INTERNATIONAL STANDARD

ISO/IEC 24752-8

First edition
2018-04

Information technology — User interfaces — Universal remote console —

Part 8: User interface resource framework

Technologies de l'information — Interfaces utilisateur — Télécommande universelle —

Partie 8: Cadre de ressources pour les interfaces utilisateur

Reference number
ISO/IEC 24752-8:2018(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword ..... **v**
Introduction ..... **vi**
1 Scope **1**
2 Normative references **1**
3 Terms and definitions **2**
4 Conformance **3**
5 Use cases (informative) **4**
5.1 General 4
5.2 User interface localisation 4
5.3 User interface personalisation 4
5.4 User interface accessibility 4
5.5 User interface responsiveness 5
5.6 User interface settings 5
6 Workflow model (informative) **5**
6.1 General 5
6.2 Storing and retrieving a user-context 5
6.3 Storing and retrieving a task-context 5
6.4 Storing and retrieving an equipment-context 6
6.5 Storing and retrieving an environment-context 6
6.6 Describing resources 6
6.7 Finding and retrieving matching resources 7
6.8 Describing user interface settings 8
6.9 Finding and retrieving matching user interface settings 8
7 REST interfaces for services **8**
7.1 General 8
7.1.1 REST architecture (informative) 8
7.1.2 Request and response parameters 9
7.1.3 Response codes 9
7.1.4 Authentication 10
7.1.5 Other general requirements 10
7.2 User-context 10
7.2.1 General 10
7.2.2 CREATE user-context (mandatory) 12
7.2.3 GET user-context (mandatory) 13
7.2.4 UPDATE user-context (mandatory) 14
7.2.5 DELETE user-context (optional) 14
7.2.6 GET user-context-list (mandatory) 15
7.3 Task-context 17
7.3.1 General 17
7.3.2 CREATE task-context (mandatory) 17
7.3.3 GET task-context (mandatory) 18
7.3.4 UPDATE task-context (mandatory) 19
7.3.5 DELETE task-context (optional) 20
7.3.6 GET task-context-list (mandatory) 20
7.4 Equipment-context 22
7.4.1 General 22
7.4.2 CREATE equipment-context (mandatory) 22
7.4.3 GET equipment-context (mandatory) 23
7.4.4 UPDATE equipment-context (mandatory) 24
7.4.5 DELETE equipment-context (optional) 25
7.4.6 GET equipment-context-list (mandatory) 26
7.5 Environment-context 28

7.5.1 General ... 28
7.5.2 CREATE environment-context (mandatory) ... 28
7.5.3 GET environment-context (mandatory) ... 29
7.5.4 UPDATE environment-context (mandatory) ... 30
7.5.5 DELETE environment-context (optional) ... 31
7.5.6 GET environment-context-list (mandatory) ... 32
7.6 Resource ... 34
7.6.1 General ... 34
7.6.2 CREATE resource (mandatory) ... 34
7.6.3 GET resource-by-ID (mandatory) ... 35
7.6.4 GET resource-from-listing (mandatory) ... 36
7.6.5 UPDATE resource (optional) ... 37
7.6.6 DELETE resource (optional) ... 38
7.7 Resource-description ... 38
7.7.1 General ... 38
7.7.2 CREATE resource-description (mandatory) ... 39
7.7.3 GET resource-description-by-ID (mandatory) ... 40
7.7.4 GET resource-description-from-listing (mandatory) ... 41
7.7.5 UPDATE resource-description (mandatory) ... 41
7.7.6 DELETE resource-description (optional) ... 42
7.8 Listing ... 43
7.8.1 General ... 43
7.8.2 CREATE listing (mandatory) ... 43
7.8.3 GET listing (mandatory) ... 44
7.8.4 DELETE listing (optional) ... 45
7.8.5 CONFIRM user-rating (optional) ... 46
8 Security considerations ... 47
Annex A (normative) **Mapping for XML ... 48**
Annex B (normative) **Mapping for JSON ... 67**
Bibliography ... 86

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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).

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

For an explanation on 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 the following URL: www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 35, *User interfaces*.

A list of all parts in the ISO/IEC 24752 series can be found on the ISO website.

Introduction

Adaptive user interfaces change their presentation and behaviour according to the specific context of use, including the user's needs and preferences, their devices and environmental parameters. Such changes are to be considered at development time. At runtime, the user interface can take all aspects of the context of use into consideration.

Changes at runtime often require that some custom-tailored user interface resources be prepared in advance. Examples include captions and audio description for videos, alternate text for images, a simplified version of an online banking app, a sign language video explaining how to take HDR photos on an online camera guide, and a help item in easy language for a tax report software. Such user interface resources will often be made available by third parties, for example human factors experts, user groups and individual users. They will upload and describe these resources on resource services from which adaptive user interface implementations can discover and retrieve them at runtime.

To make this process work, two aspects need standardization: First, user interface resources should be clearly and unambiguously described so that they can be discovered at runtime. Second, resource services hosting these user interface resources should be discoverable and have a clearly described interface for querying and retrieving user interface resources.

This document addresses both aspects in a flexible way. It specifies syntax and semantics for a RESTful resource service interface while not restricting the clients in using whatever vocabulary and terms they choose for the description of user interface resources. Since HTTP is used as the most common REST implementation, this document defines a light-weight protocol that can be used on virtually all user interface platforms, including web browsers, mobile apps and software agents.

NOTE Though this document is part of the Universal Remote Console (URC) framework, it can be used independently from the other URC technologies. In particular, a user interface implementation can benefit from a user interface resource service without being connected to a URC target and without employing the user interface socket approach.

Information technology — User interfaces — Universal remote console —

Part 8: User interface resource framework

1 Scope

This document defines a RESTful protocol for the provision and delivery of resources that are related to user interface adaptation based on context of use.

This document addresses requirements and recommendations for the following services:

— user-context service;

— task-context service;

— equipment-context service;

— environment-context service;

— resource service;

— resource-description service;

— matching service (for finding appropriate resources based on specific contexts and other match criteria).

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, *Universal Multiple-Octet Coded Character Set (UCS)*

IETF RFC 2046[1], *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*

IETF RFC 3986[2], *Uniform Resource Identifier (URI): Generic Syntax*

IETF RFC 7159[3], *The JavaScript Object Notation (JSON) Data Interchange Format*

IETF RFC 7230[4], *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*

IETF RFC 7231[5], *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*

1) November 1996, https://tools.ietf.org/html/rfc2046.

2) January 2005, https://tools.ietf.org/html/rfc3986.

3) March 2014, https://tools.ietf.org/html/rfc7159.

4) June 2014, https://tools.ietf.org/html/rfc7230.

5) June 2014, https://tools.ietf.org/html/rfc7231.

W3C XML 1.0[6], *Extensible Markup Language (XML) 1.0*, W3C Recommendation

W3C XML Schema Part 1[7], *Structures*, W3C Recommendation

W3C XML Schema Part 2[8], *Datatypes*, W3C Recommendation

3 Terms and definitions

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

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

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

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

3.1 Context of use

3.1.1
context of use
use context
users, tasks, equipment (hardware, software and materials), and the physical and social environments in which a product is used

[SOURCE: ISO 9241-11:1998, 3.5]

3.1.2
user-context
part of the context of use that describes aspects of an individual user interacting with an ICT system

3.1.3
task-context
part of the context of use that describes aspects of the task done by a user in interacting with an ICT system

3.1.4
equipment-context
part of the context of use that describes aspects of the equipment (hardware, software and materials) used by a user to interact with an ICT system

3.1.5
environment-context
part of the context of use that describes aspects of the physical and social environments in which a user interacts with an ICT system

3.2 Resources

3.2.1
resource
user interface resource
object that is used as an entity or to support decision making in the construction of a concrete user interface

[SOURCE: ISO/IEC 24752-1:2014, 3.31, modified — the alternative term 'user interface resource' has been added and an EXAMPLE and Note to entry have been removed]

6) https://www.w3.org/TR/xml/.

7) https://www.w3.org/TR/xmlschema-1/.

8) https://www.w3.org/TR/xmlschema-2/.

3.2.2
property
aspect of a context or resource used to describe the context or resource

Note 1 to entry: In this document, properties are represented as key-value pairs.

3.2.3
user-rating
explicit or implicit statement of an individual user about the degree of personal preference for a particular resource or resource description

Note 1 to entry: In this document, user-ratings are represented as float values between –1.0 and 1.0. The value –1.0 represents the highest degree of dislike, 0.0 represents no preference (neutral position), and 1.0 represents the highest degree of like.

3.3 Services

3.3.1
service
server offering RESTful operations for the management of REST resources

3.3.2
matching service
service that manages listings

3.3.3
listing
ordered list of resource descriptions that match specific criteria for user-context, task-context, equipment-context, environment-context and resource description properties

3.4 Representational State Transfer

3.4.1
Representational State Transfer
REST
paradigm for an HTTP-based protocol that is built upon HTTP methods as operations and URIs as resources

Note 1 to entry: Representational State Transfer was first introduced by Fielding (2000).

3.4.2
RESTful
conforming to the constraints of REST

3.4.3
REST resource
data object that is addressable with a URI and that can be manipulated via HTTP verbs

4 Conformance

A *user-context service* conforms to this document if all of the following is true:

— It complies with 7.1 and 7.2.

— It provides appropriate mappings for XML (A.1, A.2) and JSON (B.1, B.2).

A *task-context service* conforms to this document if all of the following is true:

— It complies with 7.1 and 7.3.

— It provides appropriate mappings for XML (A.1, A.3) and JSON (B.1, B.3).

An *equipment-context service* conforms to this document if all of the following is true:

— It complies with 7.1 and 7.4.

— It provides appropriate mappings for XML (A.1, A.4) and JSON (B.1, B.4).

An *environment-context service* conforms to this document if all of the following is true:

— It complies with 7.1 and 7.5.

— It provides appropriate mappings for XML (A.1, A.5) and JSON (B.1, B.5).

A *resource service* conforms to this document if all of the following is true:

— It complies with 7.1 and 7.6.

— It provides appropriate mappings for XML (A.1, A.6) and JSON (B.1, B.6).

A *resource-description service* conforms to this document if all of the following is true:

— It complies with 7.1 and 7.7.

— It provides appropriate mappings for XML (A.1, A.7) and JSON (B.1, B.7).

A *matching service* conforms to this document if all of the following is true:

— It complies with 7.1 and 7.8.

— It provides appropriate mappings for XML (A.1, A.8) and JSON (B.1, B.8).

5 Use cases (informative)

5.1 General

This clause contains a set of use cases for illustration of possible applications for this document. The set of use cases is not exhaustive, and does not limit the applicability of this document.

5.2 User interface localisation

— User interface localisation: A user interface label or icon is replaced at runtime by a supplementary label or icon from a resource service.

— User interface augmentation: A sign language video from a resource service is added to a user interface at runtime for the purpose of making the user interface more accessible for a deaf user.

5.3 User interface personalisation

— A browser loads an alternative style sheet from a resource service for the purpose of personalisation of a specific webpage.

— A smart-home device loads an old-style design of light controls from a resource service as a web component to replace the default design.

— A simplified user interface of an online banking application is loaded at runtime for a user with a mild cognitive disability.

5.4 User interface accessibility

— A browser downloads a long image description from a resource service for the purpose of allowing the user to listen to the image description via their screenreader.

— A video player downloads a third-party caption file from a resource service, for the purpose of displaying them with the rendition of a video from the Internet that had no captions attached.

— A video player downloads a third-party audio description track from a resource service, for the purpose of playing it in place of the default audio track of a video from the Internet that included no audio description.

5.5 User interface responsiveness

— At runtime, a drop-down menu in a webpage is replaced by a group of large graphical push buttons to make it easier to be operated on a Tablet with touch screen.

5.6 User interface settings

— An operating system retrieves and activates a set of user interface settings that accommodates the needs and preferences of an individual user although this user has never used the operating system before.

— An application (e.g. an editor) finds a suitable configuration for an individual user although the user has never used the application before.

6 Workflow model (informative)

6.1 General

This document supports applications (in the role of a client of a service) in operating with user-contexts, task-contexts, equipment-contexts, environment-contexts, resources, resource descriptions and listings.

The following subsections describe typical workflows for user interface adaptation. However, they do not limit the applicability of this document.

6.2 Storing and retrieving a user-context

In general, clients create, retrieve, update and delete user-contexts by conducting the following steps:

1) A client creates a user-context object (see 7.2.2).

2) A client retrieves a user-context object or a part of it (see 7.2.3).

3) A client updates a user-context object (see 7.2.4).

4) (optional) A client deletes a user-context object which is obsolete or not needed anymore (see 7.2.5).

Whereby:

— Steps 2 and 3 can be executed in any order, and any number of times.

NOTE Multiple clients can be involved in these steps; it does not need to be a single one.

6.3 Storing and retrieving a task-context

In general, clients create, retrieve, update and delete task-contexts by conducting the following steps:

1) A client creates a task-context object (see 7.3.2).

2) A client retrieves a task-context object or a part of it (see 7.3.3).

3) A client updates a task-context object (see 7.3.4).

4) (optional) A client deletes a task-context object which is obsolete or not needed anymore (see 7.3.5).

Whereby:

— Steps 2 and 3 can be executed in any order, and any number of times.

NOTE Multiple clients can be involved in these steps; it does not need to be a single one.

6.4 Storing and retrieving an equipment-context

In general, clients create, retrieve, update and delete equipment-contexts by conducting the following steps:

1) A client creates an equipment-context object (see 7.4.2).

2) A client retrieves an equipment-context object or a part of it (see 7.4.3).

3) A client updates an equipment-context object (see 7.4.4).

4) (optional) A client deletes an equipment-context object which is obsolete or not needed anymore (see 7.4.5).

Whereby:

— Steps 2 and 3 can be executed in any order, and any number of times.

NOTE Multiple clients can be involved in these steps; it does not need to be a single one.

6.5 Storing and retrieving an environment-context

In general, clients create, retrieve, update and delete environment-contexts by conducting the following steps:

1) A client creates an environment-context object (see 7.5.2).

2) A client retrieves an environment-context object or a part of it (see 7.5.3).

3) A client updates an environment-context object (see 7.5.4).

4) (optional) A client deletes an environment-context object which is obsolete or not needed anymore (see 7.5.5).

Whereby:

— Steps 2 and 3 can be executed in any order, and any number of times.

NOTE Multiple clients can be involved in these steps; it does not need to be a single one.

6.6 Describing resources

In general, clients describe resources by conducting the following steps:

1) A client uploads an external resource to a resource service (see 7.6.2) or to any other URI-enabled location. As a result, the resource is retrievable through a resolvable URI (see 7.6.3).

2) A client creates a resource description and links the resource to it (see 7.7.2).

3) A client retrieves a resource description or a part of it any number of times (see 7.7.3).

4) A client updates a resource description any number of times (see 7.7.5).

5) A client confirms a user-rating for a resource description any number of times (see 7.8.5).

6) (optional) A client deletes a resource description which is obsolete or not needed anymore (see 7.7.6).

7) (optional) A client deletes a resource which is obsolete or not needed anymore (see 7.6.6).

Whereby:

— Step 1 is optional since a resource description may not have a link to an external resource.

— Steps 3 to 5 can be executed in any order, and any number of times.

NOTE Multiple clients can be involved in these steps; it does not need to be a single one.

6.7 Finding and retrieving matching resources

In general, clients search for resources that match the following characteristics:

— a specific user-context;

— a specific task-context;

— a specific equipment-context;

— a specific environment-context;

— a set of arbitrary resource properties (key-value pairs) representing a query for a resource description object.

The general sequence of steps is as follows, in this order:

1) A client creates a user-context object (see 7.2.2).

2) A client creates a task-context object (see 7.3.2).

3) A client creates an equipment-context object (see 7.4.2).

4) A client creates an environment-context object (see 7.5.2).

5) A client sends a resource-description-query object to a matching service, thus creating a listing object (see 7.8.2).

6) (optional) A client retrieves (and examines) one or more matching resource descriptions from the listing object created in step 5, possibly sequentially in multiple parts, i.e. by slices (see 7.8.3).

7) A client retrieves one or multiple resources, either by looking up the resource ID from the resource descriptions of the listing object (see 7.6.3), or by referencing the first entry in the listing object (see 7.6.4).

8) (optional) A client deletes the listing object created in step 5 (see 7.8.4).

9) (optional) A client deletes the environment-context object created in step 4 (see 7.5.5).

10) (optional) A client deletes the equipment-context object created in step 3 (see 7.4.5).

11) (optional) A client deletes the task-context object created in step 2 (see 7.3.5).

12) (optional) A client deletes the user-context object created in step 1 (see 7.2.5).

Whereby:

— Steps 1 to 4 can be executed in any order.

— The block of steps 5 to 8 can be repeated any number of times.

— Steps 9 to 12 can be executed in any order.

NOTE 1 A client can omit step 6 if it is only interested in the one best matching resource (which it downloads in step 7 by reference to an index in the listing object), and does not care about its resource description.

NOTE 2 Multiple clients can be involved in these steps; it does not need to be a single one.

6.8 Describing user interface settings

A particular set of user interface settings is represented by a resource description object that has no resource attached. Such user interface settings are either created by a client, or generated on demand by a matching service upon receiving a matching request (see 6.9).

For user interface settings that are created by a client, the workflow is the same as for describing resources (see 6.6), with the following exceptions:

— Step 1 is omitted since there is no resource involved.

— In step 2, no resource is linked to the resource description.

6.9 Finding and retrieving matching user interface settings

In general, clients search for user interface settings that match the following characteristics:

— a specific user-context;

— a specific task-context;

— a specific equipment-context;

— a specific environment-context;

— a set of arbitrary user interface settings (key-value pairs) representing a query for a complete user interface settings object.

The workflow is the same as for finding and retrieving matching resources (see 6.7), with the following exceptions:

— A particular set of user interface settings is retrieved from a resource description service (step 6 which is mandatory in this case).

— Step 7 (client retrieves one or multiple resources) is omitted since there is no resource associated with the resource description.

7 REST interfaces for services

7.1 General

7.1.1 REST architecture (informative)

The interface specified in this document is based on the REST architecture (Fielding, 2000). This provides a contract between a resource service and a client that is easy to implement, test and maintain on both sides.

According to REST, HTTP request methods (see IETF RFC 7231, section 4) 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 request methods" according to IETF RFC 7231, but are applicable for HTTPS as well (see IETF RFC 7230). In fact, the use of HTTPS as a more secure protocol is recommended for all services (see [8]).

Each operation has one or multiple request parameters, and one or multiple response parameters, as specified in 7.2-7.8.

7.1.2 Request and response parameters

The request and response parameters for an operation in this clause 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.

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

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

   EXAMPLE 1 In the operation **PUT /api/user-contexts/*user-context-id*** the parameter *user-context-id* 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 (in accordance with IETF RFC 3986).

   EXAMPLE 2 In the operation **GET /api/listings/*listing-id*?start=*start*&max=*max*** the parameters *start* and *max* are URI query parameters.

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

   EXAMPLE 3 In the following HTTP response header, the parameter *user-context-uri* is submitted as content of the field "Location".

   ```
   HTTP/1.1 200 OK
   Location: https://example.com/api/user-contexts/U12345
   ```

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 (in accordance with IETF RFC 7231).

   EXAMPLE 4 In the following HTTP response body, the parameter *total* is submitted as content of the attribute `value` on element `<total>`.

   ```
   <response>
      <total value="12345">
   </response>
   ```

In addition to the request and response parameters specified in this clause, 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 service or client does not know the meaning of such proprietary information, it shall ignore it.

NOTE It is possible that future versions of this document will adopt specific proprietary extensions used in existing systems if the extensions prove to be useful and there is a need for standardization.

7.1.3 Response codes

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

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

7.1.4 Authentication

A service 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 service requires authentication, Basic authentication (in accordance with IETF RFC 7617) over Transport Layer Security (TLS) (as specified in IETF RFC 5246) should be supported, and/or other stronger HTTP authentication schemes.

7.1.5 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 (in accordance with RFC 3986, section 2.2) shall be percent-encoded (in accordance with RFC 3986, section 2.1).

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

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

— A service shall support request bodies in XML and JSON formats, alternatively. The format (MIME type, as specified in 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). This is referred to as the *request body format* parameter in the remainder of this clause.

— 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 (in accordance with IETF RFC 7231). This is referred to as the *response body format* parameter in the remainder of this clause.

— 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 (in accordance with IETF RFC 7231). This is referred to as the *preferred response body format* parameter. At a minimum, a service shall support all of the following formats:

  i) XML format (MIME type "application/xml"), according to the mapping provided normatively in Annex A;

  ii) JSON format (MIME type "application/json"), according to the mapping provided normatively in Annex B.

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).

7.2 User-context

7.2.1 General

A *user-context service* shall implement all mandatory and optional operations in 7.2, with support for all mandatory and optional parameters. It shall support both XML and JSON mapping, as specified in Annexes A and B.

A *user-context* shall be an object with the following restrictions:

— *User-context* shall be an ordered Array with any number of *option* objects, describing an individual's preferred concepts and values under various (alternative) conditions (conditions exclude each other).

— The names of the *option* objects shall be mutually unique, but are not otherwise restricted.

- Each *option* object
  - may include a *name* (type string) – for internal identification purposes,
  - shall include *preferences* with an Array of *preference* objects – specifying the individual's needs and preferences,
  - may include *conditions* with a non-empty Array of *condition* objects – defining the exact condition in which the *preferences* apply,
  - may include any other objects with proprietary content, as long as they do not interfere with the above objects within *context*.
- Each *preference* object shall consist of a set of key-value pairs, each indicating a *value* (string) for a specific preference concept (referenced by the *key*). Each *key* name shall be a URI, and shall be unique within *preferences*.
- Each *condition* object shall have exactly one *type* (type string) and an *operands* object with one or more *operand* objects.
- Each *type* shall be the string representation of one of the following operators:
  - "not" – logical NOT operator, with syntax and semantics as specified for "!" in ECMA-262, section 12.5.9;
  - "eq" – equal operator, with syntax and semantics as specified for "==" in ECMA-262, section 12.11;
  - "ne" – not-equal operator, with syntax and semantics as specified for "!=" in ECMA-262, section 12.11;
  - "lt" – less-than operator, with syntax and semantics as specified for "<" in ECMA-262, section 12.10;
  - "le" – less-than-or-equal operator, with syntax and semantics as specified for "<=" in ECMA-262, section 12.10;
  - "gt" – greater-than operator, with syntax and semantics as specified for ">" in ECMA-262, section 12.10;
  - "ge" – greater-than-or-equal operator, with syntax and semantics as specified for ">=" in ECMA-262, section 12.10;
  - "and" – local AND operator: each sub-condition is evaluated sequentially, and if one of them evaluates to false, *condition* evaluates to false immediately without evaluating the other sub-conditions; otherwise, *condition* evaluates to true;
  - "or" – local OR operator: each sub-condition is evaluated sequentially, and if one of them evaluates to true, *condition* evaluates to true immediately without evaluating the other sub-conditions; otherwise, *condition* evaluates to false;
  - "ap" – approximate operator, with two operands of any type, resulting in a Boolean value indicating whether the operands are approximately equal.
- Each *operand* shall be either a URI (in accordance with IETF RFC 3986) – in which case it shall be interpreted as the value of the concept with the URI as *key*, a *condition* object (i.e. a nested condition), or a literal.
  - If *type* is "not", *operands* shall have exactly one element.
  - If *type* is "eq", "ne", "lt", "le", "gt", "ge", or "ap", *operands* shall have exactly two elements.

  - If *type* is "and" or "or", *operands* shall have at least two elements.

- If the *conditions* object is missing or is an empty Array, the context is generic, i.e. it applies without restriction. If – in a concrete situation – there are multiple *option* objects whose *conditions* are true, the topmost *option* shall be enforced.

- *User-context* may include any other objects with proprietary content, as long as they do not interfere with the above objects within *user-context*.

NOTE 1 The approximate operator ("ap") can be used to describe conditions that are the same or almost the same (in terms of an approximation) to those that already occurred in a similar situation. This can be used by matching services (see 7.8) employing statistical methods and clustering techniques. Implementations (including matching services) are free to implement this operator in a way that best supports their matching strategy.

NOTE 2 Additional operators could be used internally by a system, e.g. "inRange". However, if a user-context is to be exposed externally, then the operators as defined in this document need to be used.

NOTE 3 ISO/IEC 24751-1 specifies how to register terms for describing equipment-context characteristics. In ISO/IEC 24751-1, a user-context is called "needs and preferences set" and a user-context service is called "preference server". It is good practice to register the *keys* of preferences as concepts in a registry server conforming to ISO/IEC 24751-1.

For processing and evaluation of a user-context, all of the following shall apply:

- For the *conditions* object to become true, all contained *condition* objects need to be true.

- Evaluation of *conditions* shall occur sequentially by *condition*, from the beginning to the end, in a depth-first fashion. A sub-condition shall become *true* immediately if a *condition* is met that is logically ORed (logical operator 'or') with the remaining *condition* object(s) and that evaluates to *true*, skipping the evaluation of the remaining *condition* object(s). Also, a sub-condition shall become *false* immediately if a *condition* object is met that is logically ANDed (logical operator 'and') with the remaining *condition* object(s) and that evaluates to *false*, skipping the evaluation of the remaining *condition* object(s).

- If a syntax error occurs in a user-context, the whole user-context shall be invalid. If a type mismatch occurs within a *condition* object, this *condition* shall evaluate to *false*. Otherwise, if a type mismatch occurs within a key-value pair of a *preferences* object, this key-value pair shall be ignored.

7.2.2 CREATE user-context (mandatory)

For creating a user-context, a user-context service shall implement the method **`POST`** on endpoint **`/api/user-contexts`**, 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 user-context

POST `/api/user-contexts`			
Parameter name	**Category**	**Type**	**Description**
user-context (mandatory)	request body	*user-context* object	Refer to 7.2.1.

EXAMPLE 1 A CREATE user-context request.

```
POST /api/user-contexts HTTP/1.1
Content-Type: request body format

[request body]
```


Table 2 — Response codes and parameters for creating a user-context

Response code: 201 Created (Success)			
Parameter name	**Category**	**Type**	**Description**
user-context-uri (mandatory)	HTTP header field "Location"	URI	Globally unique URI for the created user-context on the server. It shall have the following format: "*scheme://authority*/api/user-contexts/*user-context-id*" (with *scheme* and *authority* as defined in IETF RFC 3986, and *user-context-id* being an implementation-specific server-unique identifier for the user-context).
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A CREATE user-context response, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 201 Created
Location: https://example.com/api/user-contexts/U12345
```

NOTE The HTTP headers in the request and response of EXAMPLE 1 contain the most relevant HTTP header fields only. Other header fields can occur, see IETF RFC 7231. This applies to all examples in Clause 7.

7.2.3 GET user-context (mandatory)

For retrieving a user-context, a user-context service shall implement the method **`GET`** on endpoint **`/api/user-contexts/`*`user-context-id`***, with the request parameters as listed in Table 3, and response codes and parameters as listed in Table 4.

Table 3 — Request parameters for retrieving a user-context

GET `/api/user-contexts/`*`user-context-id`*			
Parameter name	**Category**	**Type**	**Description**
user-context-id (mandatory)	URI path	String	Server-unique identifier for a user-context on the user-context service.

EXAMPLE 1 A GET user-context request. A client wants to get the user-context with ID "U12345". Note that no request body is supplied.

```
GET /api/user-contexts/U12345 HTTP/1.1
Accept: preferred response body format
```

Table 4 — Response codes and parameters for retrieving a user-context

Response code: 200 OK (Success)			
Parameter name	**Category**	**Type**	**Description**
user-context (mandatory)	response body	*user-context* object	The user-context is returned as complete body of the response. Refer to 7.2.1.
Response code: 404 Not Found (user-context not available)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A GET user-context response, following up on EXAMPLE 1.

```
HTTP/1.1 200 OK
Content-Type: response body format

[response body]
```

7.2.4 UPDATE user-context (mandatory)

For updating a user-context, a user-context service shall implement the method **`PUT`** on endpoint **`/api/user-contexts/`*`user-context-id`***, with the request parameters as listed in Table 5, and response codes and parameters as listed in Table 6.

The UPDATE operation shall overwrite all contents of an existing user-context on the server with the contents of a *user-context* provided as request parameter ("full overwrite" as opposed to "partial overwrite"). If the referenced user-context does not exist on the server, it shall return a 404 response code (see Table 6).

Table 5 — Request parameters for updating a user-context

PUT `/api/user-contexts/`*`user-context-id`*			
Parameter name	**Category**	**Type**	**Description**
user-context-id (mandatory)	URI path	String	Server-unique identifier for the user-context on the server to be updated.
user-context (mandatory)	request body	*user-context* object	Refer to 7.2.1. All contents of the existing user-context with *user-context-id* on the server will be overwritten by the contents of request parameter *user-context*.

EXAMPLE 1 An UPDATE user-context request.

```
PUT /api/user-contexts/U12345 HTTP/1.1
Content-Type: request body format

[request body]
```

Table 6 — Response codes and parameters for updating a user-context

Response code: 204 No Content (Successfully updated)			
Parameter name	**Category**	**Type**	**Description**
user-context-uri (mandatory)	HTTP header field "Location"	URI	Globally unique URI for the updated user-context on the server. It shall have the following format: "*scheme://authority*/api/user-contexts/*user-context-id*" (with *scheme* and *authority* as defined in IETF RFC 3986, and *user-context-id* being an implementation-specific server-unique identifier for the user-context).
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).
Response code: 404 Not Found (Invalid user-context ID)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 An UPDATE user-context response, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 204 No content
Location: https://example.com/api/user-contexts/U12345
```

7.2.5 DELETE user-context (optional)

In general, a *user-context service* may dispose its user-contexts if they expire. The expiration period is implementation-specific.

For deleting a user-context, a user-context service may implement the method **`DELETE`** on endpoint **`/api/user-contexts/`*`user-context-id`***, with the request parameters as listed in Table 7, and response codes and parameters as listed in Table 8 below.

NOTE This operation is optional since a user-context service can choose to not allow to remove an existing user-context for the sake of URI stability. In any case, implementers are encouraged to think about a consistent policy on stable URIs (see e.g. the W3C URI Persistency Policy).

Table 7 — Request parameters for deleting a user-context

DELETE `/api/user-contexts/`*`user-context-id`*			
Parameter name	**Category**	**Type**	**Description**
user-context-id (mandatory)	URI path	URI	Server-unique identifier for the user-context on the server to be deleted.

EXAMPLE 1 A DELETE user-context request, requesting to delete the user-context with ID "U12345". Note that there is no request body.

```
DELETE /api/user-contexts/U12345 HTTP/1.1
```

Table 8 — Response codes and parameters for deleting a user-context

Response code: 204 No Content (Deletion enacted)			
Parameter name	**Category**	**Type**	**Description**
(no parameter)			
Response code: 404 Not Found (Invalid user-context ID)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A DELETE user-context response, indicating that the requested deletion was enacted, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 204 No Content
```

7.2.6 GET user-context-list (mandatory)

For retrieving a list of user-contexts, a user-context service shall implement the method **`GET`** on endpoint **`/api/user-contexts`**, with the request parameters as listed in Table 9, and response codes and parameters as listed in Table 10. The user-context service shall return a non-empty list of URIs of all user-contexts that the requesting user has access to (see 7.2.3). If no match is found, the service shall respond with response code 404 (see Table 10).

NOTE For user-context services that employ user authentication, publicly available user-contexts can be requested by a client taking the role of an anonymous user.

Additional restrictions may be applied by the filter parameters *owner*, *updatable*, and *deletable* (see Table 9) which may be combined. A request may also include the pagination parameters *offset* and *limit* (see Table 9). In case the request specifies an *offset* that does not exist on the resulting list of URIs, the service shall respond with response code 404 (see Table 10).

Table 9 — Request parameters for retrieving a list of concept records

GET /api/user-contexts			
Parameter name	**Category**	**Type**	**Description**
owner (optional)	URI query parameter	String	Filter parameter: identification of a user that owns the user-contexts. This filter may be combined with the other filters in Table 9. User identification and ownership of user-contexts is implementation-specific for user-context services.
updatable (optional)	URI query parameter	Boolean	Filter parameter: If *updatable* is true, only user-contexts are listed for which the requesting user has the permission to update (see 7.2.4). If *updatable* is false, only user-contexts are listed for which the requesting user does not have the permission to update (see 7.2.4). This filter may be combined with the other filters in Table 9. The existence and details of a user authentication mechanism are implementation-specific.
deletable (optional)	URI query parameter	Boolean	Filter parameter: If *deletable* is true, only user-contexts are listed for which the requesting user has the permission to delete (see 7.2.5). If *deletable* is false, only user-contexts are listed for which the requesting user does not have the permission to delete (see 7.2.5). This filter may be combined with the other filters in Table 9. The existence and details of a user authentication mechanism are implementation-specific.
offset (optional)	URI query parameter	Number	Pagination parameter: The number (0 and positive numbers) of user-contexts to skip in the list of results. If *offset* is not specified, an offset of 0 shall be assumed.
limit (optional)	URI query parameter	Number	Pagination parameter: The number (positive) of user-contexts to return. Used for pagination. If *limit* is not specified, no limit shall be assumed.

EXAMPLE 1 A GET user-context-list request. A client requests the second (`offset=1`) user-context in the list of user-contexts for which the requester has permission to read and update, and that is owned by user with the identification "UserXYZ". Note that there is no request body supplied.

```
GET /api/user-contexts?owner=UserXYZ&updatable=true&offset=1&limit=1 HTTP/1.1
Accept: preferred response body format
```

Table 10 — Response codes and parameters for retrieving a list of concept records

Response code: 200 OK			
Parameter name	**Category**	**Type**	**Description**
total number of user-contexts (mandatory)	response body	Number	Total number (≥1) of user-contexts available on the user-context service for which the filter(s) in the request apply (but without having the pagination parameters applied).
user-context-URIs (mandatory)	response body	Array of URIs	*user-context-URIs* shall be a non-empty array of URIs (String) each referring to a *user-context* object (see 7.2.1). The order of the array elements shall be significant (to allow for pagination).

Table 10 *(continued)*

Response code: 400 Bad Request (Invalid parameters)			
error message	entire response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).
Response code: 404 Not Found			
error message	entire response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A GET user-context-list response (following up on EXAMPLE 1).

```
HTTP/1.1 200 OK
Content-Type: response body format

[response body]
```

7.3 Task-context

7.3.1 General

A *task-context service* shall implement all mandatory and optional operations in 7.3, with support for all mandatory and optional parameters. It shall support both XML and JSON mapping, as specified in Annexes A and B.

The information model for a *task-context* object is as follows:

— A task-*context* shall be a (possibly empty) Array of *property* objects, each with the following fields:

  — *name* (type String);

  — *value* (type String).

— Each *property* object may have any number of *descriptor* objects, each with the following fields:

  — *name* (type String);

  — *value* (type String).

— *Task-context* may include any other objects with proprietary content, as long as they do not interfere with the above objects within *task-context*.

See A.3.1 for the mapping of a task-context in XML format, and B.3.1 for the mapping in JSON format.

NOTE ISO/IEC 24751-1 specifies how to register terms for describing task-context characteristics.

7.3.2 CREATE task-context (mandatory)

For creating a task-context, a task-context service shall implement the method **`POST`** on endpoint **`/api/task-contexts`**, with the request parameters as listed in Table 11, and response codes and parameters as listed in Table 12.

Table 11 — Request parameters for creating a task-context

`POST /api/task-contexts`			
Parameter name	**Category**	**Type**	**Description**
task-context (mandatory)	request body	*task-context* object	Refer to 7.3.1.

EXAMPLE 1 A CREATE task-context request body:

```
POST /api/task-contexts HTTP/1.1
Content-Type: request body format

[request body]
```

Table 12 — Response codes and parameters for creating a task-context

Response code: 201 Created (Success)			
Parameter name	**Category**	**Type**	**Description**
task-context-uri (mandatory)	HTTP header field "Location"	URI	Globally unique URI for the created task-context on the server. It shall have the following format: "*scheme*://*authority*/api/task-contexts/*task-context-id*" (with *scheme* and *authority* as defined in IETF RFC 3986, and *task-context-id* being an implementation-specific server-unique identifier for the task-context).
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A CREATE task-context response body, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 201 Created
Location: https://example.com/task-context/T12345
```

7.3.3 GET task-context (mandatory)

For retrieving a task-context, a task-context service shall implement the method **`GET`** on endpoint **`/api/task-contexts/`*`task-context-id`***, with the request parameters as listed in Table 13, and response codes and parameters as listed in Table 14.

Table 13 — Request parameters for retrieving a task-context

`GET /api/task-contexts/`*`task-context-id`*			
Parameter name	**Category**	**Type**	**Description**
task-context-id (mandatory)	URI path	String	Unique resource ID on the task-context service.

EXAMPLE 1 A GET task-context request. A client wants to get the task-context with ID "T12345".

```
GET /api/task-contexts/T12345 HTTP/1.1
Accept: preferred response body format
```

Table 14 — Response codes and parameters for retrieving a task-context

Response code: 200 OK (Success)			
Parameter name	**Category**	**Type**	**Description**
task-context (mandatory)	response body	as specified by *response body format*	The task-context is returned as complete body of the response. Refer to 7.3.1.
Response code: 404 Not Found (Resource not available)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A GET task-context response, following up on EXAMPLE 1.

```
HTTP/1.1 200 OK
Content-Type: response body format

[response body]
```

7.3.4 UPDATE task-context (mandatory)

For updating a task-context, a task-context service shall implement the method **`PUT`** on endpoint **`/api/task-contexts/`*`task-context-id`***, with the request parameters as listed in Table 15, and response codes and parameters as listed in Table 16.

The UPDATE operation shall overwrite all contents of an existing task-context on the server with the contents of a task-*context* provided as request parameter ("full overwrite" as opposed to "partial overwrite"). If the referenced task-context does not exist on the server, it shall return a 404 response code (see Table 16).

Table 15 — Request parameters for updating a task-context

PUT /api/task-contexts/*task-context-id*			
Parameter name	**Category**	**Type**	**Description**
task-context-id (mandatory)	URI path	String	Server-unique identifier for the task-context on the server to be updated (or created if not yet existing).
task-context (mandatory)	request body	*task-context* object	Refer to 7.3.1. All contents of the existing task-context with *task-context-id* on the server will be overwritten by the contents of request parameter *task-context*.

EXAMPLE 1 An UPDATE task-context request.

```
PUT /api/task-contexts/T12345 HTTP/1.1
Content-Type: request body format

[request body]
```

Table 16 — Response codes and parameters for updating a task-context

Response code: 204 No Content (Successfully updated)			
Parameter name	**Category**	**Type**	**Description**
task-context-uri (mandatory)	HTTP header field "Location"	URI	Globally unique URI for the updated task-context on the server. It shall have the following format: "*scheme*://*authority*/api/task-contexts/*task-context-id*" (with *scheme* and *authority* as defined in IETF RFC 3986, and *task-context-id* being an implementation-specific server-unique identifier for the task-context).
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).
Response code: 404 Not Found (Invalid task-context ID)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 An UPDATE task-context response, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 204 No content
Location: https://example.com/task-context/T12345
```

7.3.5 DELETE task-context (optional)

In general, a task-*context service* may dispose its task-contexts if they expire. The expiration period is implementation-specific.

For deleting a task-context, a task-context service may implement the method **`DELETE`** on endpoint **`/task-context/`*`task-context-id`***, with the request parameters as listed in Table 17, and response codes and parameters as listed in Table 18.

NOTE This operation is optional since a task-context service can choose to not allow to remove an existing task-context for the sake of URI stability. In any case, implementers are encouraged to think about a consistent policy on stable URIs (see e.g. the W3C URI Persistency Policy).

Table 17 — Request parameters for deleting a task-context

DELETE `/api/task-contexts/`*`task-context-id`*			
Parameter name	**Category**	**Type**	**Description**
task-context-id (mandatory)	URI path	String	Server-unique identifier for the task-context on the server to be deleted.

EXAMPLE 1 A DELETE task-context request. The task-context with ID "T12345" is requested to be deleted. Note that there is no request body.

```
DELETE /api/task-contexts/T12345 HTTP/1.1
```

Table 18 — Response codes and parameters for deleting a task-context

Response code: 204 No Content (Deletion enacted)			
Parameter name	**Category**	**Type**	**Description**
(no parameter)			
Response code: 404 Not Found (Invalid task-context ID)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A DELETE task-context response, indicating that the requested deletion was enacted, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 204 No Content
```

A task-context service may, at its own discretion, dispose a task-context on its own (without being called on the DELETE task-context operation) after it has not been used in a reasonable period of time.

7.3.6 GET task-context-list (mandatory)

For retrieving a list of task-contexts, a task-context service shall implement the method **`GET`** on endpoint **`/api/task-contexts`**, with the request parameters as listed in Table 19, and response codes and parameters as listed in Table 20. The task-context service shall return a non-empty list of URIs of all task-contexts that the requesting user has access to (see 7.3.3). If no match is found, the service shall respond with response code 404 (see Table 20).

NOTE For task-context services that employ user authentication, publicly available task-contexts can be requested by a client taking the role of an anonymous user.

Additional restrictions may be applied by the filter parameters *owner*, *updatable*, and *deletable* (see Table 19) which may be combined. A request may also include the pagination parameters *offset* and *limit* (see Table 19). In case the request specifies an *offset* that does not exist on the resulting list of URIs, the service shall respond with response code 404 (see Table 20).

Table 19 — Request parameters for retrieving a list of concept records

GET /api/task-contexts			
Parameter name	**Category**	**Type**	**Description**
owner (optional)	URI query parameter	String	Filter parameter: identification of a user that owns the task-contexts. This filter may be combined with the other filters in Table 19. User identification and ownership of task-contexts is implementation-specific for task-context services.
updatable (optional)	URI query parameter	Boolean	Filter parameter: If *updatable* is true, only task-contexts are listed for which the requesting user has the permission to update (see 7.3.4). If *updatable* is false, only task-contexts are listed for which the requesting user does not have the permission to update (see 7.3.4). This filter may be combined with the other filters in Table 19. The existence and details of a user authentication mechanism are implementation-specific.
deletable (optional)	URI query parameter	Boolean	Filter parameter: If *deletable* is true, only task-contexts are listed for which the requesting user has the permission to delete (see 7.3.5). If *deletable* is false, only task-contexts are listed for which the requesting user does not have the permission to delete (see 7.3.5). This filter may be combined with the other filters in Table 19. The existence and details of a user authentication mechanism are implementation-specific.
offset (optional)	URI query parameter	Number	Pagination parameter: The number (0 and positive numbers) of task-contexts to skip in the list of results. If *offset* is not specified, an offset of 0 shall be assumed.
limit (optional)	URI query parameter	Number	Pagination parameter: The number (positive) of task-contexts to return. Used for pagination. If *limit* is not specified, no limit shall be assumed.

EXAMPLE 1 A GET task-context-list request. A client requests the second (`offset=1`) task-context in the list of task-contexts for which the requester has permission to read and update, and that is owned by user with the identification "UserXYZ". Note that there is no request body supplied.

```
GET /api/task-contexts?owner=UserXYZ&updatable=true&offset=1&limit=1 HTTP/1.1
Accept: preferred response body format
```

Table 20 — Response codes and parameters for retrieving a list of concept records

Response code: 200 OK			
Parameter name	**Category**	**Type**	**Description**
total number of task-contexts (mandatory)	response body	Number	Total number (≥1) of task-contexts available on the task-context service for which the filter(s) in the request apply (but without having the pagination parameters applied).
task-context-URIs (mandatory)	response body	Array of URIs	*task-context-URIs* shall be a non-empty array of URIs (String) each referring to a *task-context* object (see 7.3.1). The order of the array elements shall be significant (to allow for pagination).

Table 20 *(continued)*

Response code: 400 Bad Request (Invalid parameters)			
error message	entire response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).
Response code: 404 Not Found			
error message	entire response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A GET task-context-list response (following up on EXAMPLE 1).

```
HTTP/1.1 200 OK
Content-Type: response body format

[response body]
```

7.4 Equipment-context

7.4.1 General

An equipment-*context service* shall implement all mandatory and optional operations in 7.4, with support for all mandatory and optional parameters. It shall support both XML and JSON mapping, as specified in Annexes A and B.

The information model for an equipment-*context* object is as follows:

— An equipment-*context* shall be a (possibly empty) Array of *property* objects, each with the following fields:

  — *name* (type String);

  — *value* (type String).

— Each *property* object may have any number of *descriptor* objects, each with the following fields:

  — *name* (type String);

  — *value* (type String).

— *Equipment-context* may include any other objects with proprietary content, as long as they do not interfere with the above objects within *equipment-context*.

See A.4.1 for the mapping of an equipment-context in XML format, and B.4.1 for the mapping in JSON format.

NOTE ISO/IEC 24751-1 specifies how to register terms for describing equipment-context characteristics.

7.4.2 CREATE equipment-context (mandatory)

For creating an equipment-context, an equipment-context service shall implement the method **`POST`** on endpoint **`/equipment-contexts`**, with the request parameters as listed in Table 21, and response codes and parameters as listed in Table 22.

Table 21 — Request parameters for creating an equipment-context

POST /api/equipment-contexts			
Parameter name	**Category**	**Type**	**Description**
equipment-context (mandatory)	request body	*equipment-context* object	Refer to 7.4.1.

EXAMPLE 1 A CREATE equipment-context request:

```
POST /api/equipment-contexts HTTP/1.1
Content-Type: request body format

[request body]
```

Table 22 — Response codes and parameters for creating an equipment-context

Response code: 201 Created (Success)			
Parameter name	**Category**	**Type**	**Description**
equipment-context-uri (mandatory)	HTTP header field "Location"	URI	Globally unique URI for the created equipment-context on the server. It shall have the following format: "*scheme*://*authority*/api/equipment-contexts/*equipment-context-id*" (with *scheme* and *authority* as defined in IETF RFC 3986, and *equipment-context-id* being an implementation-specific server-unique identifier for the equipment-context).
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A CREATE equipment-context response body, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 201 Created
Location: https://example.com/equipment-context/Q12345
```

7.4.3 GET equipment-context (mandatory)

For retrieving an equipment-context, an equipment-context service shall implement the method **`GET`** on endpoint **`/api/equipment-contexts/`*`equipment-context-id`***, with the request parameters as listed in Table 23, and response codes and parameters as listed in Table 24.

Table 23 — Request parameters for retrieving an equipment-context

GET /api/equipment-contexts/*equipment-context-id*			
Parameter name	**Category**	**Type**	**Description**
equipment-context-id (mandatory)	URI path	String	Unique resource ID on the equipment-context service.

EXAMPLE 1 A GET equipment-context request. A client wants to get the equipment-context with ID "Q12345".

```
GET /api/equipment-contexts/Q12345 HTTP/1.1
Accept: preferred response body format
```

Table 24 — Response codes and parameters for retrieving an equipment-context

Response code: 200 OK (Success)			
Parameter name	**Category**	**Type**	**Description**
equipment-context (mandatory)	response body	as specified by *response body format*	The equipment-context is returned as complete body of the response. Refer to 7.4.1.
Response code: 404 Not Found (Resource not available)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A GET equipment-context response, following up on EXAMPLE 1.

```
HTTP/1.1 200 OK
Content-Type: response body format

[response body]
```

7.4.4 UPDATE equipment-context (mandatory)

For updating an equipment-context, an equipment-context service shall implement the method **`PUT`** on endpoint **`/api/equipment-contexts/`*`equipment-context-id`***, with the request parameters as listed in Table 25, and response codes and parameters as listed in Table 26.

The UPDATE operation shall overwrite all contents of an existing equipment-context on the server with the contents of an equipment-*context* provided as request parameter ("full overwrite" as opposed to "partial overwrite"). If the referenced equipment-context does not exist on the server, it shall return a 404 response code (see Table 26).

Table 25 — Request parameters for updating an equipment-context

PUT `/api/equipment-contexts/`*`equipment-context-id`*			
Parameter name	**Category**	**Type**	**Description**
equipment-context-id (mandatory)	URI path	String	Server-unique identifier for the equipment-context on the server to be updated (or created if not yet existing).
equipment-context (mandatory)	request body	*equipment-context* object	Refer to 7.4.1. All contents of the existing equipment-context with *equipment-context-id* on the server will be overwritten by the contents of request parameter *equipment-context*.

EXAMPLE 1 An UPDATE equipment-context request.

```
PUT /api/equipment-contexts/Q12345 HTTP/1.1
Content-Type: request body format

[request body]
```

Table 26 — Response codes and parameters for updating an equipment-context

Response code: 204 No Content (Successfully updated)			
Parameter name	**Category**	**Type**	**Description**
equipment-context-uri (mandatory)	HTTP header field "Location"	URI	Globally unique URI for the updated equipment-context on the server. It shall have the following format: "*scheme*://*authority*/api/equipment-contexts/*equipment-context-id*" (with *scheme* and *authority* as defined in IETF RFC 3986, and *equipment-context-id* being an implementation-specific server-unique identifier for the equipment-context).
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).
Response code: 404 Not Found (Invalid equipment-context ID)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 An UPDATE equipment-context response, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 204 No content
Location: https://example.com/equipment-context/Q12345
```

7.4.5 DELETE equipment-context (optional)

In general, an equipment-*context service* may dispose its equipment-contexts if they expire. The expiration period is implementation-specific.

For deleting an equipment-context, an equipment-context service may implement the method **`DELETE`** on endpoint **`/equipment-contexts/`*`equipment-context-id`***, with the request parameters as listed in Table 27, and response codes and parameters as listed in Table 28.

NOTE This operation is optional since a equipment-context service can choose to not allow to remove an existing equipment-context for the sake of URI stability. In any case, implementers are encouraged to think about a consistent policy on stable URIs (see e.g. the W3C URI Persistency Policy).

Table 27 — Request parameters for deleting an equipment-context

DELETE `/api/equipment-contexts/`*`equipment-context-id`*			
Parameter name	**Category**	**Type**	**Description**
equipment-context-id (mandatory)	URI path	String	Server-unique identifier for the equipment-context on the server to be deleted.

EXAMPLE 1 A DELETE equipment-context request. The equipment-context with ID "Q12345" is requested to be deleted. Note that there is no request body.

```
DELETE /api/equipment-contexts/Q12345 HTTP/1.1
```

Table 28 — Response codes and parameters for deleting an equipment-context

Response code: 204 No Content (Deletion enacted)			
Parameter name	**Category**	**Type**	**Description**
(no parameter)			
Response code: 404 Not Found (Invalid equipment-context ID)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A DELETE equipment-context response, indicating that the requested deletion was enacted, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 204 No Content
```

An equipment-context service may, at its own discretion, dispose an equipment-context on its own (without being called on the DELETE equipment-context operation) after it has not been used in a reasonable period of time.

7.4.6 GET equipment-context-list (mandatory)

For retrieving a list of equipment-contexts, a equipment-context service shall implement the method **`GET`** on endpoint **`/api/equipment-contexts`**, with the request parameters as listed in Table 29, and response codes and parameters as listed in Table 30. The equipment-context service shall return a non-empty list of URIs of all equipment-contexts that the requesting user has access to (see 7.4.3). If no match is found, the service shall respond with response code 404 (see Table 30).

NOTE For equipment-context services that employ user authentication, publicly available equipment-contexts can be requested by a client taking the role of an anonymous user.

Additional restrictions may be applied by the filter parameters *owner*, *updatable*, and *deletable* (see Table 29) which may be combined. A request may also include the pagination parameters *offset* and *limit* (see Table 29). In case the request specifies an *offset* that does not exist on the resulting list of URIs, the service shall respond with response code 404 (see Table 30).

Table 29 — Request parameters for retrieving a list of concept records

GET /api/equipment-contexts			
Parameter name	**Category**	**Type**	**Description**
owner (optional)	URI query parameter	String	Filter parameter: identification of a user that owns the equipment-contexts. This filter may be combined with the other filters in Table 29. User identification and ownership of equipment-contexts is implementation-specific for equipment-context services.
updatable (optional)	URI query parameter	Boolean	Filter parameter: If *updatable* is true, only equipment-contexts are listed for which the requesting user has the permission to update (see 7.4.4). If *updatable* is false, only equipment-contexts are listed for which the requesting user does not have the permission to update (see 7.4.4). This filter may be combined with the other filters in Table 29. The existence and details of a user authentication mechanism are implementation-specific.
deletable (optional)	URI query parameter	Boolean	Filter parameter: If *deletable* is true, only equipment-contexts are listed for which the requesting user has the permission to delete (see 7.4.5). If *deletable* is false, only equipment-contexts are listed for which the requesting user does not have the permission to delete (see 7.4.5). This filter may be combined with the other filters in Table 29. The existence and details of a user authentication mechanism are implementation-specific.
offset (optional)	URI query parameter	Number	Pagination parameter: The number (0 and positive numbers) of equipment-contexts to skip in the list of results. If *offset* is not specified, an offset of 0 shall be assumed.
limit (optional)	URI query parameter	Number	Pagination parameter: The number (positive) of equipment-contexts to return. Used for pagination. If *limit* is not specified, no limit shall be assumed.

EXAMPLE 1 A GET equipment-context-list request. A client requests the second (`offset=1`) equipment-context in the list of equipment-contexts for which the requester has permission to read and update, and that is owned by user with the identification "UserXYZ". Note that there is no request body supplied.

```
GET /api/equipment-contexts?owner=UserXYZ&updatable=true&offset=1&limit=1 HTTP/1.1
Accept: preferred response body format
```

Table 30 — Response codes and parameters for retrieving a list of concept records

Response code: 200 OK			
Parameter name	**Category**	**Type**	**Description**
total number of equipment-contexts (mandatory)	response body	Number	Total number (≥1) of equipment-contexts available on the equipment-context service for which the filter(s) in the request apply (but without having the pagination parameters applied).
equipment-context-URIs (mandatory)	response body	Array of URIs	*equipment-context-URIs* shall be a non-empty array of URIs (String) each referring to a *equipment-context* object (see 7.4.1). The order of the array elements shall be significant (to allow for pagination).

Table 30 *(continued)*

Response code: 400 Bad Request (Invalid parameters)			
error message	entire response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).
Response code: 404 Not Found			
error message	entire response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A GET equipment-context-list response (following up on EXAMPLE 1).

```
HTTP/1.1 200 OK
Content-Type: response body format

[response body]
```

7.5 Environment-context

7.5.1 General

An *environment-context service* shall implement all mandatory and optional operations in 7.5, with support for all mandatory and optional parameters. It shall support both XML and JSON mapping, as specified in Annexes A and B.

The information model for an *environment-context* object is as follows:

— An *environment-context* shall be a (possibly empty) Array of *property* objects, each with the following fields:

  — *name* (type String);

  — *value* (type String).

— Each *property* object may have any number of *descriptor* objects, each with the following fields:

  — *name* (type String);

  — *value* (type String).

— *Environment-context* may include any other objects with proprietary content, as long as they do not interfere with the above objects within *environment-context*.

See A.5.1 for the mapping of an environment-context in XML format, and B.5.1 for the mapping in JSON format.

NOTE ISO/IEC 24751-1 specifies how terms for describing environment-context characteristics can be registered.

7.5.2 CREATE environment-context (mandatory)

For creating an environment-context, an environment-context service shall implement the method **`POST`** on endpoint **`/api/environment-contexts`**, with the request parameters as listed in Table 31, and response codes and parameters as listed in Table 32.

Table 31 — Request parameters for creating an environment-context

POST /api/environment-contexts			
Parameter name	**Category**	**Type**	**Description**
environment-context (mandatory)	request body	*environment-context* object	Refer to 7.5.1.

EXAMPLE 1 A CREATE environment-context request body.

```
POST /api/environment-contexts HTTP/1.1
Content-Type: request body format

[request body]
```

Table 32 — Response codes and parameters for creating an environment-context

Response code: 201 Created (Success)			
Parameter name	**Category**	**Type**	**Description**
environment-context-uri (mandatory)	HTTP header field "Location"	URI	Globally unique URI for the created environment-context on the server. It shall have the following format: "*scheme*://*authority*/api/environment-contexts/*environment-context-id*" (with *scheme* and *authority* as defined in IETF RFC 3986, and *environment-context-id* being an implementation-specific server-unique identifier for the environment-context).
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A CREATE environment-context response body, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 201 Created
Location: https://example.com/api/environment-contexts/E12345
```

7.5.3 GET environment-context (mandatory)

For retrieving an environment-context, an environment-context service shall implement the method **`GET`** on endpoint **`/api/environment-contexts/`*`environment-context-id`***, with the request parameters as listed in Table 33, and response codes and parameters as listed in Table 34.

Table 33 — Request parameters for retrieving an environment-context

GET /api/environment-contexts/*environment-context-id*			
Parameter name	**Category**	**Type**	**Description**
environment-context-id (mandatory)	URI path	String	Unique resource ID on the environment-context service.

EXAMPLE 1 A GET environment-context request for the environment-context with identifier "E12345". Note that there is no request body.

```
GET /api/environment-contexts/E12345 HTTP/1.1
Accept: preferred response body format
```

Table 34 — Response codes and parameters for retrieving an environment-context

Response code: 200 OK (Success)			
Parameter name	**Category**	**Type**	**Description**
environment-context (mandatory)	response body	as specified by *response body format*	The environment-context is returned as complete body of the response. Refer to 7.5.1 for syntax.
Response code: 404 Not Found (Resource not available)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A GET environment-context response, following up on EXAMPLE 1.

```
HTTP/1.1 200 OK
Content-Type: response body format

[response body]
```

7.5.4 UPDATE environment-context (mandatory)

For updating an environment-context, an environment-context service shall implement the method **`PUT`** on endpoint **`/api/environment-contexts/`*`environment-context-id`***, with the request parameters as listed in Table 35, and response codes and parameters as listed in Table 36.

The UPDATE operation shall overwrite all contents of an existing environment-context on the server with the contents of an *environment-context* provided as request parameter ("full overwrite" as opposed to "partial overwrite"). If the referenced environment-context does not exist on the server, it shall return a 404 response code (see Table 6).

Table 35 — Request parameters for updating an environment-context

PUT `/api/environment-contexts/`*`environment-context-id`*			
Parameter name	**Category**	**Type**	**Description**
environment-context-id (mandatory)	URI path	String	Server-unique identifier for the environment-context on the server to be updated (or created if not yet existing).
environment-context (mandatory)	request body	*environment-context* object	Refer to 7.5.1 for syntax. All contents of the existing environment-context with *environment-context-id* on the server will be overwritten by the contents of request parameter environment-context.

EXAMPLE 1 An UPDATE environment-context request, assuming that an environment-context with ID "E12345" has been created according to EXAMPLE 1 in 7.5.2.

```
PUT /api/environment-contexts/E12345 HTTP/1.1
Content-Type: request body format

[request body]
```

Table 36 — Response codes and parameters for updating an environment-context

Response code: 204 No Content (Successfully updated)			
Parameter name	**Category**	**Type**	**Description**
environment-context-uri (mandatory)	HTTP header field "Location"	URI	Globally unique URI for the updated environment-context on the server. It shall have the following format: "*scheme*://*authority*/api/environment-contexts/*environment-context-id*" (with *scheme* and *authority* as defined in IETF RFC 3986, and *environment-context-id* being an implementation-specific server-unique identifier for the environment-context).
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).
Response code: 404 Not Found (Invalid task-context ID)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 An UPDATE environment-context response, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 204 No content
Location: https://example.com/api/environment-contexts/E12345
```

7.5.5 DELETE environment-context (optional)

In general, an environment-*context service* may dispose its environment-contexts if they expire. The expiration period is implementation-specific.

For deleting an environment-context, an environment-context service may implement the method **`DELETE`** on endpoint **`/api/environment-contexts/`*`environment-context-id`***, with the request parameters as listed in Table 37, and response codes and parameters as listed in Table 38.

NOTE This operation is optional since a environment-context service can choose to not allow to remove an existing environment-context for the sake of URI stability. In any case, implementers are encouraged to think about a consistent policy on stable URIs (see e.g. the W3C URI Persistency Policy).

Table 37 — Request parameters for deleting an environment-context

DELETE `/api/environment-contexts/`*`environment-context-id`*			
Parameter name	**Category**	**Type**	**Description**
environment-context-id (mandatory)	URI path	String	Server-unique identifier for the environment-context on the server to be deleted.

EXAMPLE 1 A DELETE environment-context request. The environment-context with ID "E12345" is requested to be deleted. Note that there is no request body.

```
DELETE /api/environment-contexts/E12345 HTTP/1.1
```

Table 38 — Response codes and parameters for deleting an environment-context

Response code: 204 No Content (Deletion enacted)			
Parameter name	**Category**	**Type**	**Description**
(no parameter)			
Response code: 404 Not Found (Invalid environment-context ID)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A DELETE environment-context response, indicating that the requested deletion was enacted, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 204 No Content
```

7.5.6 GET environment-context-list (mandatory)

For retrieving a list of environment-contexts, a environment-context service shall implement the method **`GET`** on endpoint **`/api/environment-contexts`**, with the request parameters as listed in Table 39, and response codes and parameters as listed in Table 40. The environment-context service shall return a list of URIs of all environment-contexts that the requesting user has access to (see 7.5.3). If no match is found, the service shall respond with response code 404 (see Table 40).

NOTE For environment-context services that employ user authentication, publicly available environment-contexts can be requested by a client taking the role of an anonymous user.

Additional restrictions may be applied by the filter parameters *owner*, *updatable*, and *deletable* (see Table 39) which may be combined. A request may also include the pagination parameters *offset* and *limit* (see Table 39). In case the request specifies an *offset* that does not exist on the resulting list of URIs, the service shall respond with response code 404 (see Table 40).

Table 39 — Request parameters for retrieving a list of concept records

GET /api/environment-contexts			
Parameter name	**Category**	**Type**	**Description**
owner (optional)	URI query parameter	String	Filter parameter: identification of a user that owns the environment-contexts. This filter may be combined with the other filters in Table 39. User identification and ownership of environment-contexts is implementation-specific for environment-context services.
updatable (optional)	URI query parameter	Boolean	Filter parameter: If *updatable* is true, only environment-contexts are listed for which the requesting user has the permission to update (see 7.5.4). If *updatable* is false, only environment-contexts are listed for which the requesting user does not have the permission to update (see 7.5.4). This filter may be combined with the other filters in Table 39. The existence and details of a user authentication mechanism are implementation-specific.
deletable (optional)	URI query parameter	Boolean	Filter parameter: If *deletable* is true, only environment-contexts are listed for which the requesting user has the permission to delete (see 7.5.5). If *deletable* is false, only environment-contexts are listed for which the requesting user does not have the permission to delete (see 7.5.5). This filter may be combined with the other filters in Table 39. The existence and details of a user authentication mechanism are implementation-specific.
offset (optional)	URI query parameter	Number	Pagination parameter: The number (0 and positive numbers) of environment-contexts to skip in the list of results. If *offset* is not specified, an offset of 0 shall be assumed.
limit (optional)	URI query parameter	Number	Pagination parameter: The number (positive) of environment-contexts to return. Used for pagination. If *limit* is not specified, no limit shall be assumed.

EXAMPLE 1 A GET environment-context-list request. A client requests the second (`offset=1`) environment-context in the list of environment-contexts for which the requester has permission to read and update, and that is owned by user with the identification "UserXYZ". Note that there is no request body supplied.

```
GET /api/environment-contexts?owner=UserXYZ&updatable=true&offset=1&limit=1 HTTP/1.1
Accept: preferred response body format
```

Table 40 — Response codes and parameters for retrieving a list of concept records

Response code: 200 OK			
Parameter name	**Category**	**Type**	**Description**
total number of environment-contexts (mandatory)	response body	Number	Total number (≥1) of environment-contexts available on the environment-context service for which the filter(s) in the request apply (but without having the pagination parameters applied).
environment-context-URIs (mandatory)	response body	Array of URIs	*environment-context-URIs* shall be a non-empty array of URIs (String) each referring to a *environment-context* object (see 7.5.1). The order of the array elements shall be significant (to allow for pagination).

Table 40 *(continued)*

Response code: 400 Bad Request (Invalid parameters)			
error message	entire response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).
Response code: 404 Not Found			
error message	entire response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A GET environment-context-list response (following up on EXAMPLE 1).

```
HTTP/1.1 200 OK
Content-Type: response body format

[response body]
```

7.6 Resource

7.6.1 General

A *resource service* shall implement all mandatory and optional operations in 7.6, with support for all mandatory and optional parameters. It shall support both XML and JSON mapping, as specified in Annexes A and B.

NOTE 1 If the client does not know a resource's ID, it can search for matching resources through operations on a matching service (see 7.8).

NOTE 2 See 7.7 and its operations on resource description for specifying and retrieving metadata on resources.

7.6.2 CREATE resource (mandatory)

For uploading a resource, a resource service shall implement the method **`POST`** on endpoint **`/api/resources`**, with the request parameters as listed in Table 41, and response codes and parameters as listed in Table 42.

Table 41 — Request parameters for creating a resource

POST `/api/resources`			
Parameter name	**Category**	**Type**	**Description**
resource (mandatory)	request body	as specified by the HTTP header field `Content-Type`	*resource* shall be encoded as the complete body of the request message. Encoding restrictions apply (in accordance with IETF RFC 7230).

EXAMPLE 1 A CREATE resource request. The resource is the string "Power" and has the MIME type "text/plain".

```
POST /api/resources HTTP/1.1
Content-Type: text/plain

Power
```

Table 42 — Response codes and parameters for creating a resource

Response code: 201 Created (Success)			
Parameter name	**Category**	**Type**	**Description**
resource-uri (mandatory)	HTTP header field "Location"	URI	Globally unique URI for the created resource on the server. It shall have the following format: "*scheme*://*authority*/api/resources/*resource-id*" (with *scheme* and *authority* as defined in IETF RFC 3986, and *resource-id* being an implementation-specific server-unique identifier for the resource).
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A CREATE resource response, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 201 Created
Location: https://example.com/api/resources/R12345
```

7.6.3 GET resource-by-ID (mandatory)

For retrieving a resource by ID, a resource service shall implement the method **`GET`** on endpoint **`/api/resources/`*`resource-id`***, with the request parameters as listed in Table 43, and response codes and parameters as listed in Table 44.

Table 43 — Request parameters for retrieving a resource by ID

GET `/api/resources/`*`resource-id`*			
Parameter name	**Category**	**Type**	**Description**
resource-id (mandatory)	URI path	String	Unique resource ID on the resource service.

EXAMPLE 1 A GET resource-by-ID request. A client wants to get the resource with ID "R12345". Note that the type of the response body depends on the MIME type of the requested resource.

```
GET /api/resources/R12345 HTTP/1.1
```

Table 44 — Response codes and parameters for retrieving a resource by ID

Response code: 200 OK (Success)			
Parameter name	**Category**	**Type**	**Description**
resource (mandatory)	response body	as specified by the HTTP header field `Content-Type`	The resource is returned as complete body of the response. Refer to 7.6.1 for syntax.
Response code: 404 Not Found (Resource not available)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A GET resource-by-ID response, following up on EXAMPLE 1. The text "Power" is delivered as the message body, with a MIME type of "text/plain" indicated in the HTTP header of the response. Note that the "Content-Type" Header field depends on the MIME type of the requested resource, delivered as response body.

```
HTTP/1.1 200 OK
Content-Type: text/plain

Power
```

7.6.4 GET resource-from-listing (mandatory)

For retrieving a resource by reference to an item in a listing, a resource service shall implement the method **`GET`** on endpoint **`/api/resources`**, with the request parameters as listed in Table 45, and response codes and parameters as listed in Table 46.

Table 45 — Request parameters for retrieving a resource by reference to a listing item

GET `/api/resources?listing-id=`*`listing-id`*`&index=`*`index`*			
Parameter name	**Category**	**Type**	**Description**
listing-id (mandatory)	URI query parameter `listing-id`	String	*listing-id* as obtained by a previous "Get listing" request (see 7.8.3).
index (mandatory)	URI query parameter `index`	Number	Reference to the item with number *index* (with base 0).

EXAMPLE 1 A GET resource-from-listing request. A client wants to retrieve the first item (with index 0) from the listing with ID "L12345". Note that there is no preferred MIME type of the response body since the MIME type is determined by the requested resource.

```
GET /api/resources?listing-id=L12345&item=0 HTTP/1.1
```

Table 46 — Response codes and parameters for retrieving a resource by reference to a listing item

Response code: 200 OK (Success)			
Parameter name	**Category**	**Type**	**Description**
resource (mandatory)	Message body	MIME type of the resource	The resource is returned as body of the response. The resource service shall indicate its MIME type through the *Content-Type* HTTP header field in the response.
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).
Response code: 404 Not Found (Resource not available)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A GET resource-from-listing response, following up on EXAMPLE 1. The requested resource (an image) is delivered as the message body, with a MIME type of "image/png" indicated in the HTTP header of the response.

```
HTTP/1.1 200 OK
Content-Disposition: attachment; filename*=UTF-8''remex-logo-color-300x107.png;
filename="remex-logo-color-300x107.png"
Content-Transfer-Encoding: binary
Content-Length: 21516
Content-Type: image/png

[response body encoding the image]
```

7.6.5 UPDATE resource (optional)

For updating a resource, a resource service may implement the method **`PUT`** on endpoint **`/api/resources/`*`resource-id`***, with the request parameters as listed in Table 47, and response codes and parameters as listed in Table 48.

NOTE Implementations of resource services can choose to not implement the UPDATE resource operation, in order to make resources immutable, and thus to prevent old resource URIs from becoming invalid due to resource overwriting. Other implementations can choose to implement the UPDATE resource operation, but set a restriction on allowing overwrite of resources at development time only (possibly enforced by a staging model that is out of scope for this document). In any case, implementers are encouraged to think about a consistent policy on stable URIs (see e.g. the W3C URI Persistency Policy).

The UPDATE operation shall completely overwrite an existing resource on the server with the contents of a *resource* provided as request parameter. If the referenced resource does not exist on the server, it shall return a 404 response code (see Table 48).

Table 47 — Request parameters for updating a resource

PUT `/api/resources/`*`resource-id`*			
Parameter name	**Category**	**Type**	**Description**
resource-id (mandatory)	URI path	String	Server-unique identifier for the *resource* on the server to be updated
resource (mandatory)	request body	*resource* object	Refer to 7.6.1. MIME type of *resource* as specified by the HTTP header field `Content-Type`.

EXAMPLE 1 An UPDATE resource request. The resource with ID "R12345" is overwritten with the body of the request message. Note that *`request body format`* specifies the MIME type of the resource, delivered as request body.

```
PUT /api/resources/R12345 HTTP/1.1
Content-Type: request body format

[request body]
```

Table 48 — Response codes and parameters for updating a resource

Response code: 204 No Content (Successfully updated)			
Parameter name	**Category**	**Type**	**Description**
resource-uri (mandatory)	HTTP header field `Location`	URI	Globally unique URI for the updated resource on the server. It shall have the following format: "*scheme*://*authority*/api/resources/*resource-id*" (with *scheme* and *authority* as defined in IETF RFC 3986, and *resource-id* being an implementation-specific server-unique identifier for the resource).
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).
Response code: 404 Not Found (Invalid resource ID)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 An UPDATE resource response, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 204 No content
Location: https://example.com/api/resource/R12345
```

7.6.6 DELETE resource (optional)

In general, a *resource service* may dispose its resources if they expire. The expiration period is implementation-specific.

For deleting a resource, a resource service may implement the method **`DELETE`** on endpoint **`/api/resources/`*`resource-id`***, with the request parameters as listed in Table 49, and response codes and parameters as listed in Table 50.

NOTE This operation is optional since resource service implementations can choose to not allow to remove an existing resource for the sake of URI stability. See also the NOTE in 7.7.5.

Table 49 — Request parameters for deleting a resource

DELETE `/api/resources/`*`resource-id`*			
Parameter name	**Category**	**Type**	**Description**
resource-id (mandatory)	URI path	String	Server-unique identifier for the resource on the server to be deleted.

EXAMPLE 1 A DELETE resource request for the resource with ID "R12345". Note that there is no request body.

```
DELETE /api/resources/R12345 HTTP/1.1
```

Table 50 — Response codes and parameters for deleting a resource

Response code: 204 No Content (Deletion enacted)			
Parameter name	**Category**	**Type**	**Description**
(no parameter)			
Response code: 404 Not Found (Invalid resource ID)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A DELETE resource response, following up on EXAMPLE 1. The response indicates that the requested deletion was enacted. Note that there is no response body.

```
HTTP/1.1 204 No Content
```

7.7 Resource-description

7.7.1 General

A *resource-description service* shall implement all mandatory and optional operations in 7.7, with support for all mandatory and optional parameters. It shall support both XML and JSON mapping, as specified in Annexes A and B.

A resource-description is a set of metadata describing a resource. The resource may be any object that is resolvable through a URI, or it may be self-implied by the resource description. The resource-description is hosted by a resource-description service.

The information model for a *resource-description* object is as follows:

— A *resource-description* shall be a (possibly empty) Array of *property* objects, each with the following fields:

  — *name* (type String);

  — *value* (type String).

— Each *property* object may have any number of *descriptor* objects, each with the following fields:

  — *name* (type String);

— *value* (type String).

— *Resource-description* may include any other objects with proprietary content, as long as they do not interfere with the above objects within *resource-description*.

See A.7.1 for the mapping of a resource-description in XML format, and B.7.1 for the mapping in JSON format.

EXAMPLE 1 A resource-description could describe a sign language video explaining the meaning of the IBAN input field in the wire transfer form of an online banking application. A user interface implementation could integrate this video in its user interface for a user who prefers to get information in sign language rather than in written form. This resource-description would include a link to the video stored external from the resource-description.

EXAMPLE 2 Another resource-description could present a set of operating system-specific settings for a particular user. This resource-description would include settings such as font size, contrast theme, mouse speed, task bar location, etc. This resource-description would not include a link to an external resource since the resource-description itself (i.e. the set of particular settings) is the implicit resource.

NOTE 1 ISO/IEC 24751-1 provides a specification of resource-description characteristics, and of the registration of terms for describing resource-description characteristics.

NOTE 2 If the client does not know a resource-description's ID, it can search for matching resource-descriptions through operations on a resource listing (see 7.8).

NOTE 3 Resource-description and pertaining resource/object do not need to be hosted by the same server. In fact, the resource/object could be hosted by a simple webserver, not necessarily conforming to this document.

7.7.2 CREATE resource-description (mandatory)

For creating a resource-description, a resource-description service shall implement the method **`POST`** on endpoint **`/api/resource-descriptions`**, with the request parameters as listed in Table 51, and response codes and parameters as listed in Table 52.

Table 51 — Request parameters for creating a resource-description

`POST /api/resource-descriptions`			
Parameter name	**Category**	**Type**	**Description**
resource-description (mandatory)	request body	*resource-description* object	Refer to 7.7.1 for a description of the *resource-description* object. One *property* in *resource-description* may have the name "http://openurc.org/ns/uirf/resource-description#resource-uri", with the corresponding value bearing a resolvable URI for retrieving the resource (e.g. in the form "http://host/api/resources-by-ID/api/resources-id", see 7.6.3).

EXAMPLE 1 A CREATE resource-description request. Note that the property named "resource-uri" is mandatory, but does not need to be the first in the Array.

```
POST /api/resource-descriptions HTTP/1.1
Content-Type: request body format

[request body]
```

Table 52 — Response codes and parameters for creating a resource-description

Response code: 201 Created (Success)			
Parameter name	**Category**	**Type**	**Description**
resource-description-uri (mandatory)	HTTP header field `Location`	URI	Globally unique URI for the created resource-description on the server. It shall have the following format: "*scheme*://*authority*/api/resource-descriptions/*resource-description-id*" (with *scheme* and *authority* as defined in IETF RFC 3986, and *resource-description-id* being an implementation-specific server-unique identifier for the resource-description).
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A CREATE resource-description response, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 201 Created
Location: https://example.com/api/resource-descriptions/RD12345
```

7.7.3 GET resource-description-by-ID (mandatory)

For retrieving a resource-description by ID, a resource-description service shall implement the method **`GET`** on endpoint **`/api/resource-descriptions/`*`resource-description-id`***, with the request parameters as listed in Table 53, and response codes and parameters as listed in Table 54.

Table 53 — Request parameters for retrieving a resource-description by ID

GET `/api/resource-descriptions/`*`resource-description-id`*			
Parameter name	**Category**	**Type**	**Description**
resource-description-id (mandatory)	URI query parameter	String	Unique resource-description ID on the resource-description service.

EXAMPLE 1 A GET resource-description-by-ID request. A client wants to retrieve the resource-description with ID "RD12345".

```
GET /api/resource-descriptions/RD12345 HTTP/1.1
Accept: preferred response body format
```

Table 54 — Response codes and parameters for retrieving a resource-description by ID

Response code: 200 OK (Success)			
Parameter name	**Category**	**Type**	**Description**
resource-description (mandatory)	response body	as specified by *response body format*	The resource-description is returned as complete body of the response. Refer to 7.7.1 for syntax.
Response code: 404 Not Found (Resource-description not available)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A GET resource-description-by-ID response, following up on EXAMPLE 1.

```
HTTP/1.1 200 OK
Content-Type: response body format

[response body]
```

7.7.4 GET resource-description-from-listing (mandatory)

For retrieving a resource-description by reference to an item in a listing, a resource-description service shall implement the method **`GET`** on endpoint **`/api/resource-descriptions`**, with the request parameters as listed in Table 55, and response codes and parameters as listed in Table 56.

Table 55 — Request parameters for retrieving a resource-description by reference to a listing item

GET `/api/resource-descriptions?listing-id=`*`listing-id`*`&index=`*`index`*			
Parameter name	**Category**	**Type**	**Description**
listing-id (mandatory)	URI query parameter	String	*listing-id* as obtained by a previous "Get listing" request (see 7.8.3).
index (mandatory)	URI query parameter	Number	Reference to the item with number *index* (with base 0).

EXAMPLE 1 A GET resource-from-listing request. A client wants to retrieve the first item (with index 0) from the listing with ID "L12345".

```
GET /api/resource-descriptions?listing-id=L12345&item=0 HTTP/1.1
Accept: preferred response body format
```

Table 56 — Response codes and parameters for retrieving a resource-description by reference to a listing item

Response code: 200 OK (Success)			
Parameter name	**Category**	**Type**	**Description**
resource-description (mandatory)	response body	as specified by *response body format*	The resource-description is returned as complete body of the response. Refer to 7.7.1 for syntax.
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).
Response code: 404 Not Found (Resource-description not available)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A GET resource-from-listing response, following up on EXAMPLE 1.

```
HTTP/1.1 200 OK
Content-Type: response body format

[response body]
```

7.7.5 UPDATE resource-description (mandatory)

For updating a resource-description, a resource-description service shall implement the method **`PUT`** on endpoint **`/api/resource-descriptions/`*`resource-description-id`***, with the request parameters as listed in Table 57, and response codes and parameters as listed in Table 58.

The UPDATE operation shall overwrite all contents of an existing resource-description on the server with the contents of a *resource-description* provided as request parameter ("full overwrite" as opposed to "partial overwrite"). If the referenced resource-description does not exist on the server, it shall return a 404 response code (see Table 58).

Table 57 — Request parameters for updating a resource-description

PUT /api/resource-descriptions/*resource-description-id*			
Parameter name	**Category**	**Type**	**Description**
resource-description-id (mandatory)	URI path	String	Server-unique identifier for the *resource-description* on the server to be updated (or created if not yet existing).
resource-description (mandatory)	request body	*resource-description* object	Refer to 7.7.1 for syntax. All contents of the existing resource-description with *resource-description-id* on the server will be overwritten by the contents of request parameter *resource-description*.

NOTE It is possible to update the value of a property named "http://openurc.org/ns/dirf/resource-description#resource-uri" of a resource description, and thus assign a new resource or a new resource location to an existing resource description.

EXAMPLE 1 An UPDATE resource-description request for a resource-description with ID "RD12345".

```
PUT /api/resource-descriptions/RD12345 HTTP/1.1
Content-Type: request body format

[request body]
```

Table 58 — Response codes and parameters for updating a resource-description

Response code: 204 No Content (Successfully updated)			
Parameter name	**Category**	**Type**	**Description**
resource-description-uri (mandatory)	HTTP header field `Location`	URI	Globally unique URI for the updated resource-description on the server. It shall have the following format: "*scheme*://*authority*/api/resource-descriptions/*resource-description-id*" (with *scheme* and *authority* as defined in IETF RFC 3986, and *resource-description-id* being an implementation-specific server-unique identifier for the resource-description).
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).
Response code: 404 Not Found (Invalid task-context ID)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 An UPDATE resource-description response, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 204 No content
Location: https://example.com/api/resource-descriptions/RD12345
```

7.7.6 DELETE resource-description (optional)

In general, a *resource-description service* may dispose its resource-descriptions if they expire. The expiration period is implementation-specific.

For deleting a resource-description, a resource-description service may implement the method **`DELETE`** on endpoint **`/api/resource-descriptions/`*`resource-description-id`***, with the request parameters as listed in Table 59, and response codes and parameters as listed in Table 60.

NOTE This operation is optional since a resource-description service can choose to not allow to remove an existing resource description for the sake of URI stability. In any case, implementers are encouraged to think about a consistent policy on stable URIs (see e.g. the W3C URI Persistency Policy).

Table 59 — Request parameters for deleting a resource-description

DELETE /api/resource-descriptions/*resource-description-id*			
Parameter name	**Category**	**Type**	**Description**
resource-description-id (mandatory)	URI path	String	Server-unique identifier for the resource-description on the server to be deleted.

EXAMPLE 1 A DELETE resource-description request. Note that there is no request body.

```
DELETE /api/resource-descriptions/RD12345 HTTP/1.1
```

Table 60 — Response codes and parameters for deleting a resource-description

Response code: 204 No Content (Deletion enacted)			
Parameter name	**Category**	**Type**	**Description**
(no parameter)			
Response code: 404 Not Found (Invalid resource-description-id)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A DELETE resource-description response, following up on EXAMPLE 1. The response indicates that the requested deletion was enacted. Note that there is no response body.

```
HTTP/1.1 204 No Content
```

7.8 Listing

7.8.1 General

A *matching service* shall implement all mandatory and optional operations in 7.8, with support for all mandatory and optional parameters. It shall support both XML and JSON mapping, as specified in Annexes A and B.

A client can create, get and delete a listing of matching resources on a matching service.

By creating a listing, a client requests the matching service to prepare an Array of resource descriptions (each optionally including a URI reference to a resource) that meet specific criteria, given as an Array of properties (key-value pairs).

The matching service should find a reasonable number of resources that match most of the characteristics as given by the referenced user-contexts (see 7.2), the referenced tasks-contexts (see 7.3), the referenced equipment-contexts (see 7.4), the referenced environment-contexts (see 7.5), and the specific criteria as specified in the request parameter *resource-description-query* (see Table 61). The response listing shall contain zero or more matching resources, ordered by decreasing match quality. Match quality and the specific matching algorithm are implementation-specific.

NOTE A metric for the match quality could be to count the number of matching properties, though properties could also differ in their internal weights (implementation specific for matching services). A matching service may use various approaches for generating a ranked Array of resources, e.g. rule-based or statistical approaches.

7.8.2 CREATE listing (mandatory)

For creating a listing, a matching service shall implement the method **`POST`** on endpoint **`/api/listings`**, with the request parameters as listed in Table 61, and response codes and parameters as listed in Table 62 below.

Table 61 — Request parameters for creating a listing

POST `/api/listings`			
Parameter name	**Category**	**Type**	**Description**
user-context-uris (optional)	request body	URI	(Possibly empty) Array of references to existing *user-contexts* on user-context services, see 7.2.
task-context-uris (optional)	request body	URI	(Possibly empty) Array of references to existing *task-contexts* on equipment-context services, see 7.3.
equipment-context-uris (optional)	request body	URI	(Possibly empty) Array of references to existing *equipment-contexts* on equipment-context services, see 7.4.
environment-context-uris (optional)	request body	URI	(Possibly empty) Array of references to existing *environment-contexts* on environment-context services, see 7.5.
resource-description-query (mandatory)	request body	*resource-description* object	Exactly one *resource description* object to be used as a match filter. Properties can be partially filled. Refer to 7.7.1 for syntax.

NOTE The parameters *user-context-uris*, *task-context-uris*, *equipment-context-uris* and *environment-context-uris* are optional for the client since it is possible to search for a matching resource-description without considering a specific user-context, a specific equipment-context or a specific environment-context.

EXAMPLE 1 A CREATE listing request:

```
POST /api/listings HTTP/1.1
Content-Type: request body format

[request body]
```

Table 62 — Response codes and parameters for creating a listing

Response code: 201 Created (Success)			
Parameter name	**Category**	**Type**	**Description**
listing-uri (mandatory)	HTTP header field "Location"	URI	Globally unique URI for the created listing object on the server. It shall have the following format: "*scheme*://*authority*/api/listings/*listing-id*" (with *scheme* and *authority* as defined in IETF RFC 3986, and *listing-id* being an implementation-specific server-unique identifier for the listing).
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A CREATE listing response, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 201 Created
Location: https://example.com/api/listings/L12345
```

7.8.3 GET listing (mandatory)

For receiving an Array of matching resource descriptions, a matching service shall implement the method **`GET`** on endpoint **`/api/listings/`*`listing-id`***, with the request parameters as listed in Table 63, and response codes and parameters as listed in Table 64 below.

If successful (i.e. the matching service has found at least one resource description that matches the request), the matching service shall respond with an HTTP status code of 200, and an Array of resource-description-uris. Otherwise the matching service shall respond with the appropriate HTTP status code (see below) and an empty body.

Table 63 — Parameters for retrieving a listing

GET /api/listings/*listing-id*?start=*start*&max=*max*			
Parameter name	**Category**	**Type**	**Description**
listing-id (mandatory)	URI path	String	Reference to a previously created listing object, see 7.8.2.
start (optional)	URI query parameter	Number	Start index of the response listing (with base 0). If missing, defaults to 0. NOTE Together with the max parameter, this allows for clients to retrieve the listing in "slices", by sending subsequent requests on the same query, but with different start parameters.
max (optional)	URI query parameter	Number	Requested maximal number of items in the response. If missing, defaults to an implementation-specific value.

EXAMPLE 1 A GET listing request. A client wants to retrieve the first 100 entries of the listing with ID "L12345".

```
GET /api/listings/L12345?start=0&max=100 HTTP/1.1
Accept: preferred response body format
```

Table 64 — Response codes and parameters for retrieving a listing

Response code: 200 OK (Success)			
Parameter name	**Category**	**Type**	**Description**
start (mandatory)	response body	Integer	Start index of the elements in parameter *resources* (with base 0).
count (mandatory)	response body	Integer	Number of items in parameter *resources* (starting from the element with index *start*). If *count* < *max* the client shall assume that there are no further items available in the listing object beyond those in parameter *resources*.
resource-description-uris (mandatory)	response body	Non-empty Array of URIs	Non-empty Array of globally unique URIs for matching *resource-description* objects.
Response code: 204 No Content (Listing object exhausted)			
(no parameter)			
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).
Response code: 404 Not Found (Listing not available)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 GET listing response, following up on EXAMPLE 1.

```
HTTP/1.1 200 OK
Content-Type: response body format

[response body]
```

7.8.4 DELETE listing (optional)

In general, a *matching service* may dispose its listings if they expire. The expiration period is implementation-specific.

For deleting a listing, a matching service may implement the method **`DELETE`** on endpoint **`/api/listings/`*`listing-id`***, whereby ***`listing-id`*** is the unique listing ID as returned by the

create operation (see 7.8.2), with the request parameters as listed in Table 65, and response codes and parameters as listed in Table 66 below.

NOTE This operation is optional since a matching service can choose to not allow to remove an existing listing for the sake of URI stability. In any case, implementers are encouraged to think about a consistent policy on stable URIs (see e.g. the W3C URI Persistency Policy).

Table 65 — Request parameters for deleting a listing

DELETE /api/listings/*listing-id*			
Parameter name	**Category**	**Type**	**Description**
listing-id (mandatory)	URI path	String	Server-unique identifier for the listing object on the server to be deleted.

EXAMPLE 1 A DELETE listing request for a listing with ID "L12345". Note that there is no request body.

```
DELETE /api/listings/L12345 HTTP/1.1
```

Table 66 — Response codes and parameters for deleting a listing

Response code: 204 No Content (Deletion enacted)			
Parameter name	**Category**	**Type**	**Description**
(no parameter)			
Response code: 404 Not Found (Invalid listing-id)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A DELETE listing response, indicating that the requested deletion (see EXAMPLE 1) was enacted. Note that there is no response body.

```
HTTP/1.1 204 No Content
```

7.8.5 CONFIRM user-rating (optional)

For receiving a user-rating for a resource description, a matching service may implement the method **`POST`** on endpoint **`/api/user-rating`**, with the request parameters as listed in Table 67, and response codes and parameters as listed in Table 68 below.

NOTE 1 A user-rating can be any confirmation of preference by the user, as perceived by a system. Such a rating may be expressed explicitly by the user (e.g. by filling out a survey or feedback form), or may be implicitly perceived by the system (e.g. the system may assume that the user prefers a particular setting if they finished changing parameters). In the absence of a more granular rating, the two values –1.0 (for dislike) and 1.0 (for like) can be used.

NOTE 2 This operation is optional since a matching service can choose to implement a simple matching model that does not attempt to learn from previous interactions with the user.

Table 67 — Request parameters for confirming a user-rating for a resource description

POST /api/user-rating			
Parameter name	**Category**	**Type**	**Description**
user-context-uris (optional)	request body	URI	(Possibly empty) Array of references to existing *user-contexts* on user-context services, see 7.2.
task-context-uris (optional)	request body	URI	(Possibly empty) Array of references to existing *task-contexts* on equipment-context services, see 7.3.
equipment-context-uris (optional)	request body	URI	(Possibly empty) Array of references to existing *equipment-contexts* on equipment-context services, see 7.4.

Table 67 *(continued)*

POST /api/user-rating			
Parameter name	**Category**	**Type**	**Description**
environment-context-uris (optional)	request body	URI	(Possibly empty) Array of references to existing *environment-contexts* on environment-context services, see 7.5.
resource-description-uri (mandatory)	request body	URI	Reference to exactly one resource description object that was rated by the user. Refer to 7.7.1 for syntax.
user-rating (mandatory)	request body	float	Any float value in the range of [–1.0; 1.0].

EXAMPLE 1 A CONFIRM user-rating request for a resource description with ID "RD12345", and a user-rating of 1. Note that the properties "user-context-uris", "task-context-uris", "equipment-context-uris" and "environment-context-uris" are optional.

```
POST /api/user-rating HTTP/1.1
Content-Type: request body format

[request body]
```

Table 68 — Response codes and parameters for confirming a user-rating for a resource description

Response code: 201 Created (Success)			
Parameter name	**Category**	**Type**	**Description**
(no parameter)			
Response code: 400 Bad Request (Invalid parameters)			
error-message	response body	String	Human readable text explaining the error and its cause (not subject to XML or JSON mapping).

EXAMPLE 2 A CONFIRM user-rating response, following up on EXAMPLE 1. Note that there is no response body.

```
HTTP/1.1 201 Created
```

8 Security considerations

A resource service 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 resource service should provide an HTTPS-based interface (as specified in IETF RFC 7230) for all operations (see section 7). 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.

Annex A
(normative)

Mapping for XML

A.1 General

For mapping the parameters of the operations in Clause 7 to XML, the requirements in this annex shall be met.

NOTE In this annex, only parameters of category "request body" or "response body" are listed. Parameters of categories "URI path", "URI query parameter" and "HTTP header field" are not subject to XML mapping, and are to be coded as specified in Clause 7.

The message body (if applicable) and response body (if applicable) shall comply to W3C XML 1.0. Neither shall contain any namespace declaration.

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

The namespace prefix `xs:` shall be a placeholder for the standard type namespace of W3C XML Schema Part 2.

The namespace prefix `xsi:` shall be a placeholder for the XML schema instance namespace "http://www.w3.org/2001/XMLSchema-instance" (in accordance with W3C XML Schema Part 1).

The NULL value shall be mapped to the `xsi:nil` attribute with value `true` (in accordance with W3C XML Schema Part 1).

The root element of any request body shall be the XML element `<request>` containing the pertinent parameters (see operations in Clause 7).

The root element of any response body shall be the XML element `<response>` containing the pertinent parameters (see operations in Clause 7).

Entity and character references for reserved characters shall apply, when contained in a property name or value (in accordance with XML 1.0, sections 4.1 and 4.6).

EXAMPLE Use "<" for "<", ">" for ">", "&" for "&", and """ for a double quote character (").

For any mapping defined in this clause, proprietary information may be added as attribute or element of any name within any XML element (e.g. within `<request>` or `<response>`), as long as the additional information does not interfere with the information items as specified in this annex.

A.2 User-context

A.2.1 General

For the XML mapping of a *user-context* object (see 7.2.1), the following shall apply:

— The *user-context* object is the XML element `<user-context>`.

— The *option* object is the XML element `<option>` within `<user-context>`. It shall have an `id` attribute with its *identifier* (type `xs:string`).

— `<option>` may have an XML element `<name>` with the *name* of the *option* as character data content (type `xs:string`).

— *Preferences* shall be a sequence of `<preference>` elements, each including the following:

  — A `key` attribute (type `xs:anyURI`), bearing the *key* of the *preference*.

  — A `value` attribute (type `xs:string`), bearing the *value* of the *preference*.

— *Conditions* shall be a sequence of `<condition>` elements within `<option>`.

— Each `<condition>` element shall have exactly one of the following attributes:

  — `type` (type `xs:string`), indicating the *type* of the *condition* (if *condition* is an operation);

  — `value` (type `xs:string`), indicating the *value* of the *condition* (if *condition* is a literal).

— If `<condition>` has a `type` attribute, `<condition>` shall have one or more `<operand>` elements as children, with the same restrictions (attributes) as the `<condition>` element within `<option>`. In this way, `<operand>` elements may be nested.

— If `<condition>` has a `value` attribute, `<condition>` shall have no children.

EXAMPLE 1 A user-context, consisting of a single option (with ID "default") with no condition (i.e. it is always active). The option has no name. It states that speech output is always to be on.

```
<user-context>
   <option id="default">
      <preference key="http://terms.gpii.net/speech-output" value="true"/>
   </option>
</user-context>
```

EXAMPLE 2 A user-context with a single option (with ID "default"), named "high pitch for upper case". This option is active if character case is to be indicated by pitch. It states that – in case the option is active – the preferred pitch for upper-case text is “high”. Note that it is assumed that a Boolean value for indicating character case by pitch is available in the runtime context under the key "http://terms.gpii.net/indicate-case-by-pitch".

```
<user-context>
   <option id="default">
      <name>high pitch for upper case</name>
      <preference key="http://terms.gpii.net/pitch-for-upper-case" value="high"/>
      <condition type="eq">
         <operand value="http://terms.gpii.net/indicate-case-by-pitch"/>
         <operand value="true"/>
      </condition>
   </option>
</user-context>
```

EXAMPLE 3 A user-context, consisting of a single option (with ID "default") that has no name. The option becomes active if the environmental noise level is between 40 (exclusive) and 60 (inclusive). It contains a couple of preferences key-value pairs, stating (together) that – if the option is active – the preferred setting is subtitles on and volume set to 80. Note that it is assumed that a value for the context concept for noise level is available in the runtime context under the URI "http://terms.gpii.net/noise".

```
<user-context>
   <option id="default">
      <name>noise between 40 and 60</name>
      <preference key="http://terms.gpii.net/subtitles" value="true"/>
      <preference key="http://terms.gpii.net/volume" value="80"/>
      <condition type="and">
         <operand type="ge">
            <operand value="http://terms.gpii.net/noise"/>
            <operand value="40"/>
         </operand>
         <operand type="le">
            <operand value="http://terms.gpii.net/noise"/>
            <operand value="60"/>
         </operand>
      </condition>
   </option>
</user-context>
```

EXAMPLE 4 A user-context with a single option (ID is "default") which has no name. The option becomes active if the time of the day is 19:00 h or later and the environmental noise level is greater than 20, or in any case (without other restriction) if the environmental noise level is greater than 30. It states that – in case the option is active – the preferred setting is subtitles on and the volume set to 50. Note that it is assumed that a numeric value for time-of-day (value space using the "military jargon") is available in the runtime context under the URI "http://terms.gpii.net/time-of-day", and a value for noise level is available in the runtime context under the URI "http://terms.gpii.net/noise".

```
<user-context>
   <option id="default">
      <preference key="http://terms.gpii.net/subtitles" value="true"/>
      <preference key="http://terms.gpii.net/volume" value="50"/>
      <condition type="or">
         <operand type="and">
            <operand type="ge">
               <operand value="http://terms.gpii.net/time-of-day"/>
               <operand value="1900"/>
            </operand>
            <operand type="gt">
               <operand value="http://terms.gpii.net/noise"/>
               <operand value="20"/>
            </operand>
         </operand>
         <operand type="lt">
            <operand value="http://terms.gpii.net/noise"/>
            <operand value="30"/>
         </operand>
      </condition>
   </option>
</user-context>
```

EXAMPLE 5 A user-context with two contexts (with IDs "default" and "dark"). If the luminance is greater than 200, the context "default" applies, with magnification being disabled, and colours being not inverted. If the luminance is between 0 and 200 – the context "dark" applies, with various magnification settings and colour inversion. Note that it is assumed that a numeric value for the environmental luminance is available in the runtime context under the URI "http://registry.gpii.net/common/env/visual.luminance".

```
<user-context>
   <option id="default">
      <name>Default preferences</name>
      <preference key="http://registry.gpii.net/common/magnifierEnabled" value="false"/>
      <preference
         key="http://registry.gpii.net/applications/org.chrome.cloud4chrome/invertColours"
         value="false"/>
      <condition type="gt">
         <operand value="http://registry.gpii.net/common/env/visual.luminance"/>
         <operand value="200"/>
      </condition>
   </option>
   <option id="dark">
      <name>little environmental light</name>
      <preference key="http://registry.gpii.net/common/magnifierEnabled" value="true"/>
      <preference key="http://registry.gpii.net/common/magnification" value="2"/>
      <preference key="http://registry.gpii.net/common/magnifierPosition" value="TopHalf"/>
      <preference
         key="http://registry.gpii.net/applications/org.chrome.cloud4chrome/invertColours"
         value="true"/>
      <condition type="and">
         <operand type="ge">
            <operand value="http://registry.gpii.net/common/env/visual.luminance"/>
            <operand value="0"/>
         </operand>
         <operand type="le">
            <operand value="http://registry.gpii.net/common/env/visual.luminance"/>
            <operand value="200"/>
         </operand>
      </condition>
   </option>
</user-context>
```


NOTE The following XML Schema Definition file is available for automatic validation of a user-context file: http://openurc.org/ns/uirf/user-context.schema.xsd.

A.2.2 CREATE user-context (mandatory)

For the XML mapping of the request parameters (see Table 1), the following restrictions shall apply:

— *user-context* shall be the XML element `<user-context>` within the `<request>` element.

— Refer to A.2.1 for the mapping of a *user-context* object.

EXAMPLE A CREATE user-context request.

```
POST /api/user-contexts HTTP/1.1
Content-Type: application/xml

<request>
   <user-context>
      <option id="default">
         <name>Default preferences</name>
         <preference key="http://registry.gpii.net/common/onScreenKeyboardEnabled"
            value="true"/>
         <preference key="http://registry.gpii.net/common/initDelay" value="0.120"/>
         <preference key="http://registry.gpii.net/common/cursorSpeed" value="0.850"/>
         <preference key="http://registry.gpii.net/common/cursorAcceleration"
            value="0.800"/>
         <preference key="http://registry.gpii.net/common/mouseEmulationEnabled"
            value="true"/>
         <preference key="http://registry.gpii.net/common/unknown" value="true"/>
         <preference key="http://registry.gpii.net/applications/org.alsa-project/volume"
            value="14"/>
         <preference key="http://registry.gpii.net/applications/org.alsa-project/pitch"
            value="100"/>
      </option>
   </user-context>
</request>
```

NOTE 1 No XML-specific mapping restrictions apply for a CREATE user-context response since there are no parameters of category "response body" specified in Table 2 that are subject to mapping.

NOTE 2 The following XML Schema Definition file is available for automatic validation of the request body: http://openurc.org/ns/uirf/CREATE-user-context.request.schema.xsd.

A.2.3 GET user-context (mandatory)

NOTE 1 No XML-specific mapping restrictions apply for a GET user-context request since there are no parameters of category "request body" specified in Table 3.

For the XML mapping of the parameters of category "response body" (see Table 4), the following restrictions shall apply.

— *user-context* shall be a `<user-context>` element within the `<response>` element.

— Refer to A.2.1 for the mapping of a *user-context* object.

EXAMPLE A GET user-context response.

```
HTTP/1.1 200 OK
Content-Type: application/xml

<response>
   <user-context>
      <option id="default">
         <name>Default preferences</name>
         <preference key="http://registry.gpii.net/common/onScreenKeyboardEnabled"
            value="true"/>
         <preference key="http://registry.gpii.net/common/initDelay" value="0.120"/>
         <preference key="http://registry.gpii.net/common/cursorSpeed" value="0.850"/>
         <preference key="http://registry.gpii.net/common/cursorAcceleration"
```

```
            value="0.800"/>
         <preference key="http://registry.gpii.net/common/mouseEmulationEnabled"
            value="true"/>
         <preference key="http://registry.gpii.net/common/unknown" value="true"/>
         <preference key="http://registry.gpii.net/applications/org.alsa-project/volume"
            value="14"/>
         <preference key="http://registry.gpii.net/applications/org.alsa-project/pitch"
            value="100"/>
      </option>
   </user-context>
</response>
```

NOTE 2 The following XML Schema Definition file is available for automatic validation of the response body: http://openurc.org/ns/uirf/GET-user-context.response.schema.xsd.

A.2.4 UPDATE user-context (mandatory)

For the XML mapping of the parameters of category "request body" (see Table 5), the following restrictions shall apply:

— *user-context* shall be a `<user-context>` element within the `<request>` element.

— Refer to A.2.1 for the mapping of a *user-context* object.

EXAMPLE An UPDATE user-context request. Assuming that a user-context with identifier "U12345" was created on the server as provided in A.2.2 Examples 1 and 2, the following changes are now performed on the existing user-context: The preference "http://registry.gpii.net/common/mouseEmulationEnabled" is changed from `true` to `false`, and the preference "http://registry.gpii.net/applications/org.alsa-project/pitch" is deleted.

```
PUT /user-context/U12345 HTTP/1.1
Content-Type: application/xml

<request>
   <user-context>
      <option id="default">
         <name>Default preferences</name>
         <preference key="http://registry.gpii.net/common/onScreenKeyboardEnabled"
            value="true"/>
         <preference key="http://registry.gpii.net/common/initDelay" value="0.120"/>
         <preference key="http://registry.gpii.net/common/cursorSpeed" value="0.850"/>
         <preference key="http://registry.gpii.net/common/cursorAcceleration"
            value="0.800"/>
         <preference key="http://registry.gpii.net/common/mouseEmulationEnabled"
            value="false"/>
         <preference key="http://registry.gpii.net/common/unknown" value="true"/>
         <preference key="http://registry.gpii.net/applications/org.alsa-project/volume"
            value="14"/>
      </option>
   </user-context>
</request>
```

NOTE 1 No XML-specific mapping restrictions apply for an UPDATE user-context response since there are no parameters of category "response body" specified in Table 6 that are subject to mapping.

NOTE 2 The following XML Schema Definition file is available for automatic validation of the request body: http://openurc.org/ns/uirf/UPDATE-user-context.request.schema.xsd.

A.2.5 DELETE user-context (optional)

NOTE No XML-specific mapping restrictions apply for this operation since there are no parameters of category "request body" or "response body" specified in Table 7 and Table 8 that are subject to mapping.

A.2.6 GET user-context-list (mandatory)

NOTE 1 No XML-specific mapping restrictions apply for a GET user-context-list request since there are no parameters of category "request body" specified in Table 9 that are subject to mapping.

For the XML mapping of the parameters of category "response body" (see Table 10), the following restrictions shall apply:

— *total number of user-contexts* shall be a `<totalContexts>` element of type `xs:positiveInteger` within the `<response>` element.

— *user-context-uris* shall be a `<user-context-uris>` element within the `<response>` element. `<user-context-uris>` shall contain a sequence of `<uri>` elements of type `xs:anyURI`, each specifying a URI for a resulting user-context as character data content.

EXAMPLE A GET list of user-contexts response, following up on a request with the pagination parameters `offset=1` and `limit=1` (see EXAMPLE 1 in 7.2.6). The response provides the total number (9) of matching user-contexts on the user-context service, and the URI for the second user-context from that list.

```
HTTP/1.1 200 OK
Content-Type: application/xml

<response>
   <totalContexts>9</totalContexts>
   <user-context-uris>
      <uri>https://example.com/api/user-contexts/U12345</uri>
   </user-context-uris>
</response>
```

NOTE 2 The following XML Schema Definition file is available for automatic validation of the response body: http://openurc.org/ns/uirf/GET-user-context-list.response.schema.xsd.

A.3 Task-context

A.3.1 General

For the XML mapping of a task-*context* object (see 7.4.1), the following shall apply:

— The *task-context* object is the XML element `<task-context>`.

— A *property* object is an XML element `<property>` within `<task-context>`.

— A property's *name* is the value of the attribute `name` (type `xs:string`) on `<property>`.

— A property's *value* is the value of the attribute `value` (type `xs:string`) on `<property>`.

— A property's *descriptor* object is an XML element `<descriptor>` within `<property>`.

— A descriptor's *name* is the value of the attribute `name` (type `xs:string`) on `<descriptor>`.

— A descriptor's *value* is the value of the attribute `value` (type `xs:string`) on `<descriptor>`.

EXAMPLE A task-context in XML notation.

```
<task-context>
   <property name="http://openurc.org/ns/res#generalTask" value="spreadsheet"/>
   <property name="http://openurc.org/ns/res#fileName" value="Budget2016.xlsx"/>
   <property name="http://openurc.org/ns/res#application" value="MS Excel" />
</task-context>
```

NOTE The following XML Schema Definition file is available for automatic validation of a user-context file: http://openurc.org/ns/uirf/task-context.schema.xsd.

A.3.2 CREATE task-context (mandatory)

For the XML mapping of the parameters of category "request body" (see Table 11), the following restrictions shall apply:

— *task-context* shall be a `<task-context>` element (see A.3.1) within the `<request>` element.

EXAMPLE A CREATE task-context request body in XML notation:

```
POST /api/task-contexts HTTP/1.1
Content-Type: application/xml

<request>
   <task-context>
      <property name="http://openurc.org/ns/res#generalTask" value="spreadsheet"/>
      <property name="http://openurc.org/ns/res#fileName" value="Budget2016.xlsx"/>
      <property name="http://openurc.org/ns/res#application" value="MS Excel" />
   </task-context>
</request>
```

NOTE 1 No XML-specific mapping restrictions apply for a CREATE task-context response since there are no parameters of category "response body" specified in Table 12 that are subject to mapping.

NOTE 2 The following XML Schema Definition file is available for automatic validation of the request body: http://openurc.org/ns/uirf/CREATE-task-context.request.schema.xsd.

A.3.3 GET task-context (mandatory)

NOTE 1 No XML-specific mapping restrictions apply for a GET task-context request since there are no parameters of category "request body" in Table 13.

For the XML mapping of the parameters of category "response body" (see Table 14), the following restrictions shall apply:

— *task-context* shall be a `<task-context>` element (see A.3.1) within the `<response>` element.

EXAMPLE A GET task-context response in XML notation.

```
HTTP/1.1 200 OK
Content-Type: application/xml

<response>
   <task-context>
      <property name="http://openurc.org/ns/res#generalTask" value="spreadsheet"/>
      <property name="http://openurc.org/ns/res#fileName" value="Budget2016.xlsx"/>
      <property name="http://openurc.org/ns/res#application" value="MS Excel" />
   </task-context>
</response>
```

NOTE 2 The following XML Schema Definition file is available for automatic validation of a response body: http://openurc.org/ns/uirf/GET-task-context.response.schema.xsd.

A.3.4 UPDATE task-context (mandatory)

For the XML mapping of the parameters of category "request body" (see Table 15), the following restrictions shall apply:

— *task-context* shall be a `<task-context>` element (see A.3.1) within the `<request>` element.

EXAMPLE An UPDATE task-context request in XML notation. Assuming that a task-context with ID "T12345" has been created according to the EXAMPLE in A.3.2. Now, the property "http://openurc.org/ns/res#fileName" is updated to carry the value "Budget2017.xlsx".

```
PUT /api/task-contexts/T12345 HTTP/1.1
Content-Type: application/xml

<request>
   <task-context>
      <property name="http://openurc.org/ns/res#generalTask" value="spreadsheet"/>
      <property name="http://openurc.org/ns/res#fileName" value="Budget2017.xlsx"/>
      <property name="http://openurc.org/ns/res#application" value="MS Excel" />
   </task-context>
</request>
```

NOTE 1 No XML-specific mapping restrictions apply for an UPDATE task-context response since there are no parameters of category "response body" specified in Table 16 that are subject to mapping.

NOTE 2 The following XML Schema Definition file is available for automatic validation of the request body: http://openurc.org/ns/uirf/UPDATE-task-context.request.schema.xsd.

A.3.5 DELETE task-context (optional)

NOTE No XML-specific mapping restrictions apply for this operation since there are no parameters of category "request body" or "response body" specified in Table 17 and Table 18 that are subject to mapping.

A.3.6 GET task-context-list (mandatory)

NOTE 1 No XML-specific mapping restrictions apply for a GET task-context-list request since there are no parameters of category "request body" specified in Table 19 that are subject to mapping.

For the XML mapping of the parameters of category "response body" (see Table 20), the following restrictions shall apply:

— *total number of task-contexts* shall be a `<totalContexts>` element of type `xs:positiveInteger` within the `<response>` element.

— *task-context-uris* shall be a `<task-context-uris>` element within the `<response>` element. `<task-context-uris>` shall contain a sequence of `<uri>` elements of type `xs:anyURI`, each specifying a URI for a resulting task-context as character data content.

EXAMPLE A GET list of task-contexts response, following up on a request with the pagination parameters `offset=1` and `limit=1` (see EXAMPLE 1 in 7.3.6). The response provides the total number (9) of matching task-contexts on the task-context service, and the URI for the second task-context from that list.

```
HTTP/1.1 200 OK
Content-Type: application/xml

<response>
   <totalContexts>9</totalContexts>
   <task-context-uris>
      <uri>https://example.com/api/task-contexts/T12345</uri>
   </task-context-uris>
</response>
```

NOTE 2 The following XML Schema Definition file is available for automatic validation of the response body: http://openurc.org/ns/uirf/GET-task-context-list.response.schema.xsd.

A.4 Equipment-context

A.4.1 General

For the XML mapping of an equipment-*context* object (see 7.4.1), the following shall apply:

— The *equipment-context* object is the XML element `<equipment-context>`.

— A *property* object is an XML element `<property>` within `<equipment-context>`.

— A property's *name* is the value of the attribute `name` (type `xs:string`) on `<property>`.

— A property's *value* is the value of the attribute `value` (type `xs:string`) on `<property>`.

— A property's *descriptor* object is an XML element `<descriptor>` within `<property>`.

— A descriptor's *name* is the value of the attribute `name` (type `xs:string`) on `<descriptor>`.

— A descriptor's *value* is the value of the attribute `value` (type `xs:string`) on `<descriptor>`.

EXAMPLE An equipment-context in XML notation.

```
<equipment-context>
   <property name="http://openurc.org/ns/res#friendlyName" value="My iPad">
      <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="en" />
   </property>
   <property name="http://openurc.org/ns/res#friendlyName" value="Mein iPad">
      <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="de" />
```

```
   </property>
   <property name="http://openurc.org/ns/res#devicePlatform" value="iOS" />
   <property name="http://openurc.org/ns/res#deviceType" value="iPad-3gen" />
   <property name="http://openurc.org/ns/res#resolution" value="1536x2048" />
</equipment-context>
```

A.4.2 CREATE equipment-context (mandatory)

For the XML mapping of the parameters of category "request body" (see Table 21), the following restrictions shall apply:

— *equipment-context* shall be a `<equipment-context>` element (see A.4.1) within the `<request>` element.

EXAMPLE A CREATE equipment-context request body in XML notation:

```
POST /api/equipment-contexts HTTP/1.1
Content-Type: application/xml

<request>
   <equipment-context>
      <property name="http://openurc.org/ns/res#friendlyName" value="My iPad">
         <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="en" />
      </property>
      <property name="http://openurc.org/ns/res#friendlyName" value="Mein iPad">
         <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="de" />
      </property>
      <property name="http://openurc.org/ns/res#devicePlatform" value="iOS" />
      <property name="http://openurc.org/ns/res#deviceType" value="iPad-3gen" />
      <property name="http://openurc.org/ns/res#resolution" value="1536x2048" />
   </equipment-context>
</request>
```

NOTE 1 No XML-specific mapping restrictions apply for a CREATE equipment-context response since there are no parameters of category "response body" specified in Table 22 that are subject to mapping.

NOTE 2 The following XML Schema Definition file is available for automatic validation of the request body: http://openurc.org/ns/uirf/CREATE-equipment-context.request.schema.xsd.

A.4.3 GET equipment-context (mandatory)

NOTE 1 No XML-specific mapping restrictions apply for a GET equipment-context request since there are no parameters of category "request body" in Table 23.

For the XML mapping of the parameters of category "response body" (see Table 24), the following restrictions shall apply:

— *equipment-context* shall be a `<equipment-context>` element (see A.4.1) within the `<response>` element.

EXAMPLE A GET equipment-context response in XML notation.

```
HTTP/1.1 200 OK
Content-Type: application/xml

<equipment-context>
   <property name="http://openurc.org/ns/res#friendlyName" value="My iPad">
      <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="en" />
   </property>
   <property name="http://openurc.org/ns/res#friendlyName" value="Mein iPad">
      <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="de" />
   </property>
   <property name="http://openurc.org/ns/res#devicePlatform" value="iOS" />
   <property name="http://openurc.org/ns/res#deviceType" value="iPad-3gen" />
   <property name="http://openurc.org/ns/res#resolution" value="1536x2048" />
</equipment-context>
```

NOTE 2 The following XML Schema Definition file is available for automatic validation of the response body: http://openurc.org/ns/uirf/GET-equipment-context.response.schema.xsd.

A.4.4 UPDATE equipment-context (mandatory)

For the XML mapping of the parameters of category "request body" (see Table 25), the following restrictions shall apply:

— *equipment-context* shall be a `<equipment-context>` element (see A.4.1) within the `<request>` element.

EXAMPLE An UPDATE equipment-context request in XML notation. Assuming that an equipment-context with ID "Q12345" has been created according to the EXAMPLE in A.4.2. Now, the property "http://openurc.org/ns/res#resolution" is removed, and the property "http://openurc.org/ns/res#buildYear" with value "2014" is added to the existing equipment-context.

```
PUT /api/equipment-contexts/Q12345 HTTP/1.1
Content-Type: application/xml

<request>
   <equipment-context>
      <property name="http://openurc.org/ns/res#buildYear" value="2014"/>
      <property name="http://openurc.org/ns/res#resolution" xsi:nil="true"/>
   </equipment-context>
</request>
```

NOTE 1 No XML-specific mapping restrictions apply for an UPDATE equipment-context response since there are no parameters of category "response body" specified in Table 26 that are subject to mapping.

NOTE 2 The following XML Schema Definition file is available for automatic validation of the request body: http://openurc.org/ns/uirf/UPDATE-equipment-context.request.schema.xsd.

A.4.5 DELETE equipment-context (optional)

NOTE No XML-specific mapping restrictions apply for this operation since there are no parameters of category "request body" or "response body" specified in Table 27 and Table 28 that are subject to mapping.

A.4.6 GET equipment-context-list (mandatory)

NOTE 1 No XML-specific mapping restrictions apply for a GET equipment-context-list request since there are no parameters of category "request body" specified in Table 29 that are subject to mapping.

For the XML mapping of the parameters of category "response body" (see Table 30), the following restrictions shall apply:

— *total number of equipment-contexts* shall be a `<totalContexts>` element of type `xs:positiveInteger` within the `<response>` element.

— *equipment-context-uris* shall be a `<equipment-context-uris>` element within the `<response>` element. `<equipment-context-uris>` shall contain a sequence of `<uri>` elements of type `xs:anyURI`, each specifying a URI for a resulting equipment-context as character data content.

EXAMPLE A GET list of equipment-contexts response, following up on a request with the pagination parameters `offset=1` and `limit=1` (see EXAMPLE 1 in 7.4.6). The response provides the total number (9) of matching equipment-contexts on the equipment-context service, and the URI for the second equipment-context from that list.

```
HTTP/1.1 200 OK
Content-Type: application/xml

<response>
   <totalContexts>9</totalContexts>
   <equipment-context-uris>
      <uri>https://example.com/api/equipment-contexts/Q12345</uri>
   </equipment-context-uris>
</response>
```

NOTE 2 The following XML Schema Definition file is available for automatic validation of the response body: http://openurc.org/ns/uirf/GET-equipment-context-list.response.schema.xsd.

A.5 Environment-context

A.5.1 General

For the XML mapping of an *environment-context* object (see 7.5.1), the following shall apply:

— The *environment-context* object is the XML element `<environment-context>`.

— A *property* object is an XML element `<property>` within `<environment-context>`.

— A property's *name* is the value of the attribute `name` (type `xs:string`) on `<property>`.

— A property's *value* is the value of the attribute `value` (type `xs:string`) on `<property>`.

— A property's *descriptor* object is an XML element `<descriptor>` within `<property>`.

— A descriptor's *name* is the value of the attribute `name` (type `xs:string`) on `<descriptor>`.

— A descriptor's *value* is the value of the attribute `value` (type `xs:string`) on `<descriptor>`.

EXAMPLE An environment-context in XML notation.

```
<environment-context>
   <property name="http://registry.gpii.net/common/environment/visual.luminance"
      value="5" />
   <property name="http://terms.gpii.net/time-of-day" value="1905" />
</environment-context>
```

A.5.2 CREATE environment-context (mandatory)

For the XML mapping of the parameters of category "request body" (see Table 31), the following restrictions shall apply:

— *environment-context* shall be an `<environment-context>` element (see A.5.1) within the `<request>` element.

EXAMPLE A CREATE environment-context request body in XML notation:

```
POST /api/environment-contexts HTTP/1.1
Content-Type: application/xml

<request>
   <environment-context>
      <property name="http://registry.gpii.net/common/environment/visual.luminance"
         value="5" />
      <property name="http://terms.gpii.net/time-of-day" value="1905" />
   </environment-context>
</request>
```

NOTE 1 No XML-specific mapping restrictions apply for a CREATE environment-context response since there are no parameters of category "response body" specified in Table 32 that are subject to mapping.

NOTE 2 The following XML Schema Definition file is available for automatic validation of the request body: http://openurc.org/ns/uirf/CREATE-environment-context.request.schema.xsd.

A.5.3 GET environment-context (mandatory)

NOTE 1 No XML-specific mapping restrictions apply for a GET environment-context request since there are no parameters of category "request body" in Table 33.

For the XML mapping of the parameters of category "response body" (see Table 34), the following restrictions shall apply:

— *environment-context* shall be an `<environment-context>` element (see A.5.1) within the `<response>` element.

EXAMPLE A GET environment-context response in XML notation.


```
HTTP/1.1 200 OK
Content-Type: application/xml

<environment-context>
   <property name="http://openurc.org/ns/res#friendlyName" value="My iPad">
      <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="en" />
   </property>
   <property name="http://openurc.org/ns/res#friendlyName" value="Mein iPad">
      <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="de" />
   </property>
   <property name="http://openurc.org/ns/res#environmentPlatform" value="iOS" />
   <property name="http://openurc.org/ns/res#environmentType" value="iPad-3gen" />
   <property name="http://openurc.org/ns/res#resolution" value="1536x2048" />
</environment-context>
```

NOTE 2 The following XML Schema Definition file is available for automatic validation of the response body: http://openurc.org/ns/uirf/GET-environment-context.response.schema.xsd.

A.5.4 UPDATE environment-context (mandatory)

For the XML mapping of the parameters of category "request body" (see Table 35), the following restrictions shall apply:

— *environment-context* shall be a `<environment-context>` element (see A.5.1) within the `<request>` element.

EXAMPLE An UPDATE environment-context request in XML notation. Assuming that an environment-context with ID "E12345" has been created according to EXAMPLE 1 in 7.5.2. Now, the value of "http://registry .gpii.net/common/environment/visual.luminance" is changed to "4".

```
PUT /api/environment-contexts/E12345 HTTP/1.1
Content-Type: application/xml

<request>
   <environment-context>
      <property name="http://registry.gpii.net/common/environment/visual.luminance"
         value="4"/>
      <property name="http://terms.gpii.net/time-of-day" value="1905"/>
   </environment-context>
</request>
```

NOTE 1 No XML-specific mapping restrictions apply for an UPDATE environment-context response since there are no parameters of category "response body" specified in Table 36 that are subject to mapping.

NOTE 2 The following XML Schema Definition file is available for automatic validation of the request body: http://openurc.org/ns/uirf/UPDATE-environment-context.request.schema.xsd.

A.5.5 DELETE environment-context (optional)

NOTE No XML-specific mapping restrictions apply for this operation since there are no parameters of category "request body" or "response body" specified in Table 37 and Table 38 that are subject to mapping.

A.5.6 GET environment-context-list (mandatory)

NOTE 1 No XML-specific mapping restrictions apply for a GET environment-context-list request since there are no parameters of category "request body" specified in Table 39 that are subject to mapping.

For the XML mapping of the parameters of category "response body" (see Table 40), the following restrictions shall apply:

— *total number of environment-contexts* shall be a `<totalContexts>` element of type `xs:positiveInteger` within the `<response>` element.

— *environment-context-uris* shall be a `<environment-context-uris>` element within the `<response>` element. `<environment-context-uris>` shall contain a sequence of `<uri>` elements of type `xs:anyURI`, each specifying a URI for a resulting environment-context as character data content.

EXAMPLE A GET list of environment-contexts response, following up on a request with the pagination parameters `offset=1` and `limit=1` (see EXAMPLE 1 in 7.5.6). The response provides the total number (9) of matching environment-contexts on the environment-context service, and the URI for the second environment-context from that list.

```
HTTP/1.1 200 OK
Content-Type: application/xml

<response>
   <totalContexts>9</totalContexts>
   <environment-context-uris>
      <uri>https://example.com/api/environment-contexts/E12345</uri>
   </environment-context-uris>
</response>
```

NOTE 2 The following XML Schema Definition file is available for automatic validation of the response body: http://openurc.org/ns/uirf/GET-environment-context-list.response.schema.xsd.

A.6 Resource

A.6.1 General

NOTE There is no XML mapping for a user interface resource since it can have any MIME type.

A.6.2 CREATE resource (mandatory)

NOTE 1 No XML-specific mapping restrictions apply for a CREATE resource operation since the format of the request body (see Table 41) is solely dependent on the MIME type of the resource (as given by the HTTP header field `Content-Type`).

NOTE 2 No XML-specific mapping restrictions apply for a CREATE resource response since there are no parameters of category "response body" specified in Table 42 that are subject to mapping.

A.6.3 GET resource-by-ID (mandatory)

NOTE No XML-specific mapping restrictions apply for this operation since there are no parameters of category "request body" or "response body" specified in Table 43 and Table 44 that are subject to mapping.

A.6.4 GET resource-from-listing (mandatory)

NOTE No XML-specific mapping restrictions apply for this operation since there are no parameters of category "request body" or "response body" specified in Table 45 and Table 46 that are subject to mapping.

A.6.5 UPDATE resource (optional)

NOTE 1 No XML-specific mapping restrictions apply for an UPDATE resource operation since the format of the request body (see Table 47) is solely dependent on the MIME type of the resource (as given by the HTTP header field `Content-Type`).

NOTE 2 No XML-specific mapping restrictions apply for an UPDATE resource response since there are no parameters of category "response body" specified in Table 48 that are subject to mapping.

A.6.6 DELETE resource (optional)

NOTE No XML-specific mapping restrictions apply for this operation since there are no parameters of category "request body" or "response body" specified in Table 49 and Table 50 that are subject to mapping.

A.7 Resource-description

A.7.1 General

For the XML mapping of a *resource-description* object (see 7.7.1), the following shall apply:

— The *resource-description* object is the XML element `<resource-description>`.

— A *property* object is an XML element `<property>` within `<resource-description>`.

— A property's *name* is the value of the attribute `name` (type `xs:string`) on `<property>`.

— A property's *value* is the value of the attribute `value` (type `xs:string`) on `<property>`.

— A property's *descriptor* object is an XML element `<descriptor>` within `<property>`.

— A descriptor's *name* is the value of the attribute `name` (type `xs:string`) on `<descriptor>`.

— A descriptor's *value* is the value of the attribute `value` (type `xs:string`) on `<descriptor>`.

EXAMPLE A resource-description in XML notation.

```
<resource-description>
   <property name="resource-uri" value="https://res.openurc.org/api/resources/R12345"/>
   <property name="http://purl.org/dc/elements/1.1/format" value="video/mp4"/>
   <property name="http://purl.org/dc/elements/1.1/title" value="Example video">
      <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="en"/>
   </property>
   <property name="http://purl.org/dc/elements/1.1/title" value="Beispielvideo">
      <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="de"/>
   </property>
   <property name="http://openurc.org/ns/res#resolution" value="high"/>
   <property name="http://openurc.org/ns/res#includesAudio" value="true"/>
   <property name="http://www.imsglobal.org/accessibility/accessMode" value="visual"/>
   <property name="http://www.imsglobal.org/accessibility/accessMode" value="auditory"/>
   <property name="http://www.imsglobal.org/accessibility/adaptationType"
      value="captions"/>
   <property name="http://schema.org/Book/accessibilityHazard"
      value="noFlashingHazard"/>
   <property name="http://purl.org/dc/terms/modified" value="2017-02-08"/>
</resource-description>
```

NOTE This example shows the use of various namespaces for the URIs used as property names, as a means for describing a wide range of aspects of the resource. "http://purl.org/dc/elements/1.1/" refers to the Dublin Core Metadata Element Set 1.1; "http://www.w3.org/XML/1998/namespace/" refers to the XML namespace; "http://openurc.org/ns/res#" refers to the resource description vocabulary of the OpenURC Alliance; "http://www.imsglobal.org/accessibility" refers to the IMS Global AfA vocabulary; "http://schema.org" refers to the Schema.org vocabulary; and "http://purl.org/dc/terms/" refers to the Dublin Core Metadata Terms.

A.7.2 CREATE resource-description (mandatory)

For the XML mapping of the parameters of category "request body" (see Table 51), the following restrictions shall apply:

— *resource-description* shall be a `<resource-description>` element (see A.7.1) within the `<request>` element.

EXAMPLE A CREATE resource-description request in XML notation. Note that the property named "resource-uri" is mandatory, but does not need to be the first in the list.

```
POST /api/resource-descriptions HTTP/1.1
Content-Type: application/xml

<request>
   <resource-description>
     <property name="resource-uri" value="https://res.openurc.org/api/resources/R12345"/>
     <property name="http://purl.org/dc/elements/1.1/format" value="video/mp4"/>
     <property name="http://purl.org/dc/elements/1.1/title" value="Example video">
```

```
      <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="en"/>
    </property>
    <property name="http://purl.org/dc/elements/1.1/title" value="Beispielvideo">
      <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="de"/>
    </property>
    <property name="http://openurc.org/ns/res#resolution" value="high"/>
    <property name="http://openurc.org/ns/res#includesAudio" value="true"/>
    <property name="http://www.imsglobal.org/accessibility/accessMode" value="visual"/>
    <property name="http://www.imsglobal.org/accessibility/accessMode" value="auditory"/>
    <property name="http://www.imsglobal.org/accessibility/adaptationType"
      value="captions"/>
    <property name="http://schema.org/Book/accessibilityHazard"
      value="noFlashingHazard"/>
    <property name="http://purl.org/dc/terms/modified" value="2017-02-08"/>
  </resource-description>
</request>
```

NOTE 1 This example shows the use of various namespaces for the URIs used as property names, as a means for describing a wide range of aspects of the resource. For an explanation of the namespaces, see A.7.1, Note 1.

NOTE 2 No XML-specific mapping restrictions apply for a CREATE resource-description response since there are no parameters of category "response body" specified in Table 52 that are subject to mapping.

NOTE 3 The following XML Schema Definition file is available for automatic validation of the request body: http://openurc.org/ns/uirf/CREATE-resource-description.request.schema.xsd.

A.7.3 GET resource-description-by-ID (mandatory)

NOTE 1 No XML-specific mapping restrictions apply for a GET resource-description request since there are no parameters of category "request body" specified in Table 53.

For the XML mapping of the parameters of category "response body" (see Table 54), the following restrictions shall apply:

— *resource-description* shall be a `<resource-description>` element (see A.7.1) within the `<response>` element.

EXAMPLE A GET resource-description-by-ID response in XML notation. Note that the first item in the list named "resource-uri" is mandatory, but does not need to be the first item.

```
HTTP/1.1 200 OK
Content-Type: application/xml

<response>
  <resource-description>
    <property name="resource-uri" value="https://res.openurc.org/api/resources/R12345"/>
    <property name="http://purl.org/dc/elements/1.1/format" value="video/mp4"/>
    <property name="http://purl.org/dc/elements/1.1/title" value="Example video">
      <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="en"/>
    </property>
    <property name="http://purl.org/dc/elements/1.1/title" value="Beispielvideo">
      <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="de"/>
    </property>
    <property name="http://openurc.org/ns/res#resolution" value="high"/>
    <property name="http://openurc.org/ns/res#includesAudio" value="true"/>
    <property name="http://www.imsglobal.org/accessibility/accessMode" value="visual"/>
    <property name="http://www.imsglobal.org/accessibility/accessMode" value="auditory"/>
    <property name="http://www.imsglobal.org/accessibility/adaptationType"
      value="captions"/>
    <property name="http://schema.org/Book/accessibilityHazard"
      value="noFlashingHazard"/>
    <property name="http://purl.org/dc/terms/modified" value="2016-07-28"/>
  </resource-description>
</response>
```

NOTE 2 This example shows the use of various namespaces for the URIs used as property names, as a means for describing a wide range of aspects of the resource. For an explanation of the namespaces, see A.7.1, Note 1.

A.7.4 GET resource-description-from-listing (mandatory)

NOTE 1 No XML-specific mapping restrictions apply for a GET resource-description request since there are no parameters of category "request body" specified in Table 55.

For the XML mapping of the parameters of category "response body" (see Table 56), the following restrictions shall apply:

— *resource-description* shall be a `<resource-description>` element (see A.7.1) within the `<response>` element.

EXAMPLE A GET resource-description-from-listing response in XML notation. Note that the first item in the list named "resource-uri" is mandatory, but does not need to be the first item.

```
HTTP/1.1 200 OK
Content-Type: application/xml

<response>
   <resource-description>
     <property name="resource-uri" value="https://res.openurc.org/api/resources/R12345"/>
     <property name="http://purl.org/dc/elements/1.1/format" value="video/mp4"/>
     <property name="http://purl.org/dc/elements/1.1/title" value="Example video">
        <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="en"/>
     </property>
     <property name="http://purl.org/dc/elements/1.1/title" value="Beispielvideo">
        <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="de"/>
     </property>
     <property name="http://openurc.org/ns/res#resolution" value="high"/>
     <property name="http://openurc.org/ns/res#includesAudio" value="true"/>
     <property name="http://www.imsglobal.org/accessibility/accessMode" value="visual"/>
     <property name="http://www.imsglobal.org/accessibility/accessMode" value="auditory"/>
     <property name="http://www.imsglobal.org/accessibility/adaptationType"
        value="captions"/>
     <property name="http://schema.org/Book/accessibilityHazard"
        value="noFlashingHazard"/>
     <property name="http://purl.org/dc/terms/modified" value="2016-07-28"/>
   </resource-description>
</response>
```

NOTE 2 This example shows the use of various namespaces for the URIs used as property names, as a means for describing a wide range of aspects of the resource. For an explanation of the namespaces, see the NOTE in A.7.1.

A.7.5 UPDATE resource-description (mandatory)

For the XML mapping of the parameters of category "request body" (see Table 57), the following restrictions shall apply:

— *resource-description* shall be a `<resource-description>` element (see A.7.1) within the `<request>` element.

EXAMPLE An UPDATE resource-description request in XML notation. Assuming that a resource-description with ID "RD12345" has been created according to A.7.2, Examples 1 and 2, now the value of the property named "http://purl.org/dc/elements/1.1/title" with descriptor named "http://www.w3.org/XML/1998/namespace/lang" and value "en" is changed to "Sample video", the property named "http://purl.org/dc/elements/1.1/title" with descriptor named "http://www.w3.org/XML/1998/namespace/lang" and value "de" is removed, the property named "http://openurc.org/ns/res#resolution" is removed, a property named "http://schema.org/Book/accessibilityFeature" with value "audioDescription" is added, and the value of the property named "http://purl.org/dc/terms/modified" is changed to "2017-02-09".

```
PUT /api/resource-descriptions/RD12345 HTTP/1.1
Content-Type: application/xml

<request>
   <resource-description>
     <property name="resource-uri" value="https://res.openurc.org/api/resources/R12345"/>
     <property name="http://purl.org/dc/elements/1.1/format" value="video/mp4"/>
     <property name="http://purl.org/dc/elements/1.1/title" value="Sample video">
        <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="en"/>
```

```
      </property>
      <property name="http://openurc.org/ns/res#includesAudio" value="true"/>
      <property name="http://www.imsglobal.org/accessibility/accessMode" value="visual"/>
      <property name="http://www.imsglobal.org/accessibility/accessMode" value="auditory"/>
      <property name="http://www.imsglobal.org/accessibility/adaptationType"
         value="captions"/>
      <property name="http://schema.org/Book/accessibilityHazard"
         value="noFlashingHazard"/>
      <property name="http://schema.org/Book/accessibilityFeature"
         value="audioDescription"/>
      <property name="http://purl.org/dc/terms/modified" value="2017-02-09"/>
   </resource-description>
</request>
```

NOTE 1 No XML-specific mapping restrictions apply for an UPDATE resource-description response since there are no parameters of category "response body" specified in Table 58 that are subject to mapping.

NOTE 2 The following XML Schema Definition file is available for automatic validation of the request body: http://openurc.org/ns/uirf/UPDATE-resource-description.request.schema.xsd.

A.7.6 DELETE resource-description (optional)

NOTE No XML-specific mapping restrictions apply for this operation since there are no parameters of category "request body" or "response body" specified in Table 59 and Table 60 that are subject to mapping.

A.8 Listing

A.8.1 General

NOTE This subsection is empty on purpose, to match the structure of 7.8.

A.8.2 CREATE listing (mandatory)

For the XML mapping of the parameters of category "request body" (see Table 61), the following restrictions shall apply:

— *user-context-uris* shall be coded as a sequence of `<user-context-uri>` elements with character data content (type `xs:anyURI`), within the `<request>` element.

— *task-context-uris* shall be coded as a sequence of `<task-context-uri>` elements with character data content (type `xs:anyURI`), within the `<request>` element.

— *equipment-context-uris* shall be coded as a sequence of `<equipment-context-uri>` elements with character data content (type `xs:anyURI`), within the `<request>` element.

— *environment-context-uris* shall be coded as a sequence of `<environment-context-uri>` elements with character data content (type `xs:anyURI`), within the `<request>` element.

— *resource-description-query* shall be a `<resource-description>` element (see A.7.1) within the `<request>` element.

EXAMPLE A CREATE listing request in XML notation.

```
POST /api/listings HTTP/1.1
Content-Type: application/xml

<request>
   <user-context-uri>http://res.openurc.org/api/user-contexts/U12345</user-context-uri>
   <task-context-uri>http://res.openurc.org/api/task-contexts/T12345</task-context-uri>
   <task-context-uri>http://res.openurc.org/api/task-contexts/T54321</task-context-uri>
   <equipment-context-uri>http://res.openurc.org/api/equipment-contexts/Q12345
   </equipment-context-uri>
   <environment-context-uri>http://res.openurc.org/api/environment-contexts/E12345
   </environment-context-uri>
   <resource-description>
      <property name="http://purl.org/dc/elements/1.1/title" value="Beginner's tutorial">
```



```
        <descriptor name="http://www.w3.org/XML/1998/namespace/lang" value="en"/>
      </property>
      <property name="http://openurc.org/ns/res#mimeType" value="video/mp4"/>
   </resource-description>
</request>
```

NOTE 1 This example shows the use of various namespaces for the URIs used as property names. "http://purl.org/dc/elements/1.1/" refers to the Dublin Core Metadata Element Set 1.1; "http://www.w3.org/XML/1998/namespace/" refers to the XML namespace; and "http://openurc.org/ns/res#" refers to the resource description vocabulary of the OpenURC Alliance.

NOTE 2 No XML-specific mapping restrictions apply for a CREATE listing response since there are no parameters of category "response body" specified in Table 62 that are subject to mapping.

NOTE 3 The following XML Schema Definition file is available for automatic validation of the request body: http://openurc.org/ns/uirf/CREATE-listing.request.schema.xsd.

A.8.3 GET listing (mandatory)

NOTE 1 No XML-specific mapping restrictions apply for a GET listing request since there are no parameters of category "request body" specified in Table 63.

For the XML mapping of the parameters of category "response body" (see Table 64), the following restrictions shall apply:

— *start* shall be the value of attribute `start` (type `xs:int`) on element `<response>`.

— *count* shall be the value of attribute `count` (type `xs:int`) on element `<response>`.

— *resource-description-uris* shall be coded as a sequence of `<resource-description-uri>` elements with character data content (type `xs:anyURI`), within the `<response>` element.

EXAMPLE GET listing response in XML notation.

```
HTTP/1.1 200 OK
Content-Type: application/xml

<response start="0" count="1">
   <resource-description-uri> https://example.com/api/resource-descriptions/RD12345
   </resource-description-uri>
</response>
```

NOTE 2 The following XML Schema Definition file is available for automatic validation of the response body: http://openurc.org/ns/uirf/GET-listing.response.schema.xsd.

A.8.4 DELETE listing (optional)

NOTE No XML-specific mapping restrictions apply for this operation since there are no parameters of category "request body" or "response body" specified in Table 65 and Table 66 that are subject to mapping.

A.8.5 CONFIRM user-rating (optional)

For the XML mapping of the parameters of category "request body" (see Table 67), the following restrictions shall apply:

— *user-context-uris* shall be coded as a sequence of `<user-context-uri>` elements with character data content (type `xs:anyURI`), within the `<request>` element.

— *task-context-uris* shall be coded as a sequence of `<task-context-uri>` elements with character data content (type `xs:anyURI`), within the `<request>` element.

— *equipment-context-uris* shall be coded as a sequence of `<equipment-context-uri>` elements with character data content (type `xs:anyURI`), within the `<request>` element.

— *environment-context-uris* shall be coded as a sequence of `<environment-context-uri>` elements with character data content (type `xs:anyURI`), within the `<request>` element.

— *resource-description-uri* shall be a `<resource-description-uri>` element with character data content (type `xs:anyURI`), within the `<request>` element.

— *user-rating* shall be a `<user-rating>` element with character data content (type `xs:float`) within the `<request>` element.

EXAMPLE A CONFIRM user-rating request in XML notation. The value 1.0 for user-rating means that the user has the highest preference for using the resource-description with ID "RD12345" under the given contexts.

```
POST /api/user-rating HTTP/1.1
Content-Type: application/xml

<request>
   <user-context-uri>http://res.openurc.org/api/user-contexts/U12345</user-context-uri>
   <task-context-uri>http://res.openurc.org/api/task-contexts/T12345</task-context-uri>
   <task-context-uri>http://res.openurc.org/api/task-contexts/T54321</task-context-uri>
   <equipment-context-uri>http://res.openurc.org/api/equipment-contexts/Q12345
   </equipment-context-uri>
   <environment-context-uri>http://res.openurc.org/api/environment-contexts/E12345
   </environment-context-uri>
   <resource-description-uri>http://example.com/api/resource-descriptions/RD12345
   </resource-description-uri>
   <user-rating>1.0</user-rating>
</request>
```

NOTE 1 No XML-specific mapping restrictions apply for a CONFIRM user-rating response since there are no parameters of category "response body" specified in Table 68 that are subject to mapping.

NOTE 2 The following XML Schema Definition file is available for automatic validation of the request body: http://openurc.org/ns/uirf/CONFIRM-user-rating.request.schema.xsd.

Annex B
(normative)

Mapping for JSON

B.1 General

For mapping the parameters of the operations in Clause 7 to JSON, the requirements in this annex shall be met.

NOTE In this annex, only parameters of category "request body" or "response body" are listed. Parameters of categories "URI path", "URI query parameter" and "HTTP header field" are not subject to JSON mapping, and are to be coded as specified in Clause 7.

The message body (if applicable) and response body (if applicable) shall comply to JSON (in accordance with IETF RFC 7159).

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 (in accordance with 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).

Where types are specified for parameters, these shall refer to ECMA-404.

For any mapping defined in this clause, proprietary information may be added as additional members of an object, even within a nested object, as long as the additional members do not interfere with the information items specified in this clause.

B.2 User-context

B.2.1 General

For the JSON mapping of the *user-context* object, the following restrictions shall apply:

— The *user-context* shall be a JSON object.

— The *option* objects may occur as values of members (named after their IDs) of the *user-context* object.

— *name* may occur as the value (string) of the member `name` in the *option* object.

— *preferences* shall be a JSON object as the value of the member `preferences` in the *option* object, and shall be a JSON object with *keys* as member names and *values* as their values.

— *conditions* shall be the value of the member `conditions` in the *option* object, and shall be a JSON array with unnamed JSON objects (*condition* as elements).

— *type* shall be the value of the member `type` of a *condition* object, bearing the string representation of the operator.

— *operands* shall be the value of the member `operands` of a *condition* object. It shall be a JSON array with unnamed objects (*operand* as elements).

— *operand* shall be either one of the following:

  — A StringLiteral in the form of a URI – interpreted as *key* name.

  — A BooleanLiteral, NumericLiteral or StringLiteral, as specified in ECMA-262, section 12.2.4.

  — An unnamed JSON object with the members `type` and `operands`.

EXAMPLE 1 A user-context, consisting of a single option (with ID "default") with no condition (i.e. it is always active). The option has no name. It states that speech output is always to be on.

```
{
  "default": {
    "preferences":
      { "http://terms.gpii.net/speech-output": true }
  }
}
```

EXAMPLE 2 A user-context with a single option (with ID "default"), named "high pitch for upper case". This option is active if an application of category "IDE" is used, i.e. an Integrated Development Environment for programming and debugging. The user-context states that – in case the option is active – the preferred pitch for upper-case text is “high”. Note that it is assumed that a String value for indicating the application category is available in the runtime context under the key "http://terms.gpii.net/applicationCategory". Note also that if the option's condition is false – i.e. if the application category is not "IDE" or is not available – no preferences are available in the user-context.

```
{
  "default": {
    "name": "high pitch for upper case",
    "preferences":
      { "http://terms.gpii.net/pitch-for-upper-case": "high" },
    "conditions": [
      { "type": "eq",
        "operands": ["http://terms.gpii.net/applicationCategory", "IDE"]
      }
    ]
  }
}
```

EXAMPLE 3 A user-context, consisting of a single option (with ID "default") that has no name. The option becomes active if the environmental noise level is between 40 (exclusive) and 60 (inclusive). It contains a couple of preferences key-value pairs, stating (together) that – if the option is active – the preferred setting is subtitles on and volume set to 80. Note that it is assumed that a value for the context concept for noise level is available in the runtime context under the URI "http://terms.gpii.net/noise".

```
{
   "default": {
      "name": "noise between 40 and 60",
      "preferences": {
         "http://terms.gpii.net/subtitles": true,
         "http://terms.gpii.net/volume": 80
      },
      "conditions": [
         {
            "type": "and",
            "operands": [
               {
                  "type": "ge",
                  "operands": [
                     "http://terms.gpii.net/noise", 40
                  ]
               },
               {
                  "type": "le",
                  "operands": [
                     "http://terms.gpii.net/noise", 60
                  ]
               }
            ]
         }
      ]
```



```
    }
}
```

EXAMPLE 4 A user-context with a single option (ID is "default") which has no name. The option becomes active if the time of the day is 19:00 h or later and the environmental noise level is greater than 20, or in any case (without other restriction) if the environmental noise level is greater than 30. It states that – in case the option is active – the preferred setting is subtitles on and the volume set to 50. Note that it is assumed that a numeric value for time-of-day (value space using the "military jargon") is available in the runtime context under the URI "http://terms.gpii.net/time-of-day", and a value for noise level is available in the runtime context under the URI "http://terms.gpii.net/noise".

```
{
  "default": {
    "preferences": {
      "http://terms.gpii.net/subtitles": true,
      "http://terms.gpii.net/volume": 50
    },
    "conditions": [
      { "type": "or",
        "operands": [
          { "type": "and",
            "operands": [
              { "type": "ge",
                "operands": ["http://terms.gpii.net/time-of-day", 1900]
              },
              { "type": "gt",
                "operands": ["http://terms.gpii.net/noise", 20]
              }
            ]
          },
          { "type": "lt",
            "operands": ["http://terms.gpii.net/noise", 30]
          }
        ]
      }
    ]
  }
}
```

EXAMPLE 5 A user-context with two contexts (with IDs "default" and "dark"). If the luminance is greater than 200, the context "default" applies, with magnification being disabled, and colours being not inverted. If the luminance is between 0 and 200 – the context "dark" applies, with various magnification settings and colour inversion. Note that it is assumed that a numeric value for the environmental luminance is available in the runtime context under the URI "http://registry.gpii.net/common/env/visual.luminance".

```
{
  "default": {
    "name": "Default preferences",
    "preferences": {
      "http://registry.gpii.net/common/magnifierEnabled": false,
      "http://registry.gpii.net/applications/org.chrome.cloud4chrome/invertColours": false
    },
    "conditions": [
      { "type": "gt",
        "operands": ["http://registry.gpii.net/common/env/visual.luminance", 200]
      },
      {
        "type": "ap",
        "operands": ["http://registry.gpii.net/common/env/auditory.volume", 10]
      }
    ]
  },
  "dark": {
    "name": "little environmental light",
    "preferences": {
      "http://registry.gpii.net/common/magnifierEnabled": true,
      "http://registry.gpii.net/common/magnification": 2,
      "http://registry.gpii.net/common/magnifierPosition": "TopHalf",
      "http://registry.gpii.net/applications/org.chrome.cloud4chrome/invertColours": true
    },
    "conditions": [
      { "type": "and",
```