to describe hand gestures for hand gesture-based wearable applications in IoMTW. In MPEG-U, the description of hand postures and gesture patterns have been specified to support the intuitive hand based interaction for scene description. MPEG-7 specifies a set of visual descriptors to describe various visual features mainly focusing on content-based indexing and retrieval applications. In MPEG-7, the shape descriptor of the curvature scale space (CSS) descriptor and the motion trajectory descriptor are directly related to the description of hand contour and hand motion trajectory, respectively.

## MPEG - 7 Contour Description

The Curvature Scale-Space (CSS) descriptor is a contour based shape descriptor specified in MPEG-7. Basically, the CSS method treats shape boundary as a 1D signal, and analyzes this 1D signal in scale space. By examining zero crossings of curvature at different scales, the concavities/convexities of shape contour are found. These concavities/convexities are useful for shape description because they represent the perceptual features of shape contour.

The whole process of CSS description consists of the following steps: 1) obtain boundary points and scale normalization; 2) CSS contour map computation; 3) CSS peaks extraction. The CSS contour map is composed of all curvature zero-crossing points zc(t, σ), where t is the location and σ is the scale at which the zero-crossing point is obtained. Fig. 1 shows the evolution of a contour and the associated CSS image. An example of CDD contour map and its CSS map peak is shown in Fig. 2 [8]. Finally, the normalized CSS peaks are used as CSS descriptor to index the shape.

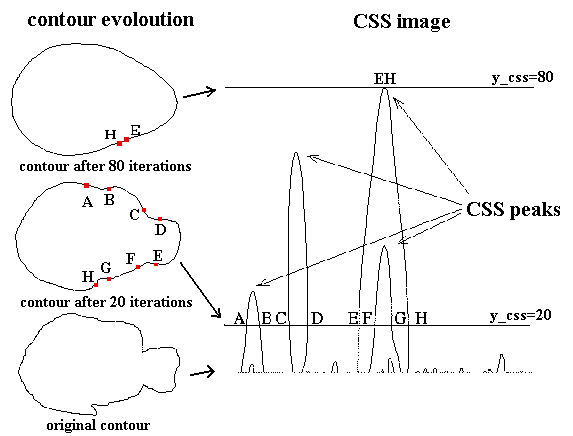


Fig. 1. CSS image formation



Fig. 2. A fish shape (left) and its CSS contour map (middle), CSS peak map (right)

## MPEG-7 Motion Trajectory Description

Hand gestures can represent a specific meaning with moving trajectory as well as the shape of hand, which is generally called dynamic hand gestures. Therefore, a schema to be devised in IoMTW to describe general hand gesture needs to support the description of hand gesture trajectory.

In MPEG-7, there four descriptors characterize various aspects of motion [7], [9]: camera motion, motion trajectory, parametric motion descriptor, and motion activity. Motion of a key point (pixel) from a moving object or region can be characterized by Motion Trajectory descriptor. The Parametric Motion descriptor characterizes an evolution of an arbitrarily shaped region over time in terms of a 2D geometric transformation.

The details on the MPEG-7 motion trajectory is given below [9]. The trajectory model is a first- or second-order piecewise approximation along time, for each spatial dimension of x and y. The core of the description is a set of keypoints, representing the successive spatio-temporal positions of the described object (positions of one representative point of the object, such as its center of mass). They are defined by their coordinates in space (2-D) and time. Additionally, interpolating parameters can be added to specify nonlinear interpolations between keypoints, using a second-order function of time. By default, in the absence of interpolating data, linear interpolation is used. Fig. 3 shows an example of trajectory representation in the x dimension and similarly for the other dimension y.



Fig. 3. Example of trajectory representation (x dimension)

Therefore, dynamic hand gestures can be interpreted as a set of points in a spatio-temporal space as a sequence of keypoints as {(x1, y1), (x2, y2), ∙∙∙∙,(xt, yt)}.

## MPEG – U

In AUI (Advanced User Interaction) interface, which has been standardized by MPEG-U part 2, aims to enhance interaction between scene descriptions and system resources. The data format of the AUI interface specified in MPEG-U is shown in Table 1. As shown in Table 1, a set of hand postures and gesture patterns have been specified to support the intuitive hand-based interaction for scene description. Fig. 4 shows a set of hand postures defined in MPEG-U.

Table. 1 Data format of AUI (Advanced User Interaction) interface defined by MPEG-U

|  |  |
| --- | --- |
| Pattern | Type |
| Geometric | Point, Line, Rect, Arc, Circle |
| Symbolic | Victory, Heart, Rock, Scissors, Paper, Okay |
| Hand Posture | OpenPalm, Fist, Pointing, Thumb-Up,  Thumb-Down, Grap |
| Hand Gesture | Push, Pull, Slap, Slap\_right, Slap\_top, Slap\_bottom,  Circle\_clockwise, Circle\_anti-clockwise, waving, check |
| Touch | Tap, Double Tab, Press, Dragg, Rotate, Flick |
| Composition |  |



Fig. 4. Hand posture of MPEG-U part 2

# Hand Gesture Description in IoMTW

In the use cases of gesture-based wearable applications in IoMTW [1], [2], it is likely required that general hand gestures having any shapes and/or trajectories should be described in an interoperable way in terms of two aspects: 1) diverse gesture-based commands that are used for interactions between a user and a device should be able to be extracted to support diverse potential use cases; 2) detection of hand gestures and their recognition can be done separately in different processing units due to the limit of computation power and other reasons. Therefore, it should be supported that the delivery/storage of general gesture description from which gesture-based commands defined by the given use case are extracted. The schema allowing hand contour and trajectory description based on Bezier curve would be candidate to meet such requirements.

The gesture postures/gestures defined in MPEG-U could be reused with some necessary extensions as gesture-based commands for the wearable use cases. However, in the view of gesture-based wearable applications in IoMTW, a limited set of hand postures and hand gestures defined in MPEG-U may not be sufficient to deal with diverse use cases to be appeared.

The CSS descriptor provided by MPEG-7 is a popular contour based shape descriptor and is effective in the content-based indexing and retrieval of images. In particular, the CSS is rotation and scale invariant, which is effective features in the indexing applications.

However, the gesture with different angle may represent different meanings in the IoMTW use cases. Therefore, hand postures and gestures should be defined depending on the degree of rotation. Furthermore, the contour of hand gestures may be reconstructed form the general description of hand gestures in the process of gesture recognition. In these sense, the MPEG-7 CSS could be used for supporting gesture-based use cases in IoMTW in the limited scope.

The MPEG-7 motion descriptor, which describes the displacement of objects in time in terms of kepoints, can be used to describe hand gesture motion trajectory. In the proposed schema, the description of hand gesture trajectory based on Bezier curve is also provided. In other words, a sequence of center positions of hand contour over time is represented by Bezier curves. In this method, Bezier curve is used for fitting the given trajectory, which is similar to the linear interpolation or the second-order interpolation used in MPEG-7.