INTERNATIONAL STANDARD

ISO/IEC 24751-4

First edition
2023-02

Information technology — Individualized adaptability and accessibility in e-learning, education and training —

Part 4:
"Access for all" framework for individualized accessibility and registry server application programming interface (API)

Reference number
ISO/IEC 24751-4:2023(E)

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 or www.iec.ch/members_experts/refdocs).

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 https://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. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 36, *Information technology for learning, education and training*.

This first edition cancels and replaces ISO/IEC TS 24751-4:2019, which has been technically revised.

The main changes are as follows:

— requirements on registries and concept submissions have been added to the scope;

— new terms have been defined in Clause 3;

— new clauses on the Access for All framework (Clause 6) and the concept registry (Clause 7) have been added.

A list of all parts in the ISO/IEC 24751 series can be found on the ISO and IEC websites.

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 and www.iec.ch/national-committees.

Introduction

This document introduces the AccessForAll (AfA) concept registry as warranted by the AfA framework whose basic principles are introduced in subclause 6.2.

The AfA framework follows an inclusive approach to participation in life by individuals. This inclusivity focuses on addressing the usability of all resources based on individual requests and requirements which would thus reverse the current practice of favouring the majority.

Typically, services and products are designed for the largest possible customer base. This leaves many marginal, or minority needs unmet. Nevertheless, many products and services can be adapted to accommodate individual requirements. The expression of unmet requirements can also prompt the diversification of available resources, increasing the pool of choices available to individuals.

The AfA approach recognizes that anyone can experience a mismatch between personal needs and a resource (including service, environment, experience or product). The approach addresses accessibility for persons with disabilities as integral to the spectrum of human diversity, and an impetus for more flexibility, responsiveness, and inclusion for everyone.

Everyone can benefit from designs that match their personally unique needs and preferences. Inclusive systems respect individual needs and preferences.

For individualized accessibility, the needs and preferences of individual users need to be described in a concise and machine-readable manner. User interfaces and their components can then read such personal AfA preference statements and accommodate them in their adaptations. In addition, other aspects of the context of use need to be described so that user interfaces and their components can take the user's tasks, their equipment and their environment into account. Also, user interface resources need to be described so that AfA services can identify the most appropriate resources for a specific context of use.

For all descriptions, vocabularies are instrumental to allow for a strict semantic and machine-readability. Therefore, this document specifies an AfA concept registry for AfA concepts for the description of AfA preference statements, other aspects of the context of use and user interface resources. For each AfA concept, a concept record contains a globally unique identifier and other characteristics of the AfA concept.

An AfA concept registry needs to be globally accessible through a well-defined API and format rules need to exist for the exchange of AfA concept records. This document specifies a RESTful API for an AfA concept registry service (a.k.a. registry server) and a JSON format for AfA concept records to be exchanged through the AfA concept registry API.

The following use cases are meant to illustrate the benefits of the AfA framework and its standardized AfA concept registry including API and concept record format. This list of use cases is not meant to restrict further uses of this document in any way.

— A person using an AfA concept registry (e.g., a developer of an assistive technology solution) registers an AfA concept on a registry server. This can be facilitated by either a web interface of the registry or by a third-party development application (e.g., an integrated development tool) running on the person's computer. The third-party development application has some advantages over the web interface since it allows for a tighter integration of development platform and registry server. It requires the definition of a concept registry API and of the format of AfA concept record.

— An infrastructure component (e.g., a tool for setting up AfA preference statements or an AfA service) looks up an AfA concept on a registry server. Thus, the definition of an AfA concept can be presented to the user or the range of allowed values of an AfA concept can be considered for the identification of matching AfA resources.

— A syntax checker (e.g., special lint tool) verifies the contents of a new AfA preference statement by validating against the AfA concept records on a registry server. By this procedure, invalid values for

AfA concepts can be detected. In case of an invalid AfA preference statement, the syntax checker can notify the user about the error or make automatic corrections.

NOTE 1 A syntax checker could be part of a service managing AfA preference statements, checking every incoming AfA preference statement for syntax errors before storing it.

— Two services managing AfA preference statements synchronize their AfA preference statements. This could be a full synchronization over all contained AfA preference statements, or it could affect only a part of the statements. To avoid the distribution of invalid content, incoming statements can be verified against the AfA concept records of an AfA concept registry server (e.g., to detect invalid values), and erroneous statements can be skipped or automatically corrected.

— Two AfA concept registry servers synchronize their entries. This could be a full synchronization over all contained AfA concept records or it could affect only a part of the entries.

NOTE 2 It is possible to run multiple registry servers globally. Some organizations have built, for security and privacy reasons, self-contained digital infrastructures (such as intranets) with only very few and well-defined gateways to the external internet. Such organizations would possibly prefer to have their own registry server running in their infrastructure, and have it synchronize with some global registry server in a secured way from time to time. Also, organizations that develop adaptive user interfaces or assistive technology solutions would likely want to have their own registry server for experimentation purposes.

Information technology — Individualized adaptability and accessibility in e-learning, education and training —

Part 4: "Access for all" framework for individualized accessibility and registry server application programming interface (API)

1 Scope

This document specifies an AfA concept registry service, in particular:

— requirements on registries;

— requirements on concept submissions;

— a data format (in JSON) for the exchange of registry entries (AfA concept records) between registry servers;

— a set of RESTful operations for AfA concept registry servers to allow for the manipulation of AfA concept registry entries by external clients other than server-internal web interfaces.

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 10646, *Information technology — Universal coded character set (UCS)*

IETF BCP 47, *Tags for Identifying Languages*

IETF RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*

IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*

IETF RFC 3987, *Internationalized Resource Identifiers (IRIs)*

IETF RFC 7159, *The JavaScript Object Notation (JSON) Data Interchange Format*

IETF RFC 7231, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*

3 Terms and definitions

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

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

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

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

3.1
access for all
AfA
approach to providing accessibility in a computer-mediated environment in which the digital resources and their method of delivery are matched to the needs and preferences of the user

[SOURCE: ISO/IEC 2382-36:2019, 3.8.1]

3.2
AfA concept
globally identifiable unique combination of characteristics relating to context, individual preference(s), and resource(s) relevant for the personalized interaction for *AccessForAll* (3.1) and *individualized accessibility* (3.12)

Note 1 to entry: The registration of an AfA concept captures its unique identifier, label(s), definitions(s), domain(s), and value space in an AfA concept record.

Note 2 to entry: The defined terms "AfA" and "individualized accessibility" incorporate the concepts of individualized needs and matching of resources.

3.3
AfA concept record
data or file associated with an *AfA concept* (3.2) in an *AfA concept registry* (3.4)

EXAMPLE AfA concept identifier, its label(s), definition(s), domain(s), and value space.

3.4
AfA concept registry
electronic directory of *AfA concept records* (3.3)

3.5
AfA concept value space
component of an *AfA concept record* (3.3) providing the specifications of the values associated with an *AfA concept* (3.2)

3.6
AfA context concept
situation or circumstance for which an *AfA preference statement* (3.8) applies

Note 1 to entry: A minimal set of AfA preference concepts can be a single preference value.

3.7
AfA preference concept
personal set of values based on a sub-set of registered *AfA concepts* (3.2)

Note 1 to entry: The collection of AfA preference concepts provides the potential choices of preference values that can be declared by an individual in an AfA preference statement.

3.8
AfA preference statement
user defined set of AfA *preference concepts* (3.7), and where appropriate, incorporating the *AfA context concept* (3.6) and resources to which the set applies

Note 1 to entry: A minimal set of AfA preference concepts can be a single preference value.

3.9
AfA resource concept
AccessForAll (3.1) approach using an *AfA concept* (3.2) to deliver a resource that satisfies the user's requirements

Note 1 to entry: A resource can include but is not limited to activities, configurations, content, environment, experience, interface, product, service, technology.

3.10
AfA concept registry server
AfA service (3.11) running an *AfA concept registry* (3.4)

3.11
AfA service
system that supports individuals in determining and declaring personal *AfA preference concepts* (3.7) found in *AfA preference statements* (3.8), and delivers the matching resource

3.12
individualized accessibility
<e-learning> facility of an IT system-based learning environment to address the needs of an individual as learner (through adaptation, reaggregation and substitution)

Note 1 to entry: Accessibility is determined by the flexibility of the education environment (with respect to presentation, control methods, structure, access model and learner supports) and the availability of equivalent content deemed to be adequate alternatives.

[SOURCE: ISO/IEC 2382-36:2019, 3.8.7]

4 Symbols and abbreviated terms

AfA	AccessForAll
API	application programming interface
IRI	internationalized resource identifier (IETF RFC 3987)
HTTP	hypertext transfer protocol (IETF RFC 7230, IETF RFC 7231)
HTTPS	HTTP secure (IETF RFC 7230, IETF RFC 2731)
JSON	JavaScript object notation (IETF RFC 7159)
MIME	multi-purpose internet mail extensions (IETF RFC 2046)
REST	representational state transfer [10]
URI	uniform resource indicator (IETF RFC 3986)

5 Conformance

A conforming *AfA concept registry* shall have the following characteristics:

— it shall meet all requirements on registries, as specified in subclause 7.2;

— it shall accept only concept submissions that meet all requirements, as specified in subclause 7.3.

A conforming *AfA concept registry server* shall meet all of the following requirements:

— REST architecture, as specified in Clause 8;

— The JSON mapping for the registry API, as specified in Clause 9.

A conforming *AfA concept registry server* may support additional format mappings for the registry API.

NOTE The requirements and recommendations for a registry server in this document apply to its API and data formats for parameters in the API. Nothing in this document is meant to constrain the design, data structure and implementation of the internal parts of a registry server, including its database of AfA concepts.

6 Access for All framework

6.1 General

This document assumes an AccessForAll (AfA) framework that enables:

— individuals to discover, explore, refine, prioritize, and request their personally unique preference statements, for a given context;

— developers to attach metadata to resources to facilitate matching to AfA preference statements;

— AfA services to satisfy the individual preferences by delivering a matching resource, e.g, by finding, assembling, developing, recruiting, or using other strategies to deliver products, services, configurations, or environments that match the preferences.

AfA is in large part motivated by the goal of addressing currently unmet or poorly met preferences (where the term *preferences* encompasses the range from critical needs to preferred configuration). This goal is in part satisfied by registries, which are open to any individual or organization, to register possible AfA concepts, including concepts associated with unanticipated, emerging, or new preferences. These registered concepts might then be used to express individual preferences within a broad diversity of AfA implementing services.

An AfA concept registry is intended to provide a living directory of possible AfA concepts (preference, context and resource concepts).

NOTE An AfA concept registry is not used for individuals to declare or store personal preferences or personal preference sets (AfA preference statement). For this purpose, a user-context service (a.k.a. preference server) is specified in ISO/IEC 24752-8.

6.2 Basic principles

The AfA framework is designed with the following basic principles in mind:

— **Description of resource functionality, not users.** The AfA approach recognizes that the preferences of a person are a declaration of functional requirements or preferences relative to resources. They are not a description of a personal deficit, medical condition, or disability. An AfA approach gives priority to the individual, and their understanding of their own preferences.

— **Flexible use of registered AfA concepts.** AfA concepts shall be freely available for use in multiple ways, including to create preference statements (using AfA preference concepts), describe contexts where preference statements apply (using AfA context concepts), and describe resources for matching preferences (using AfA resource concepts).

— **Extensibility.** The AfA approach enables practices that continuously improve the matching of resources to users' individual preferences in general or in specific contexts. It does not limit the diversity of preference concepts, associated human understandable labels, or strategies for addressing the preferences. An AfA concept registry supports the inclusion of preference concepts not previously considered.

— **Accessibility of Processes and Services.** The AfA approach is in part intended to address the lack of representation and the barriers to participation experienced by individuals with minority or uncommon needs and preferences. This exclusion can occur even in processes or services specifically intended to meet the needs of underserved individuals. To break the cycle of exclusion and lack of participation, AfA processes and services must themselves adhere to individualized accessibility principles and practices.

— **Internationalization:** An AfA concept may have multiple human-readable labels (in an unlimited number of languages) but is still globally identifiable by its IRI. On the example of preferences, more than one person may have the same requirement in mind, but they may use different languages and labels to refer to the same preference concept.

6.3 Use cases

AfA concepts are used by AfA services in terms of:

— **AfA preference concepts:** Enable individuals to create AfA preference statements or requests, (this includes discovering, exploring, evaluating and refining an understanding of the preferences that work best for them in a given context to achieve a given goal);

— **AfA context concepts:** Identify situations and circumstances for which AfA preference statements apply;

— **AfA resource concepts:** Identify resources that match AfA preference concepts (including, but not limited to, through the application of resource metadata);

— **Match making:** Match resources to personally declared AfA preference statements.

6.4 Preference concept

AfA preference concepts are possible user preferences that an individual may choose to express personal preferences through an AfA preference statement. The extensible cataloguing of possible preferences that a user may wish to express will help to guide producers and developers in addressing currently unmet user requirements. This in turn supports innovation and greater inclusion of individuals that are currently excluded or marginalized.

A preference concept's type is "PreferenceStatement".

6.5 Resource concept

AfA resource concepts describe candidate resources for matching or satisfying with an AfA preference statement in a specific context. Candidate resources are user interface resources such as user interface components, symbols, images, subtitles, audio descriptions, or custom settings. Resource concepts can also be used to describe specific techniques and processes for adaptations.

An AfA service that matches preference statements to resources in a specific context can make use of resource concepts.

A resource concept's type is "ResourceDescription".

6.6 Context concept

AfA context concepts describe situations and circumstances of user interface usages. They can be used to identify whether a specific preference statement applies in a specific context or not. They can also be used to specify the applicability of resources and resource concepts.

A context concept's type is "ContextDescription".

7 Concept registry

7.1 General

The primary purpose of an AfA concept registry is twofold:

— To enable the extensibility of AfA concepts and the registration of new or unanticipated concepts;

— To support clarity and interoperability of AfA concepts, as used by users of user interfaces and suppliers and producers of resources and AfA services.

AfA concept registries should be distinguished from other AfA services. AfA concept registries do not deliver matches for personal preferences or the means for creating and storing AfA preference statements.

An AfA concept registry will gather, store, maintain and publish AfA concepts, their definitions, the different labels used to refer to them, an assigned identifier (IRI), as well as other optional information associated with a preference concept.

7.2 Requirements on registries

Registries shall accept all registrations of AfA concepts and additional information that are submitted in good faith. Published registry policies may define and publish a process for determining 'in good faith'.

A registry shall allow free, remote, electronic interrogation of the registry by any individuals or organisations and allow free use of AfA concepts. Registry policies may limit inappropriate access, but this shall be published in the policy and shall be consistent with the provisions in this document.

Registrations shall be enabled online, such as by e-mail or via a Web-based form, and may also be accepted by facsimile or paper copy according to the policy of the particular registry.

Where a contribution is rejected for some reason, and the person making the submission is contactable, the registry authority shall notify the submitter including the reason why the contribution is rejected and provide an opportunity to appeal the rejection (unless the rejection was due to technical reasons, e.g., syntax error or missing information). Published registry policies shall define an appeals process where rejections are anticipated.

Where registries include policies for approval/rejection of submissions based on their content, these shall be defined and published in accessible forms.

Registries shall provide security for all AfA concepts and other registrations and shall publish their security policies. This includes frequency and means of backup, archival, continuity and succession plans.

As these registries are intended to serve the purpose of improving accessibility, registries that have a human interface shall ensure those user interfaces are accessible.

It is assumed that many using the registries will not be technology specialists, so it is expected that registries will support users with plain language applications to help them use the registry. In some cases, such assistance may be very comprehensive.

7.3 Requirements on concept submissions

A concept submission shall include the following information items:

1) a globally unique identifier (*IRI*) for the concept (as specified by IETF RFC 3987);

2) natural language names (*termLabels*) for the concept in at least one natural language;

3) natural language *definitions* (i.e., descriptions) of the Concept in at least one natural language;

4) contact information (*owner*) for the submitter (to facilitate addressing any issues with the submission);

5) type of concept: "PreferenceStatement", "ContextDescription", or "ResourceDescription".

A concept submission may include the following information items:

6) value space (*valueSpace*) –the potential values associated with the preference concept;

7) relation to other preference concepts (*transformationOf*, *refines*);

8) *notes;*

9) *examples.*

In all cases, natural language shall be accompanied by a language indicator. An IRI shall be assigned to each registration of a new AfA concept.

8 REST architecture

8.1 General

The interface specified in this document is based on the representational state transfer (REST) architecture (Fielding, 2000; Richardson et al., 2015). Hereby, an established relationship between a resource service and a client is provided that is easy to implement, test and maintain on both sides.

According to REST, HTTP methods (see IETF RFC 7231) are used to manipulate REST resources which are located on a resource service:

— POST creates a new resource with a new URI.

— GET retrieves a resource.

— PUT modifies a resource.

— DELETE deletes a resource.

NOTE These methods are called "HTTP methods" according to IETF RFC 7231 but are applicable for HTTPS as well. In fact, the use of HTTPS offers a more secure protocol for service implementations (see 6.4).

Each operation has one or multiple request parameters, and one or multiple response parameters, as specified in 8.2 to 8.7.

8.2 Request and response parameters

The request and response parameters for an operation in this subclause are made up of information items, each with a specific type. Complex types are Object and Array, primitive types are String, Number, Boolean, URI (as specified in IETF RFC 3986) and the *null* value (as specified in IETF RFC 7159).

Each request and response parameter, as used for the operations in this subclause, shall have one of four different categories:

1) **URI path.** The parameter is submitted as URI path in a request (as specified in IETF RFC 3986).

   EXAMPLE 1 In the operation "PUT /api/record/C12345", the parameter *conceptId* (value "C12345") is contained in the path of the URI.

2) **URI query parameter.** The parameter is submitted as one of the query parameters of the URI (as specified in IETF RFC 3986).

   EXAMPLE 2 In the operation "GET /api/records?type=PreferenceStatement&offset=1&limit=1", the parameters *type* (value "PreferenceStatement"), *offset* (value 1) and *limit* (value 1) are URI query parameters.

3) **HTTP header field.** The parameter is submitted as content of an HTTP header field (as specified in IETF RFC 7231).

   EXAMPLE 3 In the following HTTP response header, the parameter *record* (value "https://example.com/api/record/R12345") is submitted as content of the field "Location".

```
HTTP/1.1 200 OK
Location: https://example.com/api/record/R12345
```

4) **Message body.** The parameter is submitted as content of the request/response body, whereby the body's MIME type is specified through the `Content-Type` HTTP header field (as specified in IETF RFC 7231).

EXAMPLE 4 In the following HTTP response body (JSON format), the parameter *totalRows* (value 1) is submitted as the field "totalRows" in the unnamed *response* object.

```
{
    "ok": true,
    "totalRows": 1
}
```

In addition to the request and response parameters specified in this subclause, other (proprietary) request and response parameters may be added to any operation, as long as they do not interfere with the parameters as specified in this document. Also, additional (proprietary) information may be added to the parameters specified in this document, as long as they do not interfere with the parameter's content as specified in this document. If a registry server or client does not know the meaning of such proprietary information, it shall ignore it.

8.3 Response codes

Format and semantics of HTTP response codes shall adhere to IETF RFC 7231.

NOTE 1 Clause 7 lists only the most important HTTP codes for every operation. Nevertheless, a registry server can make use of the full range of HTTP response codes as listed in IETF RFC 7231.

In case of an error response code, the registry server shall return a textual description of the error as response body (see parameter *error message* in the response bodies in Clause 7).

NOTE 2 Textual descriptions of error messages are important for debugging. They include all relevant information for developers sending requests to the registry server.

8.4 Security considerations

A registry server should take measures for a level of security that is appropriate for the specific sensitivity of the data. The employed security mechanisms should be based on state-of-the-art technologies.

A registry service should provide an HTTPS-based interface (as specified in IETF RFC 7230) for all operations. For sensitive resources and operations, no HTTP-based interface should be provided.

A client should use HTTPS for communication with a resource service.

8.5 Authentication

A registry server may apply access restrictions to its operations, based on client privileges.

EXAMPLE A service can store ownership information for every REST resource. For example, only owners (creators) of a REST resource are allowed to modify and delete it.

If a registry server requires authentication, basic authentication (as specified in IETF RFC 2617) or an authentication mechanism with an equal or higher security level should be supported.

8.6 MIME type

For all operations in Clause 7, the following requirements shall be met:

— A service shall support request bodies in the JSON format. The format (MIME type, see IETF RFC 2046) of the request body is indicated by the `Content-Type` field value in the HTTP request header (as specified in IETF RFC 7231).

— A service shall indicate its response format (MIME type, as specified in IETF RFC 2046) by the `Content-Type` field value in the HTTP response header (as specified in IETF RFC 7231).

— A service shall respect a client's preferred response body format (MIME type, as specified in IETF RFC 2046), as specified by the `Accept` field value in the HTTP request header (as specified in IETF RFC 7231). At a minimum, a service shall support the JSON mapping, as specified in Clause 7.

A service may support additional formats for requests and responses. In this case, the additional formats shall be specified by different MIME types (not listed in this document).

8.7 Other general requirements

For all operations in Clause 7, the following requirements shall be met:

— If used inside a URI for a GET request, reserved characters (as specified in RFC 3986:2005, section 2.2) shall be percent-encoded (as specified in IETF RFC 3986:2005, section 2.1).

— The encoding of request and response shall be UTF-8 (as specified in ISO/IEC 10646).

— The header fields of request and response shall comply with IETF RFC 7231.

9 Registry API (JSON mapping)

9.1 General

For JSON-formatted requests and responses of a registry server, the requirements in this clause shall be met.

The message body (if applicable) and response body (if applicable) shall comply to JSON (as specified in IETF RFC 7159), except for error responses which shall be plain text.

The MIME type (as specified in IETF RFC 2046) for the JSON format shall be "application/json".

The *null* value shall be mapped to the JSON null value (as specified in IETF RFC 7159).

The root object of any request body shall be an unnamed object (hereafter referred to as the *request* object).

The root object of any response body shall be an unnamed object (hereafter referred to as the *response* object).

The MIME type (as specified in IETF RFC 2046) for an error response message (in plain text) shall be "text/plain".

Where types are specified for parameters, these shall refer to IETF RFC 7159.

For any request or response body defined in this clause, proprietary information may be added as additional parameters, or extra fields of an object, even within a nested object, as long as the additional parameters and fields do not interfere with the information items specified in this clause.

9.2 Concept record (JSON format)

For the JSON representation of a *concept record* object, when used as a parameter in the registry server's API, the following shall apply:

— A *concept record* object shall be a JSON object.

— *Concept record* shall have a `conceptId` member (String without reserved characters for URIs, see IETF RFC 3986), carrying a locally (i.e. within a registry server) unique identifier for the concept. `conceptId` shall be immutable, i.e. it cannot be modified in an update operation (see 9.4).

NOTE 1 The concatenation of the registry server domain, the path for getting a concept record (see 9.6) and the `conceptId` yields a globally unique identifier (URI, cf. IETF RFC 3986) for a concept.

EXAMPLE 1 A concept with `conceptId` “common_fontSize”, stored on a registry server with (sub-) domain name “terms.gpii.eu”, would have the globally unique URI “https://terms.gpii.eu/api/record/common_fontSize” (assuming that the HTTPS protocol is used for accessing the registry server).

— *Concept record* shall have a `type` member (String), carrying either one of the following values: “PreferenceStatement”, “ContextDescription”, “ResourceDescription”. `type` shall be immutable, i.e., it cannot be modified in an update operation (see 9.4).

— *Concept record* shall have a `subtype` member (String), carrying either one of the following values: "term", "transform". `subtype` shall be immutable, i.e., it cannot be modified in an update operation (see 9.4).

— *Concept record* may have an `origin` member (String), carrying either one of the following values: "common", "application-specific", or any other value. `origin` shall be immutable, i.e., it cannot be modified in an update operation (see 9.4).

— *Concept record* shall have a `definition` member (Array). The array shall have one or more unnamed objects, each with two members: `language` and `value`. The `language` member shall identify a language (as specified in IETF BCP 47) which may be *null*. The `value` member shall bear a textual definition (String) of the concept in the pertinent language. `definition` may contain multiple objects referring to the same language.

— *Concept record* shall have a `termLabel` member (Array). The array shall have one or more unnamed objects, each with two members: `language` and `value`. The `language` member shall identify a language (as specified in IETF BCP 47) which may be null. The `value` member shall bear a linguistic expression (String) for the concept in the pertinent language. `termLabel` may contain multiple objects referring to the same language.

— *Concept record* shall have a `datatype` member (Object), carrying either one of the following values: "Boolean", "Number", "String". `datatype` shall be immutable, i.e., it cannot be modified in an update operation (see 9.4).

— *Concept record* shall have an `owner` member, carrying the contact information of the owner(s) of the concept, to facilitate addressing any issues with the record. The type of the `owner` member is implementation specific.

NOTE 2 Initially, the submitter of a record is the owner. The submitter can extend the ownership to other parties, or pass it on to other parties, in which case the owner member is updated.

— *Concept record* may have a `valueSpace` member (Object), carrying a formal data type definition such as the JSON Schema format (as specified in JSON Schema, JSON Schema Validation, JSON Hyper-Schema). `valueSpace` shall be immutable, i.e., it cannot be modified in an update operation (see 9.4).

— *Concept record* may have a `transformationOf` member (Array). If present, the `transformationOf` array shall consist of one or more Strings, each representing a globally unique identifier (conceptId) of a concept record.

— *Concept record* may have a `refines` member (Array), carrying the identifiers of other concepts that *concept* refines. If present, the `refines` array shall consist of one or more Strings, each representing a globally unique identifier (`conceptId`) of a concept.

— *Concept record* may have a `domains` member (Array). The array shall have one or more unnamed objects, each with two members: `language` and `value`. The `language` member shall identify a language (as specified in IETF BCP 47) which may be *null*. The `value` member shall carry a textual description of the domain / application area (String) on the concept in the pertinent language. `domains` may contain multiple objects referring to the same language.

— *Concept record* may have a `notes` member (Array). The array shall have one or more unnamed objects, each with two members: `language` and `value`. The `language` member shall identify a language (as specified in IETF BCP 47) which may be null. The `value` member shall carry a textual note (String) on the concept in the pertinent language. `notes` may contain multiple objects referring to the same language.

— *Concept record* may have an `examples` member (Array). The array shall have one or more unnamed objects, each with two members: `language` and `value`. The `language` member shall identify a language (as specified in IETF BCP 47) which may be null. The `value` member shall bear a textual representation (String) of an example for the concept in the pertinent language. `examples` may contain multiple objects referring to the same language.

— The order of the members in *concept record* shall not be significant.

— *Concept record* may include additional members, as long as they do not interfere with the members specified in this subclause. If additional members carry a human-readable expression, a language code should be added (which may be null).

NOTE 3 The above list of *concept record* members is based on a concept's "basic attributes", as specified in ISO/IEC 24751-1.

EXAMPLE 2 A concept record in JSON format.

```
{
   "conceptId": "http://example.com/api/record/C12345",
   "type": "NeedAndPreference",
   "subtype": "term",
   "origin": "common",
   "definition": [
      {
         "language": "en",
         "value": "Font size in points"
      },
      {
         "language": "fr",
         "value": "Taille des caractères en points"
      }
   ],
   "termLabel": [
      {
         "language": "en",
         "value": "font-size"
      }
   ],
   "datatype": "Number",
   "valueSpace": {
      "$schema": "http://json-schema.org/draft-04/schema#",
      "description": "JSON schema for font-size type",
      "type": "integer",
      "minimum": 1,
      "maximum": 100
   },
   "transformationOf": [
      "C22222",
      "C33333"
   ],
   "refines": [
      "C44444",
      "C55555"
   ],
   "notes": [
      {
         "language": "en",
         "value": "This record was created as an example."
      }
   ],
   "examples": [
      {
         "language": "en",
         "value": "Font size applies to computers, tablets, smartphones, kiosks, etc."
      }
   ],
   "owner": [
      "Max Mustermann"
   ]
}
```

NOTE 4 The following JSON schema is available for automatic validation of a concept record in JSON format: http://openurc.org/ns/creg/concept-record.schema.json.

9.3 CREATE concept record

For creating a new record of a concept, a registry shall implement the method **POST** on endpoint **/api/record**, with the request parameters as listed in Table 1, and response codes and parameters as listed in Table 2.

Table 1 — Request parameters for creating a concept record

POST /api/record			
Parameter name	**Category**	**Type**	**Description**
concept record (mandatory)	request body	*concept record* object (see 9.2)	*concept record* shall be the (unnamed) *request* object. *conceptId* may be missing or empty, in which case the server shall provide this information in its response.

EXAMPLE 1 A CREATE concept record request in JSON format.

```
POST /api/record HTTP/1.1
Content-Type: application/json
Accept: application/json

{
   "type": "NeedAndPreference",
   "subtype": "term",
   "origin": "common",
   "definition": [
      {
         "language": "en",
         "value": "Font size in points"
      }, {
         "language": "fr",
         "value": "Taille des caractères en points"
      }
   ],
   "termLabel": [
      {
         "language": "en",
         "value": "font-size"
      }
   ],
   "datatype": "Number",
   "valueSpace": {
      "$schema": "http://json-schema.org/draft-04/schema#",
      "description": "JSON schema for font-size type",
      "type": "integer",
      "minimum": 1,
      "maximum": 100
   },
   "owner": [ "Max Mustermann" ]
}
```

NOTE 1 The following JSON Schema is available for automatic validation of a CREATE concept record request in JSON format: http://openurc.org/ns/creg/CREATE-concept-record.request.schema.json.

Table 2 — Response codes and parameters for creating a concept record

Response code: 201 Created (Success)			
Parameter name	**Category**	**Type**	**Description**
conceptUri (mandatory)	HTTP header field "Location"	URI	Unique URI for the created concept record on the server. It shall have the following format: "https://host/api/record/conceptId" (with *host* being the domain name and optional port, and *conceptId* being an implementation-specific unique ID for the concept record).
concept record (mandatory)	response body	*concept record* object (see 9.2)	*concept record* shall be the value of the member record in the response object.
Response code: 400 Bad Request (Invalid concept)			
error message	entire response body	String	Human readable text describing the error and its cause.
Response code: 404 Not Found			
error message	entire response body	String	Human readable text describing the error and its cause.

EXAMPLE 2 A CREATE concept term record response in JSON format.

```
HTTP/1.1 201 Created
Location: http://example.com/api/record/R12345
Content-Type: application/json

{
   "record": {
      "conceptId": "R12345",
      "type": "NeedAndPreference",
      "subtype": "term",
      "origin": "common",
      "definition": [
       {
         "language": "en",
         "value": "Font size in points"
       },
       {
         "language": "fr",
         "value": "Taille des caractères en points"
       }
     ],
      "termLabel": [
       {
         "language": "en",
         "value": "font-size"
       }
     ],
      "datatype": "Number",
      "valueSpace": {
         "$schema": "http://json-schema.org/draft-04/schema#",
         "description": "JSON schema for font-size type",
         "type": "integer",
         "minimum": 1,
         "maximum": 100
         },
      "owner": [ "Max Mustermann" ]
   }
}
```

NOTE 2 The following JSON Schema is available for automatic validation of a CREATE concept record response in JSON format: http://openurc.org/ns/creg/CREATE-concept-record.response.schema.json.

9.4 UPDATE concept record

For updating a record of a concept, a registry shall implement the method **`PUT`** on endpoint **`/api/record/`***`conceptId`*, with the request parameters as listed in Table 3, and response codes and parameters as listed in Table 4.

Table 3 — Request parameters for creating a concept record

PUT **`/api/record/`***`conceptId`*			
Parameter name	**Category**	**Type**	**Description**
conceptId	URI path	String	Server-unique identifier of an existing concept record.
concept record (mandatory)	request body	*concept record* object (see 9.2)	*concept record* shall be the (unnamed) *request* object. All contents of the existing concept with *conceptId* on the server will be overwritten by the contents of request parameter *concept record*.

EXAMPLE 1 An UPDATE record request in JSON format.

```
PUT /api/record/R12345 HTTP/1.1
Content-Type: application/json
Accept: application/json

{
   "conceptId": "http://example.com/api/record/R12345",
   "type": "NeedAndPreference",
   "subtype": "term",
   "origin": "common",
   "definition": [
     {
       "language": "en",
       "value": "Font size in points"
     },
     {
       "language": "fr",
       "value": "Taille des caractères en points"
     },
     {
       "language": "de",
       "value": "Schriftgröße in Punkten"
     }
   ],
   "termLabel": [
     {
       "language": "en",
       "value": "font-size"
     }
   ],
   "datatype": "Number",
   "valueSpace": {
      "$schema": "http://json-schema.org/draft-04/schema#",
      "description": "JSON schema for font-size type",
      "type": "integer",
      "minimum": 1,
      "maximum": 100
   },
   "owner": [ "Max Mustermann" ]
}
```

NOTE 1 The following JSON Schema is available for automatic validation of an UPDATE concept record request in JSON format: http://openurc.org/ns/creg/UPDATE-concept-record.request.schema.json.

Table 4 — Response codes and parameters for updating a concept record

Response code: 200 OK			
Parameter name	**Category**	**Type**	**Description**
conceptUri (mandatory)	HTTP header field "Location"	URI	Unique URI for the updated concept record on the server. It shall have the following format: "https://host/api/record/conceptId" (with *host* being the domain name and optional port, and *conceptId* being an implementation-specific unique ID for the concept record).
concept record (mandatory)	response body	*concept record* object (see 9.2)	*concept* shall be the value of member `record` of the *response* object.
Response code: 400 Bad Request (Invalid parameters)			
error message	entire response body	String	Human readable text describing the error and its cause.
Response code: 404 Not Found			
error message	entire response body	String	Human readable text describing the error and its cause.

EXAMPLE 2 An UPDATE concept record response.

```
HTTP/1.1 200 OK
Location: /api/record/R12345
Content-Type: application/json

{
   "record": {
      "conceptId": "http://example.com/api/record/R12345",
      "type": "NeedAndPreference",
      "subtype": "term",
      "origin": "common",
      "definition": [
         {
            "language": "en",
            "value": "Font size in points"
         },
         {
            "language": "fr",
            "value": "Taille des caractères en points"
         },
         {
            "language": "de",
            "value": "Schriftgröße in Punkten"
         }
      ],
      "termLabel": [
         {
            "language": "en",
            "value": "font-size"
         }
      ],
      "datatype": "Number",
      "valueSpace": {
         "$schema": "http://json-schema.org/draft-04/schema#",
         "description": "JSON schema for font-size type",
         "type": "integer",
         "minimum": 1,
         "maximum": 100
      },
      "owner": [
         "Max Mustermann"
      ]
   }
}
```

NOTE 2 The following JSON Schema is available for automatic validation of an UPDATE concept record response in JSON format: http://openurc.org/ns/creg/UPDATE-concept-record.response.schema.json.

9.5 DELETE concept record

NOTE 1 This operation is optional, since a registry server can opt to not implement the delete operation on concept records for reasons of traceability and URL stability.

For deleting a record of a concept, a registry may implement the method **`DELETE`** on endpoint **`/api/record/`***`conceptId`*, with the request parameters as listed in Table 5, and response codes and parameters as listed in Table 6.

NOTE 2 The internal result of this operation in the registry is implementation-specific. A registry can flag a concept record for deletion (so it can be recovered later) or remove it instantly.

If a registry does not implement the method **`DELETE`**, it should return the response code 404 on requests to the endpoint **`/api/record`**.

Table 5 — Request parameters for deleting a concept record

DELETE **`/api/record/`***`conceptId`*			
Parameter name	**Category**	**Type**	**Description**
conceptId	URI path	String	Server-unique identifier of an existing concept record.

EXAMPLE 1 A DELETE concept record request. Note that no request body is supplied.

```
DELETE /api/record/R12345 HTTP/1.1
Accept: application/json
```

Table 6 — Response codes and parameters for updating a concept record

Response code: 204 No Content (Deletion enacted)			
Parameter name	**Category**	**Type**	**Description**
(no parameter)			
Response code: 404 Not Found (Invalid ID)			
error message	entire response body	String	Human readable text describing the error and its cause.

EXAMPLE 2 A DELETE concept record response. Note that proprietary content could be supplied as response body.

```
HTTP/1.1 204 No Content
Content-Type: application/json
```

9.6 GET concept record

For retrieving a record of a concept, a registry shall implement the method **`GET`** on endpoint **`/api/record/`***`conceptId`*, with the request parameters as listed in Table 7, and response codes and parameters as listed in Table 8.

Table 7 — Request parameters for retrieving a concept record

GET **`/api/record/`***`conceptId`*			
Parameter name	**Category**	**Type**	**Description**
conceptId	URI path	String	Server-unique identifier of an existing concept record.

EXAMPLE 1 A GET concept record request. Note that there is no request body.