DAIA 0.5

his document specifies the outdated specification of DAIA 0.5. The current specification (DAIA 1.0) is available at <http://purl.org/NET/DAIA>

Authors: Jakob Voss (VZG) <jakob.voss@gbv.de>, Uwe Reh (Hebis) <reh@hebis.uni-frankfurt.de>

Abstract

The Document Availability Information API (DAIA) defines a data model with serializations in JSON and XML to encode information about the current availability of documents. This document defines the serialization formats DAIA/JSON and DAIA/XML and a HTTP query API to query DAIA information.

1. Introduction

Core components of the DAIA data model

The Document Availability Information API (DAIA) defines a response format that encodes information about the current availability of documents. The general structure of a DAIA response is a tree of nested elements. A description of the data model is outlined at one page: http://ws.gbv.de/daia/daiamodel.pdf

DAIA format structures can be encoded either in JSON (DAIA/JSON) or in XML (DAIA/XML). The current XML Schema is located at http://purl.org/NET/DAIA/schema.xsd. The XML namespace and URI for DAIA/XML is http://ws.gbv.de/daia/. The version attribute contains the current version of the schema (0.5).

2. Structure and Encoding

In the following paragraphs we want to give a short introduction to DAIA format. Examples of equivalent DAIA document fragments are given in DAIA/JSON and DAIA/XML. The basic information entities of DAIA format are:

- daia (root element)
- document
- item
- available and unavailable
- messages
- additional entities (institution, department, storage, and limitation)

Daia entities in DAIA/JSON are encoded as simple nodes with child nodes, daia entities in DAIA/XML are encoded as XML-elements with attributes and child-elements. XML elements include namespaces so you must use an XML parser with support of namespaces to process DAIA/XML. Below the possible and mandatory attributes and child elements or nodes of each daia entities are defined. If an entity is marked with a question mark (?) it is optional. If an entity is marked with a star (*) it is repeatable and optional. All other entities are mandatory and non-repeatable. Repeatable elements in DAIA/XML are just line up after another. Repeatable elements in DAIA/JSON must be encoded as array with one ore more content elements.

	DAIA/JSON	DAIA/XML
repeated	`"item": [ { ... }, { ... }, ... ]`	`<item ... > ... </item> <item ... > ... </item> ...`
not repeated	`"item": [ { ... } ]`	`<item ... > ... </item>`

Content of entities that must not have child nodes are encoded as Unicode strings, numbers, or boolean values. Unless a more specific limitation is defined with an XML Schema Datatype, the content must be an Unicode string (but it may be the empty string). DAIA uses the following XML Schema Datatypes:

- xs:boolean - in DAIA/XML one of true, false, 1, 0. In DAIA/JSON one of true, false (but literally instead of string).
- xs:language - must conform to the pattern [a-zA-Z]{1,8}(-[a-zA-Z0-9]{1,8})*.
- xs:date - must follow the form \d{4}[-](\d\d)[-](\d\d)((([+-])\d\d:\d\d).
- xs:dateTime - must follow the form \d{4}[-](\d\d)[-](\d\d)[T](\d\d)[:](\d\d)[:]\d\d([.]\d+)?((([+-])\d\d:\d\d).
- xs:duration - must follow the form -PnYnMnDTnHnMnS. Empty parts can be omitted.
- xs:integer - must conform to the pattern [+-]?[0-9]+ in DAIA/XML. In DAIA/JSON it is an integer.
- xs:nonNegativeInteger - must conform to the pattern ([+]?[0-9]+ in DAIA/XML. In DAIA/JSON it is a non-negative integer.
- xs:anyURI - must conform to the pattern of an URI.

For examples in DAIA/RDF the following namespace prefixes are used:

- `@prefix daia: <http://purl.org/ontology/daia/> .`
- `@prefix daiaservice: <http://purl.org/ontology/daia/Service/> .`
- `@prefix dcterms: <http://purl.org/dc/terms/> .`
- `@prefix bibo: <http://purl.org/ontology/bibo/> .`
- `@prefix foaf: <http://xmlns.com/foaf/0.1/> .`

2.1. Root element

Each full DAIA document contains exactly one root element. In DAIA/XML the root element name is **daia**, in DAIA/JSON the root element is just an unnamed object. The attributes and child elements are as follows:

- **version** (attribute) - the daia version number (currently 0.5)
- **timestamp** (attribute) - the time the document was generated. Type xs:dateTime.
- **message*** (element) - (error) message(s) about the whole response
- **institution**? (element) - information about the institution that grants or knows about services and their availability
- **document*** (element) - a group of items that can be refered to with one identifier. Please note that although the number of document elements can be zero or greater one, one single document entry should be considered as the default.

In DAIA/XML you must further specifiy the XML namespace http://ws.gbv.de/daia/ and may refer to the DAIA XML Schema http://ws.gbv.de/daia/daia.xsd as shown in the following example. In DAIA/JSON you can include a fixed child element that points to the DAIA specification and namespace:

- **schema** (attribute) - DAIA namespace string http://ws.gbv.de/daia/

Example:

DAIA/JSON	DAIA/XML
`{ "version" : "0.5", "schema" : "http://ws.gbv.de/daia/", "timestamp" : "2009-06-09T15:39:52.831+02:00", "institution" : { } }`	`<daia xmlns="http://ws.gbv.de/daia/" version="0.5" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://ws.gbv.de/daia/ http://ws.gbv.de/daia/daia.xsd" timestamp="2009-06-09T15:39:52.831+02:00"> <institution/> </daia>`

equivalent DAIA/XML with different namespace prefix	<d:daia xmlns:d="http://ws.gbv.de/daia/" version="0.5" xmlns:s="http://www.w3.org/2001/XMLSchema-instance" s:schemaLocation="http://ws.gbv.de/daia/ http://ws.gbv.de/daia/daia.xsd" timestamp="2009-06-09T15:39:52.831+02:00"> <d:institution/> </d:daia>

2.2. Document element

The document element describes a single document. Nevertheless, several *instances* of a documents (e.g. copies of a book) can exist. For the instances, have a look at the item element below.

- **id** (attribute) - each document needs an unique id to query it (e.g. ISBN, ppn, etc.). Please consider that ids have to be URIs. Type xs:anyURI.
- **href**? (attribute) - a link to the document or to additional information. Type xs:anyURI.
- **message*** (element) - (error) message(s) about the document.
- **item*** (element) - an instance or copy of the queried document (correspondends to the FRBR class of same name).

In DAIA/XML messages and items can be mixed in any order.

Example:

DAIA/JSON

```
"document" : [ {
  "href" : "https://kataloge.uni-hamburg.de/DB=1/PPNSET?PPN=57793371X",
  "id" : "gvk:ppn:57793371X",
  "item" : [ {  }, {  }, {  } ]
} ]
```

DAIA/XML

```
<document href="https://kataloge.uni-hamburg.de/DB=1/PPNSET?PPN=57793371X"
          id="gvk:ppn:57793371X">
  <item/>
  <item/>
  <item/>
</document>
```

DAIA/RDF (Turtle syntax)

```
<gvk:ppn:57793371X> a bibo:Document ;
  foaf:primaryTopicOf <https://kataloge.uni-hamburg.de/DB=1/PPNSET?PPN=57793371X> ;
  daia:exemplar [ ], [ ], [ ] .
```

2.3. Item element

The item node references a single instance (copy, URI, etc.) of a document. The availability information is of course connected to the item nodes.

- **id**? (attribute) - again, each item (instance) may have an unique ID (e.g., an individual call number for a book). Please consider that ids have to be URIs. Type xs:anyURI.
- **href**? (attribute) - a link to the item or to additional information. Type xs:anyURI.
- **part**? (attribute) - indicate that the item only contains a part of the document (`part="narrower"`) or contains more than the document (`part="broader"`)
- **message*** (element) - (error) message(s) about the item.
- **label**? (element) - a label that helps to identify and/or find the item (call number etc.)
- **department**? (element) - an administrative sub-entitity of the institution that is responsible for this item
- **storage**? (element) - a physical location of the item (stacks, floor etc.)
- **available*** (element) - information about an available service with the item.
- **unavailable*** (element) - information about an unavailable service with the item

Multiple service status can be given for an item represented by different available/unavailable elements.

Example:

DAIA/JSON	DAIA/XML	DAIA/RDF (Turtle syntax)

```
"item" : [ {
  "id" : "id:123",
  "message" : [ {
"lang": "en", "con
tent": "foo" } ],
  "department" : {
"id": "id:abc" },
  "label" : "bar",
  "available" : [ {
"service" : "prese
ntation"},
                {
"service" : "loan"
},
                {
"service" : "inter
loan"} ],
  "unavailable" : [
 {"service" : "ope
naccess"} ]
} ]
```

```
<item id="id:
123">
   <message la
ng="en">foo<
/message>
   <department
id="id:abc" />
   <label>bar<
label>
   <available
service="prese
ntation" />
   <available
service="loan"
/>
   <available
service="inter
loan" />
   <unavailable
 service="open
access" />
</item>
```

```
<id:123> a frbr:Item ;
  dct:description "foo"@en ;
  daia:label "bar" ;
  daia:heldBy <id:abc> ;
  daia:availableFor [ a daia:Service/Presentation ] ;
  daia:availableFor [ a daia:Service/Loan ] ;
  daia:availableFor [ a daia:Service/Interloan ] ;
  daia:unavailableFor [ a daia:Service/Openaccess ] ;
<id:abc> a foaf:Organization ; dcterms:isPartOf [
  a foaf:Organization ; dcterms:hasPart <id:abc> ] .
```

In DAIA/RDF, an Item element corresponds to an instance of frbr:Item. Partial items refer to items which contain less (`narrower`) or more (`broader`) than the whole document:

narrower in DAIA/XML	`<document id="x:123">`
	`  <item id="x:ABC" part="narrower"/>`
	`</document>`

narrower in DAIA/RDF	`<x:123> a bibo:Document ; daia:narrowerExemplar <x:ABC> .` `<x:123> a bibo:Document ; dcterms:hasPart [ daia:exemplar <x:ABC> ] } .`

broader in DAIA/XML	`<document id="x:123">`
	`  <item id="x:ABC" part="broader"/>`
	`</document>`

broader in DAIA/RDF	`<x:123> a bibo:Document ; daia:broaderExemplar <x:ABC> .` `<x:123> a bibo:Document ; daia:exemplar [ dcterms:hasPart <x:ABC> ] } .`

2.4. Available element

The structure of an available element is:

- **service**? (attribute) - the specific service. The value should be one of presentation, loan, openaccess, interloan or an URI. Multiple services are represented by multiple available/unavailable elements. Type enumeration or xs:anyURI.
- **href**? (attribute) - a link to perform, register or reserve the service. Type xs:anyURI.
- **delay**? (attribute) - a time period of estimated delay. Use unknown or an ISO time period. If missing, then there is probably no significant delay. Type xs:duration or the string unknown.
- **message*** (element) - (error) message(s) about the specific availability status of the item.
- **limitation*** (element) - more specific limitations of the availability status.

In DAIA/XML messages and limitations can be mixed in any order.

Services defined in DAIA

Service	URI*	Definition
presentation	http://purl.org/ontology/daia/Service/Presentation	The item is accessible within the institution (in their rooms, in their intranet etc.)
loan	http://purl.org/ontology/daia/Service/Loan	The item is accessible outside of the institution (by lending or online access) for a limited time
openaccess	http://purl.org/ontology/daia/Service/Openaccess	The item is accessible freely without any restrictions by the institution (Open Access or free copies)
interloan	http://purl.org/ontology/daia/Service/Interloan	The item is accessible mediated by another institution.
*unspecified**	http://purl.org/ontology/daia/Service	The item is accessible for an unspecified purpose by an unspecified way

It is recommended to mainly support the four specified services in non-URI form instead of using custom service URIs. If you define a new service URI you should document it as detailed as possible and discuss with other DAIA implementers to ensure interoperability. In terms of RDF all custom services must be subclasses of http://purl.org/ontology/daia/Service so a client can treat an unknown service as instance of an unspecified DAIA service. If you omit the service element then the unspecified service must be assumed (* do not use the string "unspecified" or the empty string but just omit to specify a service).

Example

DAIA/JSON	DAIA/XML	DAIA/RDF (experimental)
`"available": [ { "service": "loan", "delay":" PT2H" ] }`	`<available service="loan" delay= "PT2H" />`	`[ ] daia:availableFor [ a daia:Service/Loan ; daia:delay "PT2H"^^xsd:duration ] .`

2.5. Unavailable element

The structure of an unavailable element is identical to the structure of the available element in most cases.

- **service?** (attribute) - see above
- **href**? (attribute) - see above
- **expected** (attribute) - A time period until the service will be available again. Use unknown or an ISO time period. If missing, then the service probably won't be available in the future. Type xs:date or xs:dateTime or the string unknown.
- **message*** (element) - see above
- **limitation*** (element) - more specific limitations of the availability status
- **queue?** (attribute) - the number of waiting requests for this service. Type xs:nonNegativeInteger.

If no expected element is given, it is not sure whether the item will ever be available, so this is not the same as setting it to unknown. If no queue element is given, it may (but does not need to) be assumed as zero. In DAIA/XML messages and limitations can be mixed in any order.

Example:

DAIA/JSON	DAIA/XML
`"unavailable": [ { "service":"presentation", "delay":"PT4H" } ]`	`<unavailable service="presentation" delay= "PT4H" />`

2.6. Messages

Messages can occur at several places in a DAIA response. The structure of a message element is:

- **lang** (attribute) - a RFC 3066 language code. Type xs:language.
- **content** (string) - the message text, a simple string without further formatting.
- **errno**? (attribute) - an error code (integer value). Type xs:integer.

Notes:

- If content is an empty string, it should be removed in DAIA encodings. Applications may treat a missing content as the empty string.
- Messages are not meant to be shown to end-users, but only used for debugging. If you need a DAIA message to transport some relevant information, you likely try to use DAIA for the wrong purpose.

Example:

In DAIA/XML the message element is a repeatable XML element with optional attributes lang and errno and the string encoded as element content. In DAIA/JSON a message element is an object with lang, errno, and string as keys. Multiple messages are combined in a JSON array:

DAIA/JSON	DAIA/XML
`"message" : [ { "content":"request failed", "lang":"en" } ]`	`<message lang= "en">request failed< /message>`

2.7. Additional entities

In this section, the additional entries institution, department, storage and limitation are discussed.

- **institution** nodes refer to an institution (e.g., the University of Hamburg), referenced by an ISIL, a hyperlink and/or a name.
- **department** nodes refer to a single department of an institution, e.g., the Faculty of Computer Science of Bielefeld University. They should be used when the institution has an own library.
- **storage** nodes deliver information about the place where an item is stored ("2nd floor").
- **limitation** nodes give information of limitations of the availability of an item

The data structure of all these nodes is identical and discussed below.

Data structure

- **id**? - a (persistent) identifier for the entity. Type xs:anyURI.
- **href**? - a URI linking to the entity. Type xs:anyURI.
- **content**? - a simple message text describing the entity.

If content is an empty string, it should be removed in DAIA encodings. Applications may treat a missing content as the empty string. It is recommended to supply an id property to point to a taxonomy or authority record and a href property to provide a hyperlink to information about the specified entity.

Examples:

DAIA/JSON

```
"institution" : { "href" : "http://www.tib.uni-hannover.de" }
...
"department" : {
                 "id" : "info:isil/DE-7-022",
                 "content" : "Library of the Geographical
Institute, Goettingen University"
               }
...
"limitation"  : { "content" : "3 day loan" }
```

DAIA/XML

```
<institution href="http://www.tib.uni-hannover.de"/>
...
<department id="info:isil/DE-7-022">Library of the
Geographical Institute, Goettingen University</department>
...
<limitation>3 day loan</limitation>
```

3. DAIA/RDF

All DAIA documents, given in DAIA/JSON or DAIA/XML can also be expressed in RDF. The DAIA ontology used for this purpose is called DAIA/RDF. The ontology is located at http://purl.org/ontology/daia/.

4. Query API

DAIA Query API

A **DAIA-API** is provided in form of a **Base URL** that can be queried by HTTP (or HTTPS) GET. The Base URL may contain fixed query parameters but it must not contain the query parameters id and format. A DAIA query is constructed of the Base URL and a query parameter **id** for the document URI to be queried for and **format** with one of xml and json. The value of the `format` parameter is case insensitive. The response muste be a DAIA/XML document for `format=xml` and a DAIA/JSON document for `format=json`. If no format parameter is given or if the parameter value is no known value, a DAIA/XML document may be returned, but you can also return other formats. In particular a DAIA-API may return DAIA/RDF in RDF/XML for `format=rdfxml` and DAIA/RDF in Turtle for `format=ttl`. The HTTP response code must be 200 for all non-empty responses (DAIA/JSON and DAIA/XML) and 404 for empty responses (for instance RDF/XML in Turtle format). Multiple document URIs can be provided by concatenating them with the vertical bar (|, %7C in URL-Encoding) but a DAIA server may limit queries to any positive number of URIs.

Examples:

Base URL	Document URIs	Format	Query URL
http://example.com/	gvk:ppn:588923168	DAIA/XML	http://example.com/?id=gvk:ppn:588923168&format=xml
http://example.com/?cmd=daia	gvk:ppn:588923168	DAIA/JSON	http://example.com/?cmd=daia&id=gvk:ppn:588923168&format=json

http://example.com/	gvk:ppn:588923168 and gvk:ppn:365058963	DAIA /JSON	http://example.com/?id=gvk:ppn:588923168%7Cgvk:ppn:365058963&format=json

DAIA Storage API

The **DAIA-Storage-API** is an additional API, similar to the DAIA-API, but to retrieve Storage information only. The request parameter are also **format** and **id** but the latter can be any Unicode character string (it is up to the specific implementation of an DAIA-Storage-API what kinds of identifiers to expect). Typical applications include mapping from call number to locations. The request is limited to any combination of the elements institution, department, storage, and message. A DAIA Storage API and a DAIA API *must not* share the same base URL.

5. References

- **[DATATYPES]** XML Schema Part 2: Datatypes Second Edition. W3C Recommendation 28 October 2004.
- **[HTTP]** Hypertext Transfer Protocol - HTTP/1.1. June 1999 (RFC 2616).
- **[JSON]** JavaScript Object Notation. (RFC 4627)
- **[URI]** Uniform Resource Identifiers (URI): Generic Syntax. August 1998 (RFC 2396).
- **[UNICODE]** The Unicode Standard Version 5.0 The Unicode Consortium, 2007.
- **[XML]** Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation 26 November 2008
- **[XML-NS]** Namespaces in XML 1.0 (Second Edition). W3C Recommendation 16 August 2006

6. Notes (Informative)

- A reference implementation in Perl is available at CPAN. It includes a simple DAIA validator and converter. A public installation of this validator is available at http://daia.gbv.de/validator/.
- All DAIA-related files are combined in a project at Sourceforge: http://sourceforge.net/projects/daia/
- Date and Time values are from a subset of ISO 8601. Even in ISO 8601 there are many ways to specify data and time - so if you need to interpret DAIA data and time values you should use a ISO 8601 library to handle all its particularities and to normalize data and time values.
- The basic structure of DAIA is unlikely to change. However until DAIA 1.0 the following parts need to be finished:
  - Definition of canonical DAIA
  - Inclusion of some additional obvious constraints like uniqueness of identifiers per document.
  - The DAIA/XML namespace (currently http://ws.gbv.de/daia/) may be changed to a more stable PURL.
- For comments and public discussion about DAIA you can use this Code4LibWiki page
- See also the draft of a broader Library Ontology

6.1 Integrity rules

If department and institution have same id, the department SHOULD be ignored.