INTERNATIONAL STANDARD

ISO/IEC 23090-8

First edition
2020-12

Information technology — Coded representation of immersive media —

Part 8: Network based media processing

Reference number
ISO/IEC 23090-8:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see http://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information.*

A list of all parts in the ISO/IEC 23090 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

This document defines a framework that enables initializing and controlling media processing in a network. A network-based media processing (NBMP) source describes the requested media processing and provides information about the nature and format of the media data. Based on that, an NBMP workflow manager establishes the media processing workflow and informs the NBMP source that the workflow is ready, and that media processing can start. The media source(s) can then start transmitting their media to the network for processing.

An NBMP workflow can be understood as a connected graph of media processing tasks, each of which performs a well-defined media processing operation. The workflow manager ensures the correct operation of the workflow by configuring and monitoring each task as well as the workflow output. The workflow manager is responsible for the selection of the media processing functions and instantiating them as tasks based on the workflow description that is received from the NBMP source.

NBMP abstracts the underlying computing platform interactions to establish, load, instantiate and monitor the media processing entities that will run the media processing tasks. NBMP defines application programming interfaces (APIs) between an NBMP source and workflow manager; workflow manager and task(s); and an API to discover appropriate function(s). NBMP is media format and protocol agnostic. However, it identifies and signals the media, metadata and auxiliary information formats for data exchanged between media source, the workflow manager and tasks.

Annex C provides schema for identifying MPEG compatible functions.

Annex E provides an interface for managing function descriptions in function repository.

Information technology — Coded representation of immersive media —

Part 8:
Network-based media processing

1 Scope

The network-based media processing (NBMP) framework defines the interfaces including both data formats and application programming interfaces (APIs) among the entities connected through digital networks for media processing. Users can access and configure their operations remotely for efficient, intelligent processing. This document describes and manages workflows to be applied to the media data. This process includes uploading of media data to the network, instantiation of the media processing tasks, and configuration of the tasks. The framework enables dynamic creation of media processing pipelines, as well as access to processed media data and metadata in real-time or in a deferred way. The media and metadata formats used between the media source, workflow manager and media processing entities in a media processing pipeline are also specified.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 9899, *Information technology — Programming languages — C*

ISO/IEC 23001–7, *Information technology — MPEG systems technologies — Part 7: Common encryption in ISO base media file format files*

IETF RFC 3339:2002, *Date and Time on the Internet: Timestamps*, https://tools.ietf.org/html/rfc3339

IETF RFC 3986:2005, *Uniform Resource Identifier (URI): Generic Syntax*, https://tools.ietf.org/html/rfc3986

IETF RFC 7231, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, https://tools.ietf.org/html/rfc7231

IETF RFC 6381:2011, *The 'Codecs' and 'Profiles' Parameters for "Bucket" Media Types*

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1.1
function description
FD
logical description of the details of a NBMP function including input and output description, media processing, requirements, etc.

3.1.2
function description document
FDD
document containing function object

3.1.3
function object
FO
function description in JSON format

3.1.4
function repository
storage place where NBMP function description are retrieved from by an NBMP workflow manager or NBMP source

3.1.5
function resource
FR
REST resource identified with a URL and containing a function description document

3.1.6
media processing entity
MPE
entity that runs one or more media processing task(s)

3.1.7
media sink
entity that consumes the output of the NBMP workflow through existing delivery methods

3.1.8
media source
entity that provides media content to the NBMP workflow to be processed

3.1.9
NBMP function
implementation of a standalone and self-contained media processing operation and the corresponding description of that operation

3.1.10
NBMP descriptor
a group of NBMP parameters which describe a set of related characteristics of workflow, function or task

3.1.11
NBMP parameter
variable expressing a characteristic of workflow, function or task

3.1.12
NBMP source
entity that describes media processing in the network

3.1.13
NBMP system
system for processing media across one or more processing entities in the network and consisting of media source (s), a NBMP source, a NBMP workflow manager, a function repository, media processing entity(ies) and media sink(s)

3.1.14
NBMP workflow
graph of one or more connected task(s) that achieve the requested media processing

3.1.15
NBMP workflow manager
entity that provisions tasks and connects them to create, control, manage and monitor a complete NBMP workflow based on a workflow description document

3.1.16
port
logic input and output endpoints by specifying where the data comes in and out

3.1.17
port mapping
PM
data structure used to make references for NBMP function ports to the workflow input and output streams, especially to dynamic inputs and outputs

3.1.18
stream ID
unique string for identifying an input or output stream of workflow/function/task

3.1.19
supplementary information
metadata or auxiliary information related to the media data

3.1.20
task
runtime instance of NBMP function that gets executed inside a media processing entity

3.1.21
task description
logical description of the runtime details of a task, including input and output, requirements, configuration etc.

3.1.22
task description document
TDD
document containing task description object

3.1.23
task object
TO
task description in JSON format

3.1.24
task resource
TR
REST resource identified with an URL and containing task description document

3.1.25
workflow description
logical description of the details of the media processing including input and output description details, requested media processing, requirements etc.

3.1.26
workflow description document
WDD
document containing workflow description object

3.1.27
workflow object
WO
workflow description in JSON format

3.1.28
workflow resource
WR
REST resource identified with an URL and containing workflow description document

3.2 Abbreviated terms

API	application programming interface
CPU	central processing unit
DAG	directed acyclic graph
DASH	dynamic adaptive streaming over HTTP
GPU	graphics processing unit
HTTP	hyper-text transfer protocol
JSON	JavaScript object notation
MMT	MPEG media transport
NBMP	network-based media processing
NVP	name value pair
PCC	point-cloud compression
RTP	real-time transport protocol
TCP	transmission control protocol
UDP	user datagram protocol
URI	uniform resource identifier
URL	uniform resource locator
URN	uniform resource name
XML	eXtensible Markup Language

4 Conventions

The following naming convention apply in this document:

— Names comply to dash-case convention, i.e. words in a name are separated with '-'.

— Operations, resources, documents, descriptions and descriptors are identified by an upper-case first letter. In these names, words after '-' start with an upper-case letter. All other letters are lower-case.

— Parameters and their values are identified by lower-case letters. No uppercase letter is used in these names.

— Parameter values are identified by '', e.g. 'value'.

— JSON objects comply to dash-case convention with all lower-case letters.

The following legends are used in tables:

— cardinality: 1 = exactly one, 0-1 = zero or one, 0-N = zero or more, 1-N = one or more

— P: parameter

— O: object

— N/A: not applicable

The range of "unsigned integer" is 0 to 2^{53}-1.

5 Overview

5.1 General

The network-based media processing (NBMP) framework enables the creators, service providers and consumers of digital media to describe media processing operations that are to be performed by the media processing entities in a network, as shown in Figure 1. It provides a method to describe a workflow by composing a set of media processing functions that are accessible through NBMP application programming interfaces (APIs). A media processing entity (MPE) runs processing tasks applied on the media data and the related metadata received from media sources or other tasks. MPE provides capabilities for configuring, managing and monitoring processing tasks. A media processing task is a process applied to media and metadata input(s), producing media data and related metadata output(s) to be consumed by a media sink or other media processing tasks.

The NBMP framework is media format agnostic and supports any format of media content, including the existing MPEG codecs and MPEG formats such as ISO/IEC 13818-1 (Reference [10]), ISO/IEC 14496-12, ISO/IEC 23008-1 (Reference [12]) and ISO/IEC 23009-1 (Reference [13]).

The NBMP framework supports the delivery over IP-based networks using common transport protocols such as TCP, UDP, RTP (Reference [17]) and HTTP.

The NBMP framework also support the existing delivery methods such as streaming, file delivery, push-based progressive download, hybrid delivery, multipath and heterogeneous network environments.

5.2 Architecture

NBMP specifies interfaces to create and control media processing workflows in the network. NBMP can be split into a control plane and a media plane. The control plane covers the following APIs:

— Workflow API is used by NBMP source to create and control a media processing workflow.

— Function discovery API provides the means for workflow manager and/or NBMP source to discover media processing Functions that can be loaded as part of a media processing workflow.

— Task API is used by the workflow manager to configure and monitor a task at runtime.

On the media plane, NBMP defines the media formats, the metadata, and the supplementary information formats between the NBMP source and the task, as well as between the tasks themselves.

The discovery of the NBMP workflow manager and function repository is out of scope.

The NBMP architecture is shown in Figure 1.

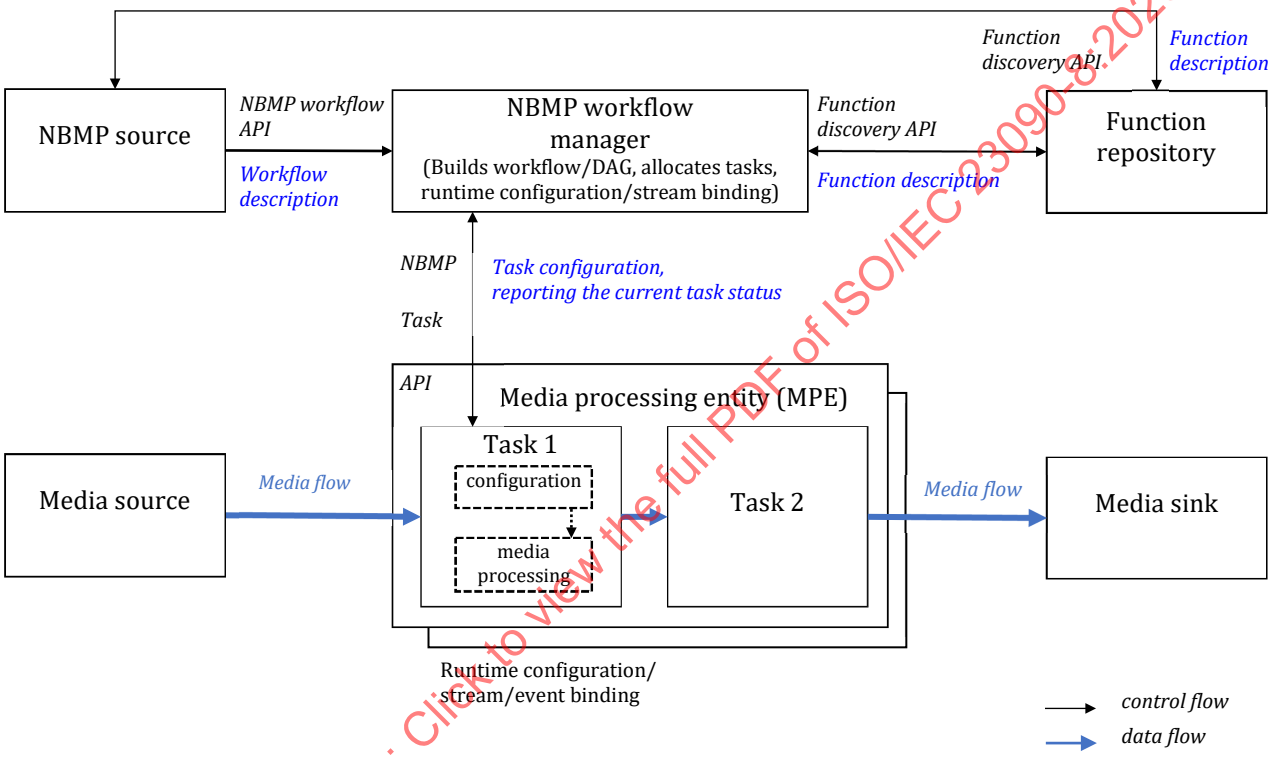

Figure 1 — NBMP reference architecture

5.3 NBMP workflow

5.3.1 General

The workflow manager receives a workflow description document from a NBMP source and builds a workflow for requested media processing. This subclause describes how media processing functions are selected, and then the corresponding tasks are configured and distributed to a set of media processing entities as part of the workflow procedure.

5.3.2 Workflow processing model

This subclause describes the detailed processing model of the workflow manager.

Since the set of functions that are provided by a function repository can be read by an NBMP source, the workflow description document can be composed in two different ways, based on use cases and actual needs.

a) The NBMP source requests the creation of a workflow using a set of functions in the function repository: with this type of workflow creation request, the NBMP source is responsible for the selection of functions that are included in the workflow. In this case, the NBMP source requests the creation of the workflow:

   1) using a description of tasks by which the workflow is to be created;

   2) specifying the connection map to define the connections of the inputs and outputs of tasks. The details of the connection map can be found in subclause 8.5. When the workflow manager receives the above information from NBMP source, it instantiates the tasks based on function names and connects the tasks according to what is defined in the connection map.

b) The NBMP source requests the creation of a workflow using a set of keywords by which the workflow manager constructs the workflow. In this case, the NBMP source may not be aware of a set of functions to be inserted into the workflow, and requests the creation of the workflow:

   1) using a set of keywords by which the workflow manager finds the appropriate functions;

   2) specifying the requirements of the workflow using descriptors from Table 4 in the workflow description document (WDD).

When the workflow manager receives the above information from the NBMP source, it will create the workflow by searching for appropriate Functions using the keywords specified in the processing descriptor. The workflow manager will then use the other descriptors in the workflow description document to provision Tasks and connect them to create the final workflow.

The processing model of the workflow manager can be described using the following steps.

i) Discovery of available media processing functions:

The NBMP function repository provides the function discovery interface as defined in subclause 7.4, to allow external entities to query for a media processing function that fulfils the requested processing. The workflow manager has access to a directory service that offers a searchable list of media processing functions. The workflow manager can use the description of the tasks in the workflow description document to find the appropriate functions for the current workflow.

ii) Selection of media processing tasks to prepare the workflow:

When a request for media processing arrives from the NBMP source, the workflow manager searches the function repository to find the list of all functions that could fulfil the workflow. Using the workflow description from the NBMP source, the workflow manager finds the functions from the function repository needed to implement the workflow. This step depends on the information for media processing from the NBMP source, such as the input and output description, and the description of the requested processing; and the information in other descriptors (as documented in Clause 8) for each function in the function directory. The mapping of the source requests to appropriate media processing tasks to be included in the workflow is part of the implementation of the NBMP in the network. To reference and link input sources with input port names and output port names at the time of task creation, the input-ports and output-ports shall be used to make references to the input streams. The architecture of the workflow manager interworking with the function repository is shown in Figure 2.

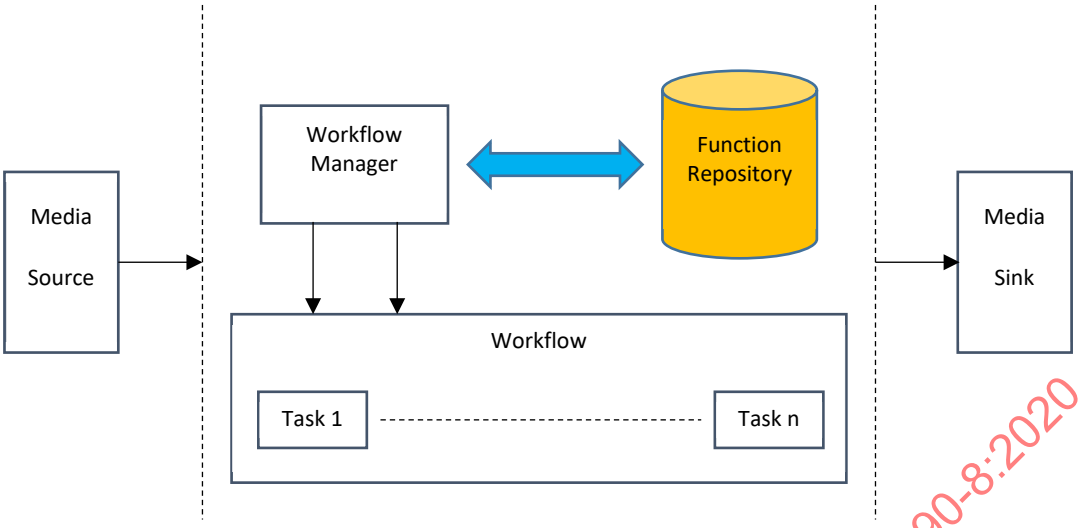

Figure 2 — Workflow manager interworking with function repository

The search for appropriate functions that need to be instantiated as tasks is performed by the workflow manager using function discovery API. Alternatively, the workflow manager may retrieve the details of all functions in the function repository using function discovery API. The workflow manager can then compare the information for media processing from the source with different descriptors of each function.

iii) Configuration of selected media processing tasks in the workflow:

Once the functions required to be included in the workflow are identified, the next step is to instantiate them as tasks and configure the tasks so they can be added to the workflow. The NBMP workflow manager extracts the configuration data from the media processing information received from the NBMP source and configures the corresponding tasks. The configuration of these tasks is performed using the task API.

5.3.3 Task allocation and distribution

The workflow manager uses the workflow to perform processing deployment and configure the media processing entities as described in subclause 5.3.2. In the case of computationally intensive media processing requests, the workflow manager may set up multiple computational instances and distribute the workload among those multiple instances. In this case, the workflow manager takes the responsibility to connect and configure all those instances as needed. This can be achieved in two different ways:

— the workflow manager allocates the same task to multiple instances and provisions a load balancer to distribute the workload among those instances using a chosen scheduling mechanism;

— the workflow manager allocates different operations of the same task to different instances (e.g., parallel operations).

In both cases, the workflow manager is responsible for setting up the workflow paths between those instances, so the required workload can be successfully realized. The workflow manager also configures the tasks to push the processed media data/streams (or make them available through a pull mechanism) to the next task in the workflow graph.

5.3.4 Workflow graph

When the workflow manager receives a workflow description document (WDD) from a NBMP source, it performs selection of media processing functions to be inserted into the workflow as described in

subclause 5.3.2. Once the list of tasks that need to be included in the workflow is compiled, the workflow manager then connects those tasks to prepare the required workflow.

The Workflow Manager can generate a directed acyclic graph (DAG) from WDD. Figure 3 is an example of a DAG. Each node of the DAG represents a processing task in the workflow. A link connecting one node to the other node in the graph represents the transfer of output of the former node as input to the later one. The details for input and output ports for a task are provided in the task's general descriptor in subclause 8.2. An NBMP workflow graph in general could have multiple inputs and outputs.

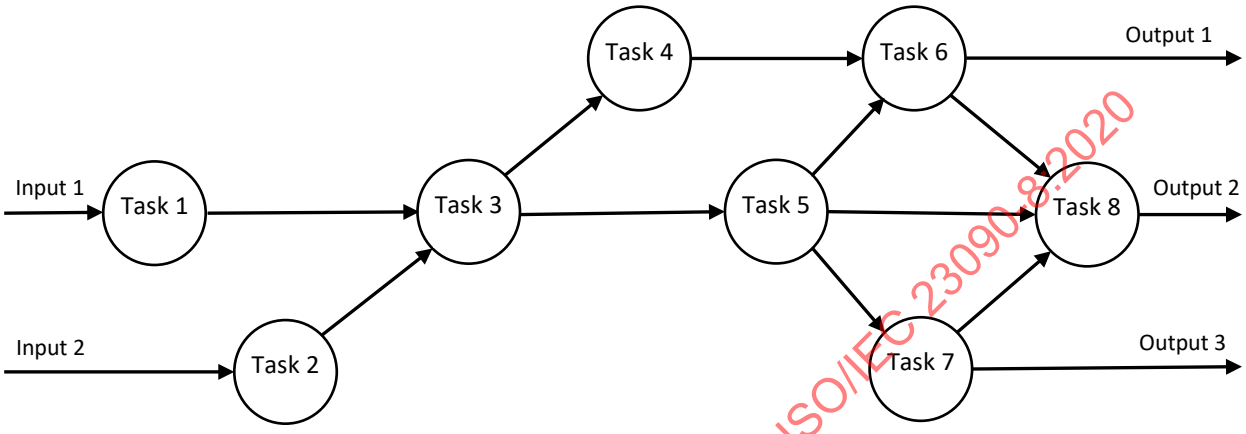

Figure 3 — Example of workflow directed acyclic graph (DAG)

5.4 Relationship between logical definitions, data objects and REST resources

Figure 4 demonstrates the relationship between logical definitions, data objects and REST resources used in this document.

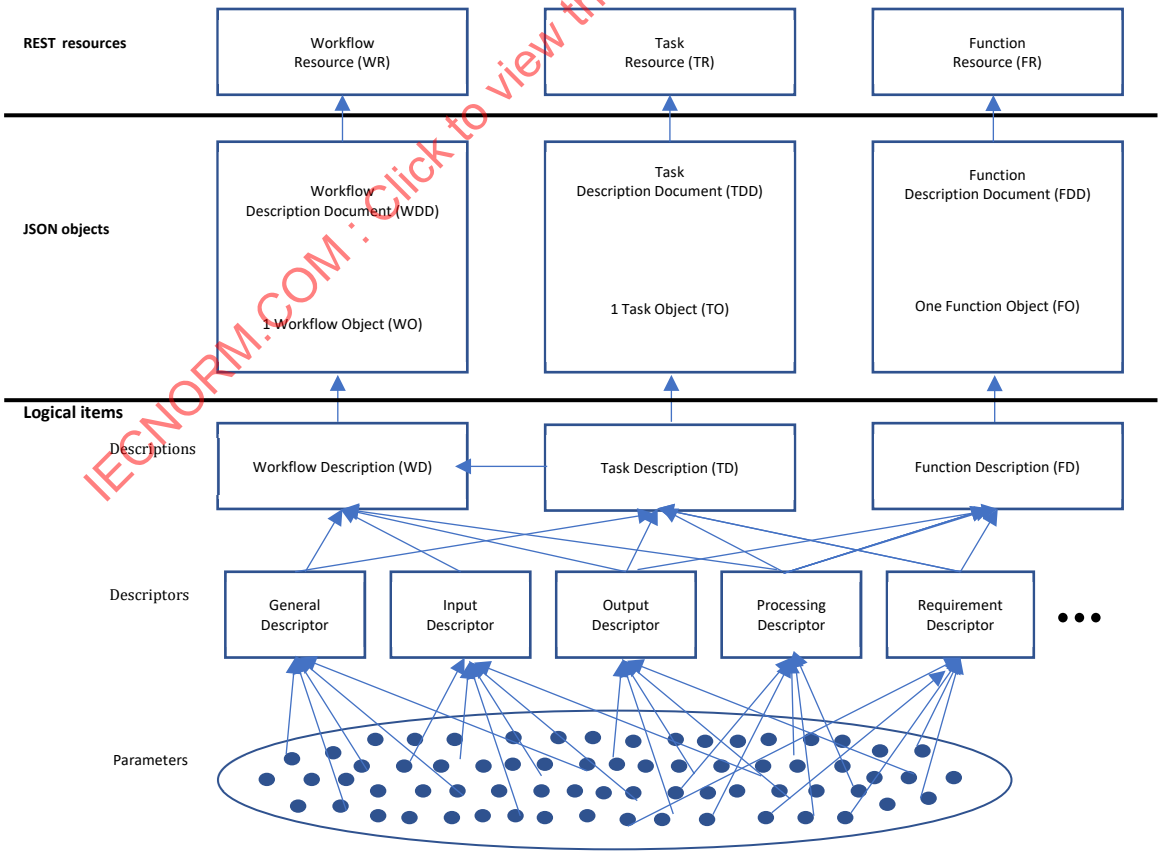

Figure 4 — The relationship between different items in this document

As shown in Figure 4, there is a one-to-one relationship between a logical description and the corresponding data object, data document and REST resource. As shown in Figure 4:

a) NBMP logical items consist of parameters, descriptors and descriptions, all of which are defined in this document.

b) A workflow object (WO), task object (TO) and function object (FO) are realization of the corresponding main descriptors as JSON (Reference [18]) objects.

c) A workflow description document (WDD), task description document (TDD) and function description document (FDD) are documents containing single WO, single TO and single FO, respectively. These documents are JSON (Reference [18]) objects.

d) A workflow resource (WR), task resource (TR) and function resource (FR) are WDD, TDD, and FDD with valid URLs, respectively and therefore REST resources.

5.5 Description of the defined entities in this document

This subclause outlines the defined entities in this document.

5.5.1 NBMP APIs

The following API of Figure 1 are in the scope of this document:

a) NBMP workflow API

b) NBMP task API

c) NBMP function discovery API

Each API has the following aspects:

1) API operations

2) API requests

3) API responses

5.5.2 Content format

The following content are in the scope of this document:

e) workflow object (WO), task object (TO) and function object (FO) in JSON

f) workflow description document (WDD), task description document (TDD), and function description document (FDD)

5.5.3 Definitions

The following items are in the scope of this document:

a) NBMP descriptions and descriptors

b) NBMP parameters' names, definitions, value ranges, and units

5.5.4 Functional behaviour

The following entities have functional aspects that are in the scope of this document:

a) workflow manager

b) NBMP source

c) function repository

d) NBMP task

The following entities are not in the scope of this document:

1) MPE

2) media source

3) media sink

6 NBMP descriptions

6.1 NBMP function description (FD)

6.1.1 General

NBMP functions (FD) that are included in the function repository shall conform to this subclause.

6.1.2 Description

An NBMP function shall be described using a set of descriptors outlined in Table 1. The descriptors are defined in Clause 8.

Table 1 — Function description (FD)

Descriptor	Additional constraints	Cardinality
Scheme	None	0-1
General	Following parameters shall not be present: — priority. The parameter id shall be a valid URI according to ETF RFC 3986.	1
Input	Following parameters shall not be present: — stream-id.	1
Output	Following parameters shall not be present: — stream-id.	1
Processing	If the function is a function group, this descriptor shall contain a connection-map object. Following parameters shall not be present: — start-time.	0-1
Requirements	In function description, only maximum or minimum values should be specified.	0-1

Descriptor	Additional constraints	Cardinality
Configuration	None	0-1
Client-Assistance	Following parameters shall not be present: — Measurement-collection-list; — Source-assistance-information.	0-1
Assertion	None	0-1
Variables	None	0-1
Events	None	0-1
Security	None	0-1

6.1.3 Function group

6.1.3.1 General

A function group is a special function that is defined as a set of functions and their connections. The contained functions in a function group are designed to work together, which allows the workflow manager to integrate them as described in a workflow. A function group is described as a sub-workflow, i.e. a DAG of connected functions. In addition to the graph, a function group also suggests the configuration parameters for each of the tasks instantiated out of the function group.

6.1.3.2 Example

Figure 5 gives an example of a function group that performs upscaling of a video. The upscaler function shown is limited to 30fps, but the input video is 60fps.

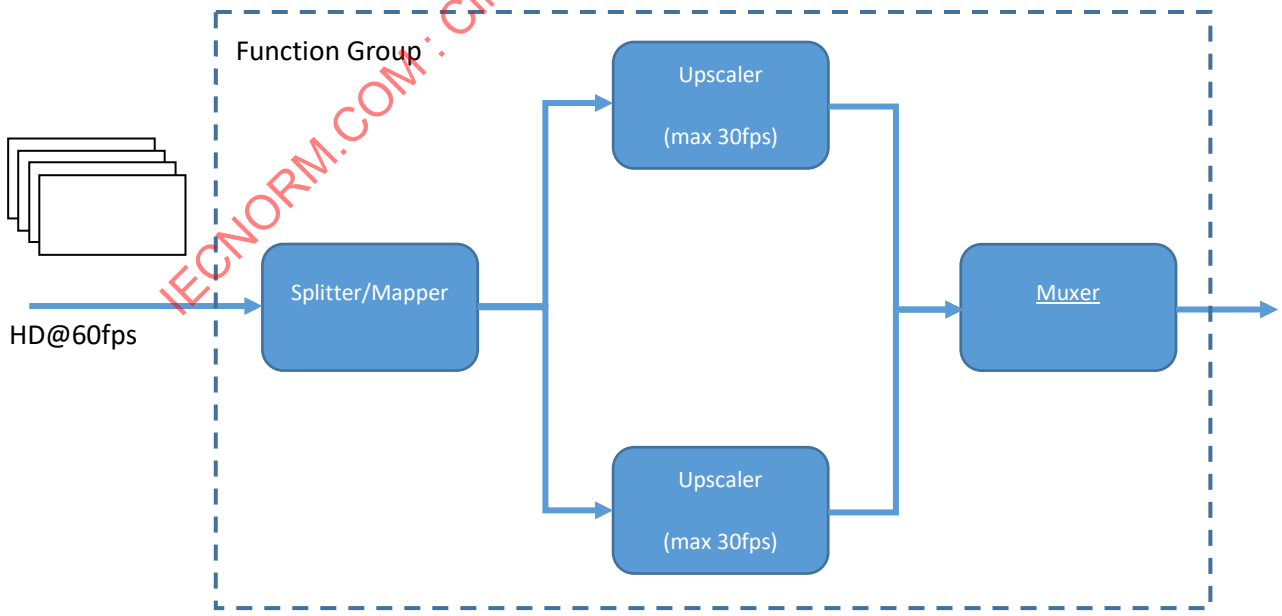

Figure 5 — Example of a function group

In Figure 5, a decode/splitter function is instantiated to prepare the content for upscaling based on the requirements and constraints of the following upscaler functions. A muxer/encode function is then used

to re-multiplex the upscaled videos into a single output stream. A joint configuration is suggested by the function group description. An additional synchronization metadata stream may be generated by the splitter/mapper and propagated down to the muxer through the upscalers to ensure that the synchronization and frame order is not lost.

6.1.3.3 Requirements

The function group shall be described with a function description from Table 1 with the additional constraints of Table 2.

Table 2 — Additional function group constraints

Descriptor	Additional constraints
General	Parameter is-group shall be set to 'true'.
Processing	Object connection-map shall be present.

Any additional restrictions for each function in a function group and ports of a function group shall be provided by the corresponding connection-map and function-restrictions objects in processing descriptor (subclause 8.5).

6.2 NBMP task description (TD)

6.2.1 General

An NBMP task shall be described according to subclause 6.2.

6.2.2 Description

A NBMP task shall be described using a set of descriptors outlined in Table 3. The descriptors are defined in Clause 8.

Table 3 — Task description (TD)

Descriptor	Additional constraints	Cardinality
Scheme	None	0-1
General	The workflow manager shall assign actual stream IDs and creates necessary ports if needed.	1
Input	None	1
Output	None	1
Processing	Following parameters shall not be present: — keywords; — function-restrictions; — connection-map.	1
Requirements	The parameter typical-delay specifies the delay requirements for Task.	0-1
Configuration	None	0-1

Descriptor	Additional constraints	Cardinality
Startup-Delay	None	0-1
Client-Assistance	None	0-1
Failover	None	0-1
Monitoring	None	0-1
Assertion	None	0-1
Reporting	None	0-1
Notification	None	0-1
Security	None	0-1
Acknowledge	None	0-1

6.2.2 Task lifecycle

An NBMP task transitions through different states at different points of its execution. This subclause describes the lifecycle of a task as shown in Figure 6.

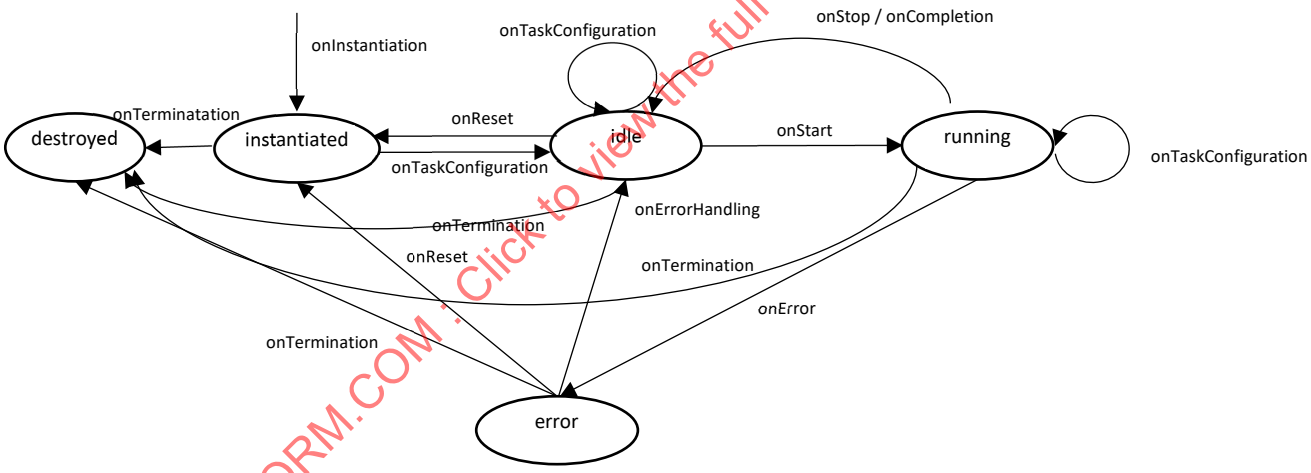

Figure 6 — Task lifecycle

The different states in the task lifecycle are as follows:

— **instantiated:** A task is in an instantiated state when it is instantiated by the workflow manager through the services of the infrastructure manager as represented using the onInstantiation transition. When the task is configured (as represented using onTaskConfiguration transition), its state changes to the idle state. Alternatively, if the task is terminated (as represented using the onTermination), it moves to the destroyed state.

— **idle:** When the task is in an instantiated state and the workflow manager performs a task configuration (as represented using the onTaskConfiguration transition), the task moves to idle state. In the idle state, the task is configured with the required processing. When the task is started (as represented using the onStart transition), the task moves to the running state. Alternatively, in the idle state, the task can be re-configured and stay in the idle state. In idle state, if the task is terminated (as represented using the onTermination transition), the task moves to destroy state.

In idle state, if the task is reset (as represented using the onReset transition), the task moves to instantiated state.

— **running:** While the task is in an idle state, and it is started (using the onStart transition), the task moves from idle state to running state. In running state, the task assigned to the media processing entity is processing data that it receives from either the previous task in the workflow or the NBMP source. Alternatively, in running state, if the workflow manager performs reconfiguration of the task (as represented using the onTaskConfiguration transition), and if the reconfiguration results in processing reconfiguration with execution on current media/metadata streams to the task, then the task stays in running state. In the running state, if the task is stopped (as represented using the onStop transition) or completed (as represented using the onCompletion transition), the task moves to the idle state. In the running state, if the task encounters an error (as represented using the onError transition), the task moves to error state. Finally, in running state, if the task is terminated (as represented using onTermination transition), it moves to the destroyed state.

— **error:** The task is in error state when the task encounters an error and cannot process the media data or metadata. Upon handling the error (as represented using the onErrorHandling transition), the task moves back to the idle state. Alternatively, while in error state, and the task is reset (as represented using onReset transition), the task moves to instantiated state. Finally, in error state, if the task is terminated, the task moves to the destroyed state.

— **destroyed:** The task is in the destroyed state when the task is terminated by the workflow manager. The task can be disposed of and cannot be re-used.

The task's state may be reflected in the general descriptor's 'state' parameter.

Each of the above transitions except the onError transition occurs by a task operation initiated by workflow manager, as outlined in subclause 10.3. The OnError transition occurs due to a task's internal state changes.

6.3 NBMP workflow description (WD)

6.3.1 General

The workflow description (WD) is passed from the NBMP source to the workflow manager. The WD describes details such as input and output data, required functions and other requirements for the workflow.

6.3.2 Description

A NBMP workflow shall be described using a set of descriptors outlined in Table 4. The descriptors are defined in Clause 8.

Table 4 — Workflow description (WD)

Descriptor	Additional constraints	Cardinality
Scheme	None	0-1
General	Following parameters shall not be present: — input-ports; — output-ports; — is-group.	1
Repository	None	0-1

Descriptor	Additional constraints	Cardinality
Input	None	1
Output	None	1
Processing	Following parameter shall not be present: — url.	1
Requirements	The parameter typical-delay specifies the end-to-end delay requirements for the workflow.	0-1
Client-Assistance	None	0-1
Failover	None	0-1
Monitoring	Following parameter shall not be present: — variable.	0-1
Assertion	None	0-1
Reporting	None	0-1
Notification	None	0-1
Acknowledge	None	0-1
Security	None	0-1
Cardinality: 1= exactly one, 0-1= zero or one		

6.3.3 Workflow lifecycle

Figure 7 describes a workflow's lifecycle:

— **instantiated**: A workflow is put in an instantiated state by the workflow manager through the services of infrastructure manager (as represented using onInstantiation transition). When the workflow is configured (as represented using onWorkflowConfig transition), the workflow moves to the idle state. Alternatively, if the workflow is terminated (as represented using onTermination transition) while in this state, it moves to the destroyed state.

— **idle**: When the workflow is in an instantiated state and the workflow manager performs workflow configuration (as represented using onWorkflowConfig transition), the workflow moves to the idle state. In idle state, media processing entities have been setup (through the help of the infrastructure manager) and the tasks running in the MPEs are provisioned and configured. When the workflow is started (as represented using onStart transition), the workflow moves to running state. Alternatively, in the idle state the workflow can be re-configured. In this case, it stays in idle state waiting for media data or metadata to arrive. In the idle state, if the workflow is terminated (as represented using onTermination transition), the workflow moves to the destroyed state. In idle state, if the workflow is reset (as represented using onReset transition), the workflow moves to the instantiated state.

— **running**: While the workflow is in the idle state, and it is started (using onStart transition), the workflow moves from the idle state to the running state. In the running state, the data from the

NBMP Source is processed by the MPEs in the workflow. Alternatively, in running state if the workflow manager performs reconfiguration of the workflow (as represented using onWorkflowConfig transition), and if the reconfiguration results in processing reconfiguration with execution on current media/metadata streams, then the workflow stays in running state. In running state, if the workflow is stopped (as represented using onStop transition), or the processing is completed (as represented using onComplete transition), the workflow moves to the idle state. In running state, if the workflow encounters an error (as represented using onError transition), the workflow moves to error state. Finally, in running state, if the workflow is terminated (as represented using onTermination transition), it moves to the destroyed state.

— **error**: The workflow is in error state when the workflow encounters an error and cannot continue with workflow processing. Upon handling the error (as represented using onErrorHandling transition), the workflow moves back to the idle state. Alternatively, while in error state, the workflow is reset (as represented using onReset transition) and the workflow moves to instantiated state. Finally, in the error state if the workflow is terminated, the workflow moves to the destroyed state.

— **Destroy**: The workflow is in destroy state when the workflow is terminated by the workflow manager. The workflow will need to be instantiated for it to be used again.

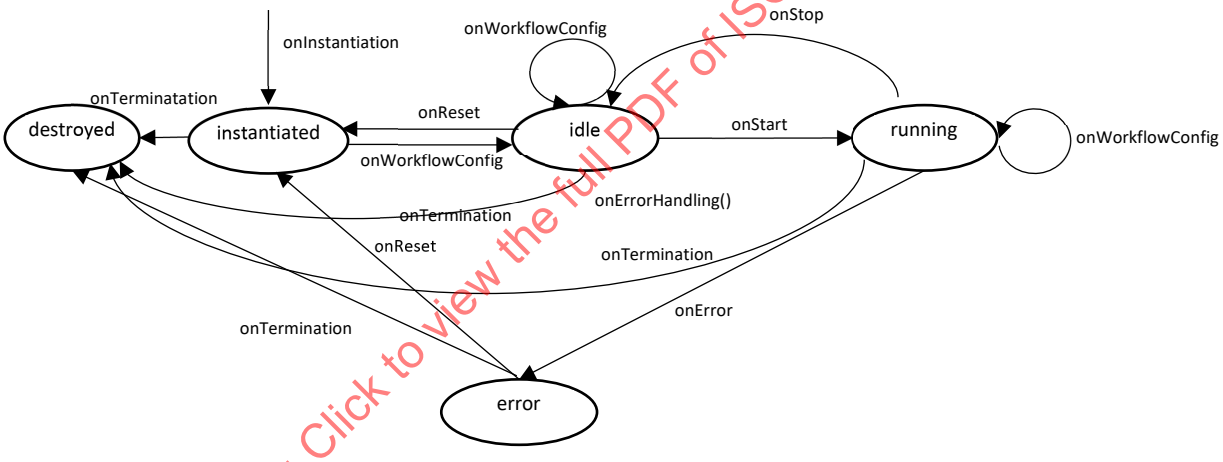

Figure 7 — Workflow lifecycle

Each of the above transitions except the onError transition occurs by a workflow operation initiated by NBMP source, as outlined in subclause 10.1. The OnError transition occurs due to a workflow's internal state changes.

7 NBMP interfaces

7.1 General

Different entities of the NBMP system such as the NBMP source, workflow manager and the media processing entities require APIs to invoke and answer media service requests. This clause describes the set of APIs supported by the NBMP system. These APIs are:

a) Workflow APIs: Between the NBMP source and workflow manager.

b) Task APIs: Between the workflow manager and media processing tasks.

c) Function discovery APIs: Between the workflow manager/NBMP source and function repository.

In response to any request over a NBMP API, when the response's body includes one or more descriptions, acknowledgement descriptor may be added to descriptions and descriptors of the response as the following:

1) If the description or descriptor is not fulfilled, not supported or partially accommodated (as defined in Table 45), one or more acknowledge Descriptors may be included in descriptions (and descriptors) to signal the status.

2) If a description or descriptor does not include an acknowledge descriptor, then it is considered fulfilled.

3) Any description may contain at most one acknowledge descriptor.

4) Any descriptor may contain at most one acknowledge descriptor.

When a description or descriptor is fulfilled, it means that all the parameters of that description or descriptor has been fulfilled (resource allocated satisfying those parameters).

Acknowledge descriptor is intended to be used to signal fulfilment of a request. Therefore, the content of an acknowledge is only meaningful when it is included in a response. However, any request over an NBMP API still may contain an item which includes acknowledge descriptors. The receiving entity shall ignore these descriptors during the processing of the item. The receiving entity shall remove or update those descriptors in the response, reflecting the latest status of included parameters.

7.2 Workflow APIs

7.2.1 General

As shown in the NBMP architecture diagram in Figure 1, the NBMP source uses the NBMP workflow API to communicate with the workflow manager for configuring and controlling media processing in the network.

When the NBMP source sends a request to the workflow manager by including a workflow resource (WR) in a workflow API's operation, the workflow manager parses the WR, the included WDD and its descriptors, and takes the appropriate actions according to the requested API operation. Then, it acknowledges the request with a response.

In this subclause, a workflow API's resources and operations are defined.

7.2.2 Workflow resources

A workflow resource (WR) is used for various workflow API operations.

The WR is a REST [REST] resource, which shall contain exactly one workflow description document (WDD).

The WR shall be in one of the following formats: JSON.

7.2.3 Workflow API operations

The workflow API is used by the NBMP source to manage workflows through a workflow manager. The workflow manager shall support the workflow API operations shown in Table 5.

Table 5 — Workflow API operations

Operation	Description	Request resource requirements	Response requirements
CreateWorkflow	Create a workflow	WR including information needed to create a workflow The general descriptor's id shall not be included in this request	If successful, shall include: 1) HTTP status code 201 2) Response's body with updated WR including: a) A value for General descriptor's Id b) Updated information including endpoint information where to send media data, metadata, and other information If failed, shall include: 1) HTTP status codes 4xx or 5xx 2) Optionally, response's body with updated WR signalling failed descriptors or parameters The response may include an HTTP status code 3xx. A new request may be made using the redirection information in the HTTP header. If accepted, but the workflow is not created immediately, shall include:
			1) HTTP status code 202 2) HTTP header Retry-After: HTTP-date / delay-seconds, in which HTTP-date /delay recommends the date or delay in seconds (as defined by RFC 7231) to get WR using operation RetrieveWorkflow. Response's body with updated WR including a value for General descriptor's id
UpdateWorkflow	Update an existing workflow	Updated WR with identical general's id, previously received in CreateWorkflow's response	If successful, shall include: 1) HTTP status code 201 2) Response's body with updated WR including: a) General descriptor's id identical to the one in the request b) Updated information including endpoint information where to send media data, metadata, and other information If failed, shall include: 1) HTTP status codes 4xx or 5xx 2) Optionally, response's body with updated WR signalling failed descriptors or parameters If accepted, but the workflow response is not ready yet, shall include: 1) HTTP status code 202 2) HTTP header Retry-After: HTTP-date / delay-seconds, in which HTTP-date

Operation	Description	Request resource requirements	Response requirements
			/delay recommends the date or delay in seconds (as defined by RFC 7231) to get WR using operation RetrieveWorkflow. Response's body with updated WR including a value for general descriptor's id
DeleteWorkflow	Terminate an existing workflow	WR with identical general's id, previously received in CreateWorkflow's response	If successful, shall include: HTTP status code 200 If failed, shall include: 1) HTTP status codes 4xx or 5xx 2) Optionally, response's body with updated WR signalling failed descriptors or parameters
RetrieveWorkflow	Retrieve an existing workflow	WR with identical general's id, previously received in CreateWorkflow's response	If successful, shall include: 1) HTTP status code 201 2) Response's body with updated WR including: a) General descriptor's identical to the one in the request b) Updated information including endpoint information where to send media data, metadata, and other information If failed, shall include: 1) HTTP status codes 4xx or 5xx 2) Optionally, response's body with updated WR signalling failed descriptors or parameters If accepted, but the workflow response is not ready yet, shall include: 1) HTTP status code 202 HTTP header Retry-After: HTTP-date / delay-seconds, in which HTTP-date /delay recommends the date or delay in seconds (as defined by RFC 7231) to get WR using operation RetrieveWorkflow

The included WDD in any response should include one "link" object, including a "ref" with value "self" and a URL according to IETF RFC 3986 indicating the location of the WDD.

7.3 Task APIs

7.3.1 General

The task API defines the interfaces for configuration of media processing tasks by the workflow manager, after the resources for the task are allocated in MPE. This subclause defines these API's operations and resources.

7.3.2 Task resource

A task resource (TR) is used for various task API operations.

The TR is a REST resource that shall contain exactly one task description document (TDD).

The TR shall be in one of the following formats: JSON.

7.3.3 Task API operations

This subclause defines task API operations. The workflow manager uses the task API operations to configure and control a task. Task API operations are defined in Table 6.

Table 6 — Task configuration API

Operation	Description	Request parameters	Response requirements
CreateTask	providing the task a configuration for media processing after MPE resources are allocated	TR including the information for task configuration General descriptor's id shall not be included in this request.	If successful, i.e. after task is instantiated to 'idle' state, shall include: 1) HTTP status code 201 2) Response's body with updated TR including: a) A value for General descriptor's Id b) Updated information including endpoint information where to send media data, metadata, and other information If failed, i.e. if task is not instantiated to 'idle' state, shall include: 1) HTTP status codes 4xx or 5xx 2) Optionally, response's body with updated TR signalling failed descriptors or parameters If accepted, but task is not created immediately, shall include: 1) HTTP status code 202 2) HTTP header Retry-After: HTTP-date / delay-seconds, in which HTTP-date /delay recommends the date or delay in seconds (as defined by RFC 7231) to get TR using operation UpdateTask. Response's body with updated TR including a value for General descriptor's id
UpdateTask	modify task's configuration	updated TR with identical general's id, previously received in CreateTask's response NOTE A parameter is added here to change the state of task from running to 'idle' state	If successful, i.e. the new configuration is in effect, shall include: 1) HTTP status code 201 2) Response's body with updated TR including: a) General descriptor's id identical to the one in the request b) Updated information including endpoint information where to send media data, metadata, and other information If failed, i.e. the new configuration did not occur, shall include: 1) HTTP status codes 4xx or 5xx 2) Response's body with updated TR signalling failed descriptors or parameters If accepted, but task is not created immediately, shall include: 1) HTTP status code 202

Operation	Description	Request parameters	Response requirements
			2) HTTP header Retry-After: HTTP-date / delay-seconds, in which HTTP-date /delay recommends the date or delay in seconds (as defined by RFC 7231) to get TR using operation UpdateTask. Response's body with updated TR including a value for General descriptor's id
GetTask	retrieve task configuration information	TR with identical general's id, previously received in CreateTask's response	If successful, i.e. was able to retrieve the current configuration, shall include: 1) HTTP status code 200 2) Response's body with updated TR including: a) General descriptor's id identical to the one in the request b) Updated report descriptors which were included in the request If failed, i.e. was not able to retrieve the current configuration, shall include: 1) HTTP status codes 4xx or 5xx Response's body with updated TR signalling failed descriptors or parameters
DeleteTask	request to destroy task	TR with identical general's id, previously received in CreateTask's response	If successful, i.e. after destroying task, shall include: 1) HTTP status code 200 If failed, i.e. not be able to destroy Task, shall include: 1) HTTP status codes 4xx or 5xx 2) Response's body with updated TR signalling failed descriptors or parameters

The included TDD in any response should include one "link" object, including a "ref" with value "self" and a URL according to IETF RFC 3986 indicating the location of the TDD.

7.4 Function discovery APIs

7.4.1 General

Function discovery API is used by the workflow manager and NBMP source for discovery of NBMP functions supported by a platform. As shown in Figure 1, these functions are catalogued in a function repository using NBMP function description (FD).

This subclause specifies the function discovery API operations, queries and responses.

7.4.2 Function discovery queries

A discovery query is used to discover one or more functions in a function repository by the properties described in the query. A query string is used to describe these properties.

The query string shall conform to IETF RFC 3986:2005, Section 3.4. It shall consist of a set of key-value pairs, separated by a single '&' character. In each key-value pair, the key and value shall be separated by a single '=' character.

Any value in a key-value pair in a Function Discovery query string may include wildcard expression syntax as indicated in Table 7.

Table 7 — Wildcard expression characters

Character	Expression
*	matches zero or more characters default character
^	matches the beginning of value
$	matches the end of value

As shown in Table 7, wildcard '*' is the default case, therefore any inquires with 'foo=bar' pair, would return all functions in the repository with their 'foo' parameter value having a substring 'bar'. To find functions with the exact match value of 'bar', the value pair 'foo=^bar$' shall be included in the query string.

Table 8 lists the supported keys in the query string.

Table 8 — Function query's keys

Descriptor and parameter used for matching		
Query keys	**Descriptor**	**Parameter**
id	general	id
name	general	name
description	general	description
brand	general	mpeg-compatibility
keywords	processing	keywords

7.4.3 Function discovery API operations

A function discovery API operation is used by the NBMP source or the workflow manager to discover the available functions in a function repository. The API operations are defined in Table 9.

Table 9 — Function discovery API operations

Operation	Description	Request parameters	Response
DiscoverFunctions	discover a set of functions	Query string shall include the key-value pairs describing the desired properties of the target function. Query string shall be empty, i.e. only '?' added to the end of function Repository's URL when it is required to discover all functions of repository.	If successful, shall include: 1) HTTP status code 200 1) Response's body including a collection of FDDs listing all accessible functions of repository with the matching values per key If failed, shall include: 1) HTTP status codes 4xx or 5xx If the search is performed and no function is found, the response shall be considered successful and an empty collection shall be included in the body.
DiscoverFunctionsInGroup	discover all functions in the function repository that belong to the given function group	Query string shall include the group's id of the group for which the function list is sought	If successful, shall include: 1) HTTP status code 200 2) Response's body including a collection of FDDs listing functions of repository that belong to the given Group id If failed, shall include: 1) HTTP status codes 4xx or 5xx If no function is found in repository, the response shall be considered successful and an empty collection shall be included in the body.
DiscoverGroupsOfFunction	discover all function groups that a given function belongs to	Query string shall include the function's id of the function for which function group information is sought	If successful, shall include: 1) HTTP status code 200 2) Response's body including a list of Group Ids that the given function is a member of If failed, shall include: 1) HTTP status codes 4xx or 5xx If no function is found in repository, the response shall be considered successful and an empty list shall be included in the body.

The discovery operations of Table 11 shall be performed with HTTP GET method.

Each included FDD in any response should include one "link" object, including a "ref" with value "self" and a URL according to IETF RFC 3986 indicating the location of the FDD.

In every included FDD in any response, each function descriptor object (FDO) should include one "link" object, including a "ref" with value "self" and a URL according to IETF RFC 3986 indicating the location of each FDO.

7.5 Supported protocols

All NBMP API operations described shall be implemented with HTTP 1.1.

Table 10 defines HTTP methods which shall be used for each NBMP API operation.

Table 10 — HTTP methods for each API

API	Operation	HTTP method
NBMP workflow API	CreateWorkflow	POST
	UpdateWorkflow	PATCH
	DeleteWorkflow	DELETE
	RetrieveWorkflow	GET
	GetReports	GET
NBMP task API	CreateTask	POST
	UpdateTask	PATCH
	GetTask	GET
	DeleteTask	DELETE
NBMP function discovery API	DiscoverFunctions	GET
	DiscoverFunctionsInGroup	GET
	DiscoverGroupsOfFunction	GET

The HTTP 'Content-Type' header shall be according to Annex D.

8 NBMP descriptors

8.1 Scheme descriptor

8.1.1 General

This descriptor provides a scheme identifier to identify the base scheme used for its descriptors. Table 11 defines this descriptor.

Table 11 — Scheme descriptor

Parameter name	Type	Cardinality
uri	P	1
defined according to the scheme identified by 'uri'	P	1-N

The URN 'urn:mpeg:mpegi:nbmp:2020' is assigned to this document, i.e. any description with this URN as its scheme's uri value, shall conform to this document.

This descriptor may be used inside any other descriptor to carry parameters of a different scheme. These parameters shall be included in the scheme descriptor as shown in Table 11. As shown in Table 11, additional parameter can be added to this descriptor. The name, definition, unit and range of each new parameter is defined by the URI owner.

8.1.2 JSON schema

```
{
  "title": "Scheme Descriptor Schema",
  "type": "object",
  "required": [
    "uri"
  ],
  "properties": {
    "uri": {
      "type": "string",
      "format":"uri"
    }
  }
}
```

8.2 General descriptor

8.2.1 General

This descriptor provides general details about the underlying resource as shown in Table 12.

Table 12 — General descriptor

Parameter name	Type	Cardinality
id	P	1
name	P	1
description	P	1
rank	P	0-1
mpeg-compatibility	P	0-1
published-time	P	0-1
priority	P	0-1
execution-time	P	0-1
input-ports	Array of object	1
output-ports	Array of object	1
is-group	P	0-1
state	P	1

The input-ports and output-ports objects specify the endpoints where the data is communicated from NBMP sources to tasks; and between tasks; and from tasks to NBMP sinks. The concrete protocols and data formats associated with the ports are specified by the input and output descriptors.

Table 13 — input-port and output-port object

Parameter name	Type	Cardinality
port-name	P	1
bind	O	1

The bind object specifies how to associate a port name to a stream, either input or output. For NBMP functions, they provide static information about the port names and their binding data formats and protocols. For NBMP tasks, they provide information about the needs for connections between ports and input and output streams by NBMP workflow manager.

Table 14 — bind object

Parameter name	Type	Cardinality
stream-id	P	0-1
name	P	1
keywords	Array of string	0-1

The Table 14 parameters of the input-ports and output-ports objects are defined in subclause 9.4 and 9.5, respectively.

8.2.2 JSON schema

```
{
  "title": "General Descriptor Schema",
  "type": "object",
  "required": [
    "id", "name", "description",
    "input-ports","output-ports", "state"
    ],
  "properties": {
    "id": {
      "type": "string"
    },
    "name": {
      "type": "string"
    },
    "description": {
      "type": "string"
    },
    "rank": {
      "type": "integer",
      "minimum": 0
```

```
},
"mpeg-compatibility": {
  "type": "string",
  "format": "uri",
  "pattenrProperties":{
    "^urn:mpeg:mpegi:nbmp:v": {"type" : "string"}
  },
  "additionalProperties": false
},
"published-time": {
  "type": "string",
  "format": "date-time"
},
"priority": {
  "type": "number"
},
"input-ports": {
  "type": "array",
  "minItems": 1,
  "uniqueItems": true,
  "items": {
    "type": "object",
    "required": ["port-name", "bind"],
    "properties": {
      "port-name": {
        "type": "string"
      },
      "bind": {
```

```
        "type": "object",
        "required": ["stream_id", "name"],
        "properties": {
          "stream-id": {
            "type": "string"
          },
          "name": {
            "type": "string"
          },
          "keywords": {
            "type": "array",
            "minItems": 1,
            "uniqueItems": true,
            "items": {
              "type": "string"
            }
          }
        }
      }
    }
  },
  "output-ports": {
    "type": "array",
    "minItems": 1,
    "uniqueItems": true,
    "items": {
      "type": "object",
```

```
      "required": ["port-name", "bind"],
      "properties": {
       "port-name": {
         "type": "string"
       },
       "bind": {
         "type": "object",
         "required": ["stream_id", "name"],
         "properties": {
          "stream-id": {
            "type": "string"
          },
          "name": {
            "type": "string"
          },
          "keywords": {
            "type": "array",
            "minItems": 1,
            "uniqueItems": true,
            "items": {
              "type": "string"
            }
          }
        }
      }
    }
  },
```

```
      "is-group": {
        "type": "boolean",
        "default": false
      },
      "state": {
        "type": "string"
      }
    }
}
```

8.3 Input descriptor

8.3.1 General

This descriptor provides the parameters of the underlying resource's inputs as shown in Table 13.

Table 13 — Input descriptor

Name	Type	Cardinality
media-parameters	array of object	0-1*
metadata-parameters	array of object	0-1*
* This descriptor shall contain at least one of the above objects.		

The input descriptor consists of two arrays of objects: one for the media inputs and one for metadata inputs, as shown in Table 14 and Table 15 respectively.

Table 14 — Input media-parameters objects

Name	Type	Cardinality
stream-id	P	1
name	P	1
keywords	P	1
mime-type	P	1
video-format	P	0-1
audio-format	P	0-1
image-format	P	0-1

Name	Type	Cardinality
codec-type	P	0-1
protocol	P	1
throughput	P	0-1
buffersize	P	0-1
caching-server-url	P	1

Table 15 — Input metadata-parameters objects

Name	Type	Cardinality
stream-id	P	1
name	P	1
keywords	P	1
mime-type	P	1
codec-type	P	0-1
protocol	P	1
max-size	P	0-1
min-interval	P	0-1
caching-server-url	P	0-1
scheme-uri	P	0-1

The schema-uri refers to a metadata-dictionary object that consists of a set of parameter-value pairs. The parameters' names, data types and value ranges are defined by the metadata scheme owner.

8.3.2 JSON schema

```
{
  "title": "Input Descriptor Schema",
  "type": "object",
  "anyOf": [
   {"required": ["media-parameters"]},
   {"required": ["metadata-parameters"]}
```

```
  ],
  "properties": {
    "media-parameters": {
      "type": "array",
      "minItems": 0,
      "uniqueItems": true,
      "items": {
        "type": "object",
        "required": [
          "stream-id", "name", "keywords", "mime-type", "data-type", "protocol","caching-server-url"
        ],
        "properties": {
          "stream-id": {
            "type": "string"
          },
          "name": {
            "type": "string"
          },
          "keywords": {
            "type": "array",
            "minItems": 1,
            "uniqueItems": true,
            "items": {
              "type": "string"
            }
          },
          "mime-type": {
            "type": "string"
```

```
      },
      "video-format": {
        "$ref": "#/configuration/parameters"
      },
      "audio-format": {
        "$ref": "#/configuration/parameters"
      },
      "image-format": {
        "$ref": "#/configuration/parameters"
      },
      "protocol" : {
        "type": "string"
      },
      "throughput": {
        "type": "integer",
        "minimum": 0
      },
      "buffersize": {
        "type": "integer",
        "minimum": 0
      },
      "caching-server-url": {
        "type": "string",
        "format": "uri"
      }
    }
  },
  "metadata-parameters": {
```

```
"type": "array",
"items": {
 "type": "object",
 "minItems": 0,
 "uniqueItems": true,
 "required": ["stream-id", "name", "keywords", "mime-type", "protocol"],
 "properties": {
  "stream-id": {
   "type": "string"
  },
  "name": {
   "type": "string"
  },
  "keywords": {
   "type": "array",
   "minItems": 1,
   "uniqueItems": true,
   "items": {
      "type": "string"
   }
  },
  "max-size": {
   "type": "integer",
   "minimum": 0
  },
  "min-interval": {
   "type": "integer",
   "minimum": 0
```

```
            },
            "caching-server-url": {
              "type": "string"
            },
            "mime-type": {
              "type": "string"
            },
            "codec-type": {
              "type": "string"
            },
            "protocol": {
              "type": "string"
            },
            "scheme-uri": {
              "type": "string",
              "format": "uri"
            }
          }
        }
      }
    }
  }
}
```

8.3.3 General

This descriptor provides the parameters of the underlying resource's outputs as shown in Table 16.

Table 16 — Output descriptor

Name	Type	Cardinality
media-parameters	array of object	0-1*
metadata-parameters	array of object	0-1*
* This descriptor shall contain at least one of the above objects.		

The output descriptor consists of two arrays of objects: one for the media outputs and one for metadata outputs, as shown in Table 17 and Table 18 respectively.

Table 17 — Output media-parameters objects

Name	Type	Cardinality
stream-id	P	1
name	P	1
keywords	P	1
mime-type	P	1
video-format	P	0-1
audio-format	P	0-1
image-format	P	0-1
codec-type	P	0-1
protocol	P	1
throughput	P	0-1
buffersize	P	0-1
caching-server-url	P	1

Table 18 — Output metadata parameters objects

Name	Type	Cardinality
stream-id	P	1
name	P	1
keywords	P	1

Name	Type	Cardinality
mime-type	P	1
codec-type	P	0-1
protocol	P	0-1
max-size	P	0-1
min-interval	P	0-1
caching-server-url	P	1
scheme-uri	P	0-1

8.3.4 JSON schema

```
{
  "title": "Output Descriptor Schema",
  "type": "object",
  "anyOf":[
   {"required": ["media-parameters"]},
   {"required": ["metadata-parameters"]}
  ],
  "properties": {
   "media-parameters": {
    "type": "array",
    "minItems": 0,
    "uniqueItems": true,
    "items": {
     "type": "object",
     "required": [
      "stream-id", "name", "keywords", "mime-type", "protocol", "caching-server-url"
     ],
     "properties": {
```

```
"stream-id": {
    "type": "string"
},
"name": {
    "type": "string"
},
"keywords": {
  "type": "array",
  "minItems": 1,
  "uniqueItems": true,
  "items": {
    "type": "string"
  }
},
"mime-type": {
  "type": "string"
},
"video-format": {
  "$ref": "#/configuration/parameters"
},
"audio-format": {
  "$ref": "#/configuration/parameters"
},
"image-format": {
  "$ref": "#/configuration/parameters"
},
"codec-type": {
  "type": "string"
```

```
        },
        "protocol" : {
          "type": "string"
        },
        "throughput": {
          "type": "integer",
          "minimum": 0
        },
        "buffersize": {
          "type": "integer",
          "minimum": 0
        },
        "caching-server-url": {
          "type": "string",
          "format": "uri"
        }
      }
    }
  },
  "metadata-parameters": {
    "type": "array",
    "items": {
      "type": "object",
      "minItems": 0,
      "uniqueItems": true,
      "required": ["stream-id", "name", "keywords", "mime-type", "protocol"],
      "properties": {
        "stream-id": {
```

```
      "type": "string"
    },
    "name": {
      "type": "string"
    },
    "keywords": {
      "type": "array",
      "minItems": 1,
      "uniqueItems": true,
      "items": {
        "type": "string"
      }
    },
    "max-size": {
      "type": "integer",
      "minimum": 0
    },
    "min-interval": {
      "type": "integer",
      "minimum": 0
    },
    "caching-server-url": {
      "type": "string"
    },
    "mime-type": {
      "type": "string"
    },
    "codec-type": {
```

```
          "type": "string"
        },
        "protocol": {
          "type": "string"
        },
        "scheme-uri": {
          "type": "string",
          "format": "uri"
        }
      }
    }
  }
}
```


8.4 Processing descriptor

8.4.1 General

This descriptor provides high-level details about the requested media processing of a workflow by listing the set of tasks to be performed on the input media data. Table 19 defines the parameters used in this descriptor.

Table 19 — Processing descriptor

Name	Type	Cardinality
keywords	P	1
image	array of object	1
start-time	P	0-1
connection-map	array of object	0-1
function-restrictions	array of object	0-1

The image object is defined in Table 20.

Table 20— Image array element

Name	Type	Cardinality
is-dynamic	P	0-1
url	P	1
static-image-info	O	0-1
dynamic-image-info	O	0-1

The image object has a static-info object if the function's image is static, i.e. the image is pre-generated. Table 21 defines the parameters of this object.

Table 21— static-image-info object

Name	Type	Cardinality
os	P	1
version	P	1
architecture	P	1
environment	P	1
patch-url	P	0-1
patch-script	O	0-1

In Table 21, the format and process of the patch-script is defined by patch-url.

The image object has a dynamic-info object if the Function's image is dynamically built, i.e. the service providing the image builds it on-request based on the information provided to the service. Table 22 defines the parameters of this object.

Table 22 — dynamic-image-info object

Name	Type	Cardinality
scheme	P	1
information	O	1

In Table 22, the format of the information object and its values are defined by scheme. The information is used to request a dynamic build of the image.

The array of connection-map object provides a description of the media workflow DAG, i.e. the connection information between different tasks in the graph. Each element in this array, which represents an edge in the DAG, is defined in Table 23.

Table 23 — connection-map array element

Name	Description	Type	Cardinality
from	specifies task/function's id and port names from which the connection is	O	1
to	specifies task/function's id and port names to which the connection is	O	1
flowcontrol-parameters	contains flow control parameters for the connection The elements of this objects shall be described using flowcontrol-requirement of Table 27.	O	0-1
co-located	Specifies the deployment of the 2 connected tasks When the value is True, the 2 tasks shall be deployed into the same MPE, Otherwise, the deployment is determined by the workflow manager based on available resources. The default is 'false'.	P	0-1
other-parameters	contains any other properties or parameters defined for the DAG edge, e.g. references to the requirement descriptor The elements of this objects shall be described using generic parameter representation of subclause 9.20.1.1.	O	0-1

Objects from and to are defined in Table 24.

Table 24 – "from" and "to" objects

Name	Description	Type	Cardinality
id	function identifier	P	1
instance	instance identifier	P	1
port-name	function logic port name It shall be output port name for "from" and input port name for "to" in connection-map object.	P	1
input-restrictions	restrictions to the input descriptor parameters as defined in Table 13 This object shall not be present in "from" objects.	O	0-1
output-restrictions	restrictions to the output descriptor parameters as defined in Table 16 This object shall not be present in "to" objects.	O	0-1

In Table 24, input-restrictions is an input descriptor's construct (i.e., identical to input descriptor, but with the object's name input-restrictions) which only includes allowed values of input parameters when the function is used in one group. Similarly, output-restrictions is an output descriptor's construct which only includes allowed values of output parameters when the function is used in one group.

The entity function-restrictions, as shown in Table 19, is an array. Each element of this array describes the additional restriction for one function instance used in a function group, as shown in Table 25.

Table 25 — function-restrictions array element

Name	Description	Type	Cardinality
instance	instance identifier	P	1
general	general descriptor restriction of this instance as defined in Table 12	O	0-1
processing	general descriptor restriction of this instance as defined in Table 19	O	0-1
requirement	requirement descriptor restriction of this instance as defined in Table 26	O	0-1
configuration	configuration descriptor restriction of this instance as defined in Table 32	O	0-1
client-assistance	Client-Assistance descriptor restriction of this instance as defined in Table 34	O	0-1
monitoring	monitoring descriptor restriction of this instance as defined in Table 39	O	0-1

In Table 25, the descriptors define allowed values for the function instance used in the function group.

8.4.2 JSON schema

```
{
  "title": "Processing Descriptor Schema",
  "type": "object",
  "required": [
    "keywords",
    "image"
  ],
  "properties": {
    "keywords": {
      "type": "array",
```

```
    "minItems": 1,
    "uniqueItems": true,
    "items": {
      "type": "string"
    }
  },
  "image": {
    "type": "array",
    "minItems": 1,
    "uniqueItems": true,
    "items": {
      "type": "object",
      "required" : [ "url"],
      "properties": {
        "is-dynamic": {
          "type": "boolean",
          "default": false
        },
        "url": {
          "type": "string",
          "format": "url"
        },
        "static-image-info":{
          "type": "object",
          "required": [
            "os", "version", "architecture", "environment"
          ],
          "properties": {
```

```
            "os":{ "type": "string"},
            "version":{ "type": "string"},
            "architecture":{ "type": "string"},
            "environment":{ "type": "string"},
            "patch-url":{
              "type": "string",
              "format": "url"
            },
            "patch-script":{
              "type": "string",
              "format": "url"
            }
          }
        },
        "dynamic-image-info":{
          "type": "object",
          "required": ["scheme", "information"],
          "properties": {
            "pscheme":{
              "type": "string",
              "format": "url"
            },
            "information":{ "type": "object"}
          }
        }
      }
    }
  },
```

```
"start-time": {
  "type": "string",
  "format": "date-time"
},
"connection-map": {
  "type": "array",
  "minItems": 1,
  "uniqueItems": true,
  "items": [
   {
    "type": "object",
    "required": [
     "from", "to"
    ],
    "properties": {
     "from":{
      "type": "object",
      "required": [
       "id",
       "instance",
       "port-name"
      ],
      "properties": {
       "id" : {
        "type": "string"
       },
       "instance": {
        "type": "string"
```

```
      },
      "port-name": {
        "type": "string"
      },
      "output-restrictions": {
        "$ref": "#/output"
      }
    }
  },
  "to": {
    "type": "object",
    "required": [
      "id",
      "instance",
      "port-name"
    ],
    "properties": {
      "id": {
        "type": "string"
      },
      "instance": {
        "type": "string"
      },
      "port-name": {
        "type": "string"
      },
      "input-restrictions": {
        "$ref": "#/input"
```

```
            }
          }
        },
        "flowcontrol": {
          "$ref": "#/requirement/flowcontrol"
        },
        "co-located": {
          "type": "boolean",
          "default": false
        },
        "other-parameters": {
          "$ref": "#/configuration/parameters"
        }
      }
    }
  ]
},
"function-restrictions": {
  "type": "array",
  "uniqueItems": true,
  "items": [
   {
     "type": "object",
     "required": [
       "instance"
     ],
     "properties": {
       "instance": {
```

```
            "type": "string"
          },
          "general": {
            "$ref": "#/general"
          },
          "processing": {
            "$ref": "#/processing"
          },
          "requirements": {
            "$ref": "#/requirements"
          },
          "configuration": {
            "$ref": "#/configuration"
          },
          "client-assistant": {
            "$ref": "#/client-assistant"
          },
          "monitoring": {
            "$ref": "#/monitoring"
          }
        }
      }
    ]
  }
 }
}
```

8.5 Requirements descriptor

8.5.1 General

This descriptor provides requirements parameters that can be configured for the underlying resource. Table 26 defines this descriptor.

Table 26 — Requirements descriptor

Parameter Name	Description	Type	Cardinality
flowcontrol-requirements	flowcontrol requirements for the resource described in subclause 9.7.1	O	0-1
hardware-requirements	hardware requirements for the resource described in subclause 9.7.2	O	0-1
security-requirements	detailed security requirements during content ingestion and content distribution for the resource described in subclause 9.7.3	O	0-1
workflow- task-requirements	detailed requirements for optimizing the workflow and tasks by the workflow manager described in subclause 9.7.4	O	0-1
resource-estimators	equations for estimating the resources by comparing to a baseline configuration described in subclause 9.7.5	O	0-1

As shown in Table 26, requirement parameters are grouped in different objects, each addressing a different class of requirements. Table 27, Table 28, Table 29 and Table 30 define the parameters of each objects.

Table 27 — Flowcontrol parameters

Name	Type	Cardinality
typical-delay	P	0-1
min-delay	P	0-1
max-delay	P	0-1
min-throughput	P	0-1
max-throughput	P	0-1
averaging-window	P	0-1

Table 28 — Hardware parameters

Name	Type	Cardinality
vcpu	P	0-1
vgpu	P	0-1
ram	P	0-1
disk	P	0-1
placement	P	0-1

Table 29 — Security parameters

Name	Type	Cardinality
tls	P	0-1
ipsec	P	0-1
cenc	P	0-1

Table 30 —Workflow/task requirement parameters

Name	Type	Cardinality
function-fusible	P	0-1
function-enhancable	P	0-1
execution-mode	P	0-1

Table 31 —Resource estimator parameters

Name	Type	Cardinality
default-values	array of object	1
computational-estimator	string	0-1*
memory-estimator	string	0-1*
bandwidth-estimator	string	0-1*
* This descriptor shall contain at least one of these entries.		

In Table 31, the Default-Values object contains the list of default name-value pairs of parameters used as the reference for estimators. These values are the values that a cloud platform has a reference hardware platform for it to run in real-time.

In Table 31, the next 3 entries contain estimator formulas E_h, E_m, and E_b for the function:

$$\{h, m\ b\} = \{ E_h(\{p\},\{c\}).h_0,\ E_m(\{p\},\{c\}).m_0,\ E_b(\{p\},\{c\}).b_0\}$$

where:

the set $\{p\}$ are input and output parameters included in input and output descriptors;

the set $\{c\}$ are configuration parameters included in configuration descriptor;

the sets $\{p_0\}$ and $\{c_0\}$ are same sets for default inputs and outputs, as well as parameters for the default configuration.

These formulas shall be described according to ISO/IEC 9899.

Parameters of Table 31 shall have the identical names, formats, and semantics used in configuration descriptor, input and output descriptors.

8.5.2 JSON schema

```
{
  "title": "Requirements Descriptor Schema",
  "type": "object",
  "properties": {
    "flowcontrol": {
      "type": "object",
      "minProperties" : 1,
      "properties": {
        "typical-delay": {
        "type": "integer",
        "minimum": 0
        },
        "min-delay": {
          "type": "integer",
          "minimum": 0
        },
        "max-delay": {
          "type": "integer",
```

```
        "minimum": 0
      },
      "typical-throughput": {
        "type": "integer",
        "minimum": 0
      },
      "min-throughput": {
        "type": "integer",
        "minimum": 0
      },
      "max-throughput": {
        "type": "integer",
        "minimum": 0
      },
      "averaging-window": {
        "type": "integer",
        "minimum": 0
      }
    }
  },
  "hardware": {
    "type": "object",
    "minProperties" : 1,
    "properties": {
      "vcpu": {
        "type": "integer",
        "minimum": 0
      },
```

```
      "vgpu": {
        "type": "integer",
        "minimum": 0
      },
      "ram": {
        "type": "integer",
        "minimum": 0
      },
      "disk": {
        "type": "integer",
        "minimum": 0
      },
      "placement": {
        "type": "string",
        "pattern": "(^[A-Z]{2}$)|(^[A-Z]{2}-.*)"
      }
    }
  },
  "security": {
    "type": "object",
    "minProperties" : 1,
    "properties": {
      "tls": {
        "type": "boolean",
        "default": false
      },
      "ipsec": {
        "type": "boolean",
```

```
        "default": false
      },
      "cenc": {
        "type": "boolean",
        "default": false
      }
    }
  },
  "workflow-task": {
    "type": "object",
    "minProperties" : 1,
    "properties": {
      "execution-mode": {
        "type": "string",
        "default": "streaming"
      },
      "function-fusible": {
        "type": "boolean",
        "default": false
      },
      "function-enhancable": {
        "type": "boolean",
        "default": false
      }
    }
  },
  "resource-estimators": {
    "type": "object",
```

```
      "required": ["default-values"],
      "properties": {
        "default-values": {
          "type": "array",
          "minItems": 1,
          "items": {
            "name" : {"type": "string"},
            "value": {"type": "string"}
          }
        },
        "computational-estimator": {
          "type": "string"
        },
        "memory-estimator": {
          "type": "string"
        },
        "bandwidth-estimator": {
          "type": "string"
        }
      }
    }
  }
}
```


8.6 Configuration descriptor

8.6.1 General

This descriptor provides configuration information for the underlying resource as shown in Table 32.

Table 32 — Configuration descriptor

Parameter name	Description	Data type	Cardinality
parameters	configuration details of parameters required for resource	array of object	1

8.6.2 JSON schema

```
{
  "title": "Configuration Descriptor Schema",
  "type": "object",
  "required": [ "parameters" ],
  "properties": {
    "parameters": {
      "type": "array",
      "uniqueItems": true,
      "items": {
        "type": "object",
        "required": [
          "name", "id", "type"
        ],
        "properties": {
          "name": {
            "type": "string"
          },
          "id": {
            "type": "integer"
          },
          "type": {
            "type": "string",
            "enum": [
```

```
      "simple", "enum", "range", "class"
    ]
  },
  "datatype": {
    "type": "string",
    "enum": [
      "boolean", "integer", "unsigned-integer", "float", "string", "object", "other"
    ]
  },
  "default": {
    "type": "object"
  },
  "restrictions": {
    "type": "object",
    "properties": {
      "enum-values": {
        "type": "array",
        "uniqueItems": true,
        "items": {
          "type": "string"
        }
      },
      "int-range": {
        "type": "object",
        "properties": {
          "min-value": {
            "type": "integer"
          },
```

```
          "max-value": {
            "type": "integer"
          },
          "increment": {
            "type": "integer"
          }
        }
      },
      "float-range": {
        "type": "object",
        "properties": {
          "min-value": {
            "type": "number"
          },
          "max-value": {
            "type": "number"
          },
          "increment": {
            "type": "number"
          }
        }
      }
    }
  },
  "conditions": {
    "type": "array",
    "uniqueItems": true,
    "description": "List of configurarion ids",
```

```
            "items": {
              "type": "integer"
            }
          },
          "exclusion": {
            "type": "array",
            "description": "List of configurarion ids",
            "uniqueItems": true,
            "items": {
              "type": "integer"
            }
          }
        }
      }
    }
  }
}
```


8.7 Startup-delay descriptor

8.7.1 General

This descriptor provides information for a delayed startup of the underlying resource and is shown in Table 33.

Table 33 — Startup-delay descriptor

Parameter-name	Type	Cardinality
startup-delay	P	1

8.7.2 JSON schema

```
{
   "title": "Startup-delay Descriptor Schema",
   "type": "object",
   "required": ["startup-delay"],
   "properties": {
    "startup-delay": {
      "type": "integer",
      "minimum": 0
    }
   }
}
```

8.8 Client-Assistance descriptor

8.8.1 General

This descriptor provides client assistance information for the underlying resource, as shown in Table 34.

Table 34 — Client-Assistance descriptor

Name	Type	Cardinality
client-assistance-flag	P	1
measurement-collection-list	O	0-1
source-assistance-information	O	0-1

Table 35 defines the two objects used in Client-Assistant descriptor.

Table 35 — Client-assistance objects

Object name	Description
measurement-collection-list	object representing the list of measurements to be collected Each element of the list represents a measurement to be collected. Following is a sample list of measurements to be supported: — viewPortCollection — deviceCapabilityCollection — userPreferencesCollection The object may include the frequency of collection for each parameter. The elements of this objects shall be described using parameter schema of subclause 9.22.1
source-assistance-information	list of objects where each object represents different type information from NBMP source The type of assistance information that is provided by the source is described in subclause 6.3. The elements of this objects shall be described using parameter schema of subclause 9.22.1

8.8.2 JSON schema

```
{
    "title": "Client-Assistance Descriptor Schema",
    "type": "object",
    "required": [
      "client-assistance-flag"
    ],
    "properties": {
      "client-assistance-flag": {
        "type": "boolean",
        "default": false
      },
      "measurement-collection-list": {
        "type": "object"
      },
```

```
      "source-assistance-information ": {
        "type": " object"
      }
    }
}
```

8.9 Failover descriptor

8.9.1 General

This descriptor provides information in case of failover of the underlying resource and is defined in Table 36.

Table 36 — Failover descriptor

Name	Type	Cardinality
failover-mode	P	1
failover-delay	P	1
backup-deployment-url	P	0-1
persistence-url	P	0-1
persistence-interval	P	0-1

8.9.2 JSON schema

```
{
  "title": "Failover Descriptor Schema",
  "type": "object",
  "required": [
    "failover-mode", "failover-delay"
  ],
  "properties": {
    "failover-mode": {
      "type": "string",
      "enum:": ["restart-immediately", "restart-with-delay", "continue-with-last-good-state", "execute-backup-deployment", "exist"]
```

```
    },
    "failover-delay": {
      "type": "integer",
      "minimum": 0
    },
    "backup-deployment-url": {
      "type": "string",
      "format": "url"
    },
    "persistence-url": {
      "type": "string",
      "format": "url"
    },
    "persistence-interval": {
      "type": "integer",
      "minimum": 0
    }
  }
}
```


8.10 Events descriptor

8.10.1 General

This descriptor provides events for the underlying resource as shown in Table 37. For a function, this descriptor describes the events that can be monitored, reported or used in notification in the task or workflow implementing this function.

Table 37— Events descriptor

Name	Type	Cardinality
events	array of object	1

The array events describes a list function events. Each element of this array has a name, and a definition. Similarly, the event of a monitoring/report/notification descriptor describes the occurred event.

8.10.2 JSON schema

```
{
  "title": "Events Descriptor Schema",
  "type": "array",
  "minItems": 1,
  "uniqueItems": true,
  "items" : {
    "properties": {
      "name": {"type": "string"},
      "definition": {"type": "string"},
      "url": {
        "type": "string",
        "format": "uri"
      }
    }
  }
}
```


8.11 Variables descriptor

8.11.1 General

This descriptor provides variables for the underlying resource as shown in Table 38. For a function, this descriptor describes the variables that can be monitored and/or reported in the function.

Table 38 — Variables descriptor

Name	Type	Cardinality
variables	array of object	1

Each element of this array shall have a name, a definition, a value, a unit, a parameter type and possibly a range. Each element may also consist of more variables.

8.11.2 JSON schema

```
{
  "title": "Variables Descriptor Schema",
  "properties" : {
   "type": "array",
   "minItems": 1,
   "uniqueItems": true,
   "items" : {
    "required": ["name", "definition", "unit", "var-type", "value"],
    "properties": {
     "name": {"type": "string"},
     "definition": {"type": "string"},
     "unit": {"type": "string"},
     "var-type": {
      "type" : "string",
      "enum" : ["string", "interger", "float", "boolean", "number"]
     },
     "value": {"type": "string"},
     "min" :{"type": "integer"},
     "max" :{"type": "integer"},
     "url": {
      "type": "string",
      "format": "url"
```

```
        },
        "children": {
          "type" : "array",
          "uniqueItems": true,
          "items" : { "$ref": "#/variables" },
          "default" : []
        }
      }
    }
  }
}
```

8.12 Monitoring descriptor

8.12.1 General

This descriptor provides monitoring information for the underlying resource as shown in Table 39.

Table 39 — Monitoring descriptor

Name	Type	Cardinality
events	array of event	0-1*
variables	array of variable	0-1*
system-events	array of event	0-1*
system-variables	array of variable	0-1*
*This descriptor shall contain at least one object.		

8.12.2 JSON schema

```
{
  "title": "Monitoring Descriptor Schema",
  "type": "object",
  "minProperties": 1,
  "properties": {
```

```
    "event": { "$ref": "#/events" },
    "variable": { "$ref": "#/variables" },
    "system-events": {
      "type": "array",
      "minItems": 1,
      "uniqueItems": true,
      "items": {
        "type": "object"
      }
    },
    "system-variables": {
      "type": "array",
      "minItems": 1,
      "uniqueItems": true,
      "items": {
        "type": "object"
      }
    }
  }
}
```


8.13 Reporting descriptor

8.13.1 General

This descriptor provides reporting information for the underlying resource as shown in Table 40.

Table 40 — Reporting descriptor

Parameter name	Type	Cardinality
events	array of event	0-1*
variables	array of variable	0-1*
system-events	array of event	0-1*
system-variables	array of variable	0-1*
report-type	P	1
reporting-interval	P	1
report-start-time	P	1
url	P	1
delivery-method	P	1
*This descriptor shall contain at least one of these objects.		

8.13.2 JSON schema

```
{
  "title": "Reporting Descriptor Schema",
  "type": "object",
  "required": [ "reporting-type", "reporting-interval", "report-start-time", "url", "delivery-method"
  ],
  "properties": {
    "event": { "$ref": "#/events" },
    "variable": { "$ref": "#/variables" },
    "system-events": {
      "type": "array",
      "minItems": 1,
      "uniqueItems": true,
      "items": {
        "type": "object"
      }
```

```
},
"system-variables": {
  "type": "array",
  "minItems": 1,
  "uniqueItems": true,
  "items": {
    "type": "object"
  }
},
"report-type": {
  "type": "string"
},
"reporting-interval": {
  "type": "integer",
  "minimum": 0
},
"report-start-time": {
  "type": "string",
  "format": "date-time"
},
"url": {
  "type": "string",
  "format": "url"
},
"delivery-method": {
  "type": "string",
  "default": "HTTP POST"
}
```

```
  }
}
```

8.14 Notification descriptor

8.14.1 General

This descriptor provides notification information for the underlying resource as shown in Table 41.

Table 41 — Notification descriptor

Parameter name	Type	Cardinality
events	array of event	0-1*
variables	array of variable	0-1*
system-events	array of event	0-1*
system-variables	array of variable	0-1*
notification-time	P	1
severity-level	P	1
notification-type	array of parameter	1
urls	P	1
notification-interval	P	0-1
*This descriptor shall contain at least one of these objects.		

8.14.2 JSON schema

```
{
  "title": "Notifications Descriptor Schema",
  "type": "object",
  "required": [
    "notification-time", "severity-level", "notification-type","urls"
  ],
  "properties": {
    "event": { "$ref": "#/events" },
    "variable": { "$ref": "#/variables" },
```

```
"system-events": {
  "type": "array",
  "minItems": 1,
  "uniqueItems": true,
  "items": {
    "type": "object"
  }
},
"system-variables": {
  "type": "array",
  "minItems": 1,
  "uniqueItems": true,
  "items": {
    "type": "object"
  }
},
"notification-time": {
  "type": "string",
  "format": "date-time"
},
"severity-level": {
  "type": "string"
},
"notification-type": {
  "type": "array",
  "minItems": 1,
  "uniqueItems": true,
  "items": {
```

```
        "type": "string",
        "enum": ["congestion", "application", "system"]
      }
    },
    "urls": {
      "type": "array",
      "minItems": 1,
      "uniqueItems": true,
      "items": {
        "type": "string",
        "format": "url"
      }
    },
    "notification-interval": {
      "type": "integer",
      "minimum": 0,
      "default": 0
    }
  }
}
```


8.15 Assertion descriptor

8.15.1 General

This descriptor provides assertion information for validating the underlying resource as shown in Table 42.

Table 42 — Assertion descriptor

Name	Type	Cardinality
min-priority	P	1
min-priority-action	P	1
support-verification	P	0-1
verification-acknowledgement	P	0-1
certificate	P	0-1
assertion	array of object	1

Possible values of the min-priority-action parameter described in Table 42 may be any value of the action parameter defined in value-predicate of assertion object described in Table 43.

The assertion object is a list of name value predicate pairs (NVPs). Each NVP pair consists of name and value-predicate as described in Table 43.

Table 43 — Assertion object

Name	Type	Cardinality
name	P	1
value-predicate	O	1

The value-predicate object represents the assertion predicate to evaluate the parameter and consists of the information in Table 44.

Table 44 — Assertion value predicate object

Name	Type	Cardinality
evaluation-condition	P	1
check-value	P	1
aggregation	P	1
offset	P	0-1
priority	P	1
action	P	1
action-parameters	P	0-N

NOTE A report on assertion failures can be provided to the NBMP source if requested.

Following is a list of assertions and their parameters which can be included in the assertion list of NVPs sent in the workflow description document from the media source as shown in Table 45.

Table 45 — Assertion list of NVPs

Type of assertions	Description	List of parameters
quality-assertions	provides description to create assertions that check the quality of media processing	list of parameters in the "QoS requirements" descriptor described in subclause 9.7
computational-assertions	provides description to create assertions that check the computational requirements of media processing	list of parameters in the "hardware requirements" section of the requirement descriptor described in subclause 9.7
input-assertions	provides description to create assertions that check whether the workflow input is of certain kind	list of media data, metadata, and other data parameters in the input descriptor described in subclause 9.4
output-assertions	provides description to create assertions that check whether the workflow output is of certain kind	list of media data, metadata, and other data parameters in the output descriptor described in subclause 9.5

8.15.2 JSON schema

```
{
  "title": "Assertion Descriptor Schema",
  "type": "object",
  "required": [
    "min-priority", "min-priority-action", "assertion"
  ],
  "properties": {
   "min-priority":{
     "type": "integer",
     "minimum": 0
   },
   "min-priority-action":{
     "type": "string"
   },
   "support-verification": {
     "type": "boolean",
```

```
    "default": false
  },
  "verification-acknowledgement": {
    "type": "string"
  },
  "certificate": {
    "type": "string"
  },
  "assertion": {
    "type": "array",
    "minItems": 1,
    "uniqueItems": true,
    "items": {
      "type": "object",
      "required": [
        "name", "value-predicate"
      ],
      "properties": {
        "name" : {
          "type": "string"
        },
        "value-predicate": {
          "type": "object",
          "required": [
            "evaluation-condition", "check-value", "aggregation", "priority", "action"
          ],
          "properties": {
            "evaluation-condition": {
```

```
      "type": "string",
      "enum": ["quality", "computational", "input", "output"]
    },
    "check-value": {
      "type": "object"
    },
    "aggregation": {
      "type": "string",
      "enum": ["sum", "min", "max", "avg"]
    },
    "offset": {
      "type": "string"
    },
    "priority": {
      "type": "integer",
      "minimum": 0
    },
    "action": {
      "type": "string",
      "enum": ["rebuild", "restart", "wait"]
    },
    "action-parameters": {
      "type": "array",
      "minItems": 1,
      "uniqueItems": true,
      "items": { "type": "string"}
    }
  }
```

```
        }
       }
      }
     }
    }
}
```

8.16 Request Descriptor

8.16.1 General

This descriptor provides information for the request sent by a task. It can be used for identifying the repeated and identical requests, the priority of request compared to other received requests, and the requesting task. Table 46 defines this descriptor.

Table 46 — Request descriptor

Parameter name	Type	Cardinality
request-id	P	1
priority	P	0-1
task-id	P	1

This descriptor shall be included in any request sent by a task.

8.16.2 JSON schema

```
{
  "title": "Request Descriptor Schema",
  "type": "object",
  "required": ["request-id", "task-id"],
  "properties": {
    "request-id": {
      "type": "integer",
      "minimum": 0
    },
    "priority": {
```

```
      "type": "integer",
      "minimum": 0
    },
    "task-id": {
      "type": "string"
    }
  }
}
```

8.17 Acknowledge descriptor

8.17.1 General

This descriptor indicated whether a description or a descriptor is fulfilled during the processing of a request. Table 47 defines this descriptor.

Table 47 — Acknowledge descriptor

Parameter name	Type	Cardinality
status	P	1
unsupported	P	0-1
failed	P	0-1
partial	P	0-1

8.17.2 JSON schema

```
{
  "title": "Acknowledge Descriptor Schema",
  "type": "object",
  "required": ["status"],
  "properties": {
    "status": {
      "type": "string",
      "enum": ["fulfilled", "failed", "not-supported", "partially-fulfilled"]
```

```
    },
    "unsupported": {
      "type": "array",
      "minItems": 1,
      "uniqueItems": true,
      "items": {"type": "string"}
    },
    "failed": {
      "type": "array",
      "minItems": 1,
      "uniqueItems": true,
      "items": {"type": "string"}
    },
    "partial": {
      "type": "array",
      "minItems": 1,
      "uniqueItems": true,
      "items": {"type": "string"}
    }
  }
}
```


8.18 Repository descriptor

8.18.1 General

This descriptor provides the list of function repositories to be used in creating a workflow. Table 48 and Table 49 define this descriptor.

Table 48 — Repository descriptor

Parameter name	Type	Cardinality
mode	P	0-1
location	array of object	1

Table 49 — Location array element

Parameter name	Type	Cardinality
url	P	1
name	P	1
description	P	1

8.18.2 JSON schema

```
{
  "title": "Repository Descriptor schema",
  "type": "object",
  "required": ["location"],
  "properties": {
    "mode": {
      "type": "string",
      "enum": ["strict", "preferred", "available"],
      "default": "available"
    },
    "location": {
      "type": "array",
      "minItems": 1,
      "uniqueItems": true,
      "items": {
        "type": "object",
        "required": ["url", "name", "description"],
```

```
        "properties": {
          "url": {
            "type": "string",
            "format": "url"
          },
          "name": {
            "type": "string"
          },
          "description": {
            "type": "string"
          }
        }
      }
    }
  }
}
```


8.19 Security descriptor

8.19.1 General

Table 50 gives the list of parameters defined for the array element of security descriptor. These should have zero or multiple authentication descriptors, depending on the concrete use cases, inputs, and outputs.

Table 50 — Security Descriptor's element

Parameter name	Type	Cardinality
name	P	1
scope	P	0-1
authentication-method	P	1
authority-url	P	0-1
certificate	P	0-1

Parameter name	Type	Cardinality
auth-token	P	0-1
client-grant	P	0-1
auth-token-expires	P	0-1
auth-token-renew	P	0-1
auth-token-rotation	P	0-1

8.19.1 JSON schema

```
{
  "title": "Security Descriptor Schema",
  "type": "object",
  "required": ["name", "authentication-method"],
  "properties": {
    "name": {"type": "string"},
    "scope": {
      "type": "string",
      "enum":["data", "function", "task"],
      "default": "data"
    },
    "authentication-method": {"type": "string"},
    "authority-url": {
      "type": "string",
      "format": "url"
    },
    "certificate": {"type": "string"},
    "auth-token": {"type": "string"},
    "client-grants": {"type": "string"},
    "auth-token-expires": {
      "type": "string",
```

```
      "format": "date-time"
    },
    "auth-token-renew": {"type ": "string"},
    "auth-token-rotation": {
      "type": "boolean",
      "default": false
    }
  }
}
```

8.20 Step descriptor

8.20.1 General

This descriptor provides information for stateful and stateless step operation of the underlying resource and is shown in Table 51.

Table 51 – Step descriptor

Parameter	Type	Cardinality
direct	P	0-1
segment-duration	P	0-1
operating-units	P	0-1

A resource with step descriptor shall include one metadata for each input for receiving the sequence number and/or start/duration of each input instance. Each input instance has a duration equal to "segment-duration".

8.20.2 JSON schema

```
{
  "title": "Step Descriptor Schema",
  "type": "object",
  "minProperties": 1,
  "properties": {
    "direct": {
```

```
      "type": "string",
      "enum":["stream", "stateful", "stateless"],
      "default": "stream"
    },
    "segment-duration": {
      "type": "integer",
      "minimum": 0
    },
    "operating-units": {
      "type": "integer",
      "minimum": 0
    }
  }
}
```


9 NBMP parameters

9.1 General

This clause describes the NBMP parameters which are defined by this document. NBMP parameters are used to express the characteristics of an NBMP workflow or its components, by including them and their values in NBMP descriptors.

Any descriptors may be extended to include new parameters. Any new parameter's name shall be a valid URI according to IETF RFC 3986. The description, data type, range and cardinality of the parameter is specified by the URI scheme used as its name.

9.2 Scheme descriptor parameters

Table 52 defines parameters, which are used as part of the scheme descriptor.

Table 52 — Scheme parameters

Name	Definition	Unit	Type	Valid range
uri	identifies the scheme for this document It shall be a valid URI according to IETF RFC 3986.	N/A	string	N/A
defined according to the scheme identified by uri	Defined according to the scheme identified by uri	N/A	string	N/A

9.3 General descriptor parameters

Table 53 defines parameters, which are used as part of the general descriptor.

Table 53 — General parameters

Name	Definition	Unit	Type	Valid range
id	unique string in the scope of repository/workflow of the resource	N/A	string	N/A
name	name for identifying the resource	N/A	string	N/A
description	a human-readable description for the resource	N/A	string	N/A
rank	rank of function/function group among functions with the same functionality A higher number means a higher rank.	N/A	number	unsigned integer
mpeg-compatibility	URN indicating the compatibility with a reference function/function group defined in Annexes A and B	URI	string	N/A
published-time	date and time of publication of this resource	As defined by IETF RFC 3339:2002, section 5.6	string	N/A
priority	priority information for the resource	N/A	number	unsigned integer
port-name	unique string among all port-names of this resource defining the logic name for input or output	N/A	string	N/A
is-group	value ‘true’ indicates containing descriptor describes a function group or task workflow If the value is ‘true’, a connection-map object shall exist in this description. The default value is ‘false’.	N/A	boolean	N/A

Name	Definition	Unit	Type	Valid range
state	current state of the resource in its lifecycle The value of this parameter shall be one of the followings in a NBMP operation's response: — null — instantiated — idle — running — in-error — destroyed If this parameter is included in a request from the NBMP source or workflow manager, the value of this parameter shall be one of the followings (subject to constraints in subclause 7.27.3): — instantiated — idle	N/A	string	N/A

9.4 Input descriptor parameters

Table 54 defines input parameters, which are used as part of input descriptor.

Table 54 — Input parameters

Name	Definition	Unit	Type	Valid range
stream-id	unique identifier, with the scope of function or task or workflow, to identify the media or metadata stream For functions, it is defined in the function descriptor. For tasks, it is assigned by the workflow manager. For workflows, it is assigned by the NBMP source.	N/A	string	N/A
name	string name assigned to this input For functions, it is defined in the function descriptor. For tasks, it is assigned by the workflow manager. For workflows, it is assigned by the NBMP source.	N/A	string	N/A
keywords	list of keywords describing this input properties The keyword should be human-readable.	N/A	array of string	N/A
mime-type	MIME media type of media or metadata in IANA registry	N/A	string	N/A

Name	Definition	Unit	Type	Valid range
video-format	format of video The parameter list is defined using generic parameter representation of subclause 9.22.1.	N/A	string	N/A
audio-format	format of audio The parameter list is defined using generic parameter representation of subclause 9.22.1.	N/A	string	N/A
image-format	format of image The parameter list is defined using generic parameter representation of subclause 9.22.1.	N/A	string	N/A
codec-type	'codecs' and 'profiles' parameters, as defined in IETF RFC 6381	N/A	string	N/A
protocol	protocol for delivery of or access to media including protocol parameters such as port number(s) Ingest protocol for timed metadata including protocol parameters such as the port number(s). Example: HTTP. When the workflow manager receives this information, it takes the responsibility of returning back with the protocol endpoint information of the appropriate media processing entity to the media source so the media source can ingest metadata using that protocol. NOTE This is only applicable for media and timed metadata.	N/A	string	N/A
throughput	maximum accepted throughput by this resource	bits/second	number	unsigned integer
buffer-size	minimum input buffer size	bytes	number	unsigned integer
caching-server-url	URL (according to IETF RFC 3986) of the server where the media will be sent from or the location from where the media can be fetched from NOTE When this parameter is missing for a workflow, the workflow manager can assign origination information of the appropriate media processing entity to the media source so the media source can ingest media using that protocol.	N/A	string	N/A

Name	Definition	Unit	Type	Valid range
max-size	maximum size of metadata in each fetch or push accepted by this input	bytes	number	unsigned integer
min-interval	minimum interval between two fetch or push accepted by this input	milliseconds	number	unsigned integer
scheme-uri	URL (according to IETF RFC 3986) or scheme identifier of metadata	N/A	string	N/A

9.5 Output descriptor parameters

Table 55 defines output parameters, which are used as part of the output descriptor.

Table 55 — Output parameters

Name	Definition	Unit	Type	Valid range
stream-id	unique identifier, with the scope of function or task or workflow, to identify the media or metadata stream For functions, it is defined in the function descriptor. For tasks, it is assigned by the workflow manager. For workflows, it is assigned by the NBMP source.	N/A	string	N/A
name	string name assigned to this output For functions, it is defined in the function descriptor. For tasks, it is assigned by the workflow manager. For workflows, it is assigned by the NBMP source.	N/A	string	N/A
keywords	list of keywords describing this output properties The keyword should be human-readable.	N/A	array of string	N/A
mime-type	MIME media type of media or metadata in IANA registry	N/A	string	N/A
video-format	formats of video The parameter list is defined using generic parameter representation of subclause 9.22.1.	N/A	array of string	N/A
audio-format	format of audio The parameter list is defined using generic parameter representation of subclause 9.22.1.	N/A	array of string	N/A
image-format	format of image The parameter list is defined using generic parameter representation of subclause 9.22.1.	N/A	array of string	N/A

Name	Definition	Unit	Type	Valid range
codec-type	'codecs' and 'profiles' parameters, as defined in IETF RFC 6381	N/A	string	N/A
protocol	protocol for delivery of or access to media including protocol parameters such as port number(s) Ingest protocol for timed metadata including protocol parameters such as the port number(s). Example: HTTP. When the workflow manager receives this information, it takes the responsibility of returning back with the protocol endpoint information of the appropriate media processing entity to the media source so the media source can ingest metadata using that protocol. NOTE This is only applicable for media and timed metadata.	N/A	string	N/A
throughput	maximum possible throughput	bits/second	number	unsigned integer
buffer-size	minimum input buffer size	bytes	number	unsigned integer
caching-server-url	URL (according to IETF RFC 3986) of the server where the media will be sent or the location to which the media be fetched to NOTE When this parameter is missing for a workflow, the workflow manager may assign destination information of the appropriate media processing entity to the media source so the media sink can ingest media using that protocol.	N/A	string	N/A
max-size	maximum size of metadata in each fetch or push by this output	bytes	number	unsigned integer
min-interval	minimum interval between two fetch or push by this output	milliseconds	number	unsigned integer
scheme-uri	URL (according to IETF RFC 3986) or scheme identifier of metadata	N/A	string	N/A

9.6 Processing descriptor parameters

Table 56 defines processing parameters, which are used as part of the processing descriptor.

Table 56 – Processing parameters

Name	Definition	Unit	Type	Valid range
keywords	list of keywords that can be used to execute a search in the function repository	N/A	array of string	N/A
start-time	resource's start time	as defined by IETF RFC 3339: 2002, section 5.6	string	N/A
is-dynamic	flag indicating whether the image is static or dynamic A value of 'true' indicates the image is built dynamically. The default value is 'false' (static image).	N/A	boolean	N/A
url	pointer to the resource implementation, according to IETF RFC 3986	N/A	string	N/A
os	operation system	N/A	string	N/A
version	version number of operation system	N/A	string	N/A
architecture	hardware architecture	N/A	string	N/A
environment	environment	N/A	string	N/A
patch-url	URL (according to IETF RFC 3986) defining the patching scheme for this image	N/A	string	N/A
scheme	URL (according to IETF RFC 3986) defining information object scheme or information needed for dynamic build	N/A	string	N/A
co-located	specifies the deployment of the 2 connected tasks When the value is True, the 2 tasks shall be deployed into the same MPE, Otherwise, the deployment is determined by the workflow manager based on available resources. The default is 'false'.	N/A	boolean	N/A
id	specifies function's id	N/A	string	N/A

Name	Definition	Unit	Type	Valid range
instance	specifies identifier for one instance of a function An instance of a function shall have unique restrictions in a function group. This identifier shall be unique for each instance in the same function group. NOTE If a function is used more than once in one function group with identical restrictions, these restrictions can be defined by one instance of that function.	N/A	string	N/A
port-name	specifies function's logic port name	N/A	string	N/A

9.7 Requirements descriptor parameters

9.7.1 Flow control parameters

Table 57 defines flow control parameters, which are used as part of the requirements descriptor.

Table 57 — Flow control parameters

Name	Definition	Unit	Type	Valid range
typical-delay	typical expected delay for the resource	millisecond	integer	unsigned integer
min-delay	minimum delay (i.e. amount time from input to output sample) adequate for this resource	millisecond	integer	unsigned integer
max-delay	maximum delay required for this resource	millisecond	integer	unsigned integer
typical-throughput	typical expected throughput for this resource	bits/second	integer	unsigned integer
min-throughput	minimum bandwidth required for this resource	bits/second	integer	unsigned integer
max-throughput	maximum bandwidth adequate for this resource	bits/second	integer	unsigned integer
averaging-window	averaging window used to calculate the throughput The default is one second.	microsecond	integer	unsigned integer

9.7.2 Hardware parameters

Table 58 defines hardware parameters, which are used as part of the requirements descriptor.

Table 58 — Hardware parameters

Name	Definition	Unit	Type	Valid range
vcpu	number of vcpus to be reserved for the execution of a task	count	number	unsigned integer
vgpu	number of vgpus to be reserved for the execution of a task	count	number	unsigned integer
ram	memory to be reserved for the execution of a task	megabytes	number	unsigned integer
disk	size of local disk to be used by a workflow or a task	gigabytes	number	unsigned integer
placement	identifier of the geographical location of the data center in which the task is to be executed The location is represented by a two-letter (alpha-2) country code (ISO 3166-1) optionally followed by '-' and the postal code conforming to the country's postal code standard.	identifier	string	N/A

9.7.3 Security requirements

Table 59 — Security parameters

Name	Definition	Unit	Type	Valid range
tls	indicates if TLS[2] or DTLS[4] shall be used for the transport of media data The default value is 'false'.	flag	boolean	N/A
ipsec	indicates if IPSec [8] tunnel model shall be used for the transport of media data The default value is 'false'.	flag	boolean	N/A
cenc	indicates if MPEG common encryption (ISO/IEC 23001-7) shall be used for the exchange of media data The default value is 'false'.	flag	boolean	N/A

9.7.4 Workflow/task requirements

Table 60 — Workflow/task parameters

Name	Definition	Unit	Type	Valid range
execution-mode	defines workflow execution modes The value shall be 'streaming', 'step', or 'hybrid'. Default mode is 'streaming'.	N/A	string	N/A
function-fusible	whether functions can be fused or split by the NBMP workflow manager When fused or enhanced, some system tasks may be added or dropped automatically and dynamically. The default value is 'false'.	flag	boolean	N/A
function-enhancable	whether the inputs and outputs of a task can be modified or enhanced with system-provided built-in functions such as media transcoding, media transport buffering for synchronization, or transporting media/metadata data between tasks/MPEs over networks The default value is 'false'.	flag	boolean	N/A

9.7.5 Resource estimator parameters

Table 61 —Resource estimator parameters

Name	Definition	Unit	Type	Valid range
name	name of the input, output or configuration parameter	N/A	string	N/A
value	default value of the input, output or configuration parameter	N/A	string	N/A

The parameters used in the resource estimator descriptor are a subset of parameters used in the input, output and configuration descriptors.

9.8 Startup-Delay descriptor parameters

Table 62 defines startup delay parameters, used in the Startup-Delay descriptor.

Table 62 — Startup-Delay parameters

Name	Definition	Unit	Type	Valid range
startup-delay	amount of delay before task startup in seconds	seconds	number	unsigned integer

9.9 Client-Assistant parameters

Table 63 defines the parameters used in the Client-Assistant descriptor.

Table 63 — Client-Assistant parameters

Name	Definition	Unit	Type	Valid range
client-assistance-flag	indicates whether the resource requires/ supports client monitoring/assistance If client-assistance-flag is set to true in WDD: the workflow manager shall insert a measurement function to collect client assistance information from the client. The workflow manager shall connect the workflow tasks with functions that support client assistance. If the workflow manager cannot support client assistance, the workflow construction shall fail. If client-assistance-flag is set to false in WDD: The insertion of measurement function in workflow is optional. If client-assistance-flag is set to true in function description: the function cannot be instantiated without client assistance information. If client-assistance-flag is set to false in function description: client assistance information is optional.	N/A	boolean	N/A

9.10 Failover parameters

Table 64 defines the parameters used in the failover descriptor.

Table 64 — Failover parameters

Name	Definition	Unit	Type	Valid range
failover-mode	indicates action upon failover of underlying resource Following are the possible values: — ‘restart-immediately’ – restart the resource — ‘restart-with-delay’ – restart the resource after a certain delay — ‘continue-with-last-good-state’ – restart the resource based on available state persistence information — ‘execute-backup-deployment’ – execute backup deployment script given by backup-deployment-url below — ‘exit’: exit the resource The default value is ‘exit’.	N/A	string	N/A

Name	Definition	Unit	Type	Valid range
failover-delay	indicates the amount of delay before starting fail-over — When failover-mode value is 'restart-immediately' or 'exit' – this value is considered to be 0. — For other failover-mode values– this value defines the amount of time before failover is taken.	second	number	unsigned integer
backup-deployment-url	URL (according to IETF RFC 3986) to an external/internal instruction file for backup deployment that needs to be executed upon failover	N/A	string	N/A
persistence-url	URL (according to IETF RFC 3986) of storage where the state information is persisted This information is optional from the media source. The workflow manager can allocate some storage and use it for state information persistence.	N/A	string	N/A
persistence-interval	defines how often the state information is written to the persistence-url	second	number	unsigned integer

9.11 Events parameters

Table 65 defines the parameters used in the events descriptor.

Table 65 — Event parameters

Name	Definition	Unit	Type	Valid range
name	event's name	N/A	string	N/A
definition	humanly readable definition of this event	N/A	string	N/A
url	unique identifier for event, according to IETF RFC 3986	N/A	string	N/A

9.12 Variables parameters

Table 66 defines the parameters used in the variables descriptor.