SMPTE Public Committee Draft

Catena — Model

1 Table of Contents Page

Foreword

SMPTE (the Society of Motion Picture and Television Engineers) is an internationally-recognized standards developing organization. Headquartered and incorporated in the United States of America, SMPTE has members in over 80 countries on six continents. SMPTE's Engineering Documents, including Standards, Recommended Practices, and Engineering Guidelines, are prepared by SMPTE's Technology Committees. Participation in these Committees is open to all with a bona fide interest in their work. SMPTE cooperates closely with other standards-developing organizations, including ISO, IEC and ITU.

SMPTE Engineering Documents are drafted in accordance with the rules given in its Standards Operations Manual. This SMPTE Engineering Document was prepared by Technology Committee TC34CS – Media Systems, Control and Services.

Introduction

This clause is entirely informative and does not form an integral part of this Engineering Document.

This document specifies a standardization of communication methods between (micro)services and full products – designed for hybrid cloud and on-premises solutions with the goal of making it easy to secure, connect and control a multi-vendor ecosystem of media processing services and microservices no matter where they are in a true plug-and-play model.

The Hybrid Environment Reality

Today's environments typically encompass a combination of on-premises and cloud environments. Solutions span a variety of platforms, and must interoperate, regardless of where they are hosted.

Need for More Seamless Multi-Vendor Integration

The number of devices, products, and services from a wide array of vendors can be daunting, particularly with the move to microservices. Media companies need technologies to simply work together.

Security Emerging as a Priority

The need to secure environments and the connections between products and services in a standardized manner is a must. As shown in Figure 1, the transport between each device is secure.

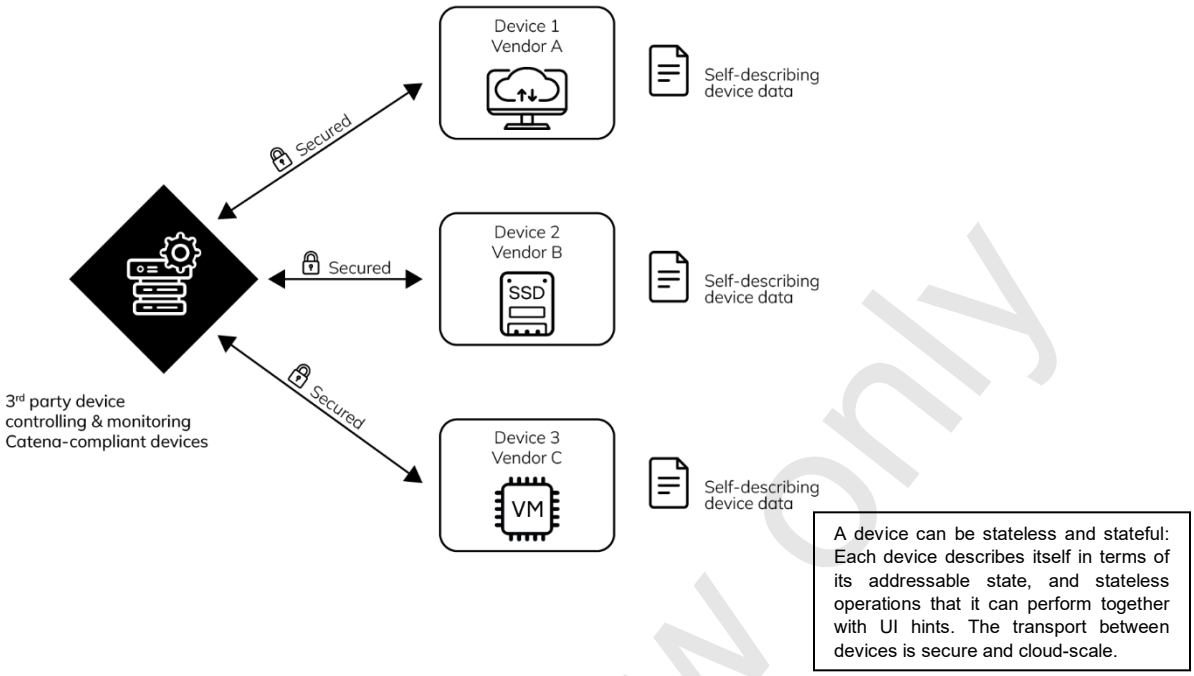


Figure 1 — Secure Architecture

Throughout this document, snippets showing JSON examples are shown. A more complete set of example files (in YAML) can be found in https://github.com/SMPTE/st2138-a.git.

At the time of publication, no notice had been received by SMPTE claiming patent rights essential to the implementation of this Engineering Document. However, attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights.

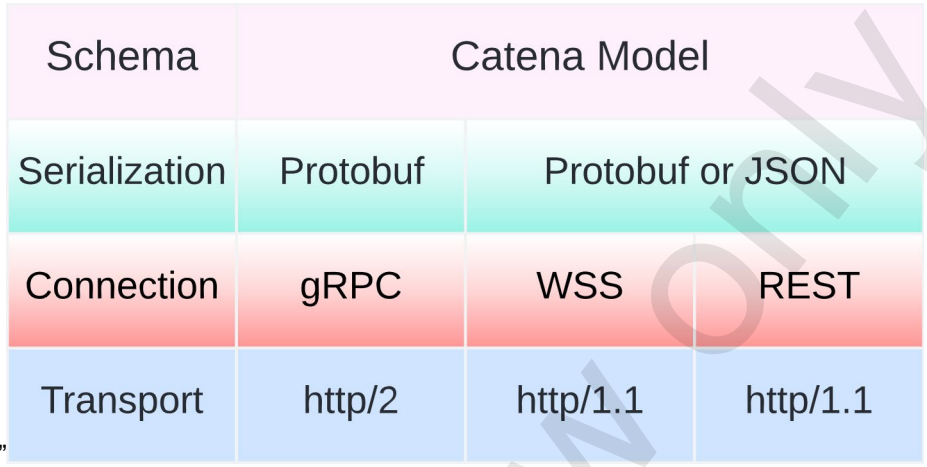
SMPTE shall not be held responsible for identifying any or all such patent rights.

1 Scope

This document specifies schema for plug-and-play communication and control of media services and devices across cloud, on-premises, and hybrid cloud/on-premises platforms.

This document also defines a number of Access Scopes that reflect how media production equipment is used in a variety of use cases.

This document is applicable to the Catena model architecture shown in Figure 2.



."

Figure 2 — Catena Architecture

2 Conformance Notation

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except: the Introduction, any clause explicitly labeled as "Informative" or individual paragraphs that start with "Note:"

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords, "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document.

The keyword “reserved” indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword “forbidden” indicates “reserved” and in addition indicates that the provision will never be defined in the future.

A conformant implementation according to this document is one that includes all mandatory provisions ("shall") and, if implemented, all recommended provisions ("should") as described. A conformant implementation need not implement optional provisions ("may") and need not implement them as described.

Unless otherwise specified, the order of precedence of the types of normative information in this document shall be as follows: Normative prose shall be the authoritative definition; tables shall be next; then formal languages; then figures; and then any other language forms.

3 Normative References

The following Standard contains provisions that, through reference in this text, constitute provisions of this standard. Dated references require that the specific edition cited shall be used as the reference. Undated citations refer to the edition of the referenced document (including any amendments) current at the date of publication of this document. All standards are subject to revision, and users of this engineering document are encouraged to investigate the possibility of applying the most recent edition of any undated reference.

IETF RFC 3066 - *Tags for the Identification of Languages*, H. Alvestrand, Internet Engineering Task Force, 2001. http://www.ietf.org/rfc/rfc3066.txt

IETF RFC 6762 - *Multicast DNS (mDNS)*, Internet Engineering Task Force, 2013. https://datatracker.ietf.org/doc/html/rfc6762

IETF RFC 6763 - *DNS-Based Service Discovery (DNS-SD)*, Internet Engineering Task Force, 2013, https://datatracker.ietf.org/doc/html/rfc6763

IETF RFC 7540 - *Hypertext Transfer Protocol Version 2 (HTTP/2)*, Internet Engineering Task Force, 2015, https://datatracker.ietf.org/doc/html/rfc7540

IETF RFC 8446 - The Transport Layer Security (TLS) Protocol Version 1.3, Internet Engineering Task Force, 2018, https://datatracker.ietf.org/doc/html/rfc8446

IETF RFC 8552 - *Scoped Interpretation of DNS Resource Records through "Underscored" Naming of Attribute Leaves*. https://datatracker.ietf.org/doc/html/rfc8552

IETF RFC 9000 - *QUIC: A UDP-Based Multiplexed and Secure Transport*, Internet Engineering Task Force, 2021, https://datatracker.ietf.org/doc/html/rfc9000

W3C CSS Color Module Level 3, https://www.w3.org/TR/css-color-3/

4 Precedence

In the event of a conflict between the schema found in https://smpte.github.io/st2138-a/interface/schemata/device.yaml and other information in this document, the schema shall take precedence.

5 Terms, Definitions and Abbreviated Forms

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

5.1
Catena service

service supporting the Catena schema

5.2
device

entity that supports Catena Services

Note 1 to entry: A device can be physical hardware, a microservice, or a fully-featured monolithic service.

5.3
client

entity that accesses resources served by devices using Catena Services

Note 1 to entry: A device can act in the capacity of a client with respect to another device

5.4
object identifier
OID

unique key of type string within a map

5.5
fully qualified object identifier
FQOID

JSON pointer that uniquely identifies an object within a Device object

5.6
RPC

Remote Procedure Call

5.7
BLOB

Binary Large Object

5.8
QUIC

Quick UDP Internet Connections

5.9
Avahi

free, open-source software suite that implements the mDNS/DNS-SD (zeroconf) protocols, enabling devices on a local network to discover each other and their services without manual configuration

5.10
discriminated union

data structure that allows a variable to hold values of different, but predefined, types, with a means to distinguish which type is currently stored

6 Precedence

Unless otherwise specified, the order of precedence of the types of normative information in this document shall be as follows. Normative prose shall be the authoritative definition. Tables shall be next, followed by formal languages, then figures, and then any other language forms. In the event of a conflict between the schema and other information in this document, the schema is authoritative.

7 Registration and Discovery

7.1 Desirable Characteristics (Informative)

7.1.1 Cloud Scale

Registration and discovery need to operate well at cloud scale. A challenge is that "cloud scale" is not a precisely defined term. It has been argued that media production operations are not truly cloud scale. A limitation of perhaps 10,000 collaborating services is likely sufficient for all use cases.

7.1.2 Fast and Dynamic

Ideally, registration and discovery would be performed using ephemeral services which are spun up, get used, then spin down. Thus, this approach needs to be able to keep up with services coming in and out of existence. Traditional DNS updates can take large fractions of an hour to propagate. Dynamic approaches such as Kubernetes DNS (and Consul, Istio, and cloud provider service directories) address this need, but they are not standards.

7.1.3 Small and Simple

The aim in a Catena environment is that when a new device is plugged in, it ought to appear very shortly thereafter under the control system being used (i.e., plug and play). It ought not to be necessary, for instance, to set up IS-04 servers, DHCP servers or DNS servers to achieve this. mDNS (as specified in RFC 6762 - Multicast DNS (mDNS)) is ideal for this use case and there are mature open-source libraries to support it.

The test for this requirement is: zero extra servers or services are needed.

Note that strictly adhering to this requirement also rules out requiring an authorization server. Any device would need to run with authorization disabled.

7.1.4 Manual Discovery

In cases where automatic discovery is not possible, users or systems need to be able to "discover" services and devices by entering their hostname or IP address into a registration dialog provided by whatever client they are using.

7.1.5 Options for Registration & Discovery

There are many options for Registration & Discovery available. The main options are outlined, along with their characteristics, in Table 1.

Table 1 — Registration & Discovery Options (Informative)

Option	Standards-Based	Cloud Scale	Fast & Dynamic	Small & Simple	Comments
DNS-SD	Yes	No	No	No	
mDNS	Yes	No	Yes	Yes	
IS-04	No	Possibly	Unknown	No	Could be suitable at cloud scale
Modern alternatives such as K8s DNS	No	Yes	Yes	No	
Event driven protocols such as MQTT	Yes	Possibly	Possibly	No, needs a broker	MQTT is not a discovery technology per se; it is a pub/sub technology, but discovery information could be what is published/subscribed. It is used with cloud services today.
NATS	No	Yes	Unknown	Unknown	

7.2 Using other Registry Technologies (Informative)

While mDNS is suitable for small-office and low-complexity use cases, it is unsuitable for operations at larger scales. There are many solutions for service registration that would be impractical to include in this standard. Therefore, this standard simply defines the information to publish a service to a discovery service, and how it is to be formatted and identified.

7.3 Discovery Recommendations

DNS-SD (as specified in RFC 6763 - DNS-Based Service Discovery (DNS-SD)) should be used for discovery in most cases, as it satisfies the majority of the desirable characteristics outlined in Subclause 7.1.

mDNS should be used for discovery in cases of facilities where DNS is not an option, although any of the options listed in Clause 7.1 may be utilized.

7.4 mDNS Overview

7.4.1 Service Naming

Each service instance shall have an instance name which shall conform to the following:

<instance-name>._st2138.<api>.<transport>.<network>.local

Where:

- <instance-name> is a human-readable identifier for the service instance
- _st2138 identifies the control protocol as SMPTE ST 2138
- <api> specifies the type of API available which shall be one of these values:
    - _grpc
    - _rest
    - _ws
- <transport> specifies the transport which shall be one of these values:
    - _http2s, _http2 for the gRPC connection type
    - _https, _http for REST and web socket connection types
    - _quic, for future compatibility if this transport gains significant adoption
- <network> specifies the network layer which shall be one of these values:
    - _tcp for http2s, http2, https and http transports
    - _udp for quic

Naming Requirements

- The instance name shall be unique on the local network.
- Implementations may include the hostname or another unique identifier.

7.4.2 Required DNS Records

Each service shall publish the following DNS records:

- PTR Record

  The PTR record allows clients to discover all available _.st2138.<api>.<transport>.<network>.local. services.

  For example, this record:

  _st2138._grpc._http2s._tcp.local. PTR router._st2138._grpc._http2s._tcp.local.

  Announces an instance called router that provides the ST 2138 service using a gRPC API with secure communications enabled.

- SRV Record

  The SRV record provides connection details for a specific instance:

  For example:

  router._st2138._grpc._http2s._tcp.local. SRV 0 0 <port> <hostname>.local.

Where:

- <port> is the TCP port number where the ST 2138 service is available
- <hostname> is the hostname of the device

- A/AAAA Record

  The A/AAAA record maps the hostname to an IP address.

  <instance-name>.local. A <IPV4 address>

  <instance-name>.local. AAAA <IPV6 address>

- TXT Record (service metadata)

  The TXT record provides additional metadata for the service.

  <instance-name>._st2138.<api>.<transport>.<network>.local. TXT <key>=<value>

Table 2 summarizes the required and optional metadata included in the TXT record.

Table 2 — ST 2138 DNS Metadata

Key	Required?	Description	Example
authz	Yes	Indicates whether the service is registered with an authorization service and is enforcing fine-grained access control	authz=on authz=off
authz_server	Yes, if authz=on	The top-level URL of the OAuth2 server with which the service is registered	Authz_server=https://authz.smpte.org
realm	Yes, if authz=on, and Keycloak is the OAuth2 implementation	The Keycloak realm	realm=production
interface	Yes	URL of the interface definition. For gRPC this would be a protobuf service definition, for REST, an openAPI specification	interface=https://github.com/smpte/st2138/services.proto
if_version	Yes	Version of the interface	
sdk	No	If an open-source SDK was used by the service, its URL may be presented here to allow connecting clients to evaluate compatibility	sdk=https://github.com/rossvideo/Catena
sdk_version	Yes, if previous key was provided	If the previous key was provided	sdk_version=cpp-v0.0.7-1

If mDNS is being used, the use case is likely a small/home office without access to DNS and OAuth2 authorization servers. This shall be the only scenario where providing a service without fine-grained access control is permitted.

If the service can find and register its resources with an OAuth2 authorization server, it shall operate with fine-grained access control enabled and advertise this fact via “authz=on”.

When a client discovers a device or service via mDNS with authz=off, it need not obtain authorization tokens to attach to every request.

7.5 Example Implementation (Informative)

The following service definition is an example of using Avahi’s configuration method in which a service or device is advertising the ST 2138 service as available over both gRPC with secure communications and REST in clear text. This works for both mDNS and DNS-SD based on the Avahi command used with the file.

```
<?xml version="1.0" standalone='no'?>
<!DOCTYPE service-group SYSTEM "avahi-service.dtd">
<service-group>
    <name replace-wildcards="yes">router</name>

    <!—ST 2138 gRPC over HTTPS/2 with TLS (secure) -->
    <service>
        <type>_st2138._grpc._http2s._tcp</type>
        <port>6254</port>
        <txt-record>authz=off</txt-record>
        <txt-record>interface=https://smpte.org/registry/st2138/service</txt-record>
        <txt-record>if_version=2025.1</txt-record>
        <txt-record>sdk=https://github.com/rossvideo/Catena</txt-record>
        <txt-record>sdk_version=cpp-v0.0.7-1</txt-record>
    </service>

    <!-- REST over HTTP (insecure) -->
    <service>
        <type>_st2138._rest._http._tcp</type>
        <port>8080</port>
        <txt-record>authz=off</txt-record>
        <txt-record>interface=https://smpte.org/registry/st2138/service</txt-record>
        <txt-record>if_version=2025.1</txt-record>
        <txt-record>sdk=https://github.com/rossvideo/Catena</txt-record>
        <txt-record>sdk_version=cpp-v0.0.7-1</txt-record>
        <txt-record>content-type=application/json</txt-record>
    </service>

</service-group>
```

7.6 Format

The registered information shall be serialized as JSON and may be stored and delivered either as a JSON object, or a stringified version of the object.

7.7 Registration Service Information

The information to publish to the registration service is tabulated in Table 3.

Table 3 — Registration Service Information

Field Name	Required?	Datatype	Definition
instance-name	Yes	string	A human-readable name for the service that is unique within the local network
api	Yes	string	The type of API used for the st2138 service, one of: gRPC, REST
transport	Yes	string	The transport. One of: http2s, http2, https, http, qui
service-name	Yes	string	The service name, e.g., switcher
hostname	Yes	string	The fully qualified domain name of the service's host
port	Yes	unsigned integer	The TCP (REST, gRPC) or UDP (quic) port on which the service is available
ipv4	One must be present	string	The IPV4 address, including CIDR information
ipv6		string	The IPV6 address
authz	Yes	boolean	Whether the service is running with authorization enabled
authz-server	Yes, if authz=true	string	The URL of the authorization server
realm	Optional	string	Where applicable, the realm with which the service is registered
audiences	Optional	array of strings	Logical groupings of services that include the service being registered. Used for traffic filtering at API gateways, if deployed.

Field Name	Required?	Datatype	Definition
interface	Optional	string	The URL of the service definition supported by the service
if-version	Optional	string	The service definition's version if this is not included in "interface"
sdk	Optional	string	The URL of the public SDK used by the service, if applicable
sdk-version	Optional	string	The version of the sdk
custom-metadata	JSON key, value store	map of string to strings which might be encodings of structured data types	A key/value store for custom metadata

7.8 Identification

If the discovery service being used provides a means to extend the information registered (and most, if not all, do), the JSON object containing the registration information should include "st2138". For example, IS-04 recommends that extensions be formatted "x-<name of extension>", which becomes "x-st2138".

7.9 Example Registration Information (Informative)

```
"x-st2138": {
"instance-name": "gallery-4-swr-2",
"api": "gRPC",
"transport": "http2s",
"service-name": "production-switcher",
"hostname": "swr2.gallery4.salford.production.ccd.co.uk",
"port": 6254,
"ipv4": "192.168.1.10/24",
"authz": true,
"authz-server": "https://authz.salford.production.ccd.co.uk",
"realm": "production",
"audiences": ["gallery4"],
"interface": "https://github.com/smpte/st2138/service.proto",
"if-version": "1.0.1"
}
```

8 The Catena Model

Figure 3 provides a high-level view of a Catena-compatible device's architecture and the Catena Model's place within it.

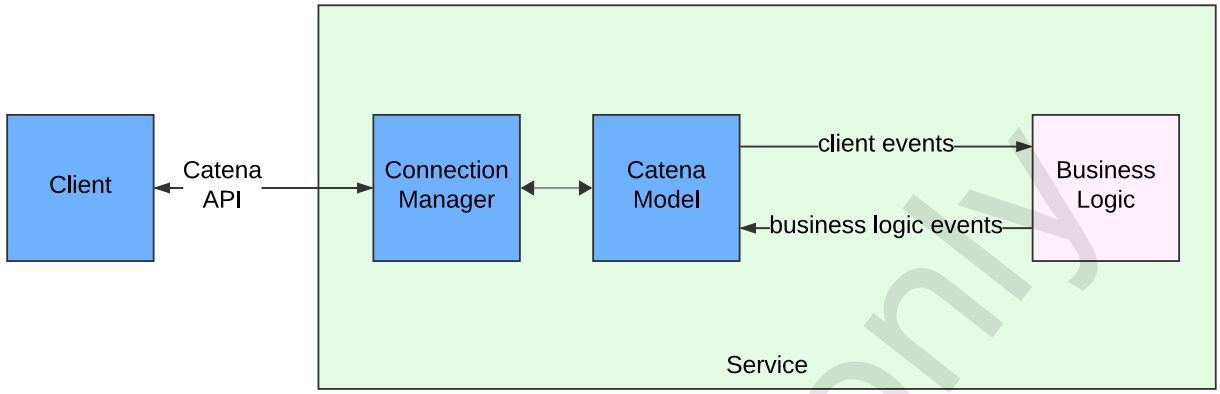


Figure 3 — Catena Device Architecture

The objective is to isolate the model from both the device's Business Logic and the Connection Manager, which could be implemented using gRPC, REST, or WSS. The Catena Model defines the schemata of the objects that travel across the Connection.

A Catena Model shall comprise at least one, and possibly multiple Device Model objects, each of which shall have a locally unique address called a "slot number".

As shown in Figure 4, slot numbers do not need to be contiguous, just locally unique.

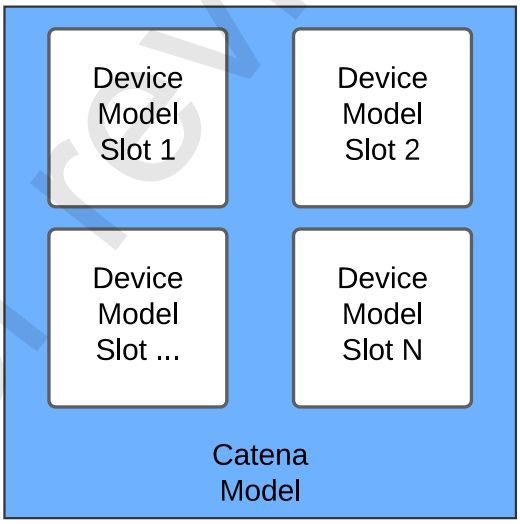


Figure 4 — Device Models

The purpose of the Catena Device Model is to describe what controls, alarms, and monitors the information that a device has, and how to access them.

Interface definitions are available as specified in Annex B.

9 Device Data Model

9.1 Device Data Model Overview

The Device Data Model is highly flexible and supports:

- Scalability from small, simple devices with minimal state data to large, complex devices with tens of MB of state data.
- Aggregation of small, possibly industry-standard components. For example, a 4-band EQ might have a well-defined and globally agreed-upon representation that may be imported into a larger Device that benefits from including one.
- Fine-grained access control by assigning every piece of state and every command within the Device Model an appropriate Access Scope. Bearer tokens that accompany each API call will either confirm that the scope in question can be accessed or not. Note that the process and business logic of how these bearer tokens are generated is outside the scope of this standard, which is limited to defining the Device policy enforcement.
- Highly scalable access control by defining a standard set of Access Scopes that all Catena Devices shall honor.

The Catena Device Data Model shall consist of a collection of objects, as defined in Clauses 9.2 through 9.12.

9.2 Device Model

Figure 5 shows a high-level view of the Catena device model.

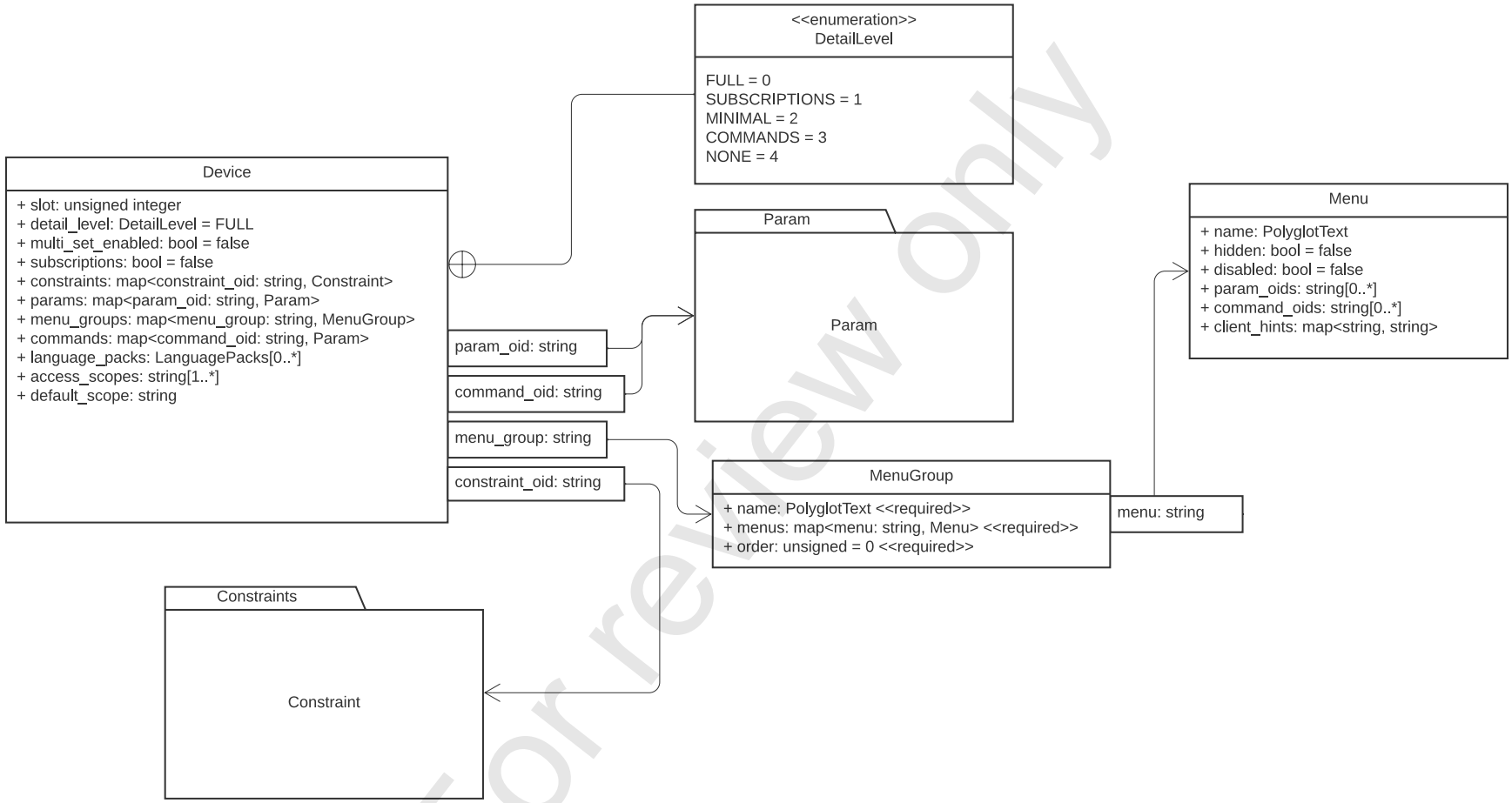


Figure 5 — The Top-level Device Model, Names plus Relations with Param and Types of MembersMenuGroup

The Device Object shown in Figure 5 is defined in Protobuf as the `Device` message in `device.proto`, and as the top-level schema at https://smpte.github.io/st2138-a/interface/schemata/device.yaml. All other schema are in the `$defs` property of this object definition and referenced thus `#/$defs/<schema name>`.

9.3 DetailLevel ObjectEnum

The Detail Level enumeration is defined in Protobuf as the `Device.DetailLevel` message in `device.proto` and by the schema at `#/$defs/detail_level`.

9.4 MenuGroups Object

The Protobuf `MenuGroups` message is defined in `menu.proto` and by the schema at `#/$defs/menu_groups`.

9.5 Object Identifiers

Local object identifiers shall conform to the schema at `#/$defs/simple_oid` and fully qualified identifiers by the schema at `#/$defs/fqoid`.

9.6 Params Object

The Params Object is a map of oids to Param objects. It is defined by the schema at `#/$defs/param_map`.

The params object is a member of the top-level Device Object. It may also appear as a child of the Param Object which means that parameters should be organized hierarchically.

As an example, consider a parameter that represents a geographical location.

In C or C++, it could be represented thus:

```
struct Location {
  struct Angle {
    int degrees;
    int minutes;
    int seconds;
  };
  Angle latitude;
  Angle longitude;
  float elevation;
}
```

This is a nested structure to demonstrate a hierarchy of parameters.

A device model that had a Location parameter may be serialized as JSON thus:

```
{
  …
  “constraints”: {
    “sexagesimal”: {
      “min_value”: 0,
      “max_value”: 59,
      “step”: 1
    }
  },
```



```
  “params”: {
    “angle”: {
      “type”: “STRUCT”,
      “params”: {
        “degrees”: {
          “type”: “INT32”
        },
        “minutes”: {
          “type”: “INT32”,
          “constraint”: {“ref_oid”: “/sexagesimal”}
        },
          “seconds”: {
          “type”: “INT32”,
          “constraint”: {“ref_oid”: “/sexagesimal”}
        },
      }
    },
    “location”: {
      “params”: {
        “latitude”: {
          “template_oid”: “/angle”
        },
        “longitude”: {
          “template_oid”: “/angle”
        },
        “elevation”: {
          “type”: “FLOAT”
        }
      },
      “value”: {
        “struct_value”: {
          “fields”: {
            “latitude”: {
              “value”: {
                “struct_value”: {
                  “fields”: {
                    “degrees”: {“value”: {“int32_value”: 0}},
                    “minutes”: {“value”: {“int32_value”: 0}},
                    “seconds”: {“value”: {“int32_value”: 0}}
```

```
                }
              }
            }
          },
          “longitude”: {
            “value”: {
              “struct_value”: {
                “fields”: {
                  “degrees”: {“value”: {“int32_value”: 0}},
                  “minutes”: {“value”: {“int32_value”: 0}},
                  “seconds”: {“value”: {“int32_value”: 0}}
                }
              }
            }
          },
          “elevation”: {“value”: {“float_value”: 0.0}}
        }
      }
    }
  },
  …
}
```

9.7 Param Object

9.7.1 Param Object Details

The Param Object structure is illustrated in Figure 6. It is defined in Protobuf as the Param message in `param.proto` and by the schema at `#/$defs/param`. To assist clients in rendering useful user interfaces, each parameter descriptor includes a widget field. The list of reserved widget names is provided in Annex A.

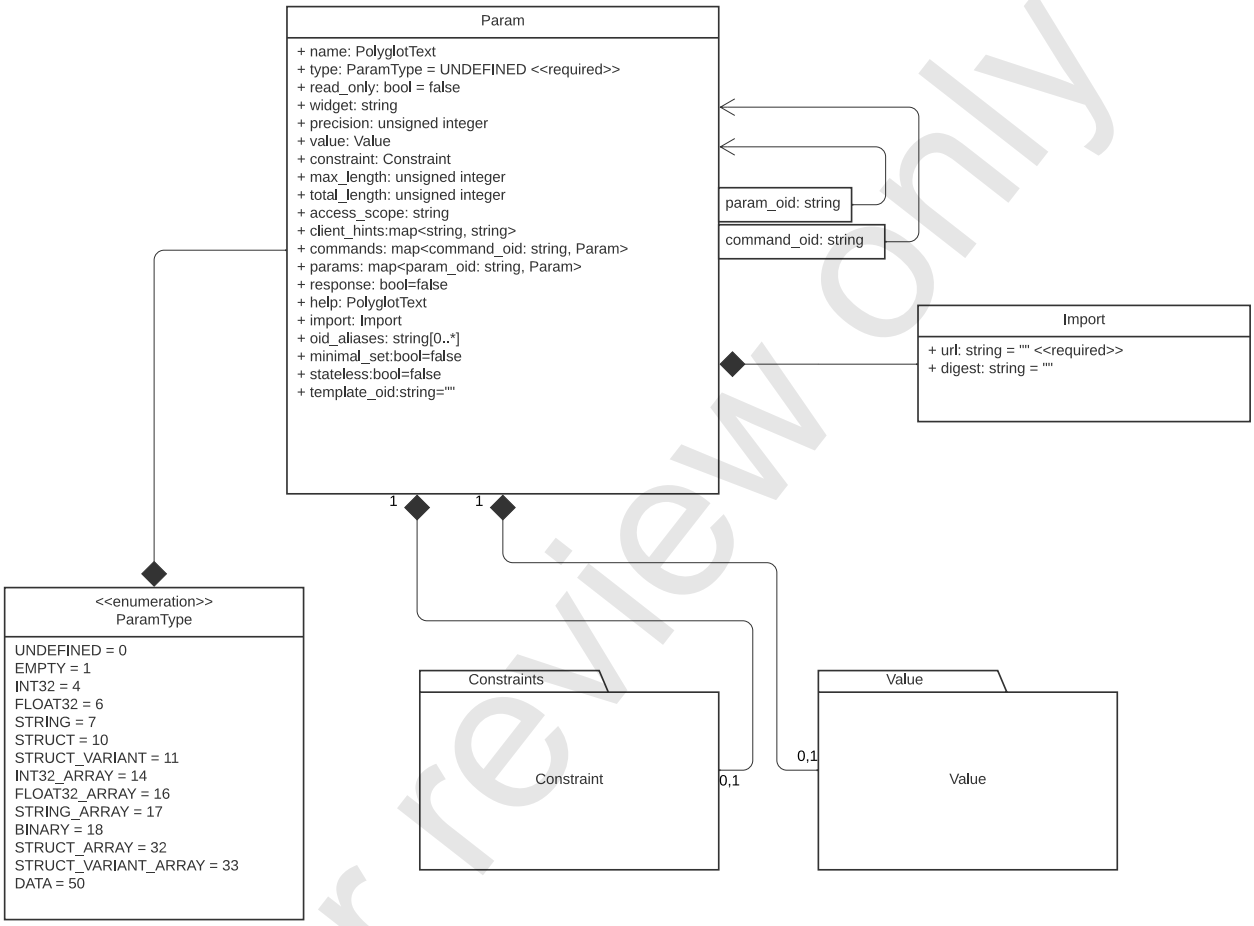


Figure 6 — Param

9.7.2 ParamType Object

The ParamType object is defined in Protobuf as the `ParamType` message in `param.proto` and by the schema at `#/$defs/param_type`.

9.7.3 Import Object

The Import Object illustrated in Figure 6, above, provides information necessary to locate the definition of a Param object. It is defined by the Protobuf `Import` message in `param.proto` and by the schema at `#/$defs/import`. Its digest field is the SHA-256 hash of the object to import.

9.8 Value Object

9.8.1 Value Object Details

The Value object is polymorphic. It may take one of many different types as illustrated in Figure 7. It is defined in Protobuf by the `Value` message in `param.proto` and by the schema at `#/$defs/value` which is specialized by the schemas at:

- `#/$defs/apply_int32`
- `#/$defs/apply_float32`
- `#/$defs/apply_string`
- `#/$defs/apply_struct`
- `#/$defs/apply_struct_variant`
- `#/$defs/apply_int32_array`
- `#/$defs/apply_float32_array`
- `#/$defs/apply_string_array`
- `#/$defs/apply_struct_array`
- `#/$defs/apply_struct_variant_array`
- `#/$defs/apply_data`

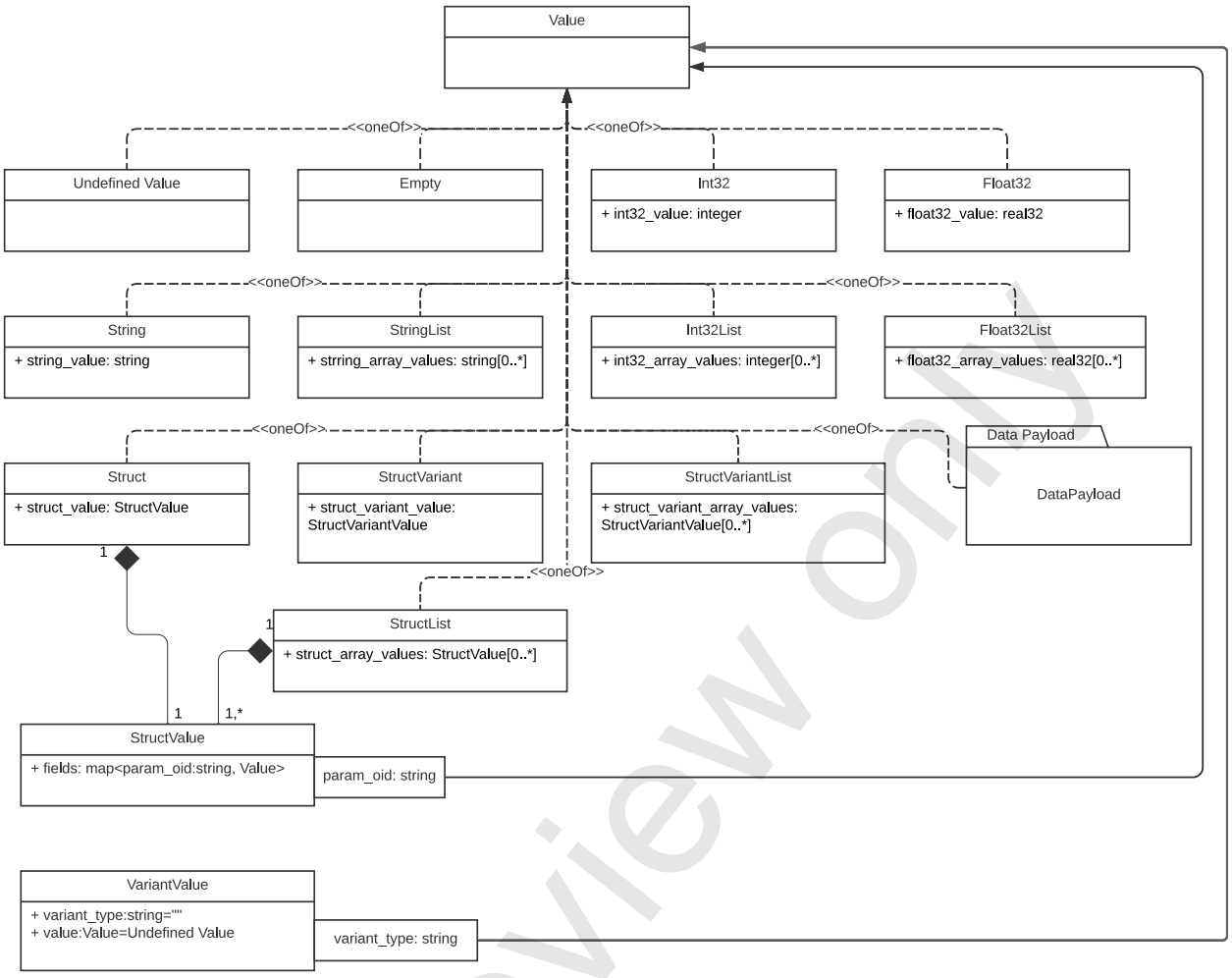


Figure 7 — Value Object Polymorphism

9.8.2 Undefined Value

This is implemented as an enum with a single default choice labeled UNDEFINED_VALUE with value 0. It is useful to signal when a parameter's value has not been defined but mighty become so in the future.

9.8.3 Struct Value

This is best explained by the example shown in 9.5.

9.8.4 Data Payload

As shown in Figure 8,the purpose of the Data Payload object is to allow a Device to send BLOBs of opaque data to a client. Both direct and by-reference methods are supported. It is defined by the Protobuf message `DataPayload` in `param.proto` and by the schema at `#/$defs/data_payload`.

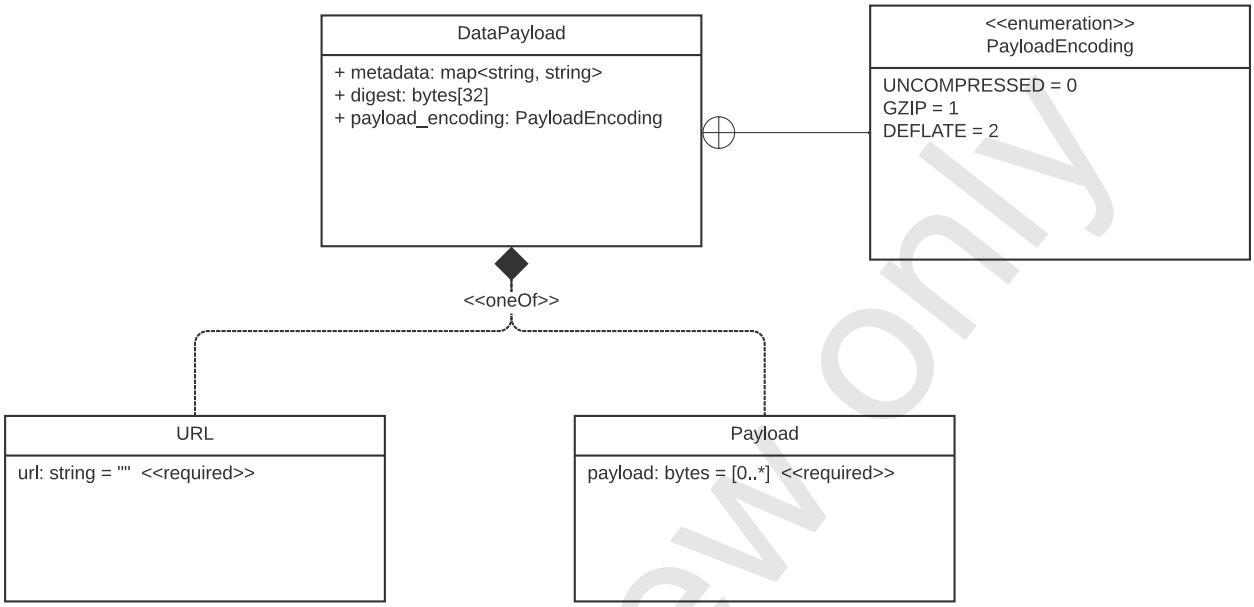


Figure 8 — DataPayload

9.9 Constraint Map

9.9.1 Constraint Map Details

Parameter values may be constrained, forcing only valid values to be allowed.

- Floating point and integer values may be constrained to a range.
- Integers may also be constrained to enumerated values.
- Integers may also be subject to an Alarm Table constraint.
- Strings may be constrained to one of two types of enumerated list:
    - STRING_CHOICE, which simply lists the string choices without multi-language support.
    - STRING_STRING_CHOICE which provides a key:value map to multi-language encoded strings.

If a constraint is defined within the body of a parameter object, it is local to that parameter and only constrains its parent parameter's value.

Constraints defined as members of the Device object's constraints object may be reused by any parameter by use of the ref_oid directive.

Constraints shared in this manner are copied into the parameter so that later changes to them do not affect the shared definition. Likewise post-initialization changes to the shared constraint will not affect the parameters that used it. Figure 9 shows the structure of the Constraint Map.

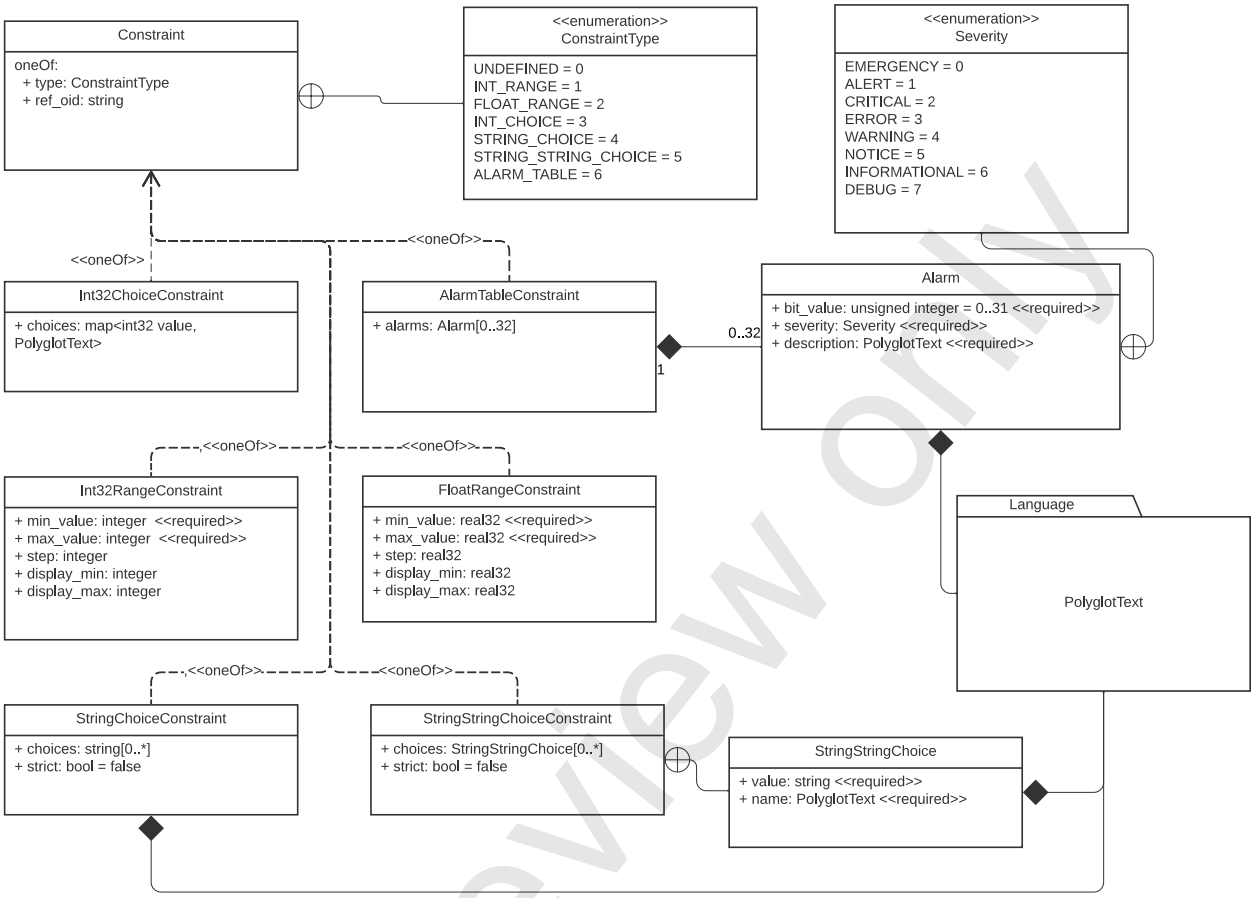


Figure 9 — Constraints

9.9.2 Range Constraints

Range constraints are defined by the Protobuf `Int32RangeConstraint` and `Float32RangeConstraint` messages in `constraint.proto` and by the schemata within `#/$defs/int32_constraint` and `#/$defs/float32_constraint`.

9.9.3 Choice constraints

Choice constraints allow for valid parameter values to be enumerated. This allows clients to provide UI elements such as drop-down menus and radio buttons for interaction.

They are defined by the Protobuf `Int32ChoiceConstraint`, `StringChoiceConstraint` and `StringStringChoiceConstraint` messages and by the schemata at `#/$defs/int32_choice`, `#/$defs/string_choice` and `#/$defs/string_string_choice`.

9.9.4 Alarm Table Constraint

Alarm tables allow up to 32 alarms to be represented by a 32-bit integer, each bit of which is mapped to a specific alarm by the Alarm object. It is defined by the Protobuf `AlarmTableConstraint` message in `constraint.proto` and within the `#/$defs/int32_constraint` schema.

9.10 Commands Map

The commands map shall be a map of string : Param object pairs. This permits a client to convey arbitrarily structured data along with a command. An example is a tape transport with a command “play” that has a value representing the play speed.

NOTE Commands are stateless; the value of the play speed in our example is a separate piece of state for each invocation of the command.

9.11 LanguagePacks Map

9.11.1 LanguagePacks Details

LanguagePacks provide multi-language support as defined in IETF RFC 3066. Supporting clients could, for example, provide a way to switch the active language rendered on the display.

They also provide the possibility of adding language support in the field by installing additional language packs.

Each language pack shall be uniquely identified by its language code, for example:

- “es” – global Spanish
- “en” – global English
- “en-us” – US English
- “fr” – global French

The structure of a LanguagePack is shown in Figure 10. They are defined by the Protobuf `LanguagePacks`, `LanguagePack` messages in `language.proto` and by the schemata at `#/$defs/language_packs`, `#/$def/language_pack`.

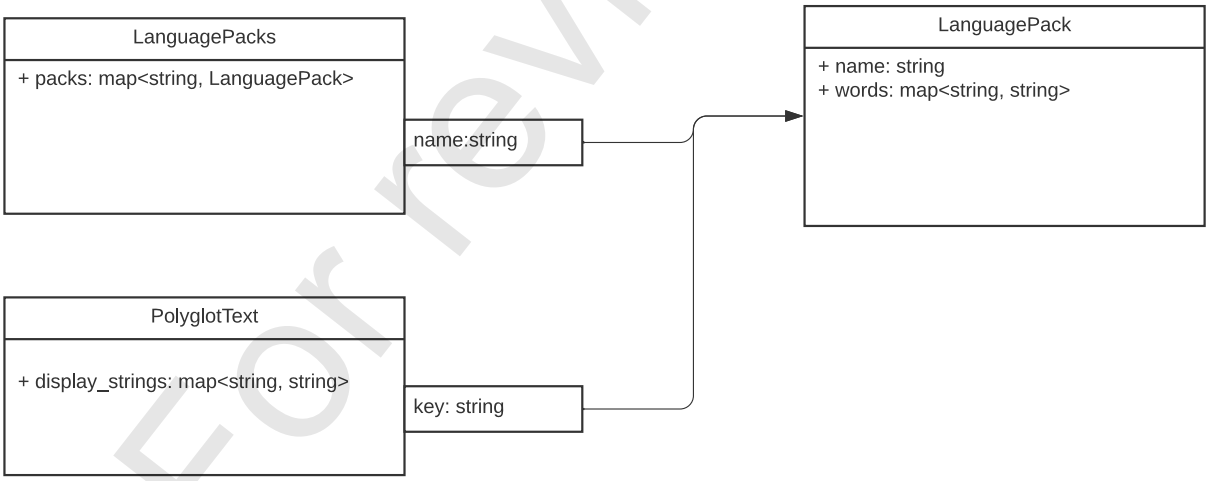


Figure 10 — Language Packs

9.11.2 PolyglotText Map

9.11.2.1 PolyglotText

PolyglotText is text that a client may display in one of multiple languages. The different options may either be defined inline for each instance of this object, or they may reference a LanguagePack. It is defined in Protobuf by the `PolyglotText` message in `language.proto` and by the schema at `#/$defs/polyglot_text`.

When defined inline, the keys shall be the language codes defined by IETF RFC 3066. For example, a key with the value "$key", that is **not** in IETF RFC 3066 indicates that the text to display can be accessed from a LanguagePack.

9.11.2.2 In-line PolyglotText Definition example

```
{
  "name": {
    "en": "Hello",
    "es": "Hola",
    "fr": "Bonjour",
    "en_CA": "G'day, Eh?"
  }
}
```

9.11.2.3 LanguagePack referencing example of PolyglotText

```
{
  "name": {
    "$key": "greeting"
  }
}
```

9.12 Access Scopes and Default Scope

Access scopes enable administrators to configure fine-grained access control schemes that may be Role-based, Attribute-Based, Conditional, and more. To make configurations scalable, Catena defines a small number of Access Scopes that reflect how media production equipment is used in a variety of use cases which include:

- Commissioning
- Troubleshooting
- Operating
- Auditing
- Patching and Upgrading
- Orchestrating (setting up signal flows)

The Catena Access Scopes comprise the following verbs:

- Monitor, e.g., Audio Level Meters, Waveform Monitors. Encoded as st2138:mon, and st2138:mon:w
- Operate, e.g., Audio Level Faders, Switcher Transition Faders, Router matrix buttons. Encoded as st2138:op, and st2138:op:w
- Configure, e.g., set up multicast IP addresses for input and output flows. Encoded as st2138:cfg, and st2138:cfg:w
- Administer, e.g., read logs, update firmware, reboot. Encoded as st2138:adm, and st2138:adm:w

The purpose of the “st2138:” prefix used with the scope names is to place the scopes within a namespace and thereby avoid clashes.

Devices shall support the Catena Access Scopes. They may augment them with extra scopes.

Every item of state (parameter) and command within the device model shall have at least one Access Scope associated with it according to these rules:

1. The parameter or command shall have an explicitly defined Access Scope if its “access_scope” member is present; otherwise
2. The parameter or command shall have a parent with an explicitly defined Access Scope, in which case it inherits that of the parent; otherwise
3. The parameter or command shall have the default access scope defined in the Device Object.

10 Mandatory Params

A Device shall provide all of the parameters shown in Table 4.

Table 4 — Mandatory Params

Fully Qualified Object Id	Type	Scope	Information
/product/name	STRING	monitor	The name of the product
/product/vendor	STRING	monitor	The name of the product’s vendor
/product/version	STRING	monitor	The product’s software version
/product/catena_sdk	STRING	monitor	Identifier for the SDK. This could be the URL to a GitHub repo, for example
/product/catena_sdk_version	STRING	monitor	Version number of the SDK
/product/serial_number	STRING	monitor	The product’s serial number.

11 Reserved OIDs

If a device uses any of these Object IDs, its use shall match the semantics specified in the Table 5.

Table 5 — Reserved OIDs

Fully Qualified Object Id	Type	Scope	Purpose
/product/options	STRING ARRAY	monitor	List of fitted options
/product/icon	STRING	monitor	Path to the product's icon. Intended for clients to access via an External Object Request.
/openGear/<oid>	Any	Any	Backwards compatibility with openGear reserved OIDs where <oid> is substituted by the openGear reserved OID

Annex A
(Normative)

Reserved Widget Identifiers

Reserved Widget Identifiers are shown in Table A.1.

Client hints are defined by schema relative to:
https://smpte.github.io/st2138-a/interface/schemata/client-hints.yaml

Table A.1 — Reserved Widget Identifier

Widget Identifier	Suggested Rendering	Suggested Client Hints
default	Client decision	
text-display	Read-only text view	
hidden	Do not render the parameter or command	
label	Parameter value formatted as a label	
text	Line of text	
password	Text entry that renders stars or circles in response to keystrokes	
combo	Drop-down list	
dot	A status LED with text	#/$defs/css_color
html-text	Text display with HTML formatting	
multiline-text	Text entry box	
toggle	Toggle button	Should be accompanied by a constraint that defines option 0 = off and option 1 = on
checkbox	Checkbox	
radio	Group of radio buttons	Should be accompanied by one of the Choice constraints
list	list	
tree	tree	
tree-popup	Pop-up tree	
color-picker	Color picker	
color-picker-popup	Pop-up color picker	
file-picker	File system navigator and file selector	
spinner	Numerical display with a pair of “nudge” buttons	
slider	slider	Orientation: horizontal | vertical

Widget Identifier	Suggested Rendering	Suggested Client Hints
wheel	Thumbwheel	Orientation: horizontal | vertical
fader	fader	Type: audio | video
date-picker	Calendar view with date picker	
time-picker	Time display with ability to set time	Clock: 12-hour | 24-hour Hours: on | off Minutes: on | off Seconds: on | off
eq-graph	A line graph showing gain in dB vs frequency with dots to show the center/cut frequency of the various members of the eq.	#/$defs/eq_graph
line-graph	A line graph showing the history of a parameter value	#/$defs/line_graph
button	A push button	
ip-v4	4-decimal number field data entry with validation that range is 0 to255 for each field.	
ip-v6	8-hex number field data entry with validation that range is 0 to 0xFFFF for each field.	
progress	Progress Meter	Orientation vertical | horizontal
level-meter	Audio meter	#/$defs/level_meter
positioner	Movable crosshairs within a rectangle (only one crosshair is possible within a rectangle)	
joystick	Joystick	
scroll	Scroll bar	Orientation: vertical | horizontal

Annex B
(Informative)

Additional Elements

B.1 Repository

Interface Definitions based on the model shown in this document are companion elements to this document. These can be found in the public repository covering all parts of the 2138 suite of documents at the following URL:

https://github.com/SMPTE/st2138-a.git

Navigate to the /interface/proto folder for the interface definitions in Protobuf 3.0.

The schemata can be found at /interface/schemata/device.yaml, but are published at the following URL: https://smpte.github.io/st2138-a/interface/schemata/device.yaml which is also the $id tag for the schemata.

The contents of st2138-a.git are normative.

The software version corresponding to this document revision is tagged as v1.0.0-pcd.

B.2 Root Directory

Throughout this document, the value of the symbol $(ROOT) is the directory into which the test materials are cloned from the repository.

Bibliography

IS-04 NMOS Discovery and Registration Specification, Advanced Media Workflow Association, v1.3.3, https://specs.amwa.tv/is-04/