TECHNICAL SPECIFICATION

ISO/IEC TS 24751-4

First edition
2019-06

Information technology for learning, education and training — AccessForAll framework for individualized accessibility —

Part 4:
Registry server API

Reference number
ISO/IEC TS 24751-4:2019(E)

Contents

Page

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 36, *Information technology for learning, education and training*.

A list of all parts in the ISO/IEC 24751 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

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. ISO/IEC DIS 24751-1 1) introduces 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 a standardized API and AfA 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 an 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 everything 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 registry server (e.g., to detect invalid values), and erroneous statements can be skipped or automatically corrected.

— Two AfA 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.

1) 2nd Edition under development. Stage at time of publication: ISO/IEC DIS 24751-1:2018.

NOTE 2 While ISO/IEC DIS 24751-1 leaves open whether there is a single registry server or multiple registry servers, 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 for learning, education and training — AccessForAll framework for individualized accessibility —

Part 4: Registry server API

1 Scope

This document specifies an API in support of ISO/IEC DIS 24751-1. In particular, this document specifies:

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

— a set of RESTful operations for AfA 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)*

ISO/IEC/DIS 24751-1,[2)]*Information technology — Information technology for learning, education and training — AccessForAll Framework For Individualized Accessibility — Part 1: Framework and Registry*

IETF RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, November 1996*, https://tools.ietf.org/html/rfc2046

IETF RFC 7231, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, June 2014*, https://tools.ietf.org/html/rfc7231

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

IETF RFC 7159, *The JavaScript Object Notation (JSON) Data Interchange Format, March 2014*. https://tools.ietf.org/html/rfc7159

IETF BCP 47, *Tags for Identifying Languages, September 2009*. https://tools.ietf.org/html/bcp47

3 Terms and definitions

For the purposes of this document, the terms and definitions of ISO/IEC DIS 24751-1, and 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/

2) 2nd Edition under development. Stage at time of publication: ISO/IEC DIS 24751-1:2018.

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

3.1
AfA concept record
concept record
record for an AfA concept in the AfA concept registry

3.2
registry server
service running an AfA concept registry

4 Symbols and abbreviated terms

AfA	AccessForAll
API	application programming interface
HTTP	hypertext transfer protocol (IETF RFC 7230, IETF RFC 2731)
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)
URI	uniform resource indicator (IETF RFC 3986)
REST	representational state transfer (Fielding, 2000)

5 Conformance

A conforming registry server shall meet the following requirements:

— REST architecture, as specified in Clause 5;

— The JSON mapping for the registry API, as specified in Clause 7.

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

NOTE The requirements and recommendations in this document apply to a registry server's 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 REST architecture

6.1 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 6.2 to 6.7.

6.2 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.

6.3 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.

6.4 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, if available.

6.5 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.

6.6 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).

6.7 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.

7 Registry API (JSON mapping)

7.1 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.

7.2 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 7.4).

NOTE 1 The concatenation of the registry server domain, the path for getting a concept record (see 7.6) 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 7.4).

— *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 7.4).

— *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 7.4).

— *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